Toward Length-Extrapolatable Transformers

Ta-Chung Chi

CMU-LTI-24-005
May 2024

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Language and Information Technology.

Keywords: Transformer, Length Extrapolation, Positional Embedding, Regular Language

Abstract

Since the advent of Transformer language models, the field of natural language processing has seen remarkable progress. Unfortunately, the complexity of training such models grows quadratically with the sequence length, making it difficult for practitioners with limited GPU resources to adopt long-sequence-length pre-training. One way to address this limitation is to allow the model to handle much longer sequences during testing without further parameter updates. This capability, known as length extrapolation, is nontrivial and presents several challenges.

Firstly, classic Transformer language models rely on per-position positional embeddings to provide positional information; this may be problematic when unseen positions are encountered during the extrapolation stage. Secondly, models pre-trained on short sequences struggle when directly fed with longer sequences due to the length distributional shift problem. Maintaining stable perplexities on longer sequences has proven challenging with existing approaches. Finally, the evaluation of length extrapolation capability often relies solely on natural language perplexity; this might not tell us the whole story as natural language is highly localized as opposed to regular language and downstream tasks such as long-context QA and code completion.

This thesis addressed the aforementioned challenges from three perspectives. **Part I** investigates the role of positional embeddings in Transformer language models. This thesis demonstrates that strong positional signals are still encoded in the hidden states of a Transformer language model, even without explicit positional embeddings. To take advantage of this, the thesis introduces a new variant of relative positional embedding named KERPLE derived from conditionally positive definite kernels. **Part II** presents a thorough analysis of existing length extrapolatable Transformers by measuring the width of models' receptive field. The key to successful length extrapolation on language modeling tasks is found to be the alignment of training and testing receptive fields. This insight leads to the proposal of a new relative positional embedding design named Sandwich, which builds upon the originally proposed Sinusoidal positional embedding. **Part III** examines the ability of Transformer's length extrapolation beyond language modeling and perplexity measurement. Motivated by the recently proposed long-context retrieval tasks, this thesis provides a better understanding of the attention mechanism and advances Transformer's implicit retrieval capability through data-dependent adjustment of the Softmax temperature. In addition, this thesis addresses Transformer's failure on formal language extrapolation tasks. Ideas from previous work such as Weight-Sharing, Adaptive-Depth, and Sliding-Window-Attention mechanisms collectively inspire a new Transformer variant named RegularGPT, which demonstrates extrapolation capability on regular language.

This thesis concludes its exploration of length-extrapolatable Transformers by suggesting various future directions. It outlines several concrete ideas that pave the way for future Transformer length extrapolation research.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Alex. Throughout my five-year journey, he never explicitly dictated what I should or shouldn't do. It was the freedom he granted me that sparked this thesis. While encouraging me to explore my interests freely, he has always been ready to offer assistance. It is not uncommon for Alex to dedicate several hours a week to brainstorming with me. Without his support, I would not have completed my Ph.D. studies. I also appreciate the guidance and intellectual contributions of my committee members, Daniel, Lei, and Dilek. They assisted and challenged me to present the best possible version of this thesis.

I extend my gratitude to Boeing for their financial support during my Ph.D. studies. They also provided me with the opportunity to apply my research to concrete aviation challenges, which significantly broadened my horizons.

I want to thank my coauthors, Ting-Han and Li-Wei, for brainstorming ideas and solving technical challenges together. Our collaboration benefited me tremendously.

Thanks go out to my friends and roommates Chai-Wei, Che-Ping, Match Punk, Strange Mochi, and Josh for their companionship and encouragement.

I am grateful to all the friends and mentors I met during my Amazon internship, including Dilek, Mahdi, Dev, Max, Mert, Prakhar, Pengfei, Yichen, Zhiyang, Vishakh, and Yen-Ting. In addition to discussing research, they made my summer wonderful and memorable.

Special thanks to my colleagues Haoyang and Xinran. You have always shared insightful thoughts with me, not only on research, but also on other aspects of life.

Finally, I would like to thank my parents and Yin, who has always been my steadfast partner, for their unconditional support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past few years, the landscape of natural language processing has undergone a transformative revolution thanks to the development of advanced Transformer-based language models [Vaswani et al., 2017b]. These models, powered by extensive self-supervised pretraining, have had a profound impact on a variety of language tasks including machine translation, text generation, sentiment analysis, and question-answering. The accuracy and fluency achieved by Transformers have reached unprecedented heights.

At the heart of these Transformer language models is the self-attention mechanism. Self-attention functions by computing interactions between all pairs of tokens, facilitating flexible message exchange. This enables models to make use of contextual nuances. Despite its empirical effectiveness, the Transformer architecture comes with a notable limitation—it demands substantial computational resources, particularly in relation to the training sequence length. This is due to the quadratic nature of the pairwise self-attention computations. Consequently, Transformers face challenges when processing lengthy input sequences, such as those found in books. Providing models trained this way is also one of the main selling points of recently released commercial language models such as GPT-4 [Achiam et al., 2023], Gemini-1.5 [Reid et al., 2024], and Claude-3 [Anthropic, 2024]. Unfortunately, their proprietary nature restricts researchers from understanding the inner workings.

The effort that the research community has put into improving the long-context processing capability of Transformer language models can be categorized into

1. **Sub-quadratic Transformer**. This line of research focuses on replacing the original self-attention mechanism with its sub-quadratic counterpart through low-rank matrix approximation [Wang et al., 2020; Choromanski et al., 2021], kernel method [Tsai et al., 2019a; Katharopoulos et al., 2020a; Choromanski et al., 2021; Peng et al., 2021], locality-sensitive hashing [Kitaev et al., 2020], and sparsification [Beltagy et al., 2020; Zaheer et al., 2020]. Because the training cost becomes sub-quadratic, these Transformer language models can be directly trained on very long sequences.

2. **Position Interpolation.** When a Transformer is fed with longer sequences, processing out-of-distribution positional indices is inevitable. One remedy is to rescale and shrink the additional indices back to the training positional index range [Chen et al., 2023]. A series of follow-up

work delves deeper into this idea [Peng et al., 2023b; Liu et al., 2024; Chen et al., 2024]. However, a costly fine-tuning process is still needed when using these methods, posing a challenge for practitioners who have limited resources.

3. **Length Extrapolation**. The development of length extrapolation often maintains unmodified self-attention and focuses on the design of positional embeddings. It allows a model trained on short sequences to handle longer sequences during inference without additional fine-tuning. Notable methods along this line often rely on novel designs of relative positional embeddings [Shaw et al., 2018; Dai et al., 2019b; Raffel et al., 2020a; Wennberg and Henter, 2021; Press et al., 2022b; Li et al., 2024].

Note that the above three directions are orthogonal but complementary: A linear Transformer might be able to further adopt the specially designed length extrapolatable positional embeddings followed by position interpolation fine-tuning to unlock even longer length extrapolation performance. Taking into account that (i) the length extrapolation idea is potentially useful for different Transformer language model architectures (ii) it obviates the need of a costly fine-tuning step, this thesis will primarily focus on investigating and improving the length extrapolation capability of Transformer language models.

1.1 Thesis Statement

Addressing the challenge of Transformer length extrapolation is the overarching goal of this thesis. A length-extrapolatable Transformer language model achieves similar or even better performance when fed with longer testing sequences, without the need for parameter updates. This thesis embarks on a journey of exploration and enhancement of the length-extrapolation prowess inherent in Transformer language models. It delves into various aspects such as positional embeddings, receptive field, self-attention distribution analysis, and a novel Transformer architecture. The advances made in this thesis facilitate the efficient processing of long sequences by Transformer language models, thus opening avenues for future long-context applications.

1.2 Outline

This thesis presents our journey on how we tackled the Transformer length extrapolation challenge. After laying the foundation of understanding positional embeddings in Chapter 2, this thesis gradually extends its scope from recency-biased tasks (Chapter 3 and 4) to a more flexible and holistic modeling of long input sequences (Chapter 5 and 6). Various methods, tools, and model architectures are proposed along the way, as detailed below:

- **Chapter 2** [Chi et al., 2023a] begins our exploration of the length extrapolation journey by completely removing positional embeddings. This is motivated by the fact that positional embeddings are the only component that is directly related to the sequence length. We find that a Transformer language model already encompasses strong positional information without positional embeddings due to the causal positional mask used to prevent future information leakage. Concretely, we measure the variance of the self-attention outputs and discover a shrinking variance effect. This finding theoretically validates the same empirical finding presented in prior work. Nevertheless, our experiment shows that a Transformer language model without positional embeddings, unfortunately, does not extrapolate well, and this forces us to turn to other solutions, such as relative positional embeddings.

- **Chapter 3** [Chi et al., 2022] presents kernelizing and generalizing the existing relative positional embeddings. We first reveal that the distance metric encoded by relative positional embeddings can be mathematically modeled by conditionally positive definite kernels. Then we transform the conditionally positive definite kernels into positive definite kernels relying on the shift-invariance property of the softmax operation of self-attention computations. By doing so, our proposed **KERPLE**, a new relative positional embedding, is the first to truly utilize longer-than-training sequence information during testing.
- **Chapter 4** [Chi et al., 2023b] dives deeper into the family of relative positional embeddings and attempts to explain why only some of them are able to extrapolate well. Inspired by the interpretability work in the computer vision field, we calculate the cumulative gradients of the input tokens to measure the empirical receptive field size of a Transformer language model. We find that extrapolation failure is often caused by the explosion of the empirical receptive field of a model. With this lesson learned, we revisit the first proposed absolute positional embedding, sinusoidal, and transform it into an extrapolatable variant named Sandwich. Unfortunately, we also find that existing extrapolatable relative positional embeddings do not model history tokens very effectively due to the local attention mechanism; this might hinder the performance on tasks that require accurate processing of every input unit, which will be addressed in the next chapter.
- **Chapter 5** [Chi et al., 2024b] investigates the implicit long-context utilization capabilities of pretrained T5 [Radford et al., 2019b] model family including Flan-T5 [Chung et al., 2022] and T5-lm-adapt [Lester et al., 2021]. Their flexible relative positional embeddings together with our proposed Softmax temperature adjustment strategies demonstrate superior performance on recently proposed long context retrieval tasks [Mohtashami and Jaggi, 2023; Li et al., 2023; Liu et al., 2023b] without any fine-tuning. In addition, our proposed method also achieves improved length extrapolation performance on multi-document QA and long code completion tasks. Finally, we provide empirical evidence and theoretical analysis that elucidate the underlying dynamics of the proposed temperature adjustment strategies.
- **Chapter 6** [Chi et al., 2023c] aims to design a new extrapolatable Transformer language model that goes beyond local attention. We draw inspiration from existing Transformer variants including Universal Transformer and Longformer and come up with the combination of three mechanisms: Weight-Sharing, Adaptive-Depth, and Sliding-Window-Attention. Our newly proposed model is named RegularGPT which can be viewed as an instantiation of working memory. Working memory is a well-known concept in psychology, in discussion of human reasoning abilities; This motivates us to rigorously benchmark RegularGPT's reasoning abilities, particularly on algorithmic tasks such as regular language. We observe much better and more robust extrapolation performance on regular language compared to

vanilla Transformer language models. Finally, RegularGPT can achieve similar language modeling and extrapolation performance on natural language, GitHub (code completion), and ArXiv (scientific articles) datasets.

- **Chapter 7** sums up our overall contributions. We will also discuss the possible future directions enabled by the work presented in this thesis.

Part I

Understanding and Improving Transformer Positional Embeddings

Chapter 2

Latent Positional Information is in the Self-Attention Variance of Transformer Language Models Without Positional Embeddings

2.1 Introduction & Related Work

Within the Transformer architecture, there are two main categories: 1) bidirectional models, such as BERT [Devlin et al., 2019a], that are trained using the masked language modeling objective, and 2) (causal) language models, such as GPT [Radford et al., 2019a], that are trained using the traditional language modeling objective. Both of these categories share the common feature of using positional embeddings to encode token distance.

Whether positional embeddings are truly essential has been a subject of ongoing research. Although they have been considered necessary for bidirectional Transformer models [Lee et al., 2019; Luo et al., 2021b; Sinha et al., 2021; Haviv et al., 2022], the situation is different for Transformer language models [Irie et al., 2019; Yang et al., 2019a; Tsai et al., 2019a; Scao et al., 2022b; Haviv et al., 2022]. In Transformer language models, removal of positional embeddings results in only a marginal decrease in performance, while enabling more efficient training [Haviv et al., 2022]. In addition to empirical evidence, it has been proven [Bhattamishra et al., 2020b] that Transformer language models without positional embeddings are Turing-complete and able to model sequences similar to recurrent neural networks [Rumelhart and McClelland, 1987; Jordan, 1986]. Despite this, it remains an open question where positional information is stored in the absence of positional embeddings. This motivates further investigation of individual operations within a Transformer layer.

In this chapter, this thesis will focus on the architecture of a pre- layer normalization (LN) [Xiong et al., 2020] multilayer Transformer language model without positional embeddings, shown in Figure 2.1.[1] We hereinafter refer to this configuration as TLM. Our primary focus is on the multi-

[1]Post-LN places layer norm at different positions. It is the configuration used in BERT [Devlin et al., 2019a].

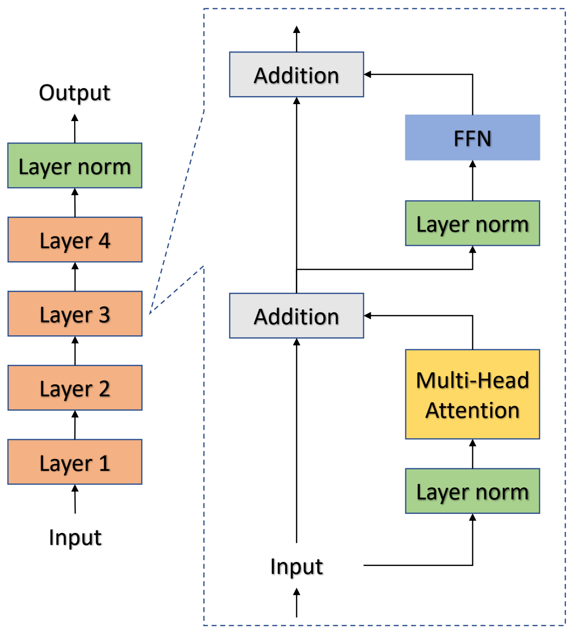


Figure 2.1: The architecture of a Pre-LN Transformer language model. All the parameters are randomly initialized and randomly sampled input is used in this chapter.

head attention (MHA) module of a randomly initialized TLM, as it is the only module that allows intertoken information exchange. To gain a deeper understanding, we compute the mean and variance of MHA outputs. To our surprise, we discover that the variance already encodes latent positional information, with later tokens in a sequence displaying smaller variance. This motivates us to quantify the variance by deriving the output distribution after MHA operations. Finally, through empirical validation using a fully pretrained TLM, we confirm that the same variance shrinkage effect persists after extensive gradient updates.

To the best of our knowledge, we are the first to identify and quantify latent positional information in TLMs. Our results provide theoretical insights into the removal of positional embeddings, enabling more efficient pretraining of future TLMs.

2.2 Probing Experiments

Given BERT and TLM (GPT) with positional embeddings removed, prior work [Haviv et al., 2022] shows that only TLM can maintain the same language modeling performance as its original version with positional embeddings. The discrepancy could be explained by the fact that only TLM encodes positional information within its layers, as shown by the position probing experiment in Haviv et al. [2022]. Since both BERT and TLM have access to the same semantic input and the only difference is the use of causal attention masks in TLM, we hypothesize that the positional information may be attributed to the interaction between causal attention masks and the TLM architecture.

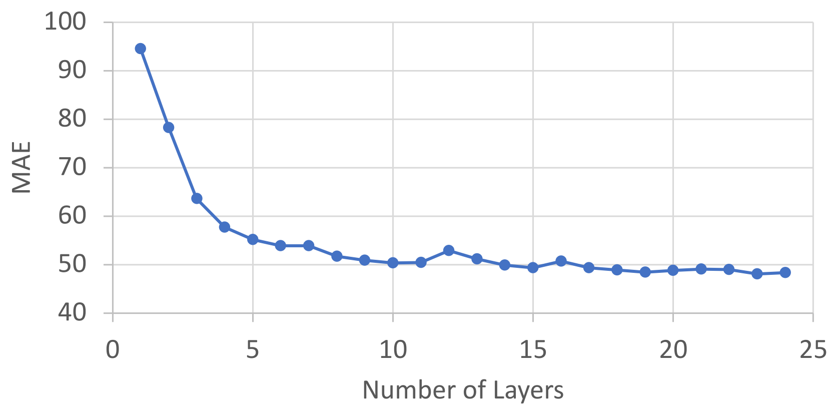


Figure 2.2: We plot the positions w.r.t their mean absolute error (MAE) for input sequence length $L = 512$. A naive baseline of predicting the middle point of $L = 256$ gives an MAE of 128. The numbers are the average of 5 seeds.

To further explore this hypothesis, we use a randomly initialized and frozen TLM to eliminate any semantic influence and focus solely on the architectural design. Additionally, to prevent the model from memorizing the order of input sequences, we do not perform embedding lookups and feed the model with randomly sampled input vectors. A trainable two-layer linear classifier with ReLU activation in between was appended to the TLM to probe the position of each token (further details can be found in § 2.8.1). We plot the mean absolute error (MAE) w.r.t. the number of Transformer layers in Figure 2.2. The plot indicates that a randomly initialized and frozen TLM with randomly sampled input vectors inherently provides positional information, with an increase in the number of layers resulting in higher probing performance. This surprising result prompts further investigation into the encoding of latent positional information within the TLM architecture.

2.3 Theoretical Analysis

We dissect the inner workings of a TLM by deriving the distribution of TLM operations in the hope that they elucidate where the latent positional information is stored. The derivation is made possible thanks to the usage of a randomly initialized and frozen TLM. We adopt the initialization settings in accordance with those employed in GPT [Radford et al., 2019a]. WLOG, our derivation is limited to the operations of the first layer in a TLM, and the FFN component is omitted (justified in § 2.3.4). The hyperparameters utilized in the simulations are: hidden dimension $d = 768$, number of attention heads $H = 12$, head dimension $d/H = 64$, sequence length $L = 512$, standard deviation for initialization $\sigma = 0.02$. All proofs of lemmas are deferred to § 2.8.

Given a sequence of randomly sampled input embeddings $\{\boldsymbol{x}_m\}_{m=1}^{L}$, where each element of $\boldsymbol{x}_m \in \mathbb{R}^d$ is sampled i.i.d. from $N(0, \sigma^2)$, a TLM consists of the following operations.

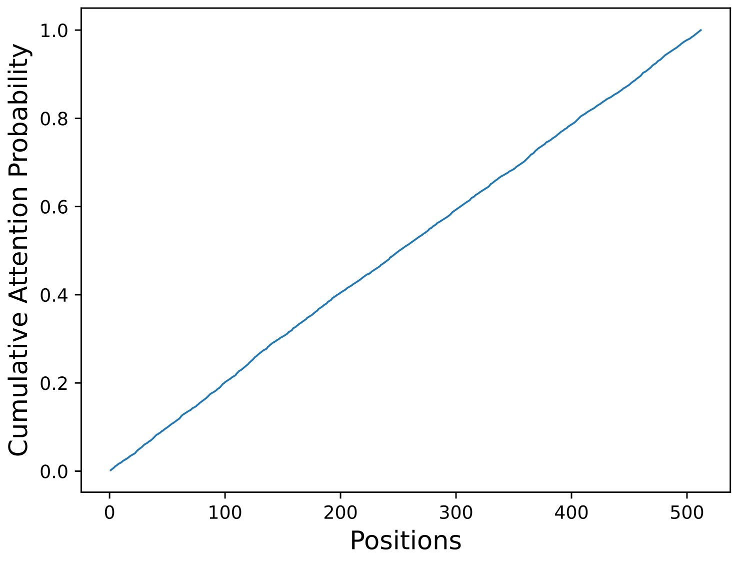


Figure 2.3: We plot the positions w.r.t their cumulative attention score for $L = 512$ averaged over 500 samples.

2.3.1 Layer Normalization

For each input embedding $\boldsymbol{x}_m$, we compute the sample mean and (biased) sample variance:

$$\overline{\boldsymbol{x}}_{m,:} = \frac{\sum_{i=1}^{d} \boldsymbol{x}_{mi}}{d}, \; S(\boldsymbol{x}_{m,:}) = \frac{\sum_{i=1}^{d} (\boldsymbol{x}_{mi} - \overline{\boldsymbol{x}}_{m,:})^2}{d}$$

Then each entry i of $\boldsymbol{x}_m$, denoted as $\boldsymbol{x}_{mi}$, is normalized by mean and variance to $\boldsymbol{e}_{mi}$:

$$\begin{aligned}\boldsymbol{e}_{mi} &= \frac{\boldsymbol{x}_{mi} - \overline{\boldsymbol{x}}_{m,:}}{\sqrt{S(\boldsymbol{x}_{m,:})}} * \gamma + \beta \\ &\overset{(*)}{\approx} \frac{\boldsymbol{x}_{mi} - \mathbb{E}[\boldsymbol{x}_{mi}]}{\sqrt{\mathbb{V}[\boldsymbol{x}_{mi}]}} \sim N(0,1),\end{aligned}$$

where $\mathbb{V}[\boldsymbol{x}]$ denotes the variance of $\boldsymbol{x}$. Since the initialization scheme sets $\gamma = 1$ and $\beta = 0$, $(*)$ holds with sufficiently large d by the law of large numbers and the continuous mapping theorem.

2.3.2 Self Attention

Each attention head computes query, key, and value vectors in $\mathbb{R}^{\frac{d}{H}}$:

$$\boldsymbol{q}_m = \boldsymbol{W}_q \boldsymbol{e}_m, \;\; \boldsymbol{k}_m = \boldsymbol{W}_k \boldsymbol{e}_m, \;\; \boldsymbol{v}_m = \boldsymbol{W}_v \boldsymbol{e}_m,$$

where $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v \in \mathbb{R}^{\frac{d}{H} \times d}$ are matrices with each element sampled i.i.d from $N(0, \sigma^2)$.

To be precise, most matrices ($\boldsymbol{W}_q^{(h)}$, $\boldsymbol{W}_k^{(h)}$, $\boldsymbol{W}_v^{(h)}$), vectors ($\boldsymbol{q}_m^{(h)}$, $\boldsymbol{k}_m^{(h)}$, $\boldsymbol{v}_m^{(h)}$), and scalars ($l_{mn}^{(h)}$, $a_{mn}^{(h)}$) are associated with a head number h. For simplicity of notation, we only show the dependency on h when necessary.

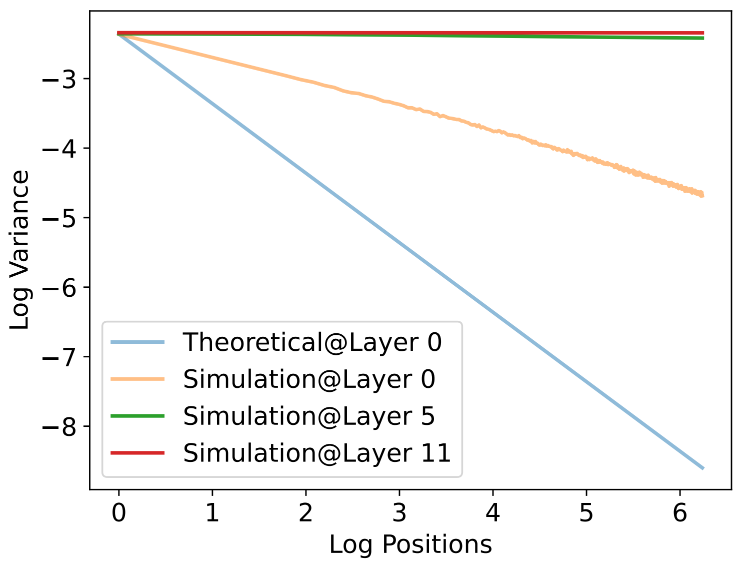


Figure 2.4: We plot the log positions (up to $L = 512$) w.r.t their log variance under the assumption of Property 2.1. The simulation aligns with the theoretical curve posited by Lemma 2.3 at the 0^{th} layer averaged over 500 samples.

Lemma 2.1. *$\boldsymbol{q}_m$, $\boldsymbol{k}_m$, and $\boldsymbol{v}_m$ have zero mean and $(d\sigma^2) \cdot I$ covariance matrix.*

The resulting vectors are processed by the self-attention module for pre-Softmax logits:

$$l_{mn} = \begin{cases} \langle \boldsymbol{q}_m, \boldsymbol{k}_n \rangle, & \text{if } m \geq n \\ -\inf, & \text{otherwise} \end{cases}$$

followed by the scaled softmax normalization:

$$a_{mn} = \frac{\exp\left(l_{mn}/\sqrt{d/H}\right)}{\sum_{i=1}^{L} \exp\left(l_{mi}/\sqrt{d/H}\right)}$$

Lemma 2.2. *l_{mn} has zero mean and $\frac{d^3\sigma^4}{H^2}$ variance. $l_{mn}/\sqrt{d/H}$ has $\frac{d^2\sigma^4}{H}$ variance.*

The numerical variance of $l_{mn}/\sqrt{d/H}$ in our case is $\frac{768^2 \cdot 0.02^4}{12} \approx 0.0079$. Lemma 2.2 suggests the following approximation:

Property 2.1. *When $\sigma^4 \ll \frac{H}{d^2}$, $l_{m,:}$ has small variance, making the attention weights $a_{m,:}$ almost evenly distributed among all positions.*[2]

In Figure 2.3, we verify Property 2.1 by showing that $\boldsymbol{a}_{mn}$ is almost evenly distributed in simulation.

Observe that the output vector $\boldsymbol{o}_m$ at position m is:

$$\boldsymbol{o}_m = \boldsymbol{W}_o \left(\oplus_{h=1}^{H} \sum_{n=1}^{L} a_{mn}^{(h)} \boldsymbol{v}_n^{(h)} \right),$$

[2]This approximation was also used in Xiong et al. [2020] except that they made a stronger assumption that $\boldsymbol{W}_q$ and $\boldsymbol{W}_k$ have to be initialized as zero matrices.

where $\oplus$ denotes the concatenation of vectors from all H attention heads. Assuming that Property 2.1 is valid and that $\boldsymbol{W}_o \in \mathbb{R}^{d\times d}$ has elements i.i.d sampled from $N(0, \sigma^2)$, we derive the distribution of $\boldsymbol{o}_m$ below.

Lemma 2.3. *$\boldsymbol{o}_m$ has zero mean and $\frac{d^2\sigma^4}{m}I$ covariance matrix.*

Figure 2.4 is a simulation that verifies Lemma 2.3 under the assumption of Property 2.1. We can see that *the variance of $\boldsymbol{o}_m$ already encodes the positional information m.*

2.3.3 Residual Connection

As indicated by the *Addition* block of Figure 2.1, the residual connection sets the output as $\boldsymbol{y}_m = \boldsymbol{x}_m + \boldsymbol{o}_m$. It allows the model to pass the first MHA output to subsequent MHA modules as well as the final classifier. As the positional information has been passed through the residual connection, we omit the FFN part in our analysis.

2.3.4 The Final Layer Normalization

Layer normalization is an operation that could eliminate the positional information derived in Lemma 2.3, which occurs before the MHA modules and position classifier. As mentioned in § 2.3.1, LN$(\boldsymbol{y}_m)$ gives:

$$\boldsymbol{y}'_{mi} \approx \frac{\boldsymbol{y}_{mi} - \mathbb{E}[\boldsymbol{y}_{mi}]}{\sqrt{\mathbb{V}[\boldsymbol{y}_{mi}]}} \approx \frac{\boldsymbol{x}_{mi} + \boldsymbol{W}_o\boldsymbol{W}_v\frac{\sum_n^m \boldsymbol{e}_{ni}}{m}}{\sqrt{\sigma^2 + \frac{d^2\sigma^4}{m}}},$$

$$\begin{aligned}\mathbb{E}[\boldsymbol{y}_{mi}] = 0,\ \mathbb{V}[\boldsymbol{y}_{mi}] &= \mathbb{V}[\boldsymbol{x}_{mi}] + \mathbb{V}[\boldsymbol{o}_{mi}]\\ &= \sigma^2 + \frac{d^2\sigma^4}{m}\end{aligned}$$

Lemma 2.4. *The variance of the j-th dimension of LN$(\boldsymbol{y}_m)$ is:*

$$\frac{m\sigma^2 + \sum_i (\boldsymbol{W}_{o,j:}\boldsymbol{W}_{v,:i})^2}{m\sigma^2 + d^2\sigma^4},$$

where $\boldsymbol{W}_{o,j:} \in \mathbb{R}^{1\times d}$ is the j-th row of $\boldsymbol{W}_o$. $\boldsymbol{W}_{v,:i} \in \mathbb{R}^{d\times 1}$ is the i-th column of $\boldsymbol{W}_v$. As long as $\sum_i(\boldsymbol{W}_{o,j:}\boldsymbol{W}_{v,:i})^2 \neq d^2\sigma^4$, the classifier should be able to exploit the discrepancy to derive m.

The readers might wonder why $\boldsymbol{W}_{o,j:}$ and $\boldsymbol{W}_{v,:i}$ in the numerator cannot be treated as random variables. The reason is that we focus only on one dimension (j-th) at a time. This means that we cannot use the law of large numbers to approximate the sample variance of $\boldsymbol{y}_{mj}$ as we did for the denominator.

2.3.5 Relaxing the Assumptions

We discuss possible relaxation of the assumptions used in § 2.3.2.

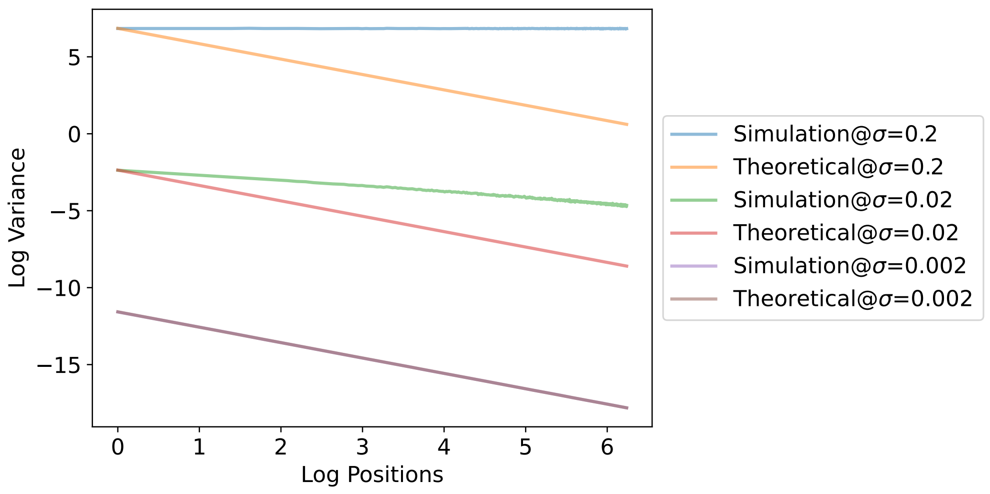


Figure 2.5: We vary the value of σ and show its effect at the 0th layer. As we can see, a smaller value of σ brings Lemma 2.3 into alignment with the corresponding simulation more closely. Note that the two lines completely overlap when $\sigma = 0.002$. Average of 500 samples.

What if Property 2.1 does not hold? Or equivalently, $\sigma^4 \not\ll \frac{H}{d^2}$. This prompts us to vary the value of σ. In Figure 2.5, we see that the smaller σ better aligns Lemma 2.3 with the simulations, which is unsurprising as Lemma 2.3 assumes small σ. Even when σ is not too small (that is, $\sigma = 0.2, 0.02$), the variance still encodes the positional information as the variance of $\boldsymbol{o}_m$ is negatively correlated with its position m.

Other Initialization Schemes So far we assume that the weight matrices ($\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v$, $\boldsymbol{W}_o$) are initialized i.i.d. from $N(0, \sigma^2)$. However, we can relax the assumption to i.i.d. samples from a distribution with zero mean and finite variance. This is because the proof in § 2.8 calculates the covariance. The variance calculation relies on $\mathbb{E}[\boldsymbol{r}_i \boldsymbol{r}_i^\top] = \sigma^2 I$ where $\boldsymbol{r}_i^\top$ is the i-th row vector of a weight matrix. This property holds for any distribution with zero mean and σ^2 variance.

2.4 Analysis of Previous Discoveries

Why are the positions of later tokens in a sequence harder to be predicted in Figure 3 of Haviv et al. [2022]? Lemma 2.3 states that the variance is inversely proportional to the position m, so the variance of later tokens (large m) plateaus, resulting in a more difficult numerical optimization problem. This also suggests a potential downside of removing positional embeddings. It might be challenging for the model to infer positional information of the later tokens in extremely long input sequences.

Why do lower layers (closer to input) give worse probing performances in both Figure 2.2 and Haviv et al. [2022]? This can be explained by Figure 2.4. Most of the positions at the 0th layer have a tiny variance ($\exp(-10) = 4.5e^{-5}$), again presenting a difficult numerical optimization problem.

Why does BERT fail to converge without positional embeddings? In a BERT model [Devlin et al., 2019a], each token has access to all other tokens, causing the variance at all positions $\frac{d^2\sigma^4}{L}$. Therefore, a BERT model cannot utilize variance differences as its positional indicator.

2.5 Post-Training Results

Our derivations apply only to the initial stage where the TLM and input embeddings are randomly initialized, and this may not hold true after gradient updates. It is therefore essential to verify the existence of variance properties and lemmas on a fully pre-trained TLM on the OpenWebText2 dataset (details in § 2.8.2).

We expect that the properties of lower layers of a pre-trained TLM should align more closely with the theoretical results for two reasons: 1) There are more steps between the lower layers and the final language modeling loss, resulting in smaller gradients and thereby fewer parameter updates, and 2) lower layers typically encode more low-level information dependent on positional information [Vulić et al., 2020; de Vries et al., 2020]. Figures 2.6 and 2.7 demonstrate that the 0^{th} (lowest) layer exhibits a highly similar cumulative attention probability and decay-with-position variance, as shown in the theoretical results. On the contrary, higher layers deviate from the analyses in § 2.3. We posit that the model learns to rely more heavily on semantic rather than positional information. This also explains why predicting positions using outputs of higher Transformer layers is more challenging, as demonstrated in Figure 2 of Haviv et al. [2022].

2.6 Conclusion

We mathematically analyzed a randomly initialized Transformer language model without positional embeddings. We showed that the variance of the self-attention output decreases as the position increases, which serves as an indicator for positional information. We validated that, after extensive gradient updates, the lower layers of a pre-trained language model still exhibit highly similar variance-reduction behavior. Our results pave the way for the pre-training of more efficient and positional embedding-free Transformer language models.

2.7 Limitations

The limitations of this chapter come mostly from our assumptions: 1) a randomly initialized and frozen TLM, and 2) Input tokens are all different and randomly sampled. These two assumptions obviously do not hold for human language and pre-trained TLMs. Therefore, we also attempt to empirically verify the existence of lemmas and properties on a pre-trained TLM without positional embeddings in § 2.5.

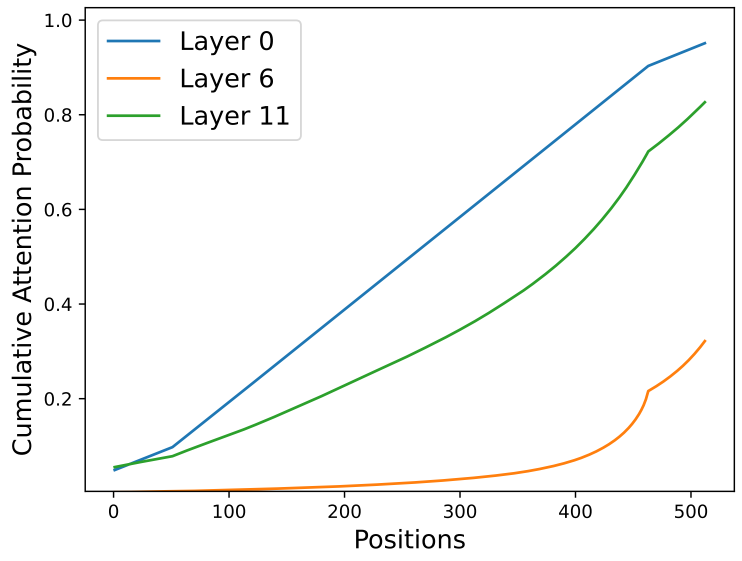


Figure 2.6: We plot the positions w.r.t their cumulative attention probability for $L = 512$ of a pre-trained TLM. We average over all heads in a layer and 500 samples.

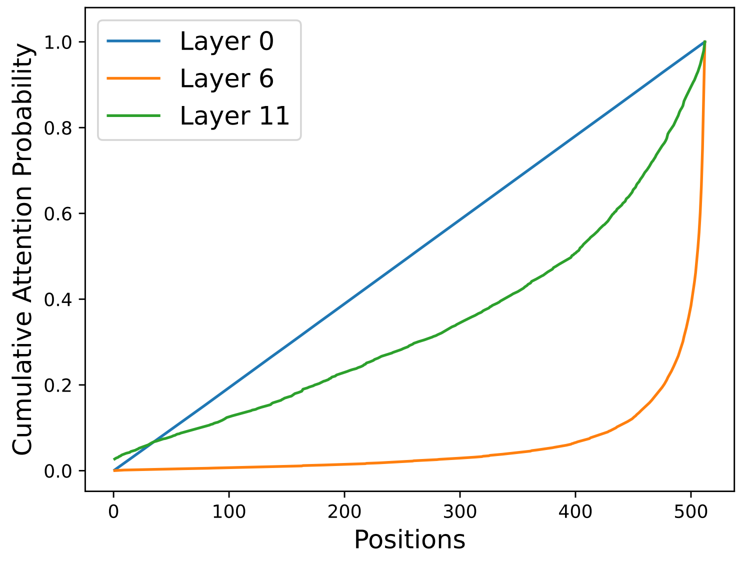


Figure 2.7: We plot the log positions w.r.t their log variance for $L = 512$ of a pre-trained TLM. We average over 500 samples.

That being said, several methods could be attempted to remove these assumptions. For example, we could analyze the training dynamics of a TLM to shed light on the model parameter distribution after pretraining. Alternately, Zipf's law or a simple n-gram language model could be used to quantify the degree of input token duplication in human languages. This might provide a more accurate estimate of the variance at different positions. We leave these ideas for future work.

# Layers	Hidden Size	# Attention Heads	Train Seq. Len.	# Trainable Params.
12	64	12	512	162M
Optimizer	Batch Size	Train Steps	Precision	Dataset
Adam (lr 6e-4)	32	50,000	bfloat16	OpenWebText2

Table 2.1: Pre-trained Model Configurations.

2.8 Proofs and Experimental Details

The proof of Lemmas 2.1 and 2.2 are head-dependent, while that of Lemma 2.3 is head-independent. For notation simplicity, in Lemmas 2.1 and 2.2, we remove the head dependency on matrices ($\boldsymbol{W}_q^{(h)}$, $\boldsymbol{W}_k^{(h)}$, $\boldsymbol{W}_v^{(h)}$), vectors ($\boldsymbol{q}_m^{(h)}$, $\boldsymbol{k}_m^{(h)}$, $\boldsymbol{v}_m^{(h)}$), and scalars ($l_{mn}^{(h)}$, $a_{mn}^{(h)}$).

Proof of Lemma 2.1 Here, we use $\boldsymbol{r}_i^\top$ to denote the i-th row vector of $\boldsymbol{W}_v$.

$$\begin{aligned}
\text{cov}(\boldsymbol{v}_m, \boldsymbol{v}_n) &= \mathbb{E}[\boldsymbol{v}_m \boldsymbol{v}_n^\top] \\
&= \mathbb{E}[\boldsymbol{W}_v \boldsymbol{e}_m \boldsymbol{e}_n^\top \boldsymbol{W}_v^\top] \\
&= \mathbb{E}\left[\begin{bmatrix} \boldsymbol{r}_1^\top \boldsymbol{e}_m \\ \vdots \\ \boldsymbol{r}_{\frac{d}{H}}^\top \boldsymbol{e}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_n^\top \boldsymbol{r}_1 & \dots & \boldsymbol{e}_n^\top \boldsymbol{r}_{\frac{d}{H}} \end{bmatrix}\right] \\
&= \left[\mathbb{E}[\boldsymbol{r}_i^\top \boldsymbol{e}_m \boldsymbol{e}_n^\top \boldsymbol{r}_j]\right]_{i,j=1}^{\frac{d}{H}} \\
&= \left[\mathbb{E}[\text{Tr}(\boldsymbol{r}_j \boldsymbol{r}_i^\top \boldsymbol{e}_m \boldsymbol{e}_n^\top)]\right]_{i,j=1}^{\frac{d}{H}} \\
&= \left[\text{Tr}(\mathbb{E}[\boldsymbol{r}_j \boldsymbol{r}_i^\top]\mathbb{E}[\boldsymbol{e}_m \boldsymbol{e}_n^\top])\right]_{i,j=1}^{\frac{d}{H}} \\
&\overset{(*)}{=} \left[\text{Tr}((\mathbb{1}_{i=j}\sigma^2) \cdot I_d \cdot \mathbb{1}_{m=n} \cdot I_d)\right]_{i,j=1}^{\frac{d}{H}} \\
&= \left[\mathbb{1}_{i=j}\mathbb{1}_{m=n} d\sigma^2\right]_{i,j=1}^{\frac{d}{H}} \\
&= (\mathbb{1}_{m=n} d\sigma^2) \cdot I_{d/H}
\end{aligned}$$

$(*)$ holds because $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ are independent when $i \neq j$ (similarly for $\boldsymbol{e}_m$ and $\boldsymbol{e}_n$) and the covariance of a Gaussian random vector is an identity matrix. I_d and $I_{d/H}$ denote $d \times d$ and $\frac{d}{H} \times \frac{d}{H}$ identity matrices.

Proof of Lemma 2.2 Here, we use $\boldsymbol{r}_i^\top$ to denote the i-th row vector of $\boldsymbol{W}_q$ and $\boldsymbol{W}_k$.

$$\begin{aligned}
&\text{cov}(l_{mn}, l_{mp}) \\
&\quad = \mathbb{E}[(\boldsymbol{e}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_n)(\boldsymbol{e}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_p)^\top] \\
&\quad = \mathbb{E}[\text{Tr}(\boldsymbol{e}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_n \boldsymbol{e}_p^\top \boldsymbol{W}_k^\top \boldsymbol{W}_q \boldsymbol{e}_m)] \\
&\quad = \mathbb{E}[\text{Tr}(\boldsymbol{e}_m \boldsymbol{e}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_n \boldsymbol{e}_p^\top \boldsymbol{W}_k^\top \boldsymbol{W}_q)] \\
&\quad = \text{Tr}(\mathbb{E}[\boldsymbol{e}_m \boldsymbol{e}_m^\top]\mathbb{E}[\boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_n \boldsymbol{e}_p^\top \boldsymbol{W}_k^\top \boldsymbol{W}_q]) \\
&\quad = \mathbb{E}[\text{Tr}(\boldsymbol{e}_n \boldsymbol{e}_p^\top \boldsymbol{W}_k^\top \boldsymbol{W}_q \boldsymbol{W}_q^\top \boldsymbol{W}_k)] \\
&\quad = \text{Tr}(\mathbb{E}[\boldsymbol{e}_n \boldsymbol{e}_p^\top]\mathbb{E}[\boldsymbol{W}_k^\top \boldsymbol{W}_q \boldsymbol{W}_q^\top \boldsymbol{W}_k)]) \\
&\quad = (\mathbb{1}_{n=p})\text{Tr}(\mathbb{E}[\boldsymbol{W}_q \boldsymbol{W}_q^\top]\mathbb{E}[\boldsymbol{W}_k \boldsymbol{W}_k^\top]) \\
&\quad \overset{(*)}{=} (\mathbb{1}_{n=p})\text{Tr}((\frac{d}{H}\sigma^2 \cdot I)(\frac{d}{H}\sigma^2 \cdot I)) \\
&\quad = (\mathbb{1}_{n=p})\frac{d^3\sigma^4}{H^2}
\end{aligned}$$

$(*)$ holds since:

$$\begin{aligned}
\mathbb{E}[\boldsymbol{W}_q \boldsymbol{W}_q^\top] &= \mathbb{E}\left[\begin{bmatrix} \boldsymbol{r}_1^\top \\ \vdots \\ \boldsymbol{r}_{\frac{d}{H}}^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 & \dots & \boldsymbol{r}_{\frac{d}{H}} \end{bmatrix}\right] \\
&= \left[\mathbb{E}[\boldsymbol{r}_i^\top \boldsymbol{r}_j]\right]_{i,j=1}^{\frac{d}{H}} = \frac{d}{H}\sigma^2 \cdot I
\end{aligned}$$

Proof of Lemma 2.3 Because $\boldsymbol{W}_o \in \mathbb{R}^{d \times d}$ is applied on a concatenation of vectors at all heads, we take $\boldsymbol{v}_i = \oplus_{h=1}^{H} \boldsymbol{v}_i^{(h)}$. $\boldsymbol{v}_i$ here is head-independent while $\boldsymbol{v}_i$ at Lemma 2.1 is head-dependent. Here, we use $\boldsymbol{r}_i^\top$ to denote the i-th row vector of $\boldsymbol{W}_o$.

$$
\begin{aligned}
&\mathrm{cov}(\boldsymbol{o}_m, \boldsymbol{o}_m) \\
&\overset{\text{Property 2.1}}{\approx} \mathbb{E}\left[\boldsymbol{W}_o \frac{\sum_{i=1}^m \boldsymbol{v}_i}{m} \frac{\sum_{j=1}^m \boldsymbol{v}_j^\top}{m} \boldsymbol{W}_o^\top\right] \\
&= \frac{1}{m^2} \sum_{i,j=1}^m \mathbb{E}[\boldsymbol{W}_o \boldsymbol{v}_i \boldsymbol{v}_j^\top \boldsymbol{W}_o^\top] \\
&= \frac{1}{m^2} \sum_{i,j=1}^m \mathbb{E}\left[\begin{bmatrix} \boldsymbol{r}_1^\top \boldsymbol{v}_i \\ \vdots \\ \boldsymbol{r}_d^\top \boldsymbol{v}_i \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_j^\top \boldsymbol{r}_1 & \dots & \boldsymbol{v}_j^\top \boldsymbol{r}_d \end{bmatrix}\right] \\
&= \frac{1}{m^2} \sum_{i,j=1}^m \left[\mathbb{E}[\boldsymbol{r}_k^\top \boldsymbol{v}_i \boldsymbol{v}_j^\top \boldsymbol{r}_l]\right]_{k,l=1}^d \\
&= \frac{1}{m^2} \sum_{i,j=1}^m \left[\mathbb{E}[\mathrm{Tr}(\boldsymbol{r}_l \boldsymbol{r}_k^\top \boldsymbol{v}_i \boldsymbol{v}_j^\top)]\right]_{k,l=1}^d \\
&= \frac{1}{m^2} \sum_{i,j=1}^m \left[\mathrm{Tr}(\mathbb{E}[\boldsymbol{r}_l \boldsymbol{r}_k^\top] \mathbb{E}[\boldsymbol{v}_i \boldsymbol{v}_j^\top])\right]_{k,l=1}^d \\
&\overset{(*)}{=} \frac{1}{m^2} \sum_{i,j=1}^m \Big[\mathrm{Tr}((\mathbb{1}_{k=l}\sigma^2) \cdot I \\
&\qquad\qquad \cdot (\mathbb{1}_{i=j} d\sigma^2) \cdot I)\Big]_{k,l=1}^d \\
&= \frac{d^2\sigma^4}{m} I
\end{aligned}
$$

$(*)$ follows from Lemma 2.1: because $\mathrm{cov}(\boldsymbol{v}_i^{(h)}, \boldsymbol{v}_j^{(h)}) = (\mathbb{1}_{i=j} d\sigma^2) \cdot I_{d/H}$, a concatenation for all $h \in H$ gives $\mathbb{E}[\boldsymbol{v}_i \boldsymbol{v}_j^\top] = (\mathbb{1}_{i=j} d\sigma^2) \cdot I_d$.

2.8.1 Probing Experiment Details

We train a randomly initialized and frozen TLM with 12 layers, $d = 768$, $H = 12$, $L = 512$, and $\sigma = 0.02$. We use the Adam optimizer [Kingma and Ba, 2014] with learning rate $1e - 3$ and 5000 gradient updates. The batch size is set to 32. We implement our model using PyTorch [Paszke et al., 2019].

2.8.2 Pre-trained Transformer Language Model Details

We use the gpt-neox library [Andonian et al., 2021] to train a TLM with no positional embeddings. Detailed hyperparameters are listed in Table 2.1. The pretraining takes 5 hours on one NVIDIA A100-40GB.

2.8.3 Scientific Artifacts

We use the gpt-neox library [Andonian et al., 2021] under Apache-2.0 license. OpenWebText2 [Gao et al., 2020a] is released by the authors of gpt-neox. The codebase and dataset are publicly released for research purposes. The steps taken to protect privacy and anonymization are discussed in Section 6 and 7 of Gao et al. [2020a]. The distribution and statistics of OpenWebext2 are also discussed in Gao et al. [2020a].

Chapter 3

Kernelizing Relative Positional Embeddings for Transformer Length Extrapolation

3.1 Introduction

We have seen how variance can serve as the implicit positional indicator for a Transformer language model without positional embeddings. However, § 2.4 suggests the difficulty in performing length extrapolation due to the fact that the extrapolated tokens share similar variance. Therefore, we turn our attention to designs of length-extrapolatable positional embeddings. While recent work on absolute positional embeddings demonstrated the length extrapolation capability [Kiyono et al., 2021; Likhomanenko et al., 2021], it is believed that relative positional embeddings are more robust to input length change [Likhomanenko et al., 2021], for example, ALiBi [Press et al., 2022b] and T5 [Raffel et al., 2020a]. Hence, we are motivated to study the inner workings of relative positional embeddings.

Relative positional embeddings (RPE) encode the idea of shift-invariance: for any shift p, $(m + p) - (n + p) = m - n$. It is often added directly to the self-attention matrix before Softmax normalization [Chen et al., 2021b]. Inspired by shift-invariance and the ability of a kernel to define a similarity function, there have been studies on shift-invariant kernels for RPE [Wennberg and Henter, 2021] with a focus on the Gaussian kernel. However, in our preliminary experiments, the Gaussian kernel demonstrates a limited length extrapolation ability (see § 3.7.3). Hence, a distinct class of shift-invariant kernels is needed to achieve adequate length extrapolation.

To this end, we note a set of well-established conditionally positive definite (CPD) kernels suitable for modeling distance metrics [Schölkopf, 2000]. However, CPD kernels do not conform to an inner product. We can remedy this issue by transforming a CPD kernel into a PD kernel by adding a sufficiently large constant. This constant offset is subsequently implicitly absorbed in the Softmax normalization (see the discussion below of Eq. (3.2)). For example, ALiBi implicitly admits a PD kernel of the form $c - |m - n|$ (see the end of § 3.4), which is reduced to a CPD kernel $-|m - n|$. The combination of CPD kernel and Softmax normalization opens the door to a sea of possible CPD kernels. We investigate structures from this class that exhibit a strong length extrapolation ability, such as ALiBi.

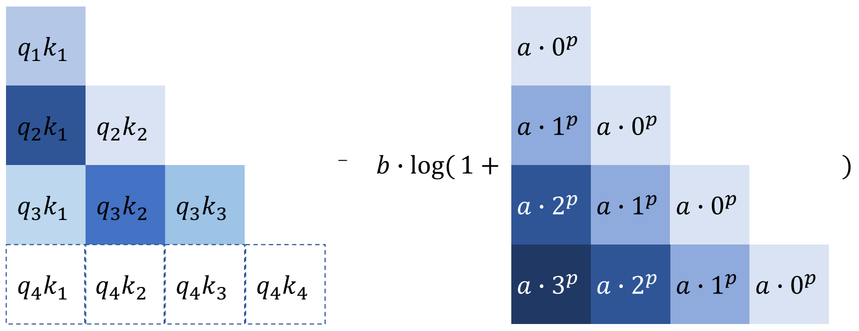


Figure 3.1: The 3-Para-Log variant of the proposed KERPLE framework. a, b, and p are learnable parameters in each attention head shared across layers. Since # of heads is H, there are $3 \cdot H$ learnable parameters. The learnable parameters are trained with length-3 sequences. At inference time, the last row (in dashed squares) becomes active, and the model extrapolates to length-4 sequences. Note that we focus on causal language modeling following ALiBi, so the matrices are triangular.

Our main result is a framework for **KE**rnelize **R**elative **P**ositional Embedding for **L**ength **E**xtrapolation (**KERPLE**). The framework elucidates key principles that encourage the length extrapolation property. We show that ALiBi is a particular instance within our framework. Our subsequent experiments suggest that the proposed method yields better length extrapolation on large datasets such as OpenWebText2, GitHub, and ArXiv.

3.2 Background and Related Work

3.2.1 Preliminary

Let $\{w_m\}_{m=1}^L$ be the input tokens to a Transformer model, where L is the total number of tokens. Each w_m is a scalar and is used to index the embedding vector $\boldsymbol{e}_m \in \mathbb{R}^d$ as input to the Transformer. A Transformer converts each $\boldsymbol{e}_m$ into query, key, and value vectors in $\mathbb{R}^d$: $\boldsymbol{q}_m = \boldsymbol{W}_q \boldsymbol{e}_m$, $\boldsymbol{k}_m = \boldsymbol{W}_k \boldsymbol{e}_m$, $\boldsymbol{v}_m = \boldsymbol{W}_v \boldsymbol{e}_m$, where $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v \in \mathbb{R}^{d \times d}$ are learnable matrices. Then, the self-attention module computes the scaled attention scores and generates the output vector $\boldsymbol{o}_m$ at position m as:

$$a_{m,n} = \frac{\exp(\boldsymbol{q}_m^\top \boldsymbol{k}_n / \sqrt{d})}{\sum_{i=1}^L \exp(\boldsymbol{q}_m^\top \boldsymbol{k}_i / \sqrt{d})}, \quad \boldsymbol{o}_m = \sum_{n=1}^L a_{m,n} \boldsymbol{v}_n.$$

Since the operation is position-agnostic, it is believed that positional information helps model token interactions [Vaswani et al., 2017b], which we survey in the next subsection.

3.2.2 Positional Embedding

Absolute. Absolute positional embeddings assign a positional vector $\boldsymbol{p}_m$ to each position m and add $\boldsymbol{p}_m$ to the embedding vector $\boldsymbol{e}_m$. The very first version of which is the predefined sinusoidal function [Vaswani et al., 2017b]. Following the success of BERT [Devlin et al., 2019b], learnable absolute positional embeddings have been applied to the task of masked language modeling [Devlin et al., 2019b; Liu et al., 2019; Clark et al., 2020; Lan et al., 2020], autoregressive decoding [Radford et al., 2018, 2019b], and sequence-to-sequence [Gehring et al., 2017; Lewis et al., 2019] settings. Recent work studied ways to extrapolate sinusoidal positional embeddings to longer sequences by randomly shifting absolute positions during training [Kiyono et al., 2021] or increasing with continuous signals [Likhomanenko et al., 2021].

Relative. As opposed to the modeling of the absolute position m, relative positional embeddings (RPE) that model the positional difference $m - n$ have become popular in the literature [Shaw et al., 2018; Huang et al., 2019; Dai et al., 2019b; Yang et al., 2019b; Huang et al., 2020; He et al., 2021; Ke et al., 2021; Chen et al., 2021b]. In particular, the T5 model that considers bucketed relative distances and log-binning has been shown to perform well on various Transformer architectures [Raffel et al., 2020a]. Rotary positional embedding [Su et al., 2021b] encodes the position with rotations: $f(\boldsymbol{q}_m, m) = R_m \boldsymbol{q}_m$ where R_m is a rotation matrix with angles proportional to m. With the rotation's property, the query-key product exhibits a positional difference: $f(\boldsymbol{q}_m, m)^\top f(\boldsymbol{k}_n, n) = \boldsymbol{q}_m^\top R_{n-m} \boldsymbol{k}_n$.

We note that the overview above focuses on the NLP domain. Recent work has applied positional embeddings to other domains such as vision [Wu et al., 2021c] and speech [Likhomanenko et al., 2021]. A survey can be found in [Dufter et al., 2022].

3.2.3 Kernel and its Application in Transformer

The kernel trick is a classic approach to generalize the inner product to high dimensional spaces [Mika et al., 1998; Schölkopf, 2000; Leslie et al., 2001; Dhillon et al., 2004; Takeda et al., 2007]. In the context of Transformers, there has been interest in applying kernels to the self-attention structure to enhance the performance. Examples of such work include kernel for positional embeddings [Tsai et al., 2019b; Wu et al., 2021a; Wennberg and Henter, 2021; Luo et al., 2021a]. Another line of research leverages the kernel's feature map [Rahimi and Recht, 2007] to linearize the self-attention module and reduce the computational cost [Katharopoulos et al., 2020b; Chen et al., 2021c; Xiong et al., 2021; Peng et al., 2021; Choromanski et al., 2021; Qin et al., 2022].

3.3 Theoretical Foundations of CPD Kernels

3.3.1 PD and CPD Kernels

In this chapter, we use shift-invariant conditionally positive definite (CPD) kernels to model the effect of relative positional differences. We propose this formulation because the notion of *relative* is modeled by a shift-invariant function: a bivariate function k over two positions (m, n) such that $k(m, n) = f(m - n)$ for some univariate f. The notion of *positional difference* $m - n$ is generalized by the CPD kernel. We review the definitions of PD and CPD kernels below.

Definition 3.1 (PD Kernel)**.** *A (real) symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel if for any integer N and any $\{x_i \in \mathcal{X}\}_{i=1}^N$, $\{c_i \in \mathbb{R}\}_{i=1}^N$, the quadratic form is nonnegative: $\sum_{i=1}^N \sum_{j=1}^N c_i c_j k(x_i, x_j) \geq 0$.*

Definition 3.2 (CPD Kernel)**.** *A (real) symmetric function $\tilde{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a conditionally positive definite kernel if for any integer N and any $\{x_i \in \mathcal{X}\}_{i=1}^N$, the quadratic form is conditionally nonnegative: $\sum_{i=1}^N \sum_{j=1}^N c_i c_j \tilde{k}(x_i, x_j) \geq 0$ for $\{c_i \in \mathbb{R}\}_{i=1}^N$ with $\sum_{i=1}^N c_i = 0$.*

Fact 3.1 (Berg et al. [1984] and Prop. 5 of Schölkopf [2000])**.** *Let $\tilde{k} : \mathcal{X} \times \mathcal{X} \to (-\infty, 0]$ be a CPD kernel with $\tilde{k}(x, x) = 0 \; \forall x \in \mathcal{X}$. Then, there exists a Hilbert space $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \to \mathcal{H}$ such that $\|\phi(x) - \phi(x')\|^2 = -\tilde{k}(x, x')$.*

Fact 3.1 suggests that CPD kernels generalize distance metrics to high dimensional spaces. Since we are interested in positional differences, we examine modeling the distance between positions using CPD kernels.

However, Fact 3.1 also implies that CPD kernels do not encode inner products as required by self-attention for the computation of pairwise relations. PD kernels represent inner products. To better understand the effect of CPD kernels on self-attention, we need to establish relations between CPD and PD kernels. As noted in Schölkopf [2000], if one takes any PD kernel and offsets it by a constant, the result is at least a CPD kernel. In the next subsection, we show that the converse is *nearly* true: if $\tilde{k}$ is CPD, so is $c + \tilde{k}$ for large enough $c \in \mathbb{R}$ (Lemma 3.1). Therefore, we may generate the CPD kernels of interest and transform them into PD kernels if needed.

3.3.2 Constructing PD Kernels From CPD Kernels via Constant Shifts

In this subsection, we review a few properties of CPD kernels and use these to generate a variety of CPD kernels. Then, we present a lemma that transforms CPD kernels into PD kernels via constant shifts. This enables the production of a family of PD kernels from CPD kernels. Finally, we present our critical observation that the exact value of the constant shift is not needed, thanks to a nice property of Softmax normalization.

Below are some important facts about CPD kernels.

Fact 3.2 (Scaling and Summation)**.** *If $\tilde{k}_1$ and $\tilde{k}_2$ are CPD, then so are $a \cdot \tilde{k}_1$ (for $a > 0$) and $\tilde{k}_1 + \tilde{k}_2$.*

Fact 3.3 (Berg et al. [1984] and Prop. 4 of Schölkopf [2000])**.** *If $\tilde{k} : \mathcal{X} \times \mathcal{X} \to (-\infty, 0]$ is CPD, then so are $-(-\tilde{k})^\alpha$ for $0 < \alpha < 1$ and $-\log(1 - \tilde{k})$.*

Fact 3.4 (Schölkopf [2000], page 3)**.** *The negative squared distance $-\|x - x'\|^2$ is CPD.*

The three Facts above jointly yield a rich family of CPD kernels as shown below.

Corollary 3.1. *The following are CPD kernels.*

(a) $\tilde{k}(x, x') = -a\|x - x'\|^p$ *with* $0 < p \leq 2$ *and* $a > 0$.
(b) $\tilde{k}(x, x') = -b \cdot \log(1 + a\|x - x'\|^p)$ *with* $0 < p \leq 2$ *and* $a, b > 0$.

We note that it is possible to keep iterating between Fact 3.2 and 3.3 and generate more complicated examples, e.g., $-a\|x - x'\|^p - b \cdot \log(1 + a\|x - x'\|^p)$ or $-b \cdot \log(1 + a\|x - x'\|^p)^c$ for $0 < c < 1$. However, since relative positional embeddings are of our interest, we only consider simple CPD kernels. Those with complicated forms are deferred to future work.

Now that Corollary 3.1 has presented a few class of CPD kernels, we prove a lemma (in § 3.7.1) that constructs PD kernels from CPD kernels through shifting. Later in Eq. (3.2), we will see that the shifting construction is combined neatly with the Softmax normalization of self-attention.

Lemma 3.1 (CPD Shift Lemma. Proof in § 3.7.1)**.** *Let $\tilde{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a CPD kernel. There exists $c \geq 0$ such that $c + \tilde{k}$ is a PD kernel.*

Lemma 3.1 implies the CPD kernels in Corollary 3.1 can be made PD if a large enough constant is added. For example, $c - \|x - x'\|^p$ for large enough c. Although Lemma 3.1 does not have an explicit construction of c, thanks to the shift-invariant property of the Softmax normalization, we can leave it as an under-determined constant in our positional embedding design (Eq. (3.1) in section 3.4). Given a set of test points $\{x_i\}_{i=1}^N$, one can do a geometric sequence search[1] to search for a c such that the $N \times N$ matrix $[c + \tilde{k}(x_i, x_j)]_{i,j=1}^N \succeq 0$. Hence, we do not need the value of c , but we can compute it if needed, e.g., deriving the feature map of $c + \tilde{k}$.

Alternative Proof of $c - \|x - x'\|^p$. While the CPD shift lemma is convenient, one can prove $c - \|x - x'\|^p$ is PD for large enough c using a kernel representation theorem in Schoenberg [1938]. See § 3.7.2 for details.

3.4 Kernelized Relative Positional Embedding

Let $\{\boldsymbol{q}_m\}_{m=1}^L$ and $\{\boldsymbol{k}_n\}_{n=1}^L$ be the input queries and keys. Let $(r_1, ..., r_\ell)$ be learnable parameters. We propose a kernelized relative positional embedding as follows.

$$a_{m,n} = \frac{\exp\left((\boldsymbol{q}_m^\top \boldsymbol{k}_n + \tilde{k}_{r_1,\dots,r_\ell}(m,n))/\sqrt{d}\right)}{\sum_{i=1}^L \exp((\boldsymbol{q}_m^\top \boldsymbol{k}_i + \tilde{k}_{r_1,\dots,r_\ell}(m,i))/\sqrt{d})}, \tag{3.1}$$

where $\tilde{k}_{r_1,\dots,r_\ell}(m,n)$ is any shift-invariant CPD kernel with ℓ parameters. Due to Lemma 3.1, Eq. (3.1) can be reformulated into its kernel form as follows.

$$\begin{aligned} a_{m,n} &\overset{(*)}{=} \frac{\exp\left((\boldsymbol{q}_m^\top \boldsymbol{k}_n + c + \tilde{k}_{r_1,\dots,r_\ell}(m,n))/\sqrt{d}\right)}{\sum_{i=1}^L \exp((\boldsymbol{q}_m^\top \boldsymbol{k}_i + c + \tilde{k}_{r_1,\dots,r_\ell}(m,i))/\sqrt{d})} \\ &\overset{\text{Lemma 3.1}}{=} \frac{\exp\left(\boldsymbol{q}_m^\top \boldsymbol{k}_n + k_{r_1,\dots,r_\ell}(m,n))/\sqrt{d}\right)}{\sum_{i=1}^L \exp(\boldsymbol{q}_m^\top \boldsymbol{k}_i + k_{r_1,\dots,r_\ell}(m,i))/\sqrt{d})} = \frac{\exp\left(k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n])/\sqrt{d}\right)}{\sum_{i=1}^L \exp\left(k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_i, i])/\sqrt{d}\right)}. \end{aligned} \tag{3.2}$$

(*) is due to the shift-invariant property of the Softmax normalization: $\frac{\exp(x_i)}{\sum_j \exp(x_j)} = \frac{\exp(x_i + c)}{\sum_j \exp(x_j + c)}$ for any $c \in \mathbb{R}$. The second equality defines a *bias kernel* which is positive definite using Lemma 3.1:

$$k_{r_1,\dots,r_\ell} = c + \tilde{k}_{r_1,\dots,r_\ell}. \tag{3.3}$$

The last equality introduces a *composite kernel* $k^{\text{comp}} : \mathbb{R}^{d+1} \times \mathbb{R}^{d+1} \to \mathbb{R}$ as

$$k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n + k_{r_1,\dots,r_\ell}(m,n). \tag{3.4}$$

[1]By geometric sequence search, we can enlarge c by 2, 4, 8, 16, and so on until we find the required large enough constant.

Interpretation. The proposed method can be interpreted as applying a composite kernel to self-attention. The composite kernel combines the information from query $\boldsymbol{q}_m$, key $\boldsymbol{k}_n$, and positions (m, n) in a way that augments the original self-attention structure by multiplicative and additive position embeddings. The augmentation allows k^{comp} to not only retain the original $\boldsymbol{q}_m^\top \boldsymbol{k}_n$ but also include positional information from the bias kernel $k_{r_1,\ldots,r_\ell}$.

Practical Choice. In section 3.5.2, we fix $\ell = 2$ and experiment on two variants of the composite kernel, Eq. (3.4), where we call these the *power* variant and the *logarithmic* variant of our proposed KERPLE framework, Eq. (3.2). These are from a combination of Corollary 3.1 and Eq. (3.3).

$$\begin{aligned}
\text{(power)} \quad & k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n + c - r_1 |m-n|^{r_2} \text{ with } r_1 > 0 \text{ and } 0 < r_2 \leq 2. \\
\text{(logarithmic)} \quad & k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n + c - r_1 \cdot \log(1 + r_2 |m-n|) \text{ with } r_1, r_2 > 0.
\end{aligned}$$

We note that these are not the only variants of the composite kernel. In section 3.5.3, we experiment with two more complicated variants, but only find lower training speeds and marginal improvement in perplexities (e.g., logarithmic variant vs. 3-para-log). Thus, based on our study, the choices above hold advantages in both performance and speed.

Connection to Prior Work. When the bias kernel, Eq. (3.3), is a triangle kernel: $c - |m - n|$, our model reduces to ALiBi [Press et al., 2022b]. Wennberg and Henter [2021] discuss the situation where the bias kernel is a Gaussian kernel. Tsai et al. [2019b] is the case where there is no bias kernel and the attention product $\boldsymbol{q}_m^\top \boldsymbol{k}_n$ is multiplied by an exponentiated inner product kernel, $\exp(\boldsymbol{x}^\top \boldsymbol{y})$. Since ALiBi is the state-of-the-art and has great input length extrapolation, we will focus on comparison with ALiBi in our experiments.

The logarithmic variant has an implicit connection to T5 positional bias [Raffel et al., 2020a]. According to the official GitHub repository [2] and the HuggingFace Transformer [Wolf et al., 2020], T5 bias is implemented with a log-binning strategy. For each head of the Transformer, they maintain a bucket of 32 learnable parameters and assign the relative positional bias b_{m-n} to these parameters as

$$b_{m-n} = \begin{cases} \text{bucket}[m-n] & \text{if } 0 \leq m-n < 16 \\ \text{bucket}[\min(31, 16 + \lfloor \frac{\log((m-n)/16)}{\log(128/16)} \cdot 16 \rfloor] & \text{if } m-n \geq 16, \end{cases}$$

where $\lfloor \cdot \rfloor$ is the floor function. Note that the log factor is approximately $7.7 \log \frac{m-n}{16}$. Therefore, T5 is using a logarithmic bucket assignment, which turns out to extrapolate to different input lengths. Compared with T5, our logarithmic variant uses fewer parameters (2x12 vs. 32x12) but cannot learn non-monotonic relations (the log function is monotonic). We will conduct additional comparisons with T5 bias in our experiments.

[2] `https://github.com/google-research/text-to-text-transfer-transformer`

3.5 Experiments

3.5.1 Dataset and Implementation Description

Dataset. We conduct experiments on OpenWebText2, GitHub, and ArXiv datasets described in Gao et al. [2020b]. OpenWebText2 includes recent content from Reddit submissions until 2020, content from multiple languages, document metadata, multiple dataset versions, and open-source replication code. GitHub includes open-source repositories written in primary coding languages such as Java, C/C++, Python, and Go. ArXiv includes papers written in LaTex in Math, Computer Science, Physics, and some related fields. These tasks are motivated by the downstream applications such as online chatting [Roller et al., 2021], code completion [Chen et al., 2021a], and academic paper summarization [Zhang et al., 2020].

	OpenWebText2	GitHub	ArXiv
Raw Size	66.77 GB	95.16 GB	56.21 GB
Type	Internet	Coding	Academic

Table 3.1: Dataset overview. Raw Size is the size before any up- or down-sampling.

Implementation. We adapt our model from GPT-NeoX [Black et al., 2021], a Transformer implementation by the EleutherAI team. The codebase is based on NVIDIA Megatron Language Model [Shoeybi et al., 2019] and further accelerated using Microsoft DeepSpeed library [Rasley et al., 2020].

Our model is trained on a machine with one NVIDIA A100 GPU with 40 GB of memory. We adopt almost all configurations of small GPT-NeoX[3], except that we change the train-micro-batch-size to 32, seq-length to 512, and max-position-embeddings to 512. Table 3.2 summarizes the important configurations fixed throughout our experiments. In particular, the floating-point

# Layers	Hidden Size	# Attention Heads	Train Seq. Len.	# Trainable Params.
12	64	12	512	162M
Optimizer	Batch Size	Train Steps	Precision	# Trainable Params. for RPEs
Adam (lr 6e-4)	32	50,000	bfloat16	at most 36

Table 3.2: 162M Model Configurations.

encoding is set as bfloat16 (Brain Floating Point, developed by Google Brain) so that the training can be accelerated by half-precision computation with reliable stability [Kalamkar et al., 2019]. Hidden size 64 means that $d = 64$ in Eq. (3.1).

[3]https://github.com/EleutherAI/gpt-neox/blob/main/configs/small_bf16.yml

3.5.2 Experimental Results (Also c.f. § 3.7.4 to 3.7.7)

We conduct experiments to cover aspects such as input length extrapolation, application on different domains, and comparison with the prior work. These are elaborated on below. (i) Motivated by the input length extrapolation demonstrated in [Press et al., 2022b], we train our model with length 512 and test on lengths ranging from 512 to 16384. We hope that the emphasis on extrapolation enables the application of Transformers to longer sequences. (ii) To evaluate the applicability of the model in different domains, we conduct experiments on OpenWebText2, GitHub, and ArXiv datasets. (iii) To validate the effectiveness of our method, we compare KERPLE with Sinusoidal [Vaswani et al., 2017b], Rotary [Su et al., 2021b], T5 [Raffel et al., 2020a], and ALiBi [Press et al., 2022b].

Table 3.3 reports the perplexities at different extrapolation lengths. We perform non-overlapping evaluation: Suppose text is segmented in a different manner for 512 and 1024 tokens, we have N sentences and N/2 correspondingly to evaluate. We also perform a paired two-sided t-test to validate the statistical significance (significance level $p = 0.05$). We compare each candidate RPE with our proposed logarithmic variant and mark the candidate with a † *if the log variant is statistically significantly better*. Table 3.4 reports the training speed. These tables lead to three conclusions. First, within the KERPLE framework, the logarithmic variant is better than the power variant. Secondly, the logarithmic variant is 9.7% faster than T5. In terms of extrapolation, the logarithmic variant generally does better than T5 but could be slightly worse than T5 at shorter lengths. Third, the logarithmic variant is slightly slower than some prior work (ALiBi, Rotary, and Sinusoidal) but consistently outperform these methods at all extrapolation lengths. More details are given below.

Logarithmic Variant vs. Power Variant. In our proposed KERPLE framework, the logarithmic variant is better than the power variant. To be precise, the logarithmic variant is 4.4% faster and has lower perplexities across all extrapolation lengths and all tasks.

Logarithmic Variant vs. T5. In terms of speed, the logarithmic variant is 9.7% faster than T5. In terms of extrapolation perplexity, the logarithmic variant is close to or slightly worse (the differences are not statistically significant) than T5 when the extrapolation length is shorter than 2048, and consistently excels T5 at longer extrapolation lengths. The tendency of extrapolation holds for each of the datasets evaluated in this chapter.

Logarithmic Variant vs. ALiBi, Rotary, and Sinusoidal. The logarithmic variant is 1.6% slower, 7.5% faster, and 3.0% slower than ALiBi, Rotary, and Sinusoidal. The speed comparison makes sense because we require only a limited amount of learnable parameters for RPEs (at most $3 \cdot H$). Also, the logarithmic variant consistently outperforms prior work at all extrapolation lengths and tasks.

3.5.3 Experiments on Complicated Kernels

In addition to the practical variants (power & logarithmic) in section 3.4, we consider two complicated versions of the composite kernel for the purpose of maximizing performance, Eq. (3.4), as follows.

OpenWebText2						
Extrp.	KERPLE		ALiBi	T5	Rotary	Sinusoidal
	(log)	(power)				
512	23.9 ± 0.6	23.9 ± 0.6	23.9 ± 0.6	**23.7 ± 0.6**	24.2 ± 0.6†	33 ± 1†
1024	22.0 ± 0.6	22.1 ± 0.7	22.4 ± 0.5†	**21.9 ± 0.6**	32.8 ± 1.7†	750 ± 346†
2048	**21.6 ± 0.3**	21.9 ± 0.2†	22.5 ± 0.2†	21.7 ± 0.2	62.4 ± 6.1†	5507 ± 2607†
4096	**21.2 ± 0.4**	21.5 ± 0.5†	22.2 ± 0.4†	22.5 ± 0.6†	111 ± 13.8†	14039 ± 2325†
8192	**21.3 ± 0.4**	21.6 ± 0.4†	22.3 ± 0.3†	25.5 ± 1.3†	185 ± 18.9†	22621 ± 1927†
16384	**21.4 ± 0.6**	21.6 ± 0.6	22.5 ± 0.5†	31.4 ± 3.1†	269 ± 33.0†	30046 ± 4824†

GitHub						
Extrp.	KERPLE		ALiBi	T5	Rotary	Sinusoidal
	(log)	(power)				
512	3.40 ± 0.20	3.42 ± 0.20	3.42 ± 0.21	**3.38 ± 0.21**	3.44 ± 0.20†	4 ± 0.2†
1024	3.04 ± 0.14	3.07 ± 0.16	3.15 ± 0.17†	**3.02 ± 0.14**	3.86 ± 0.25†	105 ± 39†
2048	2.86 ± 0.10	2.90 ± 0.08†	3.13 ± 0.10†	**2.84 ± 0.09**	5.94 ± 0.64†	1380 ± 404†
4096	**2.74 ± 0.05**	2.79 ± 0.06	3.04 ± 0.08†	2.78 ± 0.04†	11.1 ± 1.55†	5217 ± 1118†
8192	**2.71 ± 0.05**	2.76 ± 0.05	3.04 ± 0.03†	2.95 ± 0.13†	20.2 ± 2.75†	10081 ± 3583†
16384	**2.75 ± 0.16**	2.76 ± 0.13	3.02 ± 0.13†	3.35 ± 0.27†	31.3 ± 5.20†	16443 ± 8503†

ArXiv						
Extrp.	KERPLE		ALiBi	T5	Rotary	Sinusoidal
	(log)	(power)				
512	6.07 ± 0.26	6.10 ± 0.26	6.12 ± 0.26†	**6.03 ± 0.26**	6.07 ± 0.27	43 ± 44
1024	5.61 ± 0.10	5.65 ± 0.10†	5.82 ± 0.09†	**5.58 ± 0.09**	7.49 ± 0.34†	221 ± 136†
2048	5.22 ± 0.12	5.26 ± 0.13†	5.71 ± 0.14†	**5.21 ± 0.14**	14.2 ± 1.81†	730 ± 343†
4096	**5.20 ± 0.10**	5.25 ± 0.09	5.87 ± 0.08†	5.32 ± 0.16†	30.1 ± 4.32†	1998 ± 497†
8192	**5.01 ± 0.10**	5.06 ± 0.15	5.74 ± 0.13†	5.54 ± 0.39†	54.3 ± 6.22†	4228 ± 2645†
16384	**5.07 ± 0.16**	5.07 ± 0.19	5.78 ± 0.15†	6.25 ± 0.61†	85.4 ± 7.40†	6674 ± 5696

Table 3.3: Perplexity comparison on OpenWebText2, GitHub, and ArXiv. All models are trained for 50k steps with training length 512 and five random seeds. $x^\dagger$ means our log variant is statistically significantly *better* than x. The test used is the paired two-sided t test with $\alpha = 0.05$.

	KERPLE		ALiBi	T5	Rotary	Sinusoidal
	(log)	(power)				
sec/step	0.307	0.321	0.302	0.340	0.332	0.298

Table 3.4: Training time comparison on the GitHub dataset. The Log variant of KERPLE runs efficiently.

(bias+wht) bias + weight:
$k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n \cdot \exp(-r_3|m-n|^{r_4}) + c - r_1|m-n|^{r_2}$
with $r_1, r_3 > 0$ and $0 < r_2, r_4 \leq 2$.

(3-para-log) 3-parameter-logarithmic:
$k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n + c - r_1 \cdot \log(1 + r_2|m-n|^{r_3})$
with $r_1, r_2 > 0$ and $0 < r_3 \leq 2$.

Recall the tensor product property of a kernel: if k_1 is a kernel on $\mathcal{X}$ and k_2 is a kernel on $\mathcal{Y}$, then $k((x, y), (x', y')) = k_1(x, x')k_2(y, y')$ is a kernel on $\mathcal{X} \times \mathcal{Y}$. Therefore, (bias+wht) is the setting where we train a weight $\exp(-r_3|m-n|^{r_4})$ and a bias kernel $c - r_1|m-n|^{r_2}$. $\boldsymbol{q}_m^\top \boldsymbol{k}_n$ is multiplied by the weight kernel and then added with the bias kernel. (3-para-log) is the setting where we consider $|m-n|^{r_3}$ in the log. When $r_3 = 1$, it is reduced to the logarithmic variant proposed in section 3.4.

We plug in these composite kernel k^{comp} into our KERPLE framework, Eq. (3.2), and test the performance of these RPE. Compared with section 3.5.2, Table 3.5 suggests that these variants do not have clear advantage in extrapolation performance, e.g., 3-para-log is slightly better in perplexity than the (two-parameter) logarithmic variant. Thus, enlarging the complexity of kernels does not necessarily give better performance in the context of RPE.

Extrp.	OpenWebText2		GitHub		ArXiv	
	(bias+wht)	(3-para-log)	(bias+wht)	(3-para-log)	(bias+wht)	(3-para-log)
512	24.1 ± 0.6	23.8 ± 0.6	3.44 ± 0.21	3.40 ± 0.20	6.11 ± 0.27	6.06 ± 0.27
1024	22.2 ± 0.6	22.0 ± 0.7	3.08 ± 0.15	3.04 ± 0.13	5.66 ± 0.09	5.61 ± 0.10
2048	21.9 ± 0.4	21.6 ± 0.2	2.90 ± 0.12	2.85 ± 0.10	5.28 ± 0.12	5.21 ± 0.12
4096	21.5 ± 0.5	21.2 ± 0.4	2.79 ± 0.06	2.73 ± 0.05	5.31 ± 0.08	5.18 ± 0.09
8192	21.4 ± 0.5	21.3 ± 0.4	2.76 ± 0.03	2.68 ± 0.04	5.16 ± 0.18	5.00 ± 0.11
16384	OOM	OOM	OOM	OOM	OOM	OOM

Table 3.5: Perplexity comparison for KERPLE with complicated kernels on OpenWebText2, GitHub, and ArXiv. All models are trained for 50k steps with training length 512 and five seeds random. OOM means out of memory.

3.5.4 Plots of Kernel Functions

We plot kernel functions including the power, log variants, and ALiBi for different heads to see their contributions to softmax. We use the GitHub dataset for demonstration. Please see Figure 3.2, 3.3, and 3.4. Both ALiBi and its generalized power variant quickly reach a very negative value. In contrast, the log variant successfully discovers several flat kernels, effectively extending the window attention. This corroborates our previous observation that KERPLE-log can utilize more distant token information.

3.5.5 Position-wise Perplexity Evaluation

To better understand the fine-grained length extrapolation performance, we plot the position-wise perplexity with evaluation length=4096 in Figure 3.5. Please see § 3.7.6 for similar length=16384

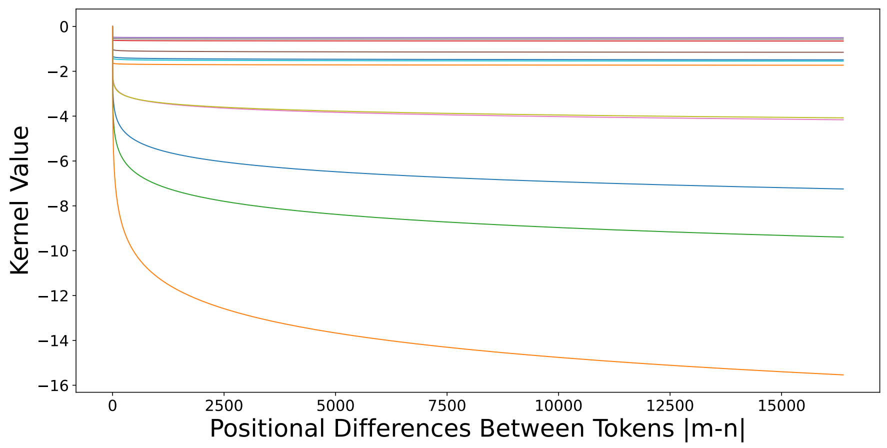


Figure 3.2: Kernel functions learned by the Log variant.

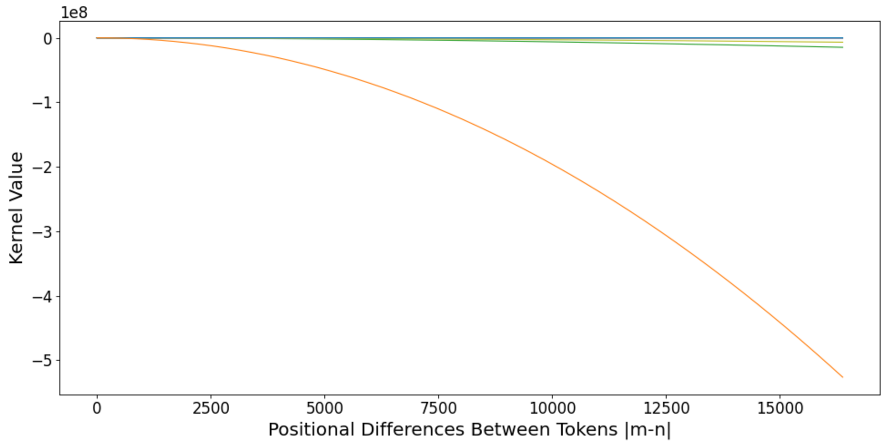


Figure 3.3: Kernel functions learned by the Power variant. Note the y-axis should be multiplied by $1e8$: the value is a very large negative number.

result. The evaluation is done by measuring the loss at each position in each sequence and averaging over the sequences.

We note that PPL@512 of KERPLE-log is the lowest among all model variants. We can derive several critical observations for evaluation length=4096 in Figure 3.5: First, KERPLE-log lies below KERPLE-log-windowed@512, indicating its usage of more distant information than window attention: If our model does not use more information other than a fixed-window=512, the y-values after position=512 should overlap with the line windowed at 512. This is clearly not the case. In addition, the PPL of KERPLE-log continues to decrease till the end of 4096 positions (Not plateauing). Second, T5 lies below KERPLE-log-windowed@512 most of the time and fluctuates around KERPLE-log-windowed@512 after length=3000. It is still worse than KERPLE-log. Third, ALiBi lies above KERPLE-log-windowed@512 for almost all the positions, indicating that window attention might be a better choice than ALiBi.

Although window attention is a strong baseline, our KERPLE-log is almost like a free lunch

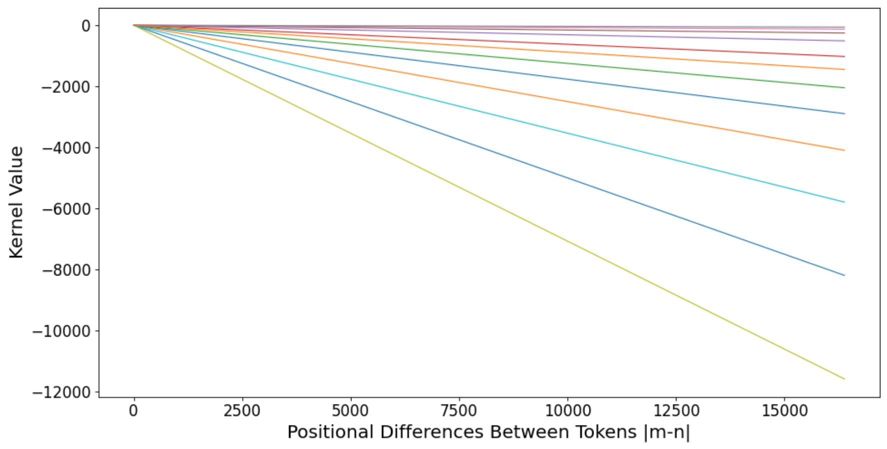


Figure 3.4: Kernel functions learned by ALiBi.

compared to window attention: With only 24 additional learnable parameters (2 parms. for each head), the almost same training speed, and the same train length=512 as window attention, it is able to achieve lower PPLs across different positions.

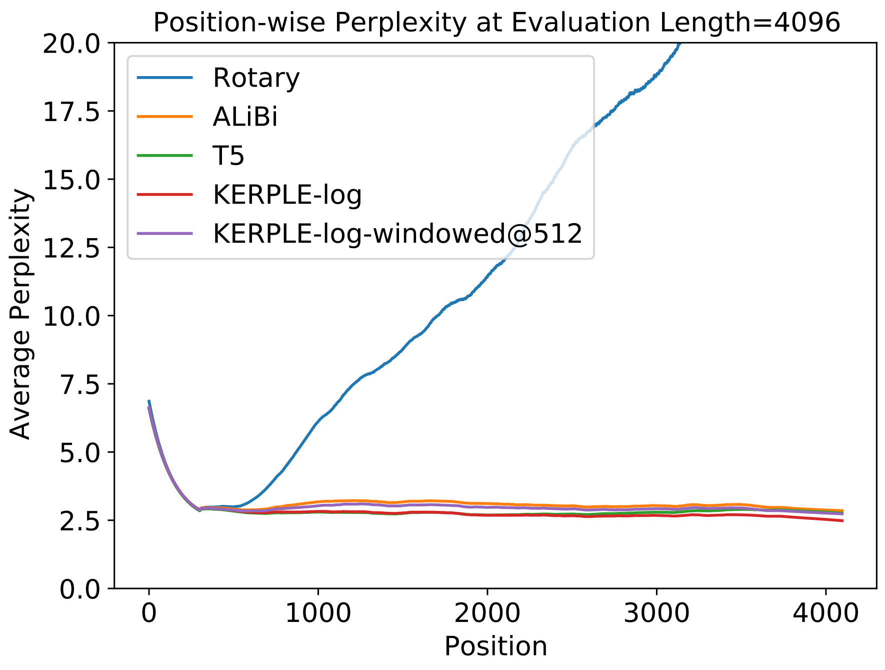


Figure 3.5: Position-wise perplexity on GitHub at evaluation length=4096 compared to window attention@512.

3.6 Conclusion

A general framework, KERPLE, is proposed to kernelize relative positional embeddings for length extrapolation. At the core of this framework is the application of CPD kernels and the derivation of practical variants. We show that these CPD kernels can be implicitly converted to PD kernels, which preserve the inner product interpretation of self-attention. We also demonstrate that the logarithmic variant achieves exceptional extrapolation performance on three large language modeling datasets. We believe our work paves the way for some interesting future directions that could resolve the observed limitations. For instance, we can consider general kernel families and model non-monotonic effects due to positional differences. In addition, the use of learnable parameters in KERPLE might enable better generalization to inputs that are higher than one-dimensional. Last but not least, there is always room for improving memory efficiency by adjusting the model architecture and training procedure.

3.7 Proofs and Experimental Details

3.7.1 Proof of CPD Shift Lemma

Lemma 3.1 (CPD Shift Lemma). *Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a conditionally positive definite (CPD) kernel. Then, there exists $c \geq 0$ such that $c + k(x, y)$ is a positive definite kernel.*

Proof. Let $K = [k(x_i, x_j)]_{i,j=1}^N$ be the matrix generated by $\{x_1, ..., x_N\}$ with $N \in \mathbb{N}$. Consider

$$f_c(v) = v^\top(c\mathbb{1}\mathbb{1}^\top + K)v = c(v^\top \mathbb{1})^2 + v^\top K v.$$

We want to show there exists a large enough c such that $f_c(v) \geq 0$ for all $v \in \{v : \|v\| = 1\}$.

(i) It is sufficient to consider $a^* = \min_{v:\|v\|=1} v^\top K v < 0$.
Let a^* be the solution to the minimization:

$$a^* = \min_{v:\|v\|=1} v^\top K v.$$

Since $v^\top K v$ is continuous in v and $\{v : \|v\| = 1\}$ is compact (i.e., closed and bounded), a^* must exist. If $a^* \geq 0$, K is positive semidefinite and $f_c(v) \geq 0$ for $c \geq 0$. Thus, without loss of generality, we assume $a^* < 0$.

(ii) It is sufficient to consider K without zero eigenvalues (i.e., full rank).
If there exists v_0 such that $Kv_0 = 0$, then $c \geq 0$ is enough to satisfy $f_c(v_0) \geq 0$. For any v_1 satisfying $v_1^\top v_0 = 0$, we have $(v_1 + v_0)^\top K(v_1 + v_0) = v_1^\top K v_1$. Therefore, whether there exists c to have $f_c(v) \geq 0$ doesn't depend on the eigenvector corresponding to zero eigenvalue (if there is such a vector). This means it is enough to consider K without zero eigenvalues.

(iii) It is sufficient to consider strict CPD.

By definition of conditional positive definiteness (CPD), we know $v^\top Kv \geq 0$ when $v^\top \mathbb{1} = 0$. Since K has no zero eigenvalue, we cannot have $v^\top Kv = 0$ when $v^\top \mathbb{1} = 0$[4]. This means the inequality is strict here: $v^\top Kv > 0$ when $v^\top \mathbb{1} = 0$, and it is enough to consider strict CPD.

(iv) It is sufficient to show there exists (small enough) $\boldsymbol{\delta > 0}$ such that $\boldsymbol{v' \in T_\delta \Rightarrow v'^\top Kv' > 0}$.

Note T_δ is defined as

$$T_\delta = \{v' : |v'^\top \mathbb{1}| < \delta,\ \|v'\| = 1\}.$$

For any $v' \in T_\delta$, if v' satisfies $v'^\top Kv' > 0$, then $c \geq 0$ is enough to have $f_c(v') \geq 0$. Conversely, when $|v^\top \mathbb{1}| \geq \delta$ and $\|v\| = 1$, observe that

$$f_c(v) = c(v^\top \mathbb{1})^2 + v^\top Kv \geq c(v^\top \mathbb{1})^2 + a^* \geq c\delta^2 + a^*$$

Then, $f_c(v) \geq 0$ when $c \geq \frac{-a^*}{\delta^2}$. Therefore, we need to prove $v' \in T_\delta \Rightarrow v'^\top Kv' > 0$ for small enough δ.

(v) Leveraging Continuity. Consider v' satisfying $\|v' - v\| < \delta_2$ with $v \in S = \{v : \|v\| = 1,\ v^\top \mathbb{1} = 0\}$. Since $v^\top Kv$ is continuous in v and $\|K\| < \infty$, for any $\epsilon > 0$ and any $v \in S$, a small enough $\delta_2 > 0$ gives $|v'^\top Kv' - v^\top Kv| < \epsilon$.

To see this, taking $v' = v + p$ with $\|p\| < \delta_2$, we have

$$|v'^\top Kv' - v^\top Kv| = |p^\top Kv + v^\top Kp + p^\top Kp| \underset{\|v\|=1}{\leq} \|K\|(2\|p\| + \|p\|^2) \leq \|K\|(2\delta_2 + \delta_2^2).$$

Therefore, $0 < \delta_2 < \sqrt{1 + \epsilon/\|K\|} - 1$ is enough to have $|v'^\top Kv' - v^\top Kv| < \epsilon$.

By definition of strict CPD, we know $\min_{v \in S} v^\top Kv = \lambda > 0$. Thus, take $\epsilon < \lambda$, a small enough δ_2 gives $v'^\top Kv' > v^\top Kv - \epsilon >= \lambda - \epsilon > 0$. In other words, there exists a small enough δ_2 such that $v'^\top Kv' > 0$ for $v' \in S_{\delta_2} = \{v' : \|v' - v\| < \delta_2,\ v \in S\}$.

(vi) Proving $\boldsymbol{\exists\, \delta > 0}$ s.t. $\boldsymbol{v' \in T_\delta \Rightarrow v'^\top Kv' > 0}$.

Due to (iv), we want to show $\exists\ \delta > 0$ s.t. $v' \in T_\delta \Rightarrow v'^\top Kv' > 0$. We will prove by the conclusion of (v). Let $\|v'\| = 1$, $v'^\top \mathbb{1} = r$ with $|r| < \delta$ and $v'' = v' - \frac{r}{n}\mathbb{1}$. We have

$$v''^\top \mathbb{1} = 0, \quad \|v' - v''\| = \frac{r}{\sqrt{n}}, \quad \|v''\| = \sqrt{\|v' - \frac{r}{n}\mathbb{1}\|^2} = \sqrt{1 - \frac{r^2}{n}}.$$

[4]By spectral decomposition, $v^\top Kv = \sum_i \lambda_i (v^\top u_i)^2 \geq 0$. Since there is no $\lambda_i = 0$, the inequality is strict.

Take $v = \frac{v''}{\|v''\|} = \frac{v''}{q}$, where $q = \sqrt{1 - \frac{r^2}{n}}$. We have $v^\top \mathbb{1} = 0, \quad \|v\| = 1$ and

$$\begin{aligned}
\|v' - v\|^2 =& \|(1 - \frac{1}{q})v' + \frac{1}{q}\frac{r}{n}\mathbb{1}\|^2 = (1 - \frac{1}{q})^2 + \frac{r^2}{nq^2} + 2(1 - \frac{1}{q})\frac{1}{q}\frac{r^2}{n} \\
=& (1 - \frac{1}{q})^2 + \frac{r^2}{n}(\frac{2}{q} - \frac{1}{q^2}) = \frac{1}{q^2}\Big((q-1)^2 + \frac{r^2}{n}(2q-1)\Big) \\
=& \frac{1}{1 - \frac{r^2}{n}}\Big(1 - \frac{r^2}{n} - 2q + 1 + \frac{r^2}{n}2q - \frac{r^2}{n}\Big) \\
=& \frac{1}{1 - \frac{r^2}{n}}\Big(1 - \frac{r^2}{n} - 2q(1 - \frac{r^2}{n}) + 1 - \frac{r^2}{n}\Big) = 2 - 2q \\
=& 2(1 - \sqrt{1 - \frac{r^2}{n}}) \approx 2(1 - 1 + \frac{1}{2}\frac{r^2}{n}) = \frac{r^2}{n} \quad (\sqrt{1-x} \approx 1 - \frac{x}{2} \text{ when } |x| \ll 1)
\end{aligned}$$

Thus, $\|v' - v\| = O(\frac{|r|}{\sqrt{n}}) \leq O(\frac{\delta}{\sqrt{n}}) < \delta_2$ for small enough δ. This implies that, with a small enough δ, for any $v' \in T_\delta$, we can find $v \in S$ such that $\|v' - v\| < \delta_2$. Thus, $v' \in S_{\delta_2}$, and by (v), we arrive at $v'^\top K v' > 0$.

□

3.7.2 Shift-invariant Kernels with Bounded and Unbounded Ranges

Definition 3.1 implies a shift-invariant kernel is generated by a univariate function $f : \mathcal{X} \to \mathbb{R}$. To characterize the set of valid univariate functions, we introduce the positive definite functions as below.

Definition 3.3 (Positive definite function). *A (real) positive definite function is a function $f : \mathcal{X} \to \mathbb{R}$ such that for any integer N and any set of N points $\{x_i \in \mathcal{X}\}_{i=1}^N$, the $N \times N$ matrix $\boldsymbol{A} = [f(x_i - x_j)]_{i,j=1}^N$ is positive semidefinite.*

We will interchange the ideas of shift-invariant kernels and positive definite functions because they are equivalent by definition. Any statement in positive definite functions can be translated into shift-invariant kernels, and vice versa. Because of this, we will use some facts about the positive definite functions to derive the shift-invariant kernels of our interest.

Generalizing Classical Bounded Shift-invariant Kernel. Applying kernels in the attention mechanism has been described in several studies[Tsai et al., 2019b; Choromanski et al., 2021; Peng et al., 2021]. One of the most common approaches is to consider the Gaussian kernel:

$$k(m, n) = \exp(-\gamma(m-n)^2), \quad \gamma > 0.$$

Note that the Gaussian kernel is bounded ($k(m, n) \in (0, 1]$ for the case above). To generalize it to a broader class of bounded shift-invariant kernels, observe that the Gaussian kernel generated by a positive definite function of the form $f(x) = \exp(-|x|^2)$. Since there is no strong reason to be limited to the power of 2, one may generalize it to a broader class of positive definite functions as below.

Fact 3.5 (Corollary 3 of Schoenberg [1938]). $\exp(-|x|^p)$ *is positive definite if* $0 < p \leq 2$ *and not positive definite if* $p > 2$.

Fact 3.5 implies that, if one wants to find a class of bounded shift-invariant kernel (i.e., $k(m,n)$ is within some fixed interval for any m, n), then $k(m,n) = \exp(-a|m-n|^p)$ with $a > 0$ and $p \in (0,2]$ may be of interest.

Constructing Unbounded Shift-invariant Kernels. A limitation of Fact 3.5 is that it only generates kernels with a bounded range (here, the range is bounded in $(0,1]$). In situations where there are no explicit bounds, one might want to consider kernels with unbounded range. To construct such kernels, we can use a kernel representation theorem presented in Schoenberg [1938]:

Fact 3.6 (Theorem 4 of Schoenberg [1938]). $f(x)$ *is bounded away from zero (*$f(x) > 0$*) and its positive powers* $f(x)^\lambda$ *(*$\lambda > 0$*) are all positive definite if and only if* $f(x)$ *is of the form*

$$f(x) = \exp(c + \psi(x)),$$

where $\psi(x)$ *is positive definite and* c *is a real constant.*

Since Fact 3.6 works for non-negative kernels, we combine it with Fact 3.5 and present the following class of shift-invariant kernel with an unbounded range.

Proposition 3.1 (Kernel from Distance Powers). *For any* $p \in (0,2]$*, there exists* $c_{\min} \in \mathbb{R}$ *such that for any* $c \geq c_{\min}$*,*

$$k(m,n) = c - |m-n|^p,$$

is a positive definite kernel. When $p > 2$*, there is no* c *to make* $k(m,n)$ *positive definite.*

Proof. Due to Fact 3.5, we know $\exp(-|x|^p)$ is positive definite when $p \in (0,2]$. Since $\exp(-|x|^p) > 0$ and $\exp(-\lambda|x|^p)$ is positive definite for any $\lambda > 0$[5], Fact 3.6 implies there exists a $c' \in \mathbb{R}$ and a positive definite $\psi(x)$ such that

$$\exp(-|x|^p) = \exp(c' + \psi(x)).$$

In other words, $-c' - |x|^p$ is a positive definite function. Take $c_{\min} = -c'$. We see that $c_{\min} - |m-n|^p$ is a shift-invariant positive definite kernel. Finally, let $k(m,n) = c - |m-n|^p$ with $c \geq c_{\min}$. The $N \times N$ matrix $[k(x_i, x_j)]_{i,j=1}^N$ generated by $k(m,n)$ on points $\{x_i \in \mathbb{R}\}_{i=1}^N$ obeys

$$[k(x_i,x_j)]_{i,j=1}^N = [c_{\min} - |x_i - x_j|^p]_{i,j=1}^N + (c - c_{\min})\mathbb{1}\mathbb{1}^\top \succeq (c - c_{\min})\mathbb{1}\mathbb{1}^\top \succeq 0,$$

where $\succeq$ is the Loewner order and $\mathbb{1} = [1, ..., 1]^\top$ in the N-dimensional vector with all ones. This shows $k(m,n)$ is a shift-invariant positive definite kernel when $0 < p \leq 2$. The conclusion on $p > 2$ is proved by contradiction. When $p > 2$, if there exists a c such that $k(m,n)$ is positive definite, then $\exp(k(m,n))$ is positive definite, which contradicts to the case of $p > 2$ in Fact 3.5. $\square$

Prop. 3.1 introduces a kernel with unbounded range ($k(m,n) \in (\infty, c]$) and is adapted from the p-th power of the distance $|m-n|$. Since the distance is a notion of "dissimilarity", $-|m-n|^p$

[5] $\exp(-\lambda|x|^p) = \exp(-|\lambda^{1/p}x|^p)$ is a constant rescaling of $\exp(-|x|^p)$ and therefore is positive definite.

becomes a notion of "similarity", which gives a sense of kernel. Thereby, we can interpret the constant c as the required value to shift $-|m-n|^p$ such that $c-|m-n|^p$ becomes a positive definite kernel.

In fact, $-|m-n|^p$ is a conditional positive definite kernel for $p \in (0,2]$ Schölkopf [2000]. Therefore, the fact that $-|m-n|^p$ can become a positive definite kernel by shifting is not a coincidence, as it has already had an intimate relation to positive definite kernels.

3.7.3 Experiments on Gaussian-like Kernels

Since the prior work on shift-invariant kernels mainly focuses on Gaussian kernels, we present preliminary experiments on Gaussian-like kernels. Compared with section 3.5.2, the perplexities of these kernels are large at every extrapolation length. This verifies our previous assertion that the Gaussian-like kernels have limited extrapolation ability.

Because the kernel can be used as a weight or a bias, we consider four kinds of the composite kernel (see section 3.4) as follows.

(2-para-bias) $r_1, r_2 > 0$.
$$k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n + r_1 \exp(-r_2|m-n|^2).$$
(3-para-bias) $r_1, r_2 > 0$ and $0 < r_3 \leq 2$.
$$k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n + r_1 \exp(-r_2|m-n|^{r_3}).$$
(1-para-wht) $r_1 > 0$.
$$k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n \cdot \exp(-r_1|m-n|^2).$$
(2-para-wht) $r_1 > 0$ and $0 < r_2 \leq 2$.
$$k^{\text{comp}}([\boldsymbol{q}_m, m], [\boldsymbol{k}_n, n]) = \boldsymbol{q}_m^\top \boldsymbol{k}_n \cdot \exp(-r_1|m-n|^{r_2}).$$
(2-para-bias) and (1-para-wht) are the settings where we put the Gaussian kernel as a bias and a weight, respectively. (3-para-bias) and (2-para-wht) generalize these settings by considering a learnable power between 0 and 2. Note we must constrain the power in (0,2]; otherwise, the function is not positive definite. See Fact 3.5 for details.

These composite kernel k^{comp} are plugged into the KERPLE framework, Eq. (3.2), and are evaluated on OpenWebText2, GitHub, and ArXiv datasets. Table 3.6 shows the Gaussian-like kernel is better to be a weight instead of a bias. As discussed in §3.7.2, the Gaussian-like kernels are bounded. To some extent, this implies that the bounded positive kernel can model a weight. However, compared with section 3.5.2, the Gaussian-like kernels have limited advantages in extrapolation. Although the performance might be improved if the power of $\exp(-|x|^p)$ is relaxed from $p = 2$ to $p \in (0,2]$, still it cannot be as good as the logarithmic variant as we demonstrate in section 3.5.2. Therefore, while the Gaussian kernel is frequently used in the literature, we need a better class of shift-invariant kernels to tackle the length extrapolation challenge.

3.7.4 Experiments on Large Model, Longer Training Length, and Wikitext-103

In this subsection, we present additional experiments on (a) large models, (b) longer training length, and (c) Wikitext-103. Below is the summary of the experiments.

Extrp.	OpenWebText2			
	1-para-wht	2-para-wht	2-para-bias	3-para-bias
512	33.8 ± 1.1	24.8 ± 0.9	58.4 ± 71.6	26.4 ± 0.5
1024	32.5 ± 0.8	23.0 ± 0.8	88.7 ± 62.6	75.3 ± 37.8
2048	34.1 ± 0.6	22.7 ± 0.4	406 ± 101	2629 ± 4024
4096	35.6 ± 0.9	22.6 ± 0.6	2590 ± 3211	37557 ± 67936
8192	39.2 ± 1.1	23.2 ± 0.3	10829 ± 18855	189216 ± 369499

Extrp.	GitHub			
	1-para-wht	2-para-wht	2-para-bias	3-para-bias
512	7.78 ± 0.48	3.56 ± 0.23	4.08 ± 0.85	3.67 ± 0.22
1024	7.85 ± 0.40	3.19 ± 0.17	4.63 ± 0.59	4.23 ± 0.57
2048	8.08 ± 0.21	3.01 ± 0.09	18.8 ± 6.8	20.0 ± 4.8
4096	8.47 ± 0.43	2.93 ± 0.09	75.8 ± 32.2	94.0 ± 24.7
8192	9.41 ± 0.75	3.05 ± 0.20	207 ± 110	261 ± 86

Extrp.	ArXiv			
	1-para-wht	2-para-wht	2-para-bias	3-para-bias
512	10.6 ± 0.4	6.18 ± 0.25	6.73 ± 0.30	7.12 ± 1.43
1024	10.7 ± 0.2	5.73 ± 0.11	7.07 ± 0.63	7.27 ± 0.69
2048	10.8 ± 0.3	5.35 ± 0.15	20.4 ± 9.3	23.5 ± 8.4
4096	11.6 ± 0.3	5.44 ± 0.14	80.6 ± 49.4	131 ± 140
8192	12.1 ± 0.2	5.50 ± 0.27	220 ± 138	437 ± 591

Table 3.6: Extrapolation of Gaussian-like kernels on OpenWebText2, GitHub, and ArXiv. All models are trained for 50k steps with training length 512 and five random seeds.

	1-para-wht	2-para-wht	2-para-bias	3-para-bias
sec/step	0.326	0.327	0.324	0.351

Table 3.7: Training time comparison for Gaussian-like kernels on GitHub.

(a) The 1.3B large model is trained on a machine with two NVIDIA A100 GPU with 40 GB of memory. We adopt almost all configurations of XL GPT-NeoX[6], except that we change the train-micro-batch-size to 16, model-parallel-size to 2, seq-length to 512, and max-position-embeddings to 512. Table 3.8 summarizes the configurations of the 1.3B model.

(b) The 162M Model with training sequence length=1024 follows the same configurations as the ones in Table 3.2 except that the train seq. length is changed to 1024.

(c) The Wikitext-103 model is implemented on ALiBi's GitHub[7] with exactly the same configurations (247M parameters), except that the function buffered_future_mask() at line 1011 of attention_with_linear_biases/fairseq/models/transformer.py is adapted to our KERPLE-log.

[6]`https://github.com/EleutherAI/gpt-neox/blob/main/configs/XL.yml`
[7]`https://github.com/ofirpress/attention_with_linear_biases`

# Layers	Hidden Size	# Attention Heads	Train Seq. Len.	# Trainable Params.
24	128	16	512	1.3B
Optimizer	**Batch Size**	**Train Steps**	**Precision**	**# Trainable Params. for RPEs**
Adam (lr 2e-4)	32	150,000	float16	48

Table 3.8: 1.3B model configurations.

Table 3.9 shows the results on the large model (1.3B). Compared with the small model results in Table 3.3, we see that T5 bias becomes weaker than KERPLE-log and ALiBi, and KERPLE-log remains stronger than ALiBi on GitHub and ArXiv datasets. This is explained by the tendency for overfitting. Observe that both T5 and KERPLE learn the positional embeddings while ALiBi uses fixed ones. T5 and KERPLE have a higher tendency of overfitting. A larger model (1.3B > 162M) or a noisy dataset (OpenWebText2 > GitHub, ArXiv) presents a higher risk of overfitting. Hence, we see that T5 bias is weak on a large model, and KERPLE-log only extrapolates well on GitHub and ArXiv.

Again, Table 3.9 shows the results on long training length (1024). compared with the short training length (512) in Table 3.3, KERPLE-log remains better than ALiBi and T5 bias, especially on longer evaluation length. This shows the robustness of KERPLE-log over different training lengths.

Table 3.10 compares KERPLE-log with ALiBi using ALiBi's implementation and configurations. The results show that KERPLE-log is superior to ALiBi on Wikitext-103.

3.7.5 Additional Analyses

Since the power and logarithmic variants derived from KERPLE achieve superior performance on length extrapolation across various datasets, we investigate the underlying reason by visualizing the *effective length* as shown in Figure 3.6. The visualization works in the following procedure.

1. For each training dataset, the learnable parameters $(r_1^{(h)}, ..., r_\ell^{(h)})$ associated with each head h (12 in total) are extracted from the model checkpoint. The CPD kernel at head h is $\tilde{k}^{(h)} = \tilde{k}_{r_1^{(h)},...,r_\ell^{(h)}}$. Both the power and the logarithmic variants in corollary 3.1 undergo a similar procedure. The only difference is that their $\tilde{k}$'s are different.
2. For each head h, we compute the *effective length* of $\tilde{k}^{(h)}$ as $\text{eff}^{(h)} = \min_{\tilde{k}^{(h)}(0,|m-n|)<-2} |m-n|$. That is, the relative positional difference $|m-n|$ such that $\tilde{k}(m,n) \overset{\text{shift-inv.}}{=} \tilde{k}(0,|m-n|)$ just becomes smaller than -2. Note $\tilde{k}(0,|m-n|)$ strictly decreases in $|m-n|$, so there is only one possible value. We pick -2 here because $\tilde{k}$ is a bias and is followed by the Softmax normalization. A bias of -2 or smaller can make a great impact on the output of Softmax[8]. $\text{eff}^{(h)}$ is interpreted as the *effective length* because, when $|m-n| < \text{eff}^{(h)}$, the attenuation due to $\tilde{k}^{(h)}$ is not strong. When $|m-n| > \text{eff}^{(h)}$, the attenuation is strong and has a large impact on $\boldsymbol{q}_m^\top \boldsymbol{k}_n + \tilde{k}^{(h)}(m,n)$.

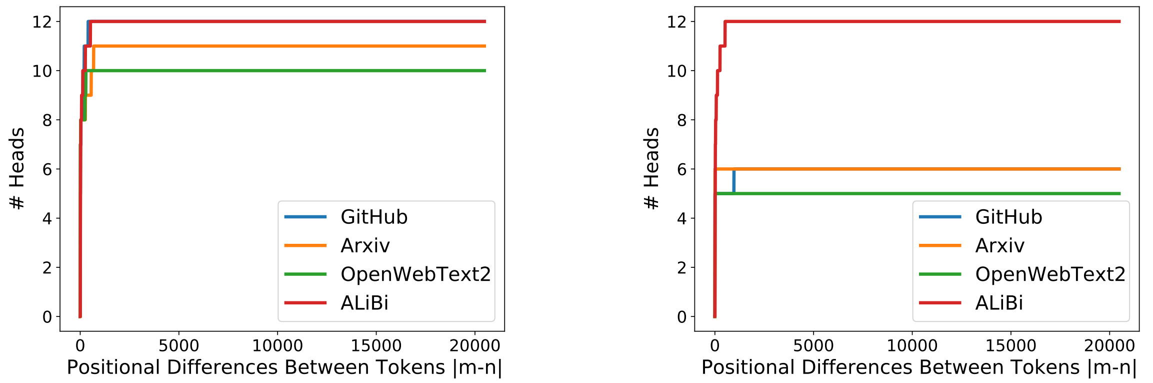
[8]Since Softmax is an exponentiated function, a -2 bias in the Softmax's argument roughly gives an attenuation of $\exp(-2) \approx 0.135$.

	162M Model. Train length, steps=1024, 50k.			1.3B Model. Train length, steps=512, 150k.		
	GitHub			GitHub		
Extrp.	KERPLE-log	ALiBi	T5 bias	KERPLE-log	ALiBi	T5 bias
512	-	-	-	**2.88 ± 0.11**	**2.88 ± 0.11**	2.93 ± 0.11†
1024	2.83 ± 0.16	2.84 ± 0.16†	**2.81 ± 0.16**	**2.60 ± 0.12**	2.62 ± 0.11†	2.64 ± 0.11†
2048	2.70 ± 0.07	2.82 ± 0.07†	**2.68 ± 0.07**	**2.44 ± 0.05**	2.58 ± 0.05†	2.47 ± 0.07†
4096	**2.53 ± 0.04**	2.77 ± 0.06†	2.54 ± 0.04	**2.46 ± 0.11**	2.65 ± 0.12†	2.49 ± 0.12
8192	**2.42 ± 0.03**	2.74 ± 0.02†	2.57 ± 0.06†	**2.44 ± 0.13**	2.57 ± 0.13†	2.57 ± 0.13†
16384	**2.48 ± 0.11**	2.80 ± 0.11†	3.10 ± 0.34†	**2.60 ± 0.07**	2.61 ± 0.07	3.16 ± 0.35†
	ArXiv			ArXiv		
Extrp.	KERPLE-log	ALiBi	T5 bias	KERPLE-log	ALiBi	T5 bias
512	-	-	-	**5.56 ± 0.15**	5.58 ± 0.16	5.62 ± 0.15†
1024	5.23 ± 0.09	5.26 ± 0.09	**5.20 ± 0.10**	**4.87 ± 0.07**	4.94 ± 0.07†	4.92 ± 0.06†
2048	4.76 ± 0.12	4.98 ± 0.18†	**4.74 ± 0.12**	**4.50 ± 0.16**	4.87 ± 0.17†	4.55 ± 0.16†
4096	**4.75 ± 0.10**	5.31 ± 0.13†	4.97 ± 0.27	**4.45 ± 0.06**	4.97 ± 0.13†	4.53 ± 0.08†
8192	**4.54 ± 0.10**	5.25 ± 0.15†	6.55 ± 0.97†	**4.47 ± 0.20**	4.94 ± 0.16†	4.65 ± 0.15†
16384	**4.62 ± 0.15**	5.35 ± 0.19†	16.0 ± 4.77†	**4.65 ± 0.24**	4.94 ± 0.07	5.25 ± 0.26†
	OpenWebText2			OpenWebText2		
Extrp.	KERPLE-log	ALiBi	T5 bias	KERPLE-log	ALiBi	T5 bias
512	-	-	-	**17.5 ± 0.3**	17.5 ± 0.4	17.8 ± 0.3†
1024	19.2 ± 0.1	19.3 ± 0.2	**19.1 ± 0.1**	**16.6 ± 0.6**	16.7 ± 0.6	16.9 ± 0.6†
2048	19.3 ± 0.2	19.5 ± 0.1	**19.2 ± 0.2**	**16.2 ± 0.4**	16.4 ± 0.4†	16.7 ± 0.4†
4096	**18.6 ± 0.3**	19.0 ± 0.3†	19.2 ± 0.4†	**16.4 ± 0.8**	16.5 ± 0.5	18.0 ± 0.9†
8192	**18.7 ± 0.5**	19.3 ± 0.4†	24.0 ± 1.1†	16.9 ± 0.7	**16.5 ± 0.1**	22.7 ± 3.7†
16384	**18.8 ± 0.5**	19.2 ± 0.3†	50.8 ± 6.5†	17.8 ± 1.2	**16.5 ± 0.3**	37.1 ± 13.1†

Table 3.9: Perplexity comparison for large models (1.3B) and long training length (1024) on GitHub, ArXiv, OpenWebText2. Due to the time constraint and limited computing resources, we are not able to obtain the numbers for the large model (1.3B) on OpenWebText2 for now. All models are trained with five random seeds. $x^\dagger$ means our log variant is statistically significantly *better* than x. The test used is the paired two-sided t-test with $\alpha = 0.05$.

	train length 512					train length 2048	
Extrp. length	512	1024	1536	2048	3072	2048	3072
ALiBi	19.73	18.81	18.50	18.48	18.40	17.91	17.64
KERPLE-log	**19.69**	**18.76**	**18.37**	**18.29**	**18.24**	**17.84**	**17.56**

Table 3.10: Perplexity comparison on Wikitext-103. To ensure a fair comparison, the model (247M) is trained on ALiBi's codebase with exactly the same configurations except for the positional embeddings. The results show that KERPLE-log is superior to ALiBi on Wikitext-103.



(a) Power Variant: $-a|m-n|^p$ (b) Logarithmic Variant: $-a\log(1+b|m-n|)$

Figure 3.6: Number of heads with effective lengths $\leq |m-n|$ for different choices of CPD kernels and datasets. See section 3.7.5 for details.

3. Then, for each $|m-n| \in [0, ..., 20480]$, we count the number of heads that satisfies $\text{eff}^{(h)} \leq |m-n|$. This gives a cumulative plot as shown in Figure 3.6, where the x-axis is $|m-n|$ and the y-axis is $\text{Count}(\{h: \ h \in [1, ..., 12], \ \text{eff}^{(h)} \leq |m-n|\})$.
4. Repeat the above steps for other datasets and kernels.

Interpretation of Curves. For a point (x, y) on a curve, it means that there are y heads with at least -2 bias when the token distance $|m-n|$ is greater than x. In other words, the slower the y converges to 12, the longer the inter-token range that the model focuses on.

The Advantage of Learnable Parameters. We observe that ALiBi [Press et al., 2022b] produces the same curve no matter which dataset is used. The reason is that ALiBi selects a fixed parameter $r = 2^{\frac{-8h}{H}}$ at head h for its linear bias $-r|m-n|$ (H heads in total) regardless of the dataset. While this strategy is useful for extrapolation, we hypothesize that different datasets might have different characteristics, e.g., the average distance of highly related tokens should differ among the datasets, as shown in Figure 3.6. These characteristics can be easier adapted by learnable parameters. Therefore, we believe that learnable parameters have more advantages in capturing the dataset-dependent characteristics.

Trends Across Datasets. We notice that both kernels trained on OpenWebText2 tend to focus more on distant relations. This makes sense because OpenWebText2 has the highest perplexity scores among all datasets, implying that more context is needed to disambiguate the next predicted token. The opposite trend holds for Arxiv and GitHub datasets, which is reasonable considering their lower perplexity scores.

Characteristics Learned by Kernels. Under any dataset, the logarithmic variant tends to focus more on distant relations than the power variant does. We can explain it through their functional forms. Because logarithm ($-a \log(1 + b|m - n|)$) decays much slower than power ($-a|m - n|^p$) does, the log variant might encourage the focus on distant relations.

3.7.6 Position-wise Perplexity for Length=16384

We further increase the sequence length of Figure 3.5 to 16384 to test models' limiting behavior. We draw similar conclusions from Figure 3.7:

1. KERPLE-log lies below KERPLE-log-windowed@512 most of the time, indicating its usage of more distant information than window attention.
2. The PPL of T5 explodes.
3. The PPL of ALiBi does not explode, but it is still worse than window attention, i.e. lies above KERPLE-log-windowed@512.

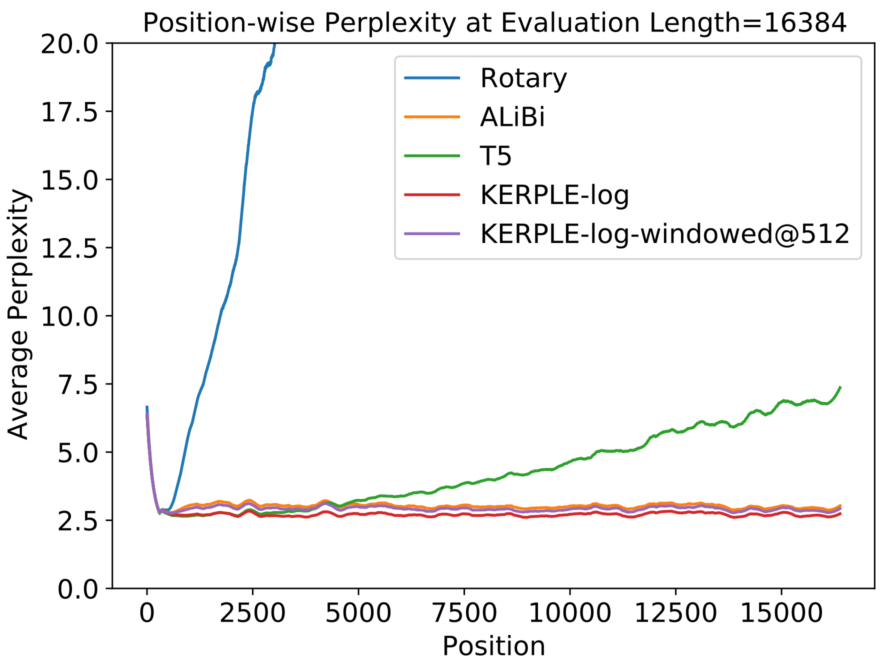


Figure 3.7: Position-wise perplexity on GitHub at evaluation length=16384 compared to window attention@512.

3.7.7 The Choice of codebase and Hyperparameters

We adopt almost all the hyperparameters (except batch size to fit in our GPU) and all implementations of the T5 bias, ALiBi, Rotary, and Sinusoidal baselines from the GPT-NeoX codebase. To ensure fair comparisons, we did not fine-tune hyper-parameters for KERPLE. The datasets we used are exactly the same as the ones released with the GPT-NeoX codebase. We just ran their prepare_data.py script to automatically download and parse the datasets. All our code was uploaded with the submission on openreview, and `https://github.com/EleutherAI/gpt-neox` is the original GitHub repository. As a side note, we chose this codebase and adopted their parameter settings because it is built by EleutherAI, which is a well-known and truly non-profit group of researchers publishing various well-regarded pretrained models for academia including GPT-J-6B and GPT-NeoX-20B.

Part II

Analyzing and Aligning Transformer Receptive Field

Chapter 4

Dissecting Transformer Length Extrapolation via the Lens of Receptive Field Analysis

4.1 Introduction

In the previous chapter, several relative positional embeddings have been discussed including ALiBi [Press et al., 2022b] and KERPLE [Chi et al., 2022]. Empirically, they extrapolate to $L_{ex} \gg L_{tr}$ much better than other absolute and relative positional embeddings including Sinusoidal [Vaswani et al., 2017b], Rotary [Su et al., 2021b], and T5 [Raffel et al., 2020a], resulting in the adoption of ALiBi for the recently released Bloom [Scao et al., 2022a] model. Despite the significant empirical success of ALiBi, there is still a lack of fundamental understanding of why it works.[1]

Figure 4.1 shows the temporal bias matrix of ALiBi. We hereinafter refer to the coefficient $\frac{1}{2^h}$ as *slope*. Intuitively, ALiBi encourages a token to focus on neighbors based on its temporal biases matrix. When two tokens are distant, ALiBi becomes highly similar to windowed attention, shown

[1]`https://github.com/ofirpress/attention_with_linear_biases#why-do-you-think-alibi-works`

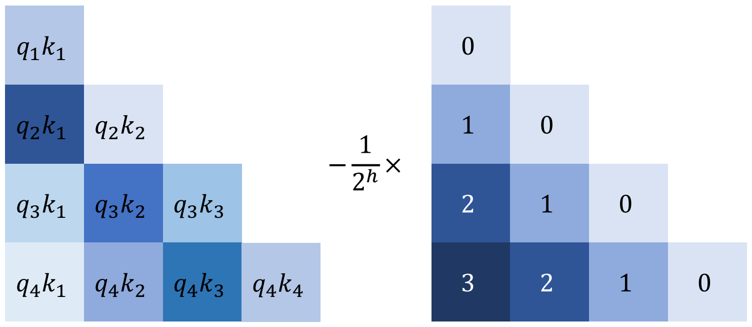


Figure 4.1: ALiBi. For a Transformer language model with H attention heads, the range of h is $n \cdot \frac{8}{H}$, where $n = \{1 \ldots H\}$. Left = self-attention matrix, right = temporal biases matrix.

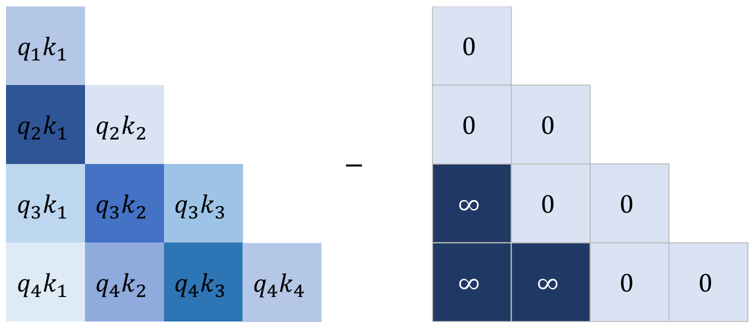


Figure 4.2: Windowed attention. This is the same design as Longformer [Beltagy et al., 2020]. We limit the context window size to $w = 2$ in this example. Left = self-attention matrix, right = temporal biases matrix.

in Figure 4.2. Experiments in § 4.4 will further establish the connection between the two.

Windowed attention allows the easy derivation of a theoretical (maximum) receptive field: wR for an R layer Transformer model with windowed attention size w. A windowed attention model can extrapolate if $L_{tr} > wR$ because 1) wR is fully covered by L_{tr} during the training stage, and 2) it simply ignores the additional $L_{ex} - wR$ tokens during the testing stage. Surprisingly, a model can still extrapolate when $L_{tr} < wR$ which we show in § 4.4. This calls for the need for empirical receptive field measurement and motivates our model-agnostic cumulative normalized gradient tool. The tool we develop can be applied back on ALiBi to show that L_{tr} covers most of its empirical receptive field.

Our analysis tool also provides critical context for explaining the length extrapolation failure [Press et al., 2022b; Chi et al., 2022] of Sinusoidal [Vaswani et al., 2017b] and Rotary [Su et al., 2021b] by showing their violation of the empirical receptive field coverage principle. Sinusoidal can be fixed by dropping the intermediate terms and keeping only the decay-with-distance biases; this leads to the creation of **Sandwich**, the first parameter-free relative positional embedding that uses information beyond L_{tr}. Sandwich shares a similar temporal bias pattern with trainable positional embeddings such as KERPLE [Chi et al., 2022] and T5 [Raffel et al., 2020a], and they jointly suggest the future design of extrapolatable Transformer positional embeddings.

4.2 Related Work

4.2.1 Length Extrapolation

In the context of language modeling, we expect token-level perplexities to remain at least the same, if not lower (i.e., better), when $L_{ex} \gg L_{tr}$ sequences are provided. Recurrent neural networks [Mikolov et al., 2010; Mikolov and Zweig, 2012; Zaremba et al., 2014] can easily perform length extrapolation. But this is not an easy task for Transformer language models, among which only those equipped with special relative positional embeddings [Press et al., 2022b; Chi et al., 2022] are length extrapolatable.

4.2.2 Positional Embeddings

It is widely believed that the design of positional embeddings is the key to successful length extrapolation of Transformer language models [Press et al., 2022b; Chi et al., 2022]. We can roughly categorize existing positional embeddings into absolute (APE) [Vaswani et al., 2017b] and relative (RPE) [Su et al., 2021b; Raffel et al., 2020a; Press et al., 2022b; Chi et al., 2022] variants. APE often assigns one positional embedding per token and combines them directly with input embeddings. In contrast, RPE adds temporal bias terms to the self-attention matrix to encode the relative distance between token pairs. For example, the right triangular matrix in Figure 4.1 shows the set of temporal bias terms. It is challenging for APE to extrapolate well without any further fine-tuning since either the beyond L positional embeddings do not exist, or the model needs to process unseen positional embeddings (e.g. unseen sinusoidal embeddings). [Press et al., 2022b; Chi et al., 2022]. In contrast, RPE usually performs better length extrapolation since it is easier to construct the additional temporal bias terms.

4.2.3 Windowed and Sparse Attention

We will see later that ALiBi can be viewed as imposing a windowed attention mask on the self-attention matrix, similar to previous Transformer models with sparse attention [Beltagy et al., 2020; Zaheer et al., 2020; Ainslie et al., 2020; Gupta and Berant, 2020]. Interpreting ALiBi from the perspective of windowed attention allows us to easily calculate the theoretical receptive field of a model.

4.2.4 Receptive Field

A model's receptive field is defined as the size of the input region that contributes the most to model outputs. It is often measured in the context of convolution neural networks [Luo et al., 2016; Dai et al., 2017; Araujo et al., 2019; Raghu et al., 2021; Dosovitskiy et al., 2021] and their dilated variants [Oord et al., 2016; Yu and Koltun, 2016; Chang et al., 2017; Beltagy et al., 2020] with the ultimate goal of receptive field size maximization. Even though we focus on Transformer language models, we borrow the idea to show that the empirical receptive field coverage of a model is crucial to its length extrapolation performance.

4.3 Background and Notations

4.3.1 Transformer Language Model

Given a sequence of $L \in \{L_{tr}, L_{ex}\}$ input embeddings $\{\boldsymbol{e}_m\}_{m=1}^{L}$ in $\mathbb{R}^d$, an R layer Transformer language model with H attention heads converts each $\boldsymbol{e}_m$ into its corresponding query, key, and value vectors in $\mathbb{R}^{\frac{d}{H}}$ at each layer:

$$\boldsymbol{q}_m = \boldsymbol{W}_q \boldsymbol{e}_m, \;\; \boldsymbol{k}_m = \boldsymbol{W}_k \boldsymbol{e}_m, \;\; \boldsymbol{v}_m = \boldsymbol{W}_v \boldsymbol{e}_m,$$

where $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v \in \mathbb{R}^{\frac{d}{H}\times d}$ are learnable matrices. The resulting vectors are processed by the self-attention module for pre-Softmax logits:

$$l_{mn} = \begin{cases} \langle \boldsymbol{q}_m, \boldsymbol{k}_n \rangle, & \text{if } m \geq n \\ -\inf, & \text{otherwise} \end{cases}$$

followed by the scaled softmax normalization:

$$a_{m,n} = \frac{\exp(l_{m,n}/\sqrt{d/H})}{\sum_{i=1}^{L} \exp(l_{m,i}/\sqrt{d/H})} \tag{4.1}$$

To be precise, the matrices ($\boldsymbol{W}_q^{(h)}$, $\boldsymbol{W}_k^{(h)}$, $\boldsymbol{W}_v^{(h)}$), vectors ($\boldsymbol{q}_m^{(h)}$, $\boldsymbol{k}_m^{(h)}$, $\boldsymbol{v}_m^{(h)}$, $\boldsymbol{o}_m^{(h)}$), and scalars ($l_{mn}^{(h)}$, $a_{mn}^{(h)}$) are associated with a head number h. For notation simplicity, we only show the dependency on h when we need it. For example, the output vector $\boldsymbol{o}_m^{(h)}$ at position m for head h is:

$$\boldsymbol{o}_m^{(h)} = \sum_{n=1}^{L} a_{m,n}^{(h)} \boldsymbol{v}_n^{(h)}$$

All the H output vectors are concatenated, denoted by $\oplus$, and transformed by $\boldsymbol{W}_o \in \mathbb{R}^{d\times d}$ to obtain $\boldsymbol{o}_m \in \mathbb{R}^d$:

$$\boldsymbol{o}_m = \boldsymbol{W}_o(o_m^{(1)} \oplus o_m^{(2)} \oplus \cdots \oplus o_m^{(H)})$$

A layer normalization [Ba et al., 2016] on $\boldsymbol{o}_m$, i.e. $\text{LayerNorm}(\boldsymbol{o}_m)$, gives the input embedding to the next layer. After R layers of propagation, the last $\boldsymbol{o}_m$ is transformed by $\boldsymbol{V} \in \mathbb{R}^{v\times d}$ and normalized by Softmax to get the distribution $\boldsymbol{p} \in \mathbb{R}^v$ over vocabulary size v:

$$\boldsymbol{p} = \text{Softmax}(\boldsymbol{V}\boldsymbol{o}_m) \tag{4.2}$$

For convenience, we set $R = 12$, $H = 12$, $d = 768$, and $L_{tr} = 512$ for all experiments reported in this thesis.

4.3.2 ALiBi

ALiBi modifies $l_{m,n}$ to be:

$$l_{mn} = \begin{cases} \langle \boldsymbol{q}_m, \boldsymbol{k}_n \rangle - \frac{1}{2^h}(m-n), & \text{if } m \geq n \\ -\inf, & \text{otherwise} \end{cases} \tag{4.3}$$

The range of h is $n \cdot \frac{8}{H}$, where $n = \{1 \ldots H\}$.

4.3.3 Windowed Attention

If the windowed attention has a size w, then:

$$l_{mn} = \begin{cases} \langle \boldsymbol{q}_m, \boldsymbol{k}_n \rangle, & \text{if } n + w > m \geq n \\ -\inf, & \text{otherwise} \end{cases}$$

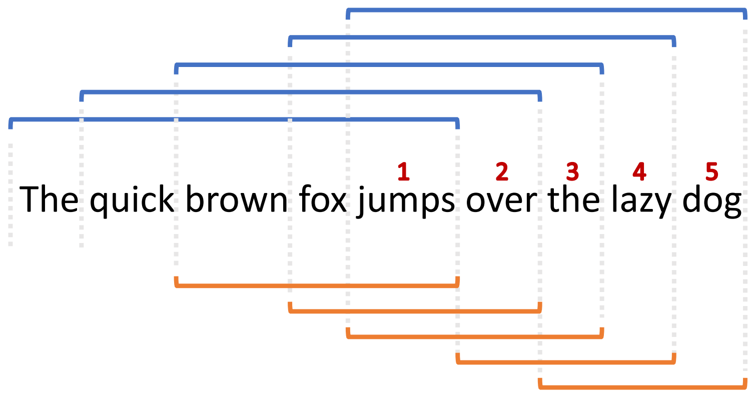


Figure 4.3: We always evaluate the perplexities of the 5 tokens numbered from 1 to 5. The upper brackets represent $L_{ex} = 5$. The lower brackets represent $L_{ex} = 3$. This formulation ensures the same 5 tokens are always evaluated with different numbers of previous tokens.

4.3.4 Evaluation of Length Extrapolation

We prepare $N = 1000$ text segments of length $L_{ex} > L_{tr}$ from the evaluation dataset. For each segment, we alter the number of previous tokens ranging from 1 to $L_{ex} - 1$ of the last token and only calculate its perplexity:

$$\text{PPL} = \exp\left(\frac{1}{N}\sum_{i=1}^{N} -\log p_i\right),$$

where p_i is the predicted probability from Eq. (4.2) of the last (L_{ex}-th) token in the i-th segment. This ensures that the same set of tokens is always used for perplexity calculation and only their number of previous tokens is varied, see Figure 4.3.[2]

L_{ex}	Shift all h by Δ						Same h for all heads					Windowed Attention with Size w					
	Δ:-3	0	2	4	6	8	h:0	2	4	6	8	w:40	80	100	120	160	320
512	5.76	5.57	5.50	5.63	5.70	5.70	9.45	6.65	5.85	5.60	5.70	8.27	7.28	7.04	6.77	6.41	6.04
1024	7.15	5.64	5.31	5.81	55.4	55.4	9.20	7.01	8.66	25.4	55.4	8.27	7.29	7.02	8.90	67.4	178
2048	7.15	5.94	5.89	6.92	94.4	94.4	9.21	7.08	8.66	31.7	94.4	8.27	7.29	7.03	8.90	67.5	202
4096	7.15	5.95	5.92	6.94	96.0	96.0	9.21	7.08	8.66	31.8	96.0	8.27	7.29	7.02	8.90	67.5	202
8192	7.15	5.95	5.92	6.94	96.0	96.0	9.21	7.08	8.66	31.8	96.0	8.27	7.29	7.02	8.90	67.5	202

Table 4.1: The three experiments conducted on the Arxiv dataset. The numbers are perplexities.

[2]There exists another evaluation protocol named non-overlapping subsequences adopted in the main experiment tables of ALiBi [Press et al., 2022b]. It is not the most suitable protocol for length extrapolation evaluation as it suffers from the "early token" curse. Please refer to §B of ALiBi [Press et al., 2022b] for details.

4.4 ALiBi and Windowed Attention

Here, we alter the slope ($\frac{1}{2^h}$) of ALiBi to check if the length extrapolation property persists and reveal the connection between ALiBi and windowed attention. We present three experiments on two datasets, ArXiv and OpenWebText2 which represent two vastly different domains (§ 4.9.1), to ensure that the observations are general, shown in Table 4.1 and 4.4.

4.4.1 Slope Shift (Shift all h by Δ)

We first investigated whether slope diversity (each attention head has one slope) is the key to length extrapolation. We shift h by a fixed amount Δ and find that the model, unfortunately, fails to extrapolate beyond a certain amount of Δ. This implies that diversity itself might not be the deciding factor, but that the actual slope value is more important.

4.4.2 Slope Equalization (Same h for all heads)

To identify the slope magnitude that enables length extrapolation, we set all slopes to be the same instead of the original geometric sequence. We then steadily increase the slope value from 0 to 8 and find that only large slopes ($\frac{1}{2^h}$), or equivalently small h, allow a model to extrapolate well. Large slopes implicitly enforce a narrow windowed bias on the self-attention matrix such that distant tokens cannot interact with each other.

4.4.3 Windowed Attention (Size w)

We make the implicit window effect explicit as shown by Eq. (4.3), which is also adopted by Longformer [Beltagy et al., 2020]. We define the windowed attention size to be w. The model underperforms at small w and diverges on long L_{ex} at large w. The same trend holds in the first two experiments when h is too small or large.

4.4.4 Other Observations

First, ALiBi does not in fact extrapolate since its perplexities all increase instead of staying the same when $L_{ex} > L_{tr}$. In contrast, windowed attention models are extrapolatable up to $w = 100$. Second, we can clearly see that once L_{ex} passes a certain threshold, the perplexity either remains the same or explodes. This suggests that the model is either ignoring tokens beyond a certain length (same)[3] or not using it properly (explosion). In the next section, we will use the concept of receptive field to explain these observations.

[3]A limited but similar observation was made in §B.2 of ALiBi [Press et al., 2022b].

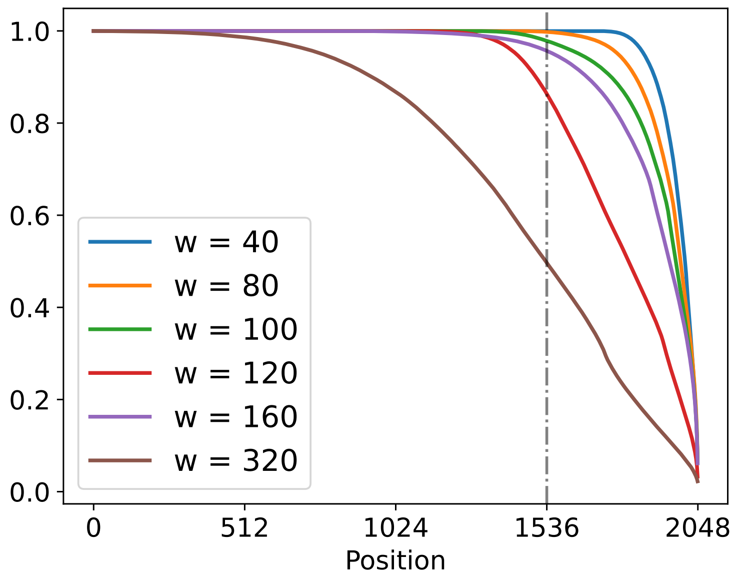


Figure 4.4: Cumulative normalized gradient on ArXiv when predicting the next (2048-th) token.

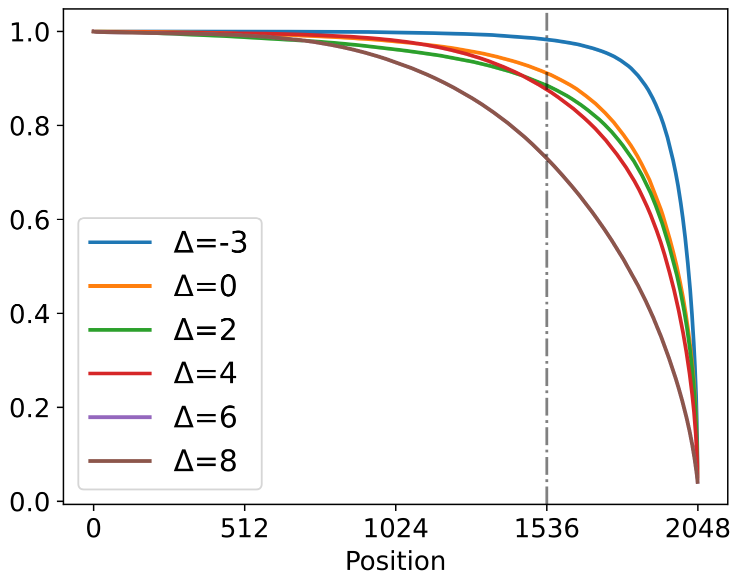


Figure 4.5: Cumulative normalized gradient on ArXiv when predicting the next (2048-th) token.

4.5 Receptive Field Measurement

Following the definition of windowed attention size w, an R layer Transformer has a theoretical receptive field (TRF) of wR, which is the maximum number of tokens that contribute to the prediction of the next token. In practice, a neural model often uses a subset of TRF, named empirical receptive field (ERF). While previous work [Luo et al., 2016; Dai et al., 2017; Araujo et al., 2019; Raghu et al., 2021; Dosovitskiy et al., 2021; Beltagy et al., 2020] aims to increase ERF to match TRF, we show that decreasing ERF could serve as one feasible approach to enable successful length extrapolation.

Consider the case where TRF $\leq L_{tr}$: This model can extrapolate easily because its TRF is fully covered and trained. Concretely, if we set $R = 12$, $L_{tr} = 512$ in Table 4.1 and 4.4, we know that as long as $w < 42.6 = 512/12$, TRF will be fully covered by L_{tr}. Surprisingly, the model is still able to extrapolate up to $w = 100$, leading to a TRF of $100 * 12 = 1200 \gg 512$. This can be explained by the ERF and TRF discrepancy discussed above; this calls for the need to quantify ERF.

4.5.1 Quantifying Empirical Receptive Field

We first calculate the normalized gradient [Luo et al., 2016] of each input token w.r.t the prediction of the next token:

$$s_m = \frac{\|\boldsymbol{g}_m\|_2}{\sum_{n=1}^{L_{ex}} \|\boldsymbol{g}_n\|_2},$$

where $\boldsymbol{g}_m$ is the gradient vector of the input embedding $\boldsymbol{e}_m$. We then calculate the cumulative sum as:

$$c_m = \sum_{n=m}^{L_{ex}} s_n, \quad 0 \leq c_m \leq 1,$$

Visualizations of c_m for the slope shift and windowed attention experiments are shown in Figures 4.4 and 4.5. We define the ERF of a model as:

$$\text{ERF} = \min\{m \mid c_m > 0.99\}.$$

Figure 4.4 demonstrates how we derive the model's ERF when it is predicting the 2048-th token. For models with $w \in [40, 80, 100]$, the most recent $L_{ex} = L_{tr} = 512$ (1536-th to 2047-th) covers more than 99% of the total (1.0) normalized gradient, so their ERF is smaller than 512. In contrast, models with $w \in [120, 160, 320]$ have ERF = 768, 1024, and 1536 tokens, respectively. Since $L_{tr} = 512$ does not fully cover their ERFs, they fail to extrapolate well.

We next focus on the more complex Figure 4.5, in which neither of the configurations reaches 0.99 within the most recent $L_{tr} = 512$ tokens. Generally, this explains why the perplexity often bumps up when L_{ex} goes from 512 to 1024: Models cannot perfectly process more tokens than they were trained on. If we take a closer look, the $\Delta = -3$ model has the strongest windowing effect and the smallest ERF=768 tokens, therefore its perplexity plateaus the soonest at $L_{ex} = 1024$ in Table 4.1. The remaining models all need ERF=2048 tokens to reach $c_m = 0.99$, which explains why their perplexities become stable only after $L_{ex} = 2048$ (Table 4.1). For $\Delta \in [6, 8]$ models specifically, the difference between L_{tr} and ERF is too large to be handled, resulting in exploded perplexities.

4.5.2 Fixing Failed Cases

We fix the failed cases in Table 4.1 section 1 (varying Δ) and section 3 (varying w) by increasing L_{tr} to cover their ERFs. We increase L_{tr} to 1024 for windowed attention with $w = 160$. For shifted ALiBi with $\Delta = 6$, we need $L_{tr} = 2048$ tokens. Table 4.2 shows that both are now able to maintain stable perplexities.

L_{ex}	Shift all h by $\Delta = 6$		Windowed Attention $w = 160$	
	Arxiv	OpenWebText2	Arxiv	OpenWebText2
2048	4.4	15.2	6.2	19.9
4096	6.2	19.8	6.2	19.9
8192	6.2	19.9	6.2	19.9

Table 4.2: Fixing failed cases with longer L_{tr}: $L_{tr} = 2048$ for ALiBi with $\Delta = 6$ and $L_{tr} = 1024$ for windowed attention with $w = 160$.

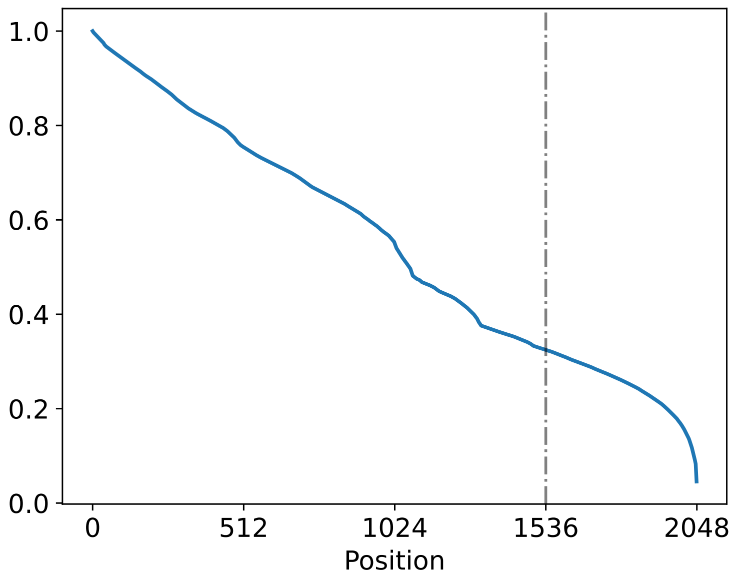


Figure 4.6: Cumulative normalized gradient of Rotary on ArXiv when predicting the last (2048-th) token with $L_{tr} = 512$.

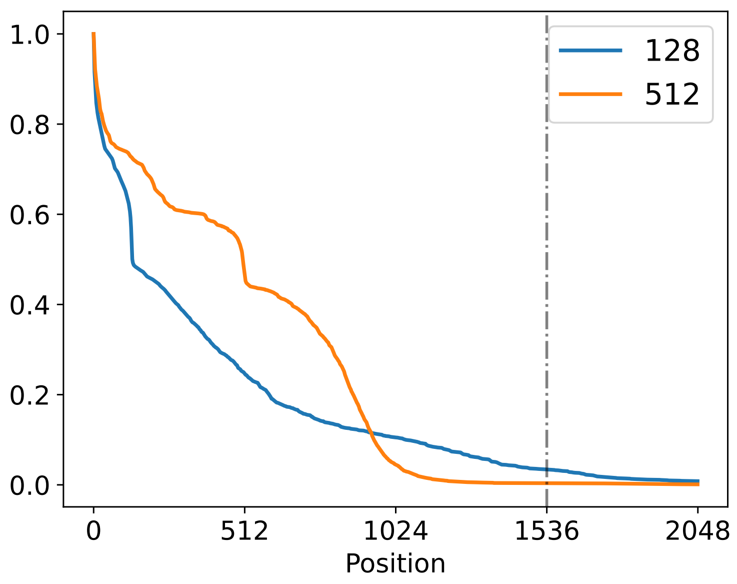


Figure 4.7: Cumulative normalized gradient of Sinusoidal on ArXiv when predicting the last (2048-th) token with $L_{tr} \in [128, 512]$.

$$(\boldsymbol{W}_q(\boldsymbol{e}_m + \boldsymbol{p}_m))^\top(\boldsymbol{W}_k(\boldsymbol{e}_n + \boldsymbol{p}_n)) = \tag{4.4}$$
$$\underbrace{\boldsymbol{e}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_n^\top}_{\text{semantic info.}} + \underbrace{\boldsymbol{e}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{p}_n + \boldsymbol{p}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_n + \boldsymbol{p}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{p}_n}_{\text{mixture of semantic and positional info.}} \approx \underbrace{\boldsymbol{e}_m^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{e}_n^\top}_{\text{semantic info.}} + \underbrace{\boldsymbol{p}_m^\top \boldsymbol{p}_n}_{\text{positional info.}}$$

4.5.3 Analyses of Sinusoidal and Rotary

Sinusoidal [Vaswani et al., 2017b] constructs the positional embedding at position m and $\forall i \in [1, d/2]$ as:

$$\begin{aligned} \boldsymbol{p}_{m,2i} &= \sin\left(\frac{m}{10000^{2i/d}}\right), \\ \boldsymbol{p}_{m,2i+1} &= \cos\left(\frac{m}{10000^{2i/d}}\right) \end{aligned} \tag{4.5}$$

They will be added with the input embeddings $\{\boldsymbol{e}_m\}_{m=1}^L$ followed by the query and key transformations as shown in Eq. (4.4). Unlike addition, Rotary [Su et al., 2021b] multiplies each token embedding $\boldsymbol{e}_m$ with a position-specific rotation matrix $\boldsymbol{R}_m\boldsymbol{e}_m$.

What could c_m tell us when it is applied to the non-extrapolatable Sinusoidal and Rotary positional embeddings? As we can see in Figure 4.6 and 4.7, they both fail to focus on the most recent L_{tr} tokens because neither of their formulations guarantees a L_{tr}-bounded receptive field. Figure 4.7 tells additional stories: To predict the last token (2048-th), Sinusoidal focuses on the 512-th token when $L_{tr} = 512$ and the 128-th token when $L_{tr} = 128$ as indicated by the sudden jump on their normalized gradient plots. This is because the model has only seen at most L_{tr} positional embeddings and overfitted on them, which provides explicit evidence to the Sinusoidal, or APE in general, overfitting hypothesis made by the author of ALiBi[4]. It also explains why RPE is a better choice for length extrapolatable Transformers: They cannot overfit on the positional embeddings.

4.6 A New RPE for Length Extrapolation

4.6.1 Introduction to Sandwich

We fix the overfitting issue of Sinusoidal by transforming it into a new RPE, **Sandwich**, shown in Eq. (4.4). Specifically, we drop the cross terms and keep only the inner product of two positional embeddings[5] at m and n. Now $\boldsymbol{p}_m^\top \boldsymbol{p}_n$ with $m, n \in [1, L]$ become the temporal bias terms of Sandwich:

$$\begin{aligned}\boldsymbol{p}_m^\top \boldsymbol{p}_n &= \sum_{i=1}^{\bar{d}/2} \sin\left(\frac{m}{10000^{2i/\bar{d}}}\right)\sin\left(\frac{n}{10000^{2i/\bar{d}}}\right) + \\ &\qquad \cos\left(\frac{m}{10000^{2i/\bar{d}}}\right)\cos\left(\frac{n}{10000^{2i/\bar{d}}}\right) \\ &= \sum_{i=1}^{\bar{d}/2} \cos\left(\frac{m-n}{10000^{2i/\bar{d}}}\right)\end{aligned}$$

A similar observation was previously made in a context different from length extrapolation [Yan et al., 2019].

The largest value of $\boldsymbol{p}_m^\top \boldsymbol{p}_n$ happens at the point where $m - n = 0$, which gives the maximum value of $\bar{d}/2$. To align L_{tr} with the ERF of Sandwich, we need to further check that $\boldsymbol{p}_m^\top \boldsymbol{p}_n$ demonstrates a similar windowed attention effect as ALiBi. This can be done by subtracting all $\boldsymbol{p}_m^\top \boldsymbol{p}_n$ by $\bar{d}/2$ and further dividing them by a set of predefined compression ratios. for the sake of simplicity, we set the compression ratios to be the same as ALiBi's $h = n \cdot \frac{8}{H}$ with $n \in \{1 \ldots H\}$:

$$\frac{\boldsymbol{p}_m^\top \boldsymbol{p}_n - \bar{d}/2}{h} \tag{4.6}$$

Eq. (4.6) is added after the scaled softmax is done in Eq. (4.1). Figures 4.8 and 4.9 show a visualization of Sandwich when $h = 8$. Sandwich indeed has the same decay-with-distance pattern as ALiBi.[6]

Note that we deliberately decouple this $\bar{d}$ from d in Eq. (4.5) since we treat $\bar{d}$ as a hyperparameter that controls the shape of Sandwich. A larger $\bar{d}$ leads to a stronger windowed attention effect as shown in Figure 4.10. We set $\bar{d} = 128$ in this chapter for all the experiments. We also experiment with smaller and larger $\bar{d}$ and only find worse performance. Finally, readers can find the reference Python implementation in § 4.9.5.

[4]`https://twitter.com/OfirPress/status/1435690039925567489`

[5]We set $\boldsymbol{p}_{m,n}$ to $2d$ as doing so gives better empirical performance; it only needs to be computed once before training.

[6]Fun fact: We imagine different compression ratios as the ways we eat sandwiches: For a huge sandwich, we have to squeeze it more to fit in our mouths!

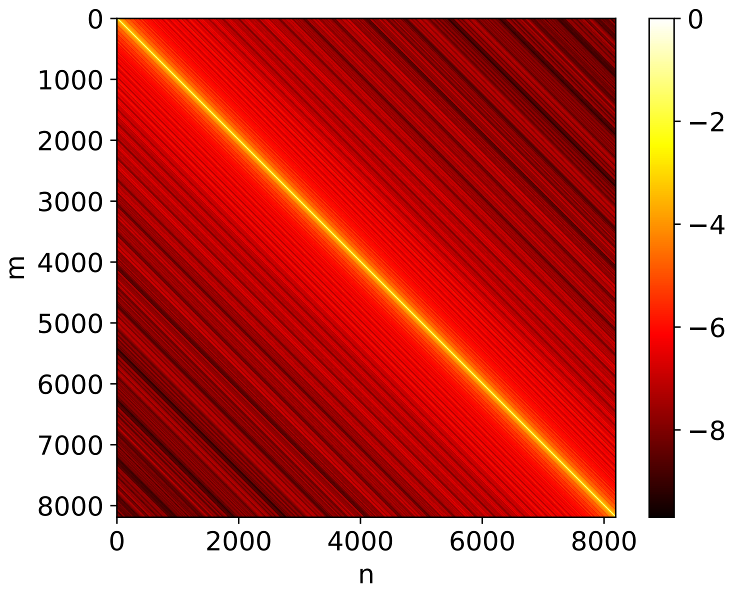


Figure 4.8: The visualization of Eq. (4.6) when the compression ratio $h = 8$ and $\bar{d} = 128$.

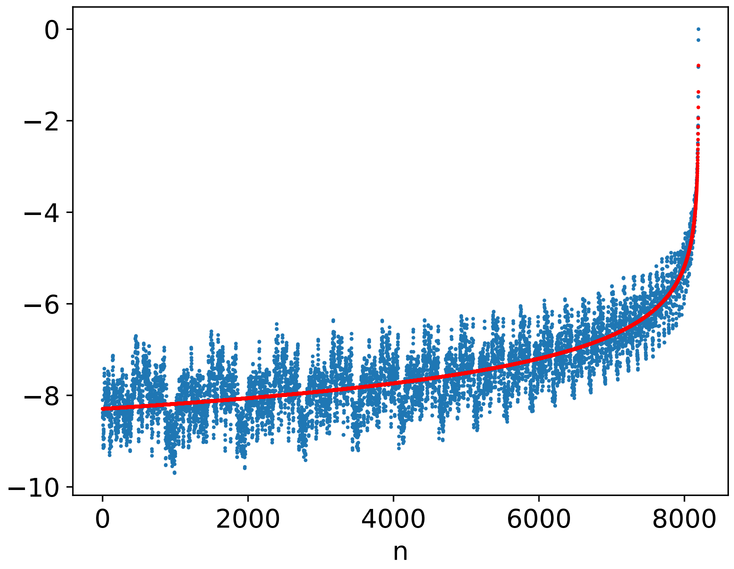


Figure 4.9: We plot the last row in Figure 4.8. The red curve is the least-squared fitted log function: $y = -0.825 \cdot \log(|m - n|) + 1) - 0.8$ with $m = 8192$ in this example.

4.6.2 Experiments and Discussion

To verify the performance of Sandwich, we train a Transformer language model following previous work [Press et al., 2022b; Chi et al., 2022]. Table 4.3 presents the results; the left part contains all

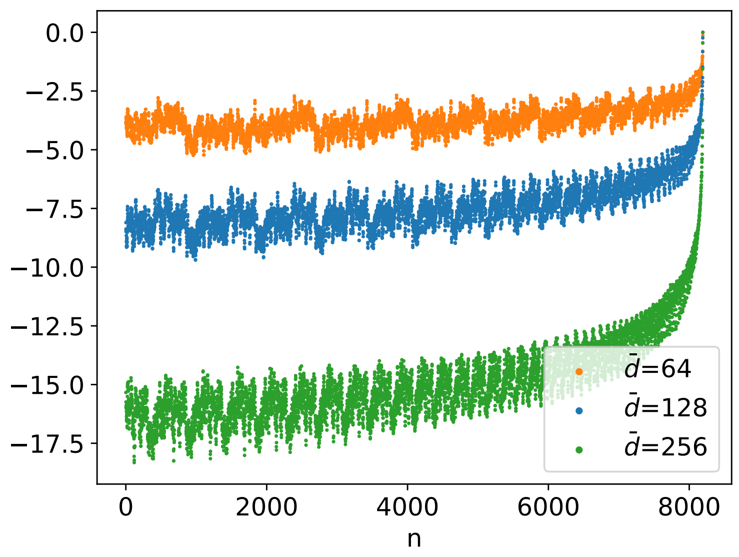


Figure 4.10: We experiment with different $\bar{d}$ following the construction of Figure 4.8 and find they create different windowed attention effect.

models without learnable parameters, and the right part contains models with learnable parameters. These numbers should not be compared across sections.

In general, models on the right achieve lower perplexities across the three datasets. This is expected as they can adapt to individual datasets more easily thanks to the additional learnable parameters. However, there is no free lunch: They often consume more GPU memory and run much slower. For example, T5 is 10% slower than Sandwich during the training stage. Note that Sandwich can also be equipped with learnable parameters such as learnable compression ratios h; this is left to future work. We now shift our focus to the left section. When $L_{ex} = L_{tr} = 512$, Sandwich is comparable to other models except that Rotary performs a bit better on OpenWebText2. Once we increase L_{ex}, Sandwich begins to reveal its advantages: On ArXiv and GitHub, it is consistently better than all the baselines but only marginally worse than ALiBi when $L_{ex} \geq 4096$ on OpenWebText2.

It is worth mentioning that Sandwich is the first parameter-free RPE that truly makes use of distant token information beyond $L_{tr} = 512$. To see this, notice that lower (better) perplexities occur at $L_{ex} > L_{tr} = 512$. The gradient analysis tool in § 4.5.1 further corroborates this in Figure 4.11, which reveals a receptive field pattern distinct from that of ALiBi and windowed attention. Even though Sandwich allocates about 60% of the total cumulative gradient on the most recent $L_{tr} = 512$ tokens, distant tokens beyond L_{tr} still contribute substantially to the model prediction.

Why do ALiBi and windowed attention need to have their ERFs covered by L_{tr} while Sandwich does not? To answer this question, we revisit Figure 4.9 and approximate (least-squared) the original temporal bias pattern using a log curve, which gives a snug fit[7]: $y = -0.825 \cdot \log(1 + |m - n|) - 0.8$.

[7]In the actual implementation, we fit the curve using the most recent 50 points of Sandwich. The reason is because the most recent tokens are more important, and we want them to be closer to the original Sandwich.

OpenWebText2							
L_{ex}	Sandwich	Smoothed	ALiBi	Sinusoidal	Rotary	KERPLE	T5
512	23.5 ± 3.8	23.2 ± 3.7	**22.8 ± 3.3**	$26 \pm 1^\dagger$	$23.0 \pm 3.4^*$	**22.6 ± 3.5***	$22.6 \pm 3.6^*$
1024	**23.0 ± 3.6**	23.1 ± 3.6	23.3 ± 3.4	$14168^\dagger$	$61^\dagger$	**22.0 ± 3.3***	$22.2 \pm 3.3^*$
2048	23.3 ± 3.5	**23.2 ± 3.2**	23.5 ± 3.3	$20370^\dagger$	$96^\dagger$	**21.9 ± 3.1***	23.0 ± 3.1
4096	23.8 ± 3.3	23.6 ± 3.0	**23.5 ± 3.3***	$42003^\dagger$	$232^\dagger$	**22.1 ± 2.9***	$26.8 \pm 3.2^\dagger$
8192	24.7 ± 3.4	24.0 ± 2.9	**23.5 ± 3.3***	$67869^\dagger$	$343^\dagger$	**22.3 ± 2.9***	$38.6 \pm 7.2^\dagger$

ArXiv							
L_{ex}	Sandwich	Smoothed	ALiBi	Sinusoidal	Rotary	KERPLE	T5
512	5.27 ± 0.33	5.33 ± 0.32	**5.25 ± 0.33**	$5.8^\dagger$	**5.25 ± 0.33**	5.22 ± 0.37	**5.16 ± 0.37***
1024	**5.05 ± 0.33**	5.13 ± 0.32	$5.41 \pm 0.36^\dagger$	$1070^\dagger$	$16.02^\dagger$	$4.95 \pm 0.34^*$	**4.91 ± 0.35***
2048	**5.02 ± 0.34**	5.15 ± 0.36	$5.58 \pm 0.40^\dagger$	$1784^\dagger$	$33.76^\dagger$	**4.83 ± 0.35***	$4.92 \pm 0.35^*$
4096	**5.15 ± 0.39**	5.33 ± 0.39	$5.58 \pm 0.40^\dagger$	$18050^\dagger$	$71.96^\dagger$	**4.84 ± 0.34***	5.35 ± 0.36
8192	**5.28 ± 0.44**	5.45 ± 0.42	$5.58 \pm 0.40^\dagger$	$44100^\dagger$	$111^\dagger$	**4.90 ± 0.33***	$6.74 \pm 0.90^\dagger$

GitHub							
L_{ex}	Sandwich	Smoothed	ALiBi	Sinusoidal	Rotary	KERPLE	T5
512	2.88 ± 0.12	2.88 ± 0.17	$2.83 \pm 0.11^\dagger$	$4^\dagger$	**2.82 ± 0.11**	$2.81 \pm 0.14^*$	**2.76 ± 0.14***
1024	2.71 ± 0.09	**2.70 ± 0.07**	$2.97 \pm 0.11^\dagger$	$8342^\dagger$	$3.86 \pm 0.25^\dagger$	$2.67 \pm 0.10^*$	**2.61 ± 0.08***
2048	**2.69 ± 0.11**	2.74 ± 0.08	$3.01 \pm 0.10^\dagger$	$9179^\dagger$	$5.94 \pm 0.64^\dagger$	$2.65 \pm 0.10^*$	**2.65 ± 0.05**
4096	**2.73 ± 0.12**	2.78 ± 0.08	$3.01 \pm 0.10^\dagger$	$11017^\dagger$	$11.1 \pm 1.55^\dagger$	**2.70 ± 0.09**	2.91 ± 0.12
8192	**2.79 ± 0.15**	2.83 ± 0.08	$3.01 \pm 0.10^\dagger$	$11270^\dagger$	$20.2 \pm 2.75^\dagger$	**2.75 ± 0.08**	$3.68 \pm 0.50^\dagger$

Table 4.3: Perplexity comparison on the OpenWebText2, GitHub, and ArXiv datasets. All models are trained for 50k steps with a training length of 512 and five random seeds. The models in the left section have parameter-free positional embeddings. In contrast, both KERPLE and T5 are equipped with learnable parameters. A fair comparison should only be made within the same section. $x^\dagger$ means sandwich is statistically significantly *better* than x. x^* means sandwich is statistically significantly *worse* than x. The test used is paired two-sided t-test with $\alpha = 0.05$. More details about the datasets and hyperparameters are provided in § 4.9.3 and 4.9.4.

Table 4.3 shows its language modeling performance under the "smoothed" column. Pictorially, the log curve decays relatively fast when two tokens are nearby and plateaus when the distance between them increases. In other words, tokens that are far away from the last one ($m = 8192$) share similar temporal biases, possibly leading to beneficial averaging and denoising effects. Note that the averaging effect does not come out of thin air during the extrapolation stage: The almost linear segment ranging from 1536 to 1792 suggests that Sandwich was trained to perform averaging within L_{tr}; it just needs to average over more historical tokens when it extrapolates to longer L_{ex}. In contrast, ALiBi's linear bias lacks the middle ground to learn the averaging behavior: It either decays so fast that distant tokens are masked out or so slow that the ERF becomes much greater than L_{tr}. The averaging hypothesis also explains why Sandwich, KERPLE, and T5's perplexities go up in Table 4.3 instead of continuing to decrease after some L_{ex} (4096 on ArXiv for example): While averaging and denoising improve performance, doing so over too many historical tokens (very large L_{ex}) will reintroduce noises.

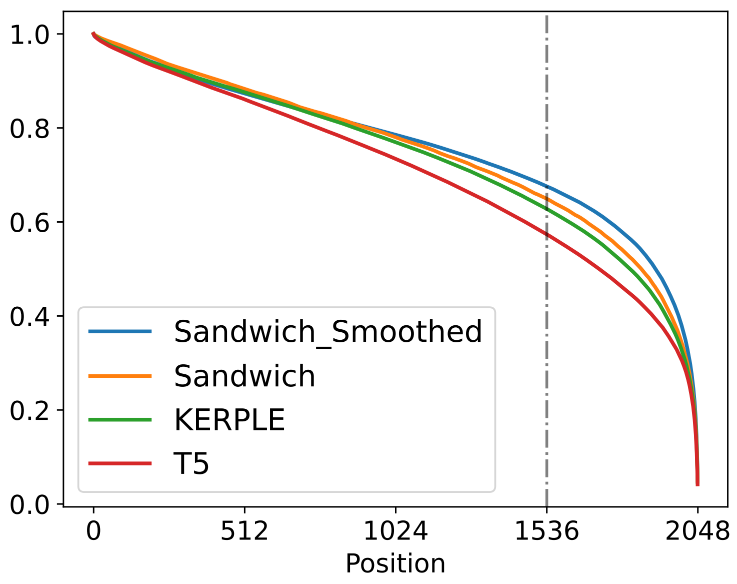


Figure 4.11: Cumulative normalized gradient of Sandwich, Smoothed Sandwich, KERPLE, and T5 on ArXiv when predicting the last (2048-th) token with $L_{tr} = 512$.

4.6.3 Connection to KERPLE and T5

KERPLE [Chi et al., 2022] has the formulation of $c - r_1 \cdot \log(1 + r_2|m - n|)$. The -0.8 in our fitted log curve term can be absorbed by c, as Softmax is shift-invariant, and if we set $r_1 = 0.825$ and $r_2 = 1$, Sandwich becomes a special case of KERPLE. T5 [Raffel et al., 2020a] adopts the log-binning strategy that assigns distinct bins to nearby tokens whereas distant tokens all share the same bin. In spirit, T5 treats distant tokens similarly to Sandwich. Figure 4.11 verifies that all three of them share a similar empirical receptive field pattern.

4.7 Conclusion

In this chapter, we first establish the connection between ALiBi and windowed attention through their constructions and language modeling performance. We then develop a cumulative normalized gradient tool to measure the empirical receptive field. This shows that length extrapolation of ALiBi and windowed attention is possible when the training sequence length covers the empirical receptive field. It also reveals the models' limitation of not utilizing information beyond the training sequence length. Fortunately, this is overcome by our new relative positional embedding, Sandwich, which is simplified from the earliest proposed Sinusoidal positional embedding. Finally, Sandwich demonstrates a log-decaying temporal bias pattern similar to that previously seen in the design of KERPLE and T5, and such pattern is likely to be the secret to successful length extrapolation. Together these findings supports more effective design of future extrapolatable Transformer language models.

4.8 Limitations

Although Sandwich, KERPLE, and T5 use information beyond training sequence length, their receptive fields still highly favor the most recent tokens. While this recency bias is beneficial to the modeling of human-written text, it is problematic in other scenarios.

Let us consider the task of *parity* prediction: A model needs to predict whether a bit string has an even or odd number of ones. For example, the parity of [1, 1, 0, 1] is odd (or 1) and the parity of [1, 0, 1, 0] is even (or 0). Unlike human-written text, every single bit is equally important. Transformer language models with current RPEs still struggle on this simple task [Anil et al., 2022]. Its difficulty can be explained by the recency bias effect that we described. Devising a new positional embedding or Transformer model architecture that solves this problem is a promising direction for future work.

4.9 Additional Experimental Detail

4.9.1 Results on OpenWebText2

L_{ex}	Shift all h by Δ						Same h for all heads					Windowed Attention Size w					
	Δ:-3	0	2	4	6	8	h:0	2	4	6	8	w:40	80	100	120	160	320
512	18.6	19.0	19.5	20.0	20.5	20.5	32.7	22.2	19.7	19.7	20.5	25.3	23.7	23.1	24.0	22.9	21.9
1024	21.6	19.3	19.6	24.8	232	232	32.8	23.2	24.9	146	232	25.3	23.7	23.2	137	234	353
2048	21.6	19.7	20.5	29.3	299	299	32.8	23.2	24.9	165	299	25.3	23.7	23.2	137	236	408
4096	21.6	19.7	20.5	29.4	299	299	32.9	23.2	24.9	165	299	25.3	23.7	23.2	137	236	408
8192	21.6	19.7	20.5	29.4	299	299	32.9	23.2	24.9	165	299	25.3	23.7	23.2	137	236	408

Table 4.4: The three experiments on the OpenWebText2 dataset.

Table 4.4 includes the three experiments conducted in § 4.4 on OpenWebText2. Their corresponding receptive field plots are shown in Figure 4.12 and 4.13.

4.9.2 Efficient Inference

Although ALiBi might not be using token information further than L_{tr}, it has the nice property of efficient inference [Press, 2022]. Tables 4.1 and 4.4 show that ALiBi perplexities remain constant when $L_{ex} \geq 2048$. This suggests a cache window size $\bar{w} = 2048$ for inference. The generation of the first $\bar{w}$ tokens remains the same, and we can still cache all $\boldsymbol{q}_m$, $\boldsymbol{k}_m$, and $\boldsymbol{v}_m$ vectors for $m \in [1, 2048]$. When it comes to generating the $\bar{w} + 1$-th token, we simply discard the first cached $\boldsymbol{q}_1$, $\boldsymbol{k}_1$, and $\boldsymbol{v}_1$ and use the rest of $\bar{w} - 1$ tokens along with the newly added token to perform self-attention. If we want to generate a length L_{ex} text snippet, the complexity is $O(\bar{w} \times L_{ex})$ instead of $O(L_{ex}^2)$. This complexity is also better than that of an APE model, which is $O(\bar{w}^2 \times L_{ex})$ since an APE model needs to completely re-encode the previous $\bar{w}$ vectors when generating new tokens following the first $\bar{w}$ ones.

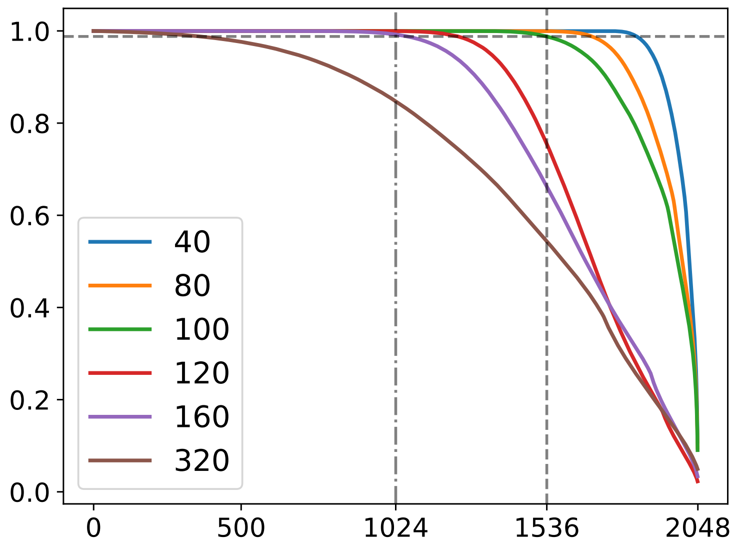


Figure 4.12: Cumulative normalized gradient on OpenWebText2 when predicting the last (2048-th) token. Windowed Attention Size $w = 40, 80, \cdots, 320$. This graph is constructed in the same way as Figure 4.4.

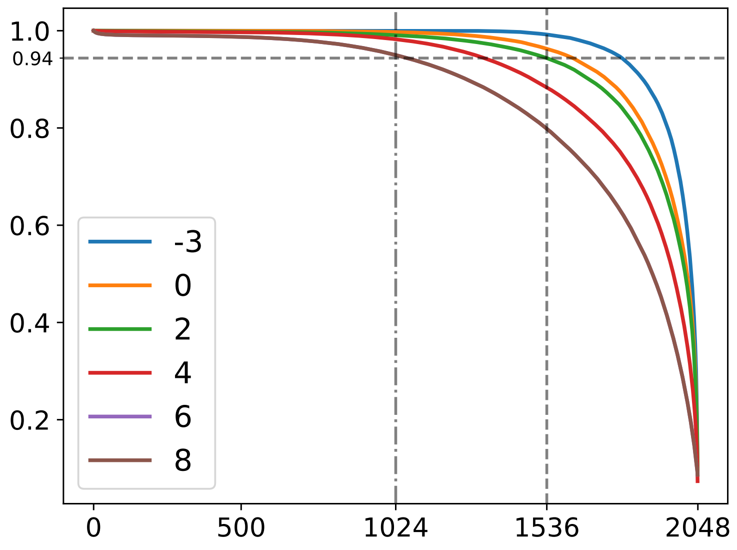


Figure 4.13: Cumulative normalized gradient on OpenWebText2 when predicting the last (2048-th) token. Shift all h by $\Delta = -3, 0, \cdots, 8$. This graph is constructed in the same way as Figure 4.5.

We implement the process discussed above to verify that ALiBi indeed allows for efficient inference. The results, along with ones for Sandwich, are presented in Table 4.5. Both ALiBi and Sandwich permit efficient inference by setting $\bar{w} = 2048$. It is worth pointing out that the performance of Sandwich at $L_{ex} = 4096$ becomes a bit worse compared to that in Table 4.3. This is more evidence that Sandwich is using longer than L_{tr} token information.

L_{ex}	OpenWebText2		Arxiv		GitHub	
	Sandwich	ALiBi	Sandwich	ALiBi	Sandwich	ALiBi
4096	23.9	23.5	5.31	5.59	2.79	3.01
8192	24.1	23.5	5.35	5.59	2.81	3.01
16384	24.1	23.5	5.35	5.59	2.81	3.01

Table 4.5: Efficient Inference with $\bar{w} = 2048$. The numbers are perplexities.

4.9.3 Scientific Artifacts

	OpenWebText2	GitHub	ArXiv
Raw Size	66.77 GB	95.16 GB	56.21 GB
Type	Internet	Coding	Academic

Table 4.6: Dataset overview. Raw Size is the size before any up- or down-sampling.

We use the gpt-neox library [Andonian et al., 2021] under Apache-2.0 license and the datasets [Gao et al., 2020a] released by the authors of gpt-neox. The codebase and datasets (Table 4.6) are publicly released for research purposes. The steps taken to protect the privacy and anonymization are discussed in Gao et al. [2020a] section 6 and 7. Finally, Gao et al. [2020a] section 5 also discusses the distribution and statistics of the datasets used in this chapter.

4.9.4 Implementation Details

The configurations and hyperparameters are outlined in Table 4.7. The pretraining takes 5 hours on a single NVIDIA A-100 GPU. We do not tune any hyperparameters and just use the default ones.

# Layers	Hidden Size	# Attention Heads	Train Seq. Len.	# Trainable Params.
12	64	12	512	162M
Optimizer	Batch Size	Train Steps	Precision	# Trainable Params. for RPEs
Adam (lr 6e-4)	32	50,000	bfloat16	0

Table 4.7: 162M model configurations.

4.9.5 Python Implementation of Sandwich

```
import numpy as np

base = 1e4
heads = 12
seq_len = 8192
```

```
positions = np.arange(seq_len)[..., None]
bar_d = 128 # This is the hyperparameter of Sandwich
i = np.arange(bar_d // 2)

pos_embs = np.concatenate([np.sin(positions / base ** (2 * i / bar_d)),
                          np.cos(positions / base ** (2 * i / bar_d))],
                          axis=-1)
sandwich = np.matmul(pos_embs, pos_embs.T)
compression_ratio = np.arange(1, heads + 1) * 8 / heads
multi_head_sandwich = sandwich[None, ...] \\
                      / compression_ratio[..., None, None]
```

Part III

Length Extrapolation Beyond Natural Language Modeling

Chapter 5

Attention Alignment and Flexible Positional Embeddings Improve Transformer Length Extrapolation

5.1 Introduction

So far, all the length-extrapolatable Transformer designs are tailored for natural language modeling, a task known to have strong recency bias, and they often do not perform well on other seemingly simple tasks such as passkey, topic, and line retrieval [Mohtashami and Jaggi, 2023; Li et al., 2023]. To circumvent the recency bias, we sift through the positional embeddings of existing open-source large pre-trained Transformer language models, shown in Table 5.2, to find a flexible design, and the T5 family [Raffel et al., 2020b] comes to our attention. As visualized in Figure 5.1, the flexibility of T5's positional embeddings allows it to encourage recency bias on one head and discourage that on another head. However, there is no free lunch: T5 suffers from the dispersed attention issue as shown in Table 5.1. That is, the attention distributions of long input sequences tend to be flatter than those of short input sequences. As a remedy, we propose two fine-tuning-free attention alignment strategies via Softmax temperature scaling [Yao et al., 2021; Su, 2021] to mitigate the dispersed attention issue: maximum probability ($\mathrm{P}_{\max}$) and entropy (H) alignment.

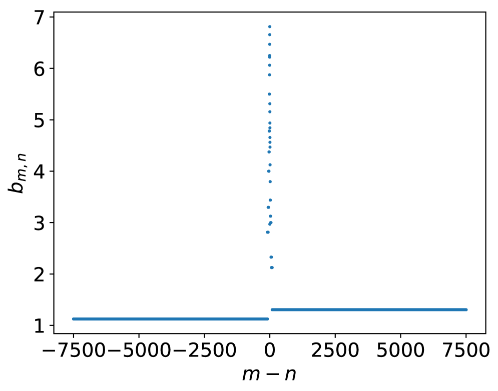
	Retrieval Tasks					
Criteria	Topic		Line		Passkey	
	512	15k	512	15k	512	15k
$\mathrm{P}_{\max}$	0.28	0.12	0.27	0.11	0.32	0.24
H	3.47	6.63	3.47	7.04	3.09	5.97

Table 5.1: The dispersed attention issue of Flan-T5-XL encoder. $\mathrm{P}_{\max}$ is the average maximum probability and H is the average entropy. After increasing the sequence length from 512 to 15k, we observe larger entropy and smaller maximum probability, implying a flatter self-attention distribution.

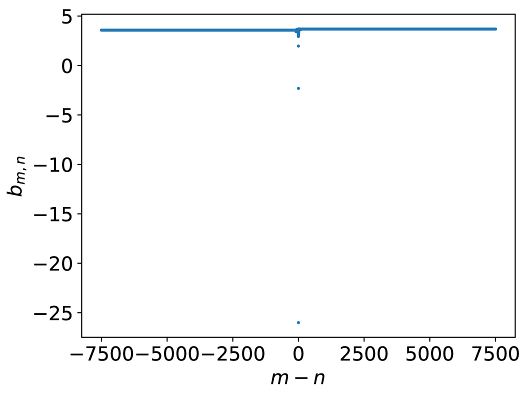
Models	T5 2020b	OPT 2022	ChatGLM 2022	LLaMA 2023	Falcon 2023	Pythia 2023	XGen 2023	BLOOM 2022a	MPT 2023
PE.	Learned Relative	Learned Absolute	Rotary Relative	Rotary Relative	Rotary Relative	Rotary Relative	Rotary Relative	ALiBi Relative	ALiBi Relative

Table 5.2: Positional embeddings of open-source Transformer language models. T5 is the only model equipped with learnable relative positional embeddings, which enable its long-context utilization capability.

We validate the effectiveness of our alignment strategies on tasks including language modeling, retrieval, multi-document question answering, and code completion. We also provide a theoretical analysis of how the alignment strategies work under the hood by investigating the relation between the Softmax temperature and data distribution.



(a) 1st Attention Head



(b) 27nd Attention Head

Figure 5.1: Visualization of T5 positional embeddings. To plot figures of $b_{m,n}$, we set $m = 7500$ and vary the value of n from 0 to 15k. Each attention head of a Flan-T5-XL encoder learns a set of positional embeddings that capture different attention bias. For example, the positional embeddings in the left figure encourage the model to focus on nearby tokens. In contrast, the ones in the right figure let the model focus on only remote tokens.

5.2 Related Work

Transformer Positional Embeddings Transformer-based models rely on positional embeddings to encode positional information. We summarize open-source large pre-trained Transformer language models and their positional embeddings in Table 5.2. The relative variants are widely adopted due to their better empirical performance [Su et al., 2021a] and possible length-extrapolation capability [Press et al., 2022a]. In this chapter, we place special focus on the T5 positional embeddings due to their flexibility as shown in Figure 5.1.

Transformer Length Extrapolation Existing research on Transformer length extrapolation is mostly confined to the task of natural language modeling [Press et al., 2022a; Chi et al., 2022, 2023b]. Unfortunately, the reported positive results do not carry over to long-context retrieval [Mohtashami and Jaggi, 2023; Li et al., 2023]. This contrastive observation can be explained by models' short empirical receptive field [Chi et al., 2023b]. Specifically, the strong decaying prior of positional embeddings prevents models from accessing distant tokens that may be necessary for retrieval tasks. In this chapter, we improve the flexible positional embeddings of T5 to get around this limitation.

Transformer Position Interpolation Instead of performing direct length extrapolation, a different line of research conducts model fine-tuning on long input sequences [Chen et al., 2023], where the main focus is to identify the most efficient fine-tuning scheme that can improve long-context utilization. Positive results have been reported on retrieval tasks [Li et al., 2023]. However, we argue that fine-tuning incurs additional costs since it needs 1) GPU resources to perform long sequence fine-tuning with large models and 2) a pre-defined target sequence length, which still imposes a sequence length upper limit. Our proposed methods can circumvent these two limitations.

Retrieval Tasks with Transformers Transformer-based approaches often consist of a retriever and a reader to overcome the context length restriction [Guu et al., 2020; Lewis et al., 2020; Izacard and Grave, 2021; Borgeaud et al., 2022]. The retriever retrieves relevant text snippets from a very large database and the reader digests the retrieved information to generate the correct output. Our proposed attention alignment strategy can be used to significantly increase the input sequence length of the reader, thereby allowing more retrieved information to participate in the decision process. For small-scale retrieval problems, our methods even obviate the need for context segmentation and the external key-value store used in prior work [Mohtashami and Jaggi, 2023], serving as a more elegant approach.

Softmax Temperature Scaling To increase the length extrapolation capability of Transformers, previous work [Yao et al., 2021; Su, 2021; Peng et al., 2023b] scales the temperature of Softmax logarithmically w.r.t the sequence length. Our entropy alignment strategy is also inspired by this line of research except that we adopt a different procedure outlined below in Algorithm 1. Interestingly, our results in § 5.7 show that the logarithmic temperature scaling scheme is more similar to our proposed maximum probability alignment strategy.

5.3 Long-context Retrieval Tasks with T5

5.3.1 Why Retrieval?

As suggested by recent work [Mohtashami and Jaggi, 2023; Li et al., 2023], the task of long-context retrieval serves as a controllable benchmark to measure how well a Transformer language model utilizes long-context inputs. One prominent characteristic of retrieval tasks is that only a subset of the input is of interest, requiring a model to accurately pick up the necessary information. The other characteristic is that the key information can sit anywhere in an input, requiring a model to attend flexibly. Finally, the controllable aspect allows us to gradually increase the input sequence length to test the models' length extrapolation capability.

5.3.2 Why T5?

To solve retrieval tasks using Transformer language models, it is necessary to choose a positional embedding design that permits accurate and flexible length-extrapolatable attention. After checking through the existing positional embeddings in Table 5.2, we find that the T5 family [Raffel et al., 2020b] fits our needs. As for other candidates, learnable absolute positional embeddings [Vaswani et al., 2017b; Zhang et al., 2022] must be evaluated within the training length. ALiBi [Press et al., 2022a] and Rotary [Su et al., 2021a] have a recency bias; they cannot extrapolate easily without fine-tuning.

For each attention head, T5 encoder maintains a bucket (B) of 32 learnable parameters and assigns the relative positional bias (rpe bias) $b_{m,n}$ as[1]

$$b_{m,n} = \begin{cases} \mathrm{B}[m-n], \text{ if } 0 \leq m-n < 8 \\ \mathrm{B}[n-m+16], \text{ if } -8 < m-n < 0 \\ \mathrm{B}[\min(15, 8 + \lfloor \frac{\log((m-n)/8)}{\log(128/8)} \cdot 8 \rfloor)], \text{ if } 8 \leq m-n \\ \mathrm{B}[\min(31, 24 + \lfloor \frac{\log((n-m)/8)}{\log(128/8)} \cdot 8 \rfloor)], \text{ if } m-n \leq -8, \end{cases}$$

where $0 \leq m < L$ and $0 \leq n < L$ are two position indices. $b_{m,n}$ will be added to the (m,n)-th entry of the $L \times L$ self-attention matrix. The summation becomes the input to the temperature-scaled Softmax. We plot the learned rpe bias of a T5 encoder in Figure 5.1. We can tell that its attention heads encode rich attention patterns. For example, head 1 learns to focus on the nearby tokens whereas head 27 learns to ignore the nearby tokens and allow access to faraway tokens.

5.3.3 The Dispersed Attention Issue of T5 Encoder

Unfortunately, directly applying T5 models on retrieval tasks does not yield perfect results. Upon inspecting the intermediate model states, we find that a longer input sequence consists of more tokens competing for the same amount (i.e., Softmax sums to 1) of attention, resulting in the dispersed attention issue. In Table 5.1, we see that the longer the input sequence, the flatter the self-attention distribution. The situation is not hopeless if the desired information still attains a higher attention weight than the remaining tokens. Our proposed solution in § 5.4 will let the key information stand out.

[1]`https://github.com/huggingface/transformers/blob/v4.33.2/src/transformers/models/t5/modeling_t5.py#L390`

Algorithm 1 Attention Alignment Strategies

```
Require: A short sequence of length $L_{tr}$ and a long sequence of length $L_{ex} > L_{tr}$. Encoder $E$. Alignment mode $M$.
Ensure: The Softmax temperature $\tau$
  function FindS($\tau$, $M$)
      Set temperature of all Softmax to $\tau$
      $s \leftarrow [\,]$
      for operation in $E$ do
          Perform the operation
          if operation is Softmax$_\tau(l)$ then
              if $M$ is Maximum Probability then
                  Append $\max($Softmax$_\tau(l))$ to $s$
              else if $M$ is Entropy then
                  Append $\mathrm{H}($Softmax$_\tau(l))$ to $s$
              end if
          end if
      end for
      return avg$(s)$
  end function
  $\overline{S^{tr}}(1) \leftarrow \textsc{FindS}(1.0, M)$
  for $\tau_{ex} = 1.0, 0.95, 0.9, \cdots, 0.5$ do
      $\overline{S^{ex}}(\tau_{ex}) = \textsc{FindS}(\tau_{ex}, M)$
  end for
  return $\tau_{ex}$ s.t. $\overline{S^{ex}}(\tau_{ex}) \approx \overline{S^{tr}}(1)$
```

5.4 Proposed Methods

A natural solution to the dispersed attention issue described in § 5.3 is to sharpen the self-attention distribution. This can be achieved by reducing the temperature τ during extrapolation. We set the extrapolation temperature τ_{ex} such that the sharpness during training with $\tau_{tr} = 1$ and that during extrapolation with $\tau_{ex} < 1$ are roughly the same. As a measurement of sharpness, we explore the maximum probability or entropy of a distribution. In other words, our proposed solution is to align the maximum probability or entropy of training and extrapolation distributions by adjusting τ_{ex}.

Concretely, let $l^{(i)} \in \mathbb{R}^L$ be the i-th pre-Softmax logit vector of a T5 encoder, where $L \in \{L_{tr}, L_{ex}\}$ is the sequence length. The post-Softmax distribution of $l^{(i)}$ is $\mathrm{P}^{(i)}(\tau) = \mathrm{Softmax}_\tau(l^{(i)})$. The maximum probability and entropy of $\mathrm{P}^{(i)}(\tau)$ are $\mathrm{P}^{(i)}_{\max}(\tau)$ and $\mathrm{H}^{(i)}(\tau)$, respectively.

Take the maximum probability alignment strategy as an example: We first run the forward pass and compute the average maximum probability under temperature τ over all logit vectors: $\overline{\mathrm{P}_{\max}}(\tau) = (1/N)\sum_i \mathrm{P}^{(i)}_{\max}(\tau)$ where $N = R \times H \times L$ is the number of logit vectors in a T5 encoder with R layers, H heads, and length-L sequences. Since the temperature is 1 during training and τ_{ex} during extrapolation, we denote the average maximum probability during training as $\overline{\mathrm{P}^{tr}_{\max}}(1)$ and that during extrapolation as $\overline{\mathrm{P}^{ex}_{\max}}(\tau_{ex})$. Finally, to align the maximum probabilities, we adjust τ_{ex} s.t. $\overline{\mathrm{P}^{ex}_{\max}}(\tau_{ex}) \approx \overline{\mathrm{P}^{tr}_{\max}}(1)$. In practice, we do a grid search on τ_{ex} from 1.0 to 0.5. We outline the procedure of the alignment strategies in Algorithm 1.

Note that our proposed methods do not require any model fine-tuning or gradient computations. The only overhead is estimating the temperature τ_{ex} using Algorithm 1 and a few length L_{ex} sequences. Once the temperature is decided, it will be held fixed, rendering our methods simple and efficient. In addition, our fine-tuning free methods do not lead to performance regression on short L_{tr} sequences commonly observed on long-context fine-tuned models [Roziere et al., 2023].

5.5 Experiments

We compare the two alignment strategies against the length-only Softmax temperature scaling scheme $\tau = \log_{L_{ex}} L_{tr}$ [Yao et al., 2021; Su, 2021] and LongChat-13B-16K [Li et al., 2023]. Note that LongChat-13B-16K [Li et al., 2023], the best baseline, was fine-tuned from LLaMA [Touvron et al., 2023] on long sequences of length 16k while our proposed methods do not need any fine-tuning. Our experiments are conducted on an A6000 GPU.

5.5.1 Language Modeling

We use the LM-Adapted T5 models for this experiment[2]. We set $L_{tr} = 512$. Following previous work on Transformer length extrapolation, we perform an intrinsic evaluation on language modeling [Press et al., 2022a; Chi et al., 2022, 2023b]. Ideally, our proposed methods should alleviate the perplexity explosion problem during extrapolation. As we can see in Table 5.3, both alignment

[2]https://github.com/google-research/text-to-text-transfer-transformer/blob/main/released_checkpoints.md#lm-adapted-t511lm100k

Models	Language Modeling					
	Sequence Length (L_{ex})					
	1024	2048	4096	8192	15000	Avg.
T5-Large-LM	35.9	40.1	>1k	>1k	>1k	> 1k
w/ $\mathrm{P}_{\max}$	34.7	45.5	45.2	45.5	52.7	**44.7**
w/ H	40.2	43.9	45.6	54.6	56.0	48.1
w/ $\log_{L_{ex}} L_{tr}$	39.8	38.2	47.4	45.3	55.9	45.3
T5-XL-LM	28.3	>1k	>1k	>1k	>1k	> 1k
w/ $\mathrm{P}_{\max}$	30.2	36.0	31.6	41.7	50.0	37.9
w/ H	30.4	36.8	38.4	53.3	63.4	44.4
w/ $\log_{L_{ex}} L_{tr}$	27.3	29.4	31.7	39.3	45.8	**34.7**
T5-XXL-LM	109	>1k	>1k	>1k	>1k	> 1k
w/ $\mathrm{P}_{\max}$	32.2	29.7	29.5	36.6	44.3	34.5
w/ H	26.8	28.1	34.2	37.8	43.8	**34.1**
w/ $\log_{L_{ex}} L_{tr}$	27.1	36.1	33.9	246	43.8	77.5

Table 5.3: Language modeling performance. We report the average perplexity of 500 sequences. The lower the better.

Models	Retrieval Tasks																
	Topic, # of topics					Line, # of lines						Passkey, # of sentences					Avg.
	5	10	15	20	25	200	300	400	500	600	680	20k	30k	40k	50k	55k	
Flan-T5-Large	99	100	97	97	83	97	100	92	96	93	92	62	47	31	16	9	76
w/ $\mathrm{P}_{\max}$	96	90	86	94	98	99	98	98	98	98	100	84	90	85	79	85	**92**
w/ H	59	32	16	0	3	97	90	94	83	93	88	29	25	21	15	22	48
w/ $\log_{L_{ex}} L_{tr}$	88	79	75	61	55	99	99	98	99	97	98	74	63	51	41	35	76
Flan-T5-XL	100	100	100	100	100	96	90	77	57	45	26	100	100	100	100	100	87
w/ $\mathrm{P}_{\max}$	100	100	100	100	100	97	90	89	80	70	62	100	99	100	100	100	**93**
w/ H	99	98	97	96	96	95	87	88	79	70	71	100	100	100	100	100	92
w/ $\log_{L_{ex}} L_{tr}$	99	100	100	100	100	98	88	81	86	60	67	100	100	100	100	99	92
Flan-T5-XXL	100	100	100	99	99	100	100	98	95	84	82	100	100	100	100	100	97
w/ $\mathrm{P}_{\max}$	100	100	100	99	99	97	99	96	97	94	95	100	98	100	100	100	**98**
w/ H	100	100	97	98	94	99	92	92	76	58	58	100	100	100	100	100	92
w/ $\log_{L_{ex}} L_{tr}$	100	100	99	98	92	100	98	94	93	84	90	100	100	100	100	100	97
LongChat	100	100	100	99	89	100	91	93	83	78	59	100	100	99	100	99	93

Table 5.4: Performance of retrieval tasks. Each number is the averaged accuracy computed over 100 sequences. The LongChat model corresponds to LongChat-13B-16K [Li et al., 2023]. It is a LLaMA-13B [Touvron et al., 2023] model fine-tuned on sequences of length 16k using positional interpolation [Chen et al., 2023]. Flan-T5-XXL has 11B parameters. The maximum sequence lengths (L_{ex}) of the three tasks are around 14.5k to 15.5k tokens.

strategies dramatically improve (lower) the perplexity. We also observe that scaling the temperature solely based on sequence lengths is not the optimal strategy, as indicated by the sudden perplexity increase of the $\log_{L_{ex}} L_{tr}$ strategy. We will provide an in-depth discussion on this topic in § 5.7.

Note that perplexity is not our primary focus since it often cannot accurately reflect the long-context utilization capability of Transformers on practical tasks [Li et al., 2023].

5.5.2 Long-context Retrieval

The tasks are formulated in the Question Answering (QA) format; therefore, we use the Flan-T5 models to leverage their instruction-following capability. We set $L_{tr} = 512$. Inspired by recently proposed retrieval tasks, we evaluate the proposed alignment strategies on three of these. Topic retrieval requires a model to retrieve the first topic in a long and multi-topic conversation [Li et al., 2023]. Line retrieval has a long series of key-value pairs, and a model needs to retrieve the value corresponding to the given key [Li et al., 2023]. Passkey retrieval hides a passkey in a long junk text snippet, and a model needs to return that passkey [Mohtashami and Jaggi, 2023].

As we can see in Table 5.4, the retrieval performance is greatly boosted after the Flan-T5 models are equipped with our proposed attention alignment strategies. In particular, the maximum probability alignment strategy provides better results across the board. Other baselines such as MPT [Team, 2023] and ChatGLM2 [Du et al., 2022] perform worse than LongChat. Please refer to Li et al. [2023] for more details. We also present the optimal temperature given by Algorithm 1 in Table 5.9 in § 5.10.5. In short, the temperature decreases when the input sequence length increases. We will provide additional temperature analysis below, in §5.7.

Multi-document Question Answering										
Models	10 Docs	20 Docs	30 Docs, golden doc at different positions							
	Avg.	Avg.	0	4	9	14	19	24	29	Avg.
Flan-T5-Large	52.4	43.3	52.6	42.0	36.5	34.0	33.9	33.9	37.9	38.7
w/ $\mathrm{P_{max}}$	**53.1**	**44.2**	50.8	44.5	39.5	36.4	35.9	35.8	37.0	**40.0**
Improvement	+0.7	+0.9	-1.8	+1.5	+3.0	+2.4	+2.0	+1.9	-0.9	+1.3
w/ H	52.1	43.2	47.6	41.1	35.2	33.5	32.2	33.3	34.2	36.7
w/ $\log_{L_{ex}} L_{tr}$	53.2	44.5	50.6	44.1	39.3	36.3	35.8	35.8	37.2	39.9
Flan-T5-XL	59.4	51.2	58.4	44.6	40.0	39.9	41.7	46.4	54.8	46.5
w/ $\mathrm{P_{max}}$	**61.1**	**53.6**	60.9	49.1	46.0	44.9	46.3	49.1	55.7	**50.3**
Improvement	+1.7	+2.4	+2.5	+4.5	+6.0	+5.0	+4.6	+2.7	+0.9	+3.8
w/ H	60.5	52.4	52.4	43.5	42.1	40.3	42.0	42.9	51.3	44.9
w/ $\log_{L_{ex}} L_{tr}$	60.9	**53.6**	61.0	49.1	46.1	44.7	46.1	48.7	55.4	50.2
Flan-T5-XXL	63.6	56.9	58.9	49.1	48.1	47.5	48.9	53.1	61.2	52.4
w/ $\mathrm{P_{max}}$	63.7	**57.7**	60.4	52.5	51.0	50.2	51.3	53.5	59.1	**54.0**
Improvement	+0.1	+0.8	+1.5	+3.4	+2.9	+2.7	+2.4	+0.4	-2.1	+1.6
w/ H	63.6	57.1	61.0	53.4	50.8	50.3	50.7	51.9	55.7	53.4
w/ $\log_{L_{ex}} L_{tr}$	**63.9**	57.6	61.5	53.3	51.3	50.3	51.1	53.0	57.2	**54.0**

Table 5.5: Performance of multi-document QA. Numbers are accuracy. Full score is 100. The maximum sequence length (L_{ex}) of 30 documents is around 5k. The improvement row represents the absolute accuracy improvement after a Flan-T5 model is equipped with our proposed maximum probability alignment strategy. For the full performance breakdown, please refer to Table 5.14 in § 5.10.7.

5.5.3 Multi-document Question Answering

We again use the Flan-T5 models to leverage their instruction-following capability. We set $L_{tr} = 512$. We follow the multi-document question-answering task settings and data splits detailed in Liu et al. [2023b]. In short, the input consists of a question Q and multiple documents extracted from NaturalQuestions [Kwiatkowski et al., 2019] related to Q, where one of the documents (golden doc) contains the ground truth answer to Q. As shown in Table 5.5, when a model is equipped with the proposed maximum probability alignment strategy, it consistently outperforms the original model across model sizes and number of input documents.

Apart from the better task performance, we believe that the attention dispersed attention issue discussed in § 5.3 can help demystify the lost-in-the-middle phenomenon [Liu et al., 2023b] of this task: Transformer models tend to perform worse when the ground truth sits near the middle of the input context. Let us recall the relative positional embedding of head 27 learned in Figure 5.1, if the ground truth answer sits in the middle, it will have long contexts from both sides competing for the attention weight. If this hypothesis is correct, we can expect the performance boost to be more prominent when the answer appears near the middle. We reveal the performance breakdown when the number of input documents is 30. As we can see in the improvement row, those cases indeed achieve greater improvements.

Our strategies are not always perfect: The performance drops if the ground truth answer is at position 29. We believe T5 might have already handled this case pretty well due to the recency bias learned on some attention heads, and our additional temperature scaling sharpens the distribution too aggressively.

5.5.4 Code Key Retrieval and Completion

To test the generalizability of the alignment strategies, we apply our methods to the CodeT5+ model [Wang et al., 2023] that was pre-trained on code data with 770M parameters.[3] We set $L_{tr} = 768$. We do not experiment with larger CodeT5+ models since they do not follow the T5 architecture, but use other positional embeddings. We conduct two experiments on the LCC dataset [Guo et al., 2023], which is highly similar to the classic PY150 dataset [Raychev et al., 2016] except that the input context length is much longer.

For the code key retrieval experiment, we sample several code files from LCC along with a special function that only returns an integer from 1 to 100. We concatenate them and ask a model to generate the returned integer at the end [Roziere et al., 2023]. Considering that this is essentially a passkey retrieval task in the code domain, we briefly report the average accuracy of 100 test cases when the input sequence length is around 16k: 0 (Original CodeT5+), **87** (w/ $\mathrm{P}_{\max}$), 80 (w/ H), and 85 (w/ $\log_{L_{ex}} L_{tr}$). We can see that the maximum probability alignment strategy performs the best.

For the code completion experiment, a model needs to generate the next line of code given some prior code as the context. The metrics are Exact Match (EM) and Edit similarity (ES) on a per line basis [Svyatkovskiy et al., 2020]. We report the results in Table 5.6 using the context length bucketing format. While both alignment strategies improve the performance substantially, $\mathrm{P}_{\max}$ is

[3] `https://huggingface.co/Salesforce/codet5p-770m-py`

better; however, its EM performance lags behind $\log_{L_{ex}} L_{tr}$ when the sequence length increases. We additionally include an extrapolation-free baseline, *truncation*, that truncates the long input context to the most recent $L_{tr} = 768$ tokens. Both $\mathrm{P}_{\max}$ and $\log_{L_{ex}} L_{tr}$ perform better than this baseline when $L_{ex} < 6000$, indicating that they can indeed benefit from longer ($6000/768 = 7.8$x) contexts without any fine-tuning.

Code Completion Exact Match

Models	Sequence Length (L_{ex})					
	1k	2k	3k	4k	5k	6k
CodeT5+	19.6	19.0	11.3	2.6	0.1	0.0
w/ $\mathrm{P}_{\max}$	21.1	22.5	21.7	21.5	19.3	22.7
w/ H	19.5	18.7	13.7	9.0	7.9	9.0
w/ $\log_{L_{ex}} L_{tr}$	**21.6**	**23.0**	**22.1**	**22.0**	**20.6**	**24.3**
w/ *truncation*	20.0	19.2	19.3	19.2	17.1	21.4

Code Completion Edit Similarity

Models	Sequence Length (L_{ex})					
	1k	2k	3k	4k	5k	6k
CodeT5+	62.4	59.6	53.1	38.9	18.3	10.4
w/ $\mathrm{P}_{\max}$	65.9	65.7	65.3	65.6	**63.1**	64.9
w/ H	64.8	62.5	54.1	43.0	43.0	44.8
w/ $\log_{L_{ex}} L_{tr}$	**66.3**	**66.1**	**65.2**	**66.4**	63.0	66.1
w/ *truncation*	65.3	64.2	64.2	65.6	62.2	**66.9**

Table 5.6: Code completion performance. Full score is 100. We set $L_{tr} = 768$. The bucket *n*k contains the data with length in [*n*k, (*n*+1)k), $n \in [1, 6]$. For example, the bucket 3k contains data with length in [3000, 4000). See Table 5.12 and 5.13 in § 5.10.6 for the full performance breakdown of L_{ex} up to 16k tokens.

5.5.5 Overall Observations

First, the maximum probability alignment strategy is the most reliable and best-performing method across most tasks and settings, echoing our discussion in § 5.3.1: For most data, only a subset of the input is useful for a model process at a time. The maximum probability alignment strategy captures this characteristic naturally, thereby outperforming the entropy alignment strategy that cares more about the holistic distribution.

Second, deciding the optimal temperature solely based on sequence lengths, e.g. $\tau = \log_{L_{ex}} L_{tr}$, is not robust enough. For example, the perplexity of $\log_{L_{ex}} L_{tr}$ suddenly increases (worse) on T5-XXL-LM, in Table 5.3, while the other strategies maintain stable results. As another example, it fails to improve the retrieval performance on the Flan-T5-Large model, shown in Table 5.4.

5.6 Theoretical Analysis

5.6.1 Assumptions

To shed more light on the underlying mechanisms of the two alignment strategies, we establish the connection between the softmax temperature τ and data distribution under empirically verified assumptions. We focus on the 0-th layer (closest to the input embeddings) and take the average over all logit vectors across attention heads. Note that this is just a crude approximation of Algorithm 1 for analysis purposes since 1) a Transformer language model typically encompasses multiple layers, and 2) in Algorithm 1, we take the maximum probability or entropy of individual logit vectors as opposed to the average one.

Assumption 5.1. *The length L average logit vector is normally distributed, i.e., its entry $l_i \sim N(0, \sigma^2)$.*

To compute the *average logit vector*, we start with a input sequence of length L. Using a Transformer model with H attention heads (specifically, a T5 Encoder in our context), we generate $H \times L$ pre-softmax logit vectors, each with a length of L. Here, the number of layers is 1 because we focus on the 0-th layer. These logit vectors are then individually sorted, and we subsequently calculate the average of all $H \times L$ sorted logit vectors, resulting in the average logit vector of length L.

To assess whether the average logit entries follow a Gaussian distribution, we make use of QQ plots, as illustrated in Figure 5.2. The linearity of the plot serves as an indicator – the closer the points are to the identity line, the more Gaussian the distribution.

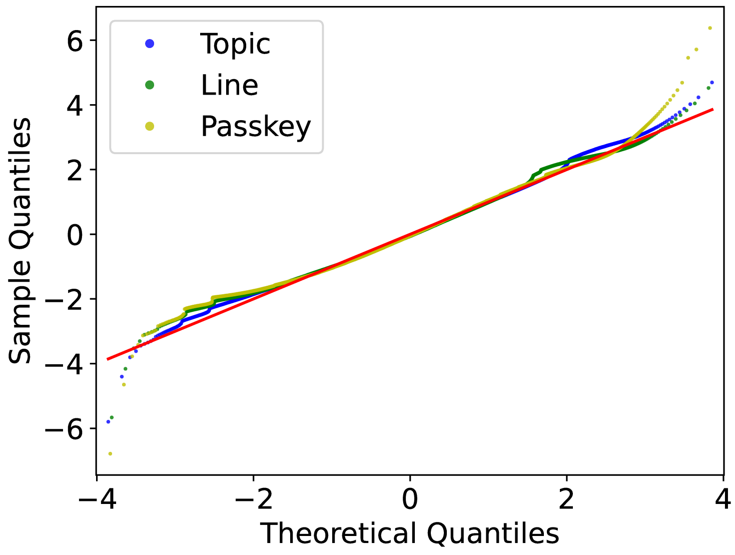


Figure 5.2: QQ plots of Flan-T5-XL. We experiment with short and long sequences. The red reference line is y=x. We use sequences of length around 512 for this plot. The plot for sequences of length around 15k looks highly similar. Please refer to § 5.10.1 for details.

Assumption 5.2. *The largest logit entry of the average logit vector during training and extrapolation is the same: $l^{ex}_{\max} = l^{tr}_{\max}$. See Table 5.7.*

Criteria	**Retrieval Tasks**					
	Topic		Line		Passkey	
	512	15k	512	15k	512	15k
$l_{\max}$	8.61	8.80	8.71	8.97	8.75	8.85

Table 5.7: Largest logit entry of Flan-T5-XL. $l_{\max}$ is the largest logit entry of the average logit vector.

5.6.2 Maximum Probability Alignment

Proposition 5.1. *Under Assumption 5.1 and 5.2, we can adjust the temperature* τ *to align the maximum probability* $\mathrm{P}_{\max}^{tr} = \mathrm{P}_{\max}^{ex}$

$$\begin{aligned}\tau &\approx \frac{\log L_{tr} + \log \mathrm{P}_{\max}^{tr} + \sigma_{tr}^2/2}{\log L_{ex} + \log \mathrm{P}_{\max}^{tr} + \sigma_{ex}^2/(2\tau^2)}. \\ &= \frac{B}{A + \frac{C}{\tau^2}} = \frac{B\tau^2}{A\tau^2 + C}.\end{aligned}$$

Assuming $\tau \neq 0$*, we solve the quadratic equation* $A\tau^2 - B\tau + C = 0$ *to get* τ*. We pick the larger root as our final solution. See proof in § 5.10.2.*

5.6.3 Entropy Alignment

Proposition 5.2. *Under Assumption 5.1, we can adjust the temperature* τ *to align the entropy* $H_{tr} = H_{ex}$

$$\tau \approx \frac{\sigma_{ex}}{\sqrt{\sigma_{tr}^2 + 2\log\frac{L_{ex}}{L_{tr}}}}$$

See proof in § 5.10.3.

5.7 Discussion

The objective of this section is to explain the observations made in § 5.5 through the lens of temperature analysis. We visualize Proposition 5.1 and 5.2 by plotting the temperature curves in Figures 5.3 and 5.4. We evaluated $\mathrm{P}_{\max}^{tr}$ and σ_{tr} at the training length and σ_{ex} at every extrapolation length considering only the 0-th layer. You may find the temperature curves for the other tasks in § 5.10.4.

First, while both proposed strategies lower the temperature when the input sequence length increases, the entropy alignment strategy does so more aggressively, possibly leading to its inferior performance observed in Tables 5.4 and 5.5 (w/ H). This can be seen by comparing the curves from Propositions 5.1 and 5.2 or the dots from Algorithm 1.

Second, deciding the optimal temperature based on sequence lengths, e.g. $\tau = \log_{L_{ex}} L_{tr}$, is not the most robust method. It gives too high of a temperature in Figure 5.3 compared to Algorithm 1.

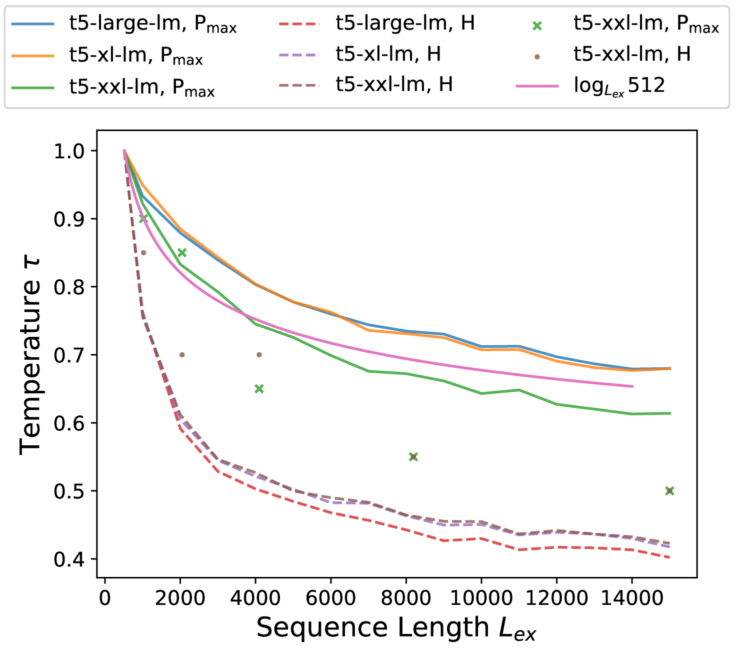


Figure 5.3: Language modeling temperature analysis. Curves are from Proposition 5.1 & 5.2. Dots and crosses are from Algorithm 1.

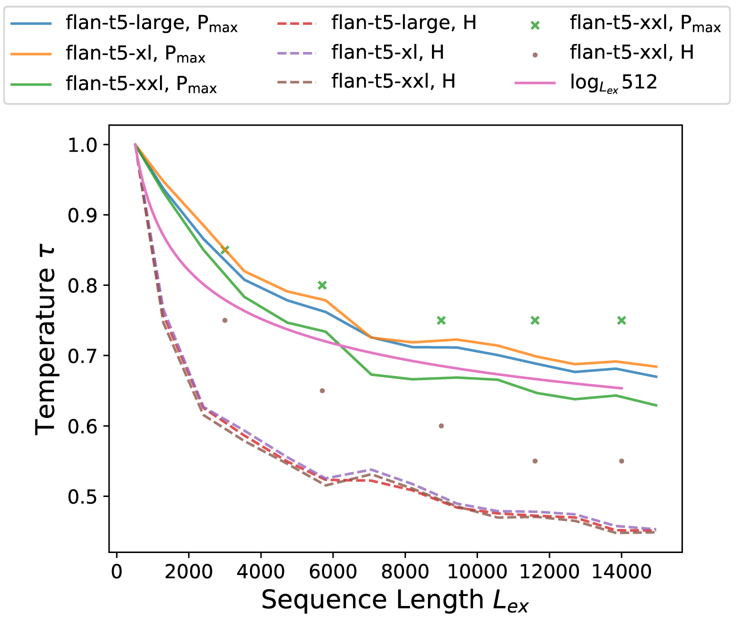


Figure 5.4: Topic retrieval temperature analysis. Curves are from Proposition 5.1 & 5.2. Dots and crosses are from Algorithm 1.

In other words, it does not sharpen the distribution enough, possibly explaining its perplexity spike in Table 5.3. On the other hand, it overly lowers the temperature in Figure 5.4, thus failing to improve the retrieval performance of Flan-T5-Large in Table 5.4.

5.8 Conclusion

In this chapter, we show that the T5 model family has great potential when it comes to Transformer length extrapolation. We propose the maximum probability and entropy alignment strategies to fix T5's dispersed attention issue without model fine-tuning. We conduct experiments on natural language modeling, retrieval, multi-document question answering, and code completion tasks to demonstrate the effectiveness of our proposed methods. Finally, we present a simplified theoretical analysis to elucidate how the temperature is scaled to achieve attention alignment. We hope that our work can inspire future length-extrapolatable Transformer designs.

5.9 Limitations

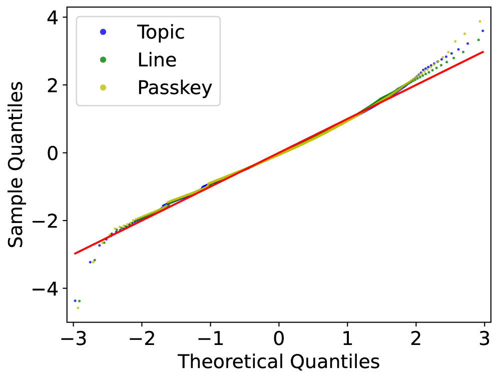
We base our theoretical analysis on a simplified Transformer language model, which might be further improved by taking all the layers and their interactions into account. In addition, we find that different layers have different degrees of distribution flatness, which could be leveraged in future work to perform per-layer fine-grained attention alignment. Finally, our temperature scaling scheme sometimes sharpens a distribution too aggressively in the multi-document question-answering and code-completion experiments. This drawback could possibly be improved by designing a more fine-grained attention alignment strategy.

5.10 Proofs and Experimental Details

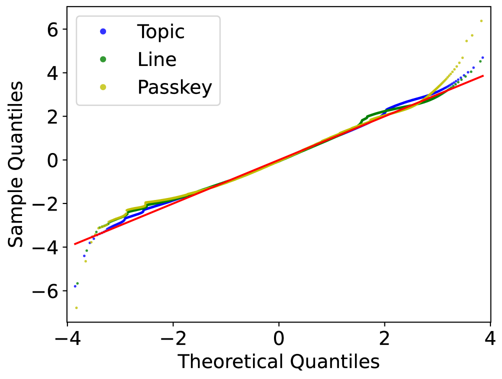
5.10.1 QQ Plots for Assumption 5.1

A QQ plot [Wilk and Gnanadesikan, 1968] is a graphical technique used for comparing two probability distributions by plotting their quantiles against each other. A point (x, y) corresponds to a quantile from the second distribution (y-coordinate) plotted against the same quantile from the first distribution (x-coordinate). When the two distributions under comparison are similar, the points in the QQ plot will roughly align with the identity line, y = x. In our case, where we aim to determine the degree of Gaussian behavior in the average logit vector, the linearity of the plot serves as an indicator – the closer the points are to the identity line, the more Gaussian the distribution.

We present the QQ plots for two lengths, 512 and 15k, on the three retrieval tasks in Figure 5.5. They are all close to the red reference line, indicating that their form is highly Gaussian.



(a) Short sequences around 512



(b) Long sequences around 15k

Figure 5.5: QQ plots of Flan-T5-XL. We experiment with short and long sequences. The red reference line is y=x. The more closely the scatter plots follow the red reference line, the more Gaussian they are.

5.10.2 Detailed Derivation of Proposition 5.1

Let $l_{\max}$ be the largest value in the logit vector l. Let τ be the temperature of the Softmax function. The probability of the largest entry is

$$\mathrm{P}_{\max} = \frac{e^{l_{\max}/\tau}}{\sum_{i=1}^{L} e^{l_i/\tau}}.$$

Since Softmax is shift-invariant, the logit vector can always be made zero-mean: $\sum_i l_i = 0$. Next, according to Assumption 5.1, the denominator of Softmax can be approximated as

$$\sum_{i=1}^{L} e^{l_i/\tau} \approx L \cdot \mathbb{E}[e^{l_i/\tau}] = L \cdot e^{\sigma^2/(2\tau^2)} \qquad (5.1)$$

This implies $\mathrm{P}_{\max}$ is approximately

$$\mathrm{P}_{\max} \approx \frac{e^{l_{\max}/\tau}}{L e^{\sigma^2/(2\tau^2)}}$$

During the training stage, the temperature τ is 1

$$\mathrm{P}^{tr}_{\max} \approx \frac{e^{l^{tr}_{\max}}}{L_{tr} e^{\sigma^2_{tr}/2}},$$

which gives an expression of the largest logit entry during the training stage

$$l^{tr}_{\max} \approx \log\left(\mathrm{P}^{tr}_{\max} L_{tr} e^{\sigma^2_{tr}/2}\right) \qquad (5.2)$$

According to Assumption 5.2, the largest probability during the extrapolation stage can be simplified as

$$
\begin{aligned}
\mathrm{P}_{\max}^{ex} &\approx \frac{e^{l_{\max}^{ex}/\tau}}{L_{ex}e^{\sigma_{ex}^2/(2\tau^2)}} \overset{\text{A. 5.2}}{=} \frac{e^{l_{\max}^{tr}/\tau}}{L_{ex}e^{\sigma_{ex}^2/(2\tau^2)}} \\
&\overset{(5.2)}{\approx} \frac{\left(\mathrm{P}_{\max}^{tr}L_{tr}e^{\sigma_{tr}^2/2}\right)^{1/\tau}}{L_{ex}e^{\sigma_{ex}^2/(2\tau^2)}}
\end{aligned}
$$

Since τ is a free parameter during extrapolation, we adjust it to carry out the maximum probability alignment strategy. Rearranging the terms gives Proposition 5.1.

5.10.3 Detailed Derivation of Proposition 5.2

The entropy of a discrete probability computed by Softmax is

$$
H = -\sum_i \frac{e^{l_i/\tau}}{D}\log\frac{e^{l_i/\tau}}{D} = \log D - \frac{\sum_i \frac{l_i}{\tau}e^{l_i/\tau}}{D},
$$

where $D = \sum_i e^{l_i/\tau}$ is the denominator of Softmax, which can be approximated using Eq. (5.1). On the other hand, we note that $\sum_i l_i e^{l_i} \approx L\mathbb{E}[le^l]$. When $l \sim N(0,\sigma^2)$, $\mathbb{E}[le^l]$ is approximated as

$$
\begin{aligned}
\mathbb{E}[le^l] &= \int_{-\infty}^{\infty}\frac{le^l}{\sigma\sqrt{2\pi}}e^{-\frac{l^2}{2\sigma^2}}dl \\
&= \int_{-\infty}^{\infty}\frac{l}{\sigma\sqrt{2\pi}}e^{\frac{2\sigma^2 l-l^2}{2\sigma^2}}dl \\
&= \int_{-\infty}^{\infty}\frac{l}{\sigma\sqrt{2\pi}}e^{-\frac{(l-\sigma^2)^2-\sigma^4}{2\sigma^2}}dl \\
&= e^{\sigma^2/2}\int_{-\infty}^{\infty}\frac{l}{\sigma\sqrt{2\pi}}e^{-\frac{(l-\sigma^2)^2}{2\sigma^2}}dl \\
&= e^{\sigma^2/2}\sigma^2
\end{aligned}
\tag{5.3}
$$

Thus, combining Eq. (5.1) and (5.3), the entropy H is approximated as

$$
\begin{aligned}
H &\approx \log L + \frac{\sigma^2}{2\tau^2} - \frac{Le^{\sigma^2/(2\tau^2)}\frac{\sigma^2}{\tau^2}}{Le^{\sigma^2/(2\tau^2)}} \\
&= \log L - \frac{\sigma^2}{2\tau^2}
\end{aligned}
$$

Since τ is set to 1 during the training stage, we have $H_{tr} \approx \log L_{tr} - \frac{\sigma_{tr}^2}{2}$. During extrapolation, we align the entropy (i.e., $H_{ex} = H_{tr}$) by adjusting τ.

$$
\log L_{ex} - \frac{\sigma_{ex}^2}{2\tau^2} \approx H_{ex} = H_{tr} \approx \log L_{tr} - \frac{\sigma_{tr}^2}{2}.
$$

Since τ is a free parameter during extrapolation, we adjust it to apply the entropy alignment strategy. Rearranging the terms gives Proposition 5.2.

5.10.4 More Real-world Temperature Plots

We verify Proposition 5.1 and 5.2 on the remaining tasks by plotting the temperature curves in Figure 5.6, 5.7, 5.8, and 5.9. We empirically evaluate σ_{tr} at the training length and σ_{ex} every extrapolation length considering only the 0-th layer.

The real temperatures given by Algorithm 1 are usually higher than those derived from the two propositions. After checking the per-layer attention distributions, we find that the 0-th layer has flatter distributions compared to higher layers. Because the two propositions are derived based on the 0-th layer and a flatter distribution needs a lower temperature to correct, the temperatures given by them tend to be lower than the ones given by Algorithm 1 that takes the average of temperatures across all layers.

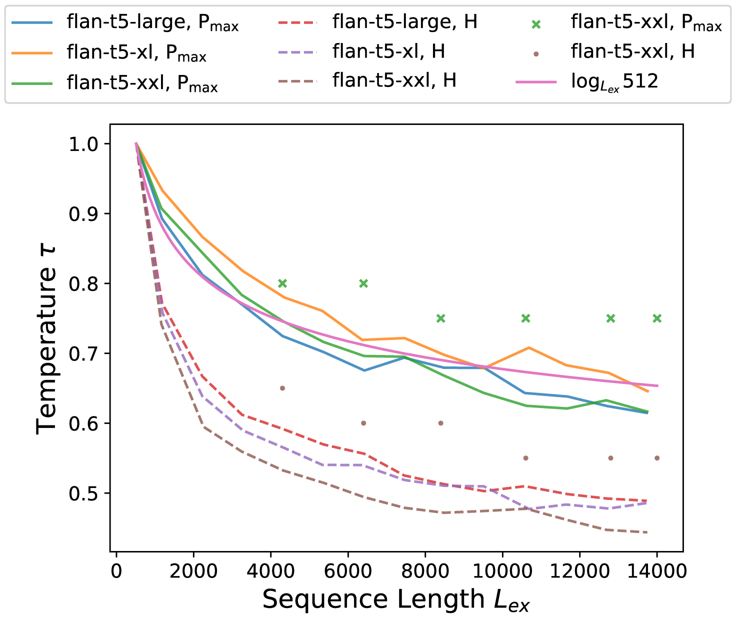


Figure 5.6: Line retrieval temperature analysis. Curves are given by Proposition 5.1 and 5.2. Cross signs and dots are given by Algorithm 1. $\log_L 512$ is given by Yao et al. [2021]; Su [2021].

5.10.5 Detailed Temperature Breakdown

We report the temperatures for all tasks across model sizes given by Algorithm 1 in Table 5.8, 5.9, 5.10, and 5.11.

5.10.6 Performance Breakdown of Code Completion

We report the performance breakdown of Exact Match and Edit Similarity across lengths in Table 5.12 and 5.13.

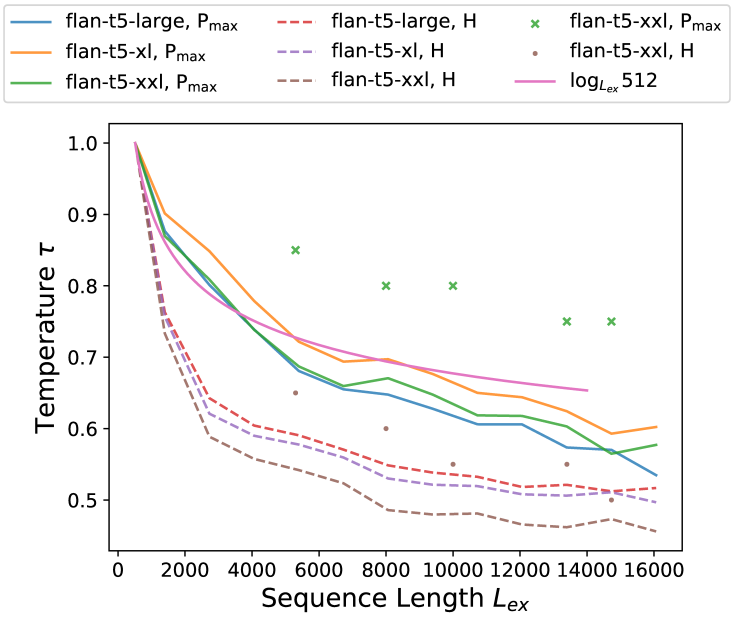


Figure 5.7: Passkey retrieval temperature analysis. Curves are given by Proposition 5.1 and 5.2. Cross signs and dots are given by Algorithm 1. $\log_L 512$ is given by Yao et al. [2021]; Su [2021].

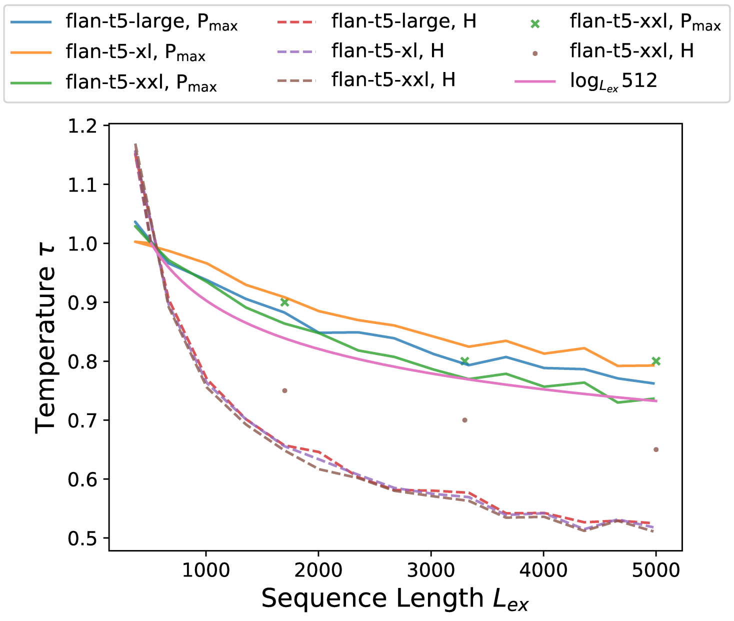


Figure 5.8: Multi-doc QA temperature analysis. Curves are from Proposition 5.1 & 5.2. Dots and crosses are from Algorithm 1.

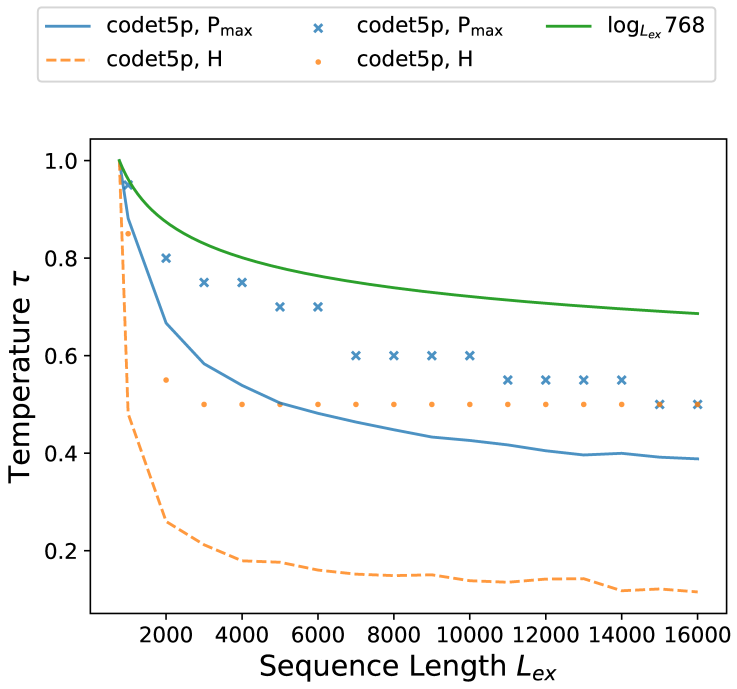


Figure 5.9: Code completion temperature analysis. Curves are given by Proposition 5.1 and 5.2. Cross signs and dots are given by Algorithm 1. $\log_L 768$ is given by Yao et al. [2021]; Su [2021].

Language Modeling					
Models	Sequence Length (L_{ex})				
	1024	2048	4096	8192	15000
T5-Large-LM					
w/ $\mathrm{P}_{\mathrm{max}}$	0.9	0.85	0.8	0.75	0.7
w/ H	0.8	0.7	0.6	0.5	0.5
T5-XL-LM					
w/ $\mathrm{P}_{\mathrm{max}}$	0.9	0.85	0.75	0.7	0.6
w/ H	0.85	0.7	0.55	0.5	0.5
T5-XXL-LM					
w/ $\mathrm{P}_{\mathrm{max}}$	0.9	0.85	0.65	0.55	0.5
w/ H	0.85	0.7	0.7	0.55	0.5
w/ $\log_{L_{ex}} L_{tr}$	0.9	0.82	0.75	0.69	0.65

Table 5.8: Temperatures of language modeling. We search the optimal temperature from 1.0, 0.95, 0.9, $\cdots$, 0.5. We set $L_{tr} = 512$.

5.10.7 Performance Breakdown of Multi-document Question Answering

We report the performance breakdown of different numbers of input documents in Table 5.14.

Models	Topic, # of topics					Line, # of lines						Passkey, # of sentences				
	5	10	15	20	25	200	300	400	500	600	680	20k	30k	40k	50k	55k
Flan-T5-Large																
w/ $\mathrm{P_{max}}$	0.85	0.8	0.75	0.75	0.75	0.85	0.8	0.8	0.75	0.75	0.75	0.85	0.80	0.80	0.75	0.75
w/ H	0.7	0.6	0.55	0.5	0.5	0.65	0.55	0.55	0.5	0.5	0.5	0.6	0.55	0.5	0.5	0.5
Flan-T5-XL																
w/ $\mathrm{P_{max}}$	0.8	0.75	0.7	0.65	0.65	0.8	0.75	0.75	0.7	0.70	0.7	0.85	0.8	0.75	0.75	0.75
w/ H	0.7	0.55	0.55	0.5	0.5	0.6	0.55	0.55	0.5	0.5	0.5	0.7	0.65	0.6	0.6	0.6
Flan-T5-XXL																
w/ $\mathrm{P_{max}}$	0.85	0.8	0.75	0.75	0.75	0.8	0.8	0.75	0.75	0.75	0.75	0.85	0.8	0.8	0.75	0.75
w/ H	0.75	0.65	0.6	0.55	0.55	0.65	0.6	0.6	0.55	0.55	0.55	0.65	0.6	0.55	0.55	0.5
w/ $\log_{L_{ex}} L_{tr}$	0.79	0.72	0.69	0.67	0.65	0.74	0.71	0.69	0.67	0.66	0.65	0.73	0.69	0.67	0.66	0.65

Retrieval Tasks

Table 5.9: Temperatures of retrieval tasks. We search the optimal temperature from 1.0, 0.95, 0.9, $\cdots$, 0.5. The maximum lengths of the three tasks are all around 14.5k to 15.5k tokens (L_{ex}). We set $L_{tr} = 512$.

Multi-document Question Answering

Models	10 Docs	20 Docs	30 Docs
	$L_{ex} = 1700$	$L_{ex} = 3300$	$L_{ex} = 5000$
Flan-T5-Large			
w/ Max.	0.9	0.85	0.8
w/ Ent.	0.75	0.65	0.6
Flan-T5-XL			
w/ Max.	0.85	0.75	0.75
w/ Ent.	0.75	0.65	0.55
Flan-T5-XXL			
w/ Max.	0.9	0.8	0.8
w/ Ent.	0.75	0.7	0.65
w/ $\log_{L_{ex}} L_{tr}$	0.84	0.77	0.73

Table 5.10: Temperatures of multi-document question answering. We search the optimal temperature from 1.0, 0.95, 0.9, $\cdots$, 0.5. Different golden document positions have the same temperature. We set $L_{tr} = 512$.

5.11 Scientific Artifacts

The pretrained models we used belong to the T5 model family, which is released under the Apache 2.0 license. The models are used in this chapter for research purposes only. For the data used to train T5 models, please refer to Raffel et al. [2020b]; Lester et al. [2021]; Chung et al. [2022] for details. Except for the LCC Python data, other task data is written in English. We already report the

Code Completion																
Models	Sequence Length (L_{ex})															
	1k	2k	3k	4k	5k	6k	7k	8k	9k	10k	11k	12k	13k	14k	15k	16k
CodeT5+																
w/ $\mathrm{P}_{\max}$	0.95	0.8	0.75	0.75	0.7	0.7	0.6	0.6	0.6	0.6	0.55	0.55	0.55	0.55	0.5	0.5
w/ H	0.85	0.55	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
w/ $\log_{L_{ex}} L_{tr}$	0.96	0.87	0.83	0.8	0.78	0.76	0.75	0.74	0.73	0.72	0.71	0.71	0.7	0.7	0.69	0.69

Table 5.11: Temperatures of code completion. We search the optimal temperature from 1.0, 0.95, 0.9, $\cdots$, 0.5. The maximum length is around 16k tokens (L_{ex}). We set $L_{tr} = 768$.

Code Completion Exact Match															
Models	Sequence Length (L_{ex})														
	1k	2k	3k	4k	5k	6k	7k	8k	9k	10k	11k	12k	13k	14k	15k
CodeT5+	19.6	19.0	11.3	2.6	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
w/ $\mathrm{P}_{\max}$	21.1	22.5	21.7	21.5	19.3	22.7	16.1	14.4	13.4	20.6	16.0	15.3	12.3	16.7	4.5
w/ H	19.5	18.7	13.7	9.0	7.9	9.0	10.3	8.8	10.8	12.1	11.7	10.2	9.2	11.1	2.3
w/ $\log_{L_{ex}} L_{tr}$	21.6	23.0	22.1	22.0	20.6	24.3	20.7	18.6	19.1	22.4	13.8	20.3	15.4	19.4	11.4
w/ *truncation*	20.0	19.2	19.3	19.2	17.1	21.4	21.1	18.0	19.1	25.2	18.1	20.3	16.9	27.8	15.9

Table 5.12: Full exact match breakdown of code completion edit similarity. We set $L_{tr} = 768$. Numbers in red are higher than their counterpart in the w/*truncation* row. The bucket *n*k contains the data with length in [*n*k, (*n*+1)k), $n \in [1, 15]$.

Code Completion															
Models	Sequence Length (L_{ex})														
	1k	2k	3k	4k	5k	6k	7k	8k	9k	10k	11k	12k	13k	14k	15k
CodeT5+	62.4	59.6	53.1	38.9	18.3	10.4	6.1	4.0	4.5	5.0	6.7	5.1	6.4	4.4	3.5
w/ $\mathrm{P}_{\max}$	65.9	65.7	65.3	65.6	63.1	64.9	60.0	60.0	58.1	57.5	56.2	56.0	52.1	56.9	39.9
w/ H	64.8	62.5	54.1	43.0	43.0	44.8	47.7	47.0	47.6	51.2	44.3	49.7	50.3	57.4	42.0
w/ $\log_{L_{ex}} L_{tr}$	66.3	66.1	65.2	66.4	63.0	66.1	61.9	58.8	61.6	57.8	54.2	57.9	48.7	52.2	48.6
w/ *truncation*	65.3	64.2	64.2	65.6	62.2	66.9	66.8	61.8	64.1	65.1	63.5	63.9	61.5	67.6	60.8

Table 5.13: Full edit similarity breakdown of code completion. We set $L_{tr} = 768$. Numbers in red are higher than their counterpart in the w/*truncation* row. The bucket *n*k contains the data with length in [*n*k, (*n*+1)k), $n \in [1, 15]$.

number of data instances in § 5.5 for the language modeling and retrieval tasks. As for the multi-doc QA and code related tasks, we follow the original data splits.

Models	10 Docs			20 Docs					30 Docs						
	0	4	9	0	4	9	14	19	0	4	9	14	19	24	29
Multi-document Question Answering															
Flan-T5-Large	60.6	48.5	48.0	54.5	44.0	39.6	38.0	40.2	52.6	42.0	36.5	34.0	33.9	33.9	37.9
w/ Max.	60.9	49.8	48.6	53.5	45.6	40.8	39.7	41.3	50.8	44.5	39.5	36.4	35.9	35.8	37.0
w/ Ent.	58.9	50.1	47.3	52.4	45.2	40.4	38.0	40.0	47.6	41.1	35.2	33.5	32.2	33.3	34.2
w/ $\log_{L_{ex}} L_{tr}$	60.2	51.1	48.4	53.8	46.0	41.4	39.4	41.7	50.6	44.1	39.3	36.3	35.8	35.8	37.2
Flan-T5-XL	64.0	55.4	58.9	60.6	47.9	45.1	47.3	55.3	58.4	44.6	40.0	39.9	41.7	46.4	54.8
w/ Max.	65.3	57.3	60.8	62.2	51.6	49.0	49.4	56.0	60.9	49.1	46.0	44.9	46.3	49.1	55.7
w/ Ent.	64.7	56.7	60.0	59.3	50.1	47.9	49.8	55.1	52.4	43.5	42.1	40.3	42.0	42.9	51.3
w/ $\log_{L_{ex}} L_{tr}$	65.1	57.0	60.6	62.2	51.7	48.8	49.5	56.0	61.0	49.1	46.1	44.7	46.1	48.7	55.4
Flan-T5-XXL	65.1	61.0	64.6	61.1	53.9	52.4	54.7	62.4	58.9	49.1	48.1	47.5	48.9	53.1	61.2
w/ Max.	66.2	61.8	63.2	62.8	55.9	54.4	55.6	59.6	60.4	52.5	51.0	50.2	51.3	53.5	59.1
w/ Ent.	67.3	62.1	61.3	63.2	56.1	54.1	54.3	57.6	61.0	53.4	50.8	50.3	50.7	51.9	55.7
w/ $\log_{L_{ex}} L_{tr}$	66.7	61.9	63.1	63.1	56.0	54.7	55.1	59.0	61.5	53.3	51.3	50.3	51.1	53.0	57.2

Table 5.14: Full performance breakdown of multi-document question answering. The numbers are accuracy. Full score is 100. 0, 4, 9... indicate the position of the golden document that contains the answer to a question.

Chapter 6

Transformer Working Memory Enables Regular Language Reasoning and Natural Language Length Extrapolation

6.1 Introduction

In preceding chapters, we delved into various natural language tasks and explored different designs of positional embeddings. In this chapter, we extend our scope beyond natural language to encompass the family of regular languages. This expansion enables us to investigate the compositional capabilities of Transformer language models Valvoda et al. [2022], a pivotal benchmark for evaluating the models' capacity to combine primitives in modeling complex scenarios. Our proposed Transformer model architecture draws inspiration from the concept of working memory, demonstrating proficiency in accurately modeling regular language structures.

It has long been believed that *Working Memory* (WM), a term coined in the 1960s to liken human minds to computers, plays an important role in human reasoning ability and the guidance of decision-making behavior [Baddeley and Hitch, 1974; Baddeley, 1992; Ericsson and Kintsch, 1995; Cowan, 1998; Miyake et al., 1999; Oberauer, 2002; Diamond, 2013; Adams et al., 2018]. Although no single definition encompasses all applications of WM [Adams et al., 2018], the following one should be shared by all theories of interest:

> *Working memory is a system of components that holds a* ***limited amount*** *of information temporarily in a heightened state of availability for use in ongoing processing.* - Adams et al. [2018]

WM is instantiated in the two major driving forces of sequence modeling: Recurrent neural networks'(RNN) [Elman, 1990; Jordan, 1997; Hochreiter and Schmidhuber, 1997] short term memory modulated by their recurrent nature and gate design [Rae and Razavi, 2020b; Nematzadeh et al., 2020; Armeni et al., 2022], and Transformers' [Vaswani et al., 2017a] salient tokens heightened by self-attention.

In reality, self-attention often attends broadly [Clark et al., 2019], violating the limited amount of information notion of WM. Our hypothesis is that such violation is to blame for Transformers'

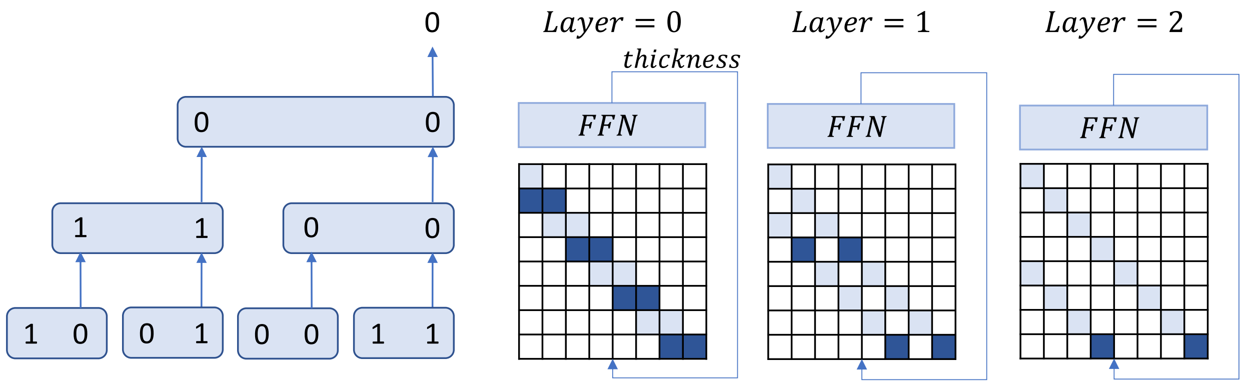


Figure 6.1: This is the divide-and-conquer approach that solves the PARITY problem. Lightly shaded blue cells represent $M_{mn}^{(l)}$ in eq. (6.1). The darkened blue cells represent the routing path to specifically solve the result for the last bit. As we can see, this approach requires at most $\log_2 T$ layers to obtain the result for a length T input sequence, making it a more efficient approach compared to the combination of scratchpad and recency biases.

failure on algorithmic reasoning of regular languages [Deletang et al., 2023; Liu et al., 2023a] such as PARITY, a seemingly simple task that checks if the number of 1s in a bit string is even. Surprisingly, a Transformer can only correctly count the number of 1s when the sequence length is kept fixed at the training sequence length T_{tr}, and it fails miserably when the test sequence length is extrapolated to $T_{ex} > T_{tr}$ [Hahn, 2020; Bhattamishra et al., 2020a; Chiang and Cholak, 2022; Deletang et al., 2023; Liu et al., 2023a]. In contrast, an RNN can be perfectly extrapolated.

The goal of this chapter is therefore to improve Transformers' WM by limiting the amount of accessible information at a time. Existing attempts that use a combination of scratchpad and recency biases [Wei et al., 2022; Nye et al., 2022; Anil et al., 2022; Liu et al., 2023a] are not optimal as they completely forego the parallelization property of a Transformer, making it as computationally inefficient as an RNN.

This begs the question: *Does there exist a more efficient Transformer working memory design?* The answer is affirmative thanks to the proposed **RegularGPT**, which boils down to the three design choices: Weight-Sharing, Adaptive-Depth, and Sliding-Dilated-Attention; Each of them has been proposed previously, but it is the unique combination that sparks the successful and efficient learning of regular languages. We will further demonstrate its: 1) similar recursive parallel structure as linear RNN [Orvieto et al., 2023b], resulting in $\log T_{tr}$ or $\log T_{ex}$ layers, and 2) generalizability by showing strong performance on the task of Transformer natural language length extrapolation [Press et al., 2022a; Chi et al., 2022, 2023b].

6.1.1 Regular Language and Algorithmic Reasoning

The Chomsky hierarchy [Chomsky, 1956b] classifies formal languages into different hierarchies based on their increasing complexity. Each hierarchy represents a family of formal languages that can be solved by the corresponding automaton. At the lowest level, there lies the family of regular languages, which can be expressed using a finite state automaton (FSA), a computational model comprising a set of states and transitions connecting them.

PARITY, or the family of regular languages in general, is a type of language that strictly follows certain rules, like grammar. The successful modeling of a regular language is important, since it implies a model's ability to learn the underlying rules of the data. In terms of generalization, there is also the task of modular arithmetic within the same language family. Concretely, if the training data consists of arithmetic operations such as $1 + 2 \times 3$, a model should learn the rules of $a + b$, $a \times b$, and that $\times$ has a higher priority than $+$. Learning unambiguous rules behind the data is a critical step toward sequence modeling with regulated output.

Our primary objective is to enhance the algorithmic reasoning of the Transformer model in regular languages by testing its language transduction capability under the extrapolation setting. Concretely, the model is trained only to predict desired outputs on a set of short length-T sequences with $T \leq T_{tr}$. It must also predict the correct output for longer testing sequences of length $T_{ex} \gg T_{tr}$. It is worth noting that we evaluate our model through language transduction following recent work [Deletang et al., 2023; Liu et al., 2023a], instead of the conventional language recognition protocol. Both settings are equally hard as they are underpinned by the same finite state semiautomaton. Interested readers may refer to Deletang et al. [2023] for further details regarding the two evaluation protocols. We also reveal the connection between RegularGPT and finite state semiautomaton later in § 6.6.

6.1.2 Failure Mode and An Inefficient Fix

The PARITY task involves a length T bit string $\sigma_1\sigma_2\cdots\sigma_T$ where each bit σ_i is randomly sampled from a Bernoulli distribution with $\mathbb{P}(\sigma_i = 1) = 0.5$. The goal is to determine whether the sequence contains an even or odd number of 1s.

It has been observed that a Transformer is incapable of performing length extrapolation on PARITY, but what could be its potential failure mode? Previous work sheds light on this by showing that a Transformer might settle on the naive-summation approach [Anil et al., 2022; Deletang et al., 2023; Liu et al., 2023a]. Concretely, it sums up all the bits and outputs the summation modulo 2. This approach fails since unseen summations will be produced when the model takes sequences of length $T_{ex} > T$ as input or $\mathbb{P}(S_i)$ deviates from 0.5.

To the best of our knowledge, the existing remedy [Liu et al., 2023a; Anil et al., 2022] is to use scratchpad [Wei et al., 2022; Nye et al., 2022] along with recency biases [Press et al., 2022a] to enforce the correct learning: They create a scratchpad that interleaves the sequence of input bits and intermediate answers $(\sigma_1, q_1, \sigma_2, q_2, \cdots, \sigma_T, q_T)$, where $q_i = \text{solve}(\sigma_1 \cdots \sigma_i)$. The model is trained to predict all the $\sigma_{i\in[T]}$. Recency biases play the role of limiting a Transformer's receptive field to only a few most recent σ and q at every timestep i. This is to prevent self-attention from ignoring q and giving the same naive-summation solution.

Scratchpad and recency biases jointly create the notion of WM along the temporal dimension similar to RNNs, thereby enabling successful extrapolation on regular languages. Nevertheless, we note that this fix is inefficient during inference since all the intermediate answers q_i have to be generated sequentially before reaching the final answer q_T. A desirable fix should only take in the input bits $(\sigma_1, \sigma_2, \cdots, \sigma_n)$ and directly generate the final answer q_T. In other words, our goal is to find an efficient WM design for a Transformer.

6.1.3 A Desirable Fix for PARITY (Figure 6.1)

An alternative solution to the PARITY problem is based on the spirit of divide-and-conquer, where we first divide the sequence into T/C chunks with each chunk of length $C < T$, and we compose the final answer by recursively merging the chunk outputs. This approach does not suffer from the unseen summation issue as the model was trained to handle a fixed amount of C bits at a time in its WM (chunk). It then recursively applies the already-seen results to compose the final solution when it encounters longer sequences during inference. More importantly, it is more efficient than the scratchpad and recency biases approach since it only requires $\log_C T$ layers of parallel computations instead of $2T$ steps of sequential decoding.

6.2 Proposed Architecture of RegularGPT

In this chapter, we use $[N]$ to denote the list of non-negative integers $[0, \ldots, N-1]$. The Transformer model used in this chapter is always causal. It takes an input sequence of $T \leq T_{tr}$ units (can be tokens or bits) $\sigma_{i\in[T]}$, passes them through a fixed number of L Transformer layers, and finally computes the distribution over the vocabulary V through the prediction head W_o. We present our modifications to the vanilla Transformer below. Only the related operations will be expanded, and we follow all the other details of GPT2 [Radford et al., 2019b].

6.2.1 Sliding-Dilated-Attention

A Transformer layer at layer l consists of a self-attention operation denoted as $\text{SA}^{(l)}$ and feed-forward network denoted as $\text{FFN}^{(l)}$. Originally, $\text{SA}^{(l)}$ computes the inter-token relationships across all T units. Instead, we set the chunk size to C and produce T/C non-overlapping chunks;[1] Only the units within the same chunk inter-attend with each other. In practice, this can be achieved by an attention mask $M^{(l)} \in \mathbb{R}^{T\times T}$ at layer l. $M^{(l)}$ shares the same shape as the self-attention matrix (see Figure 6.1) and is defined as:

$$M^{(l)}_{mn} = \begin{cases} r_{(m-n)/C^\ell}, & \text{if } \frac{m-n}{C^l} \in [C] \\ -\inf, & \text{otherwise} \end{cases} \tag{6.1}$$

Note that M is a lower triangular matrix due to the causal nature of our model. r_i's with $i \in [C]$ are learnable relative positional scalars. To be precise, each attention head has a different set of learnable biases r_i's. Here, we drop the dependency on the head for notational simplicity.

The use of r_i's is similar to the positional scalars of T5 [Rae and Razavi, 2020b] except that we do not use the log-binning strategy over $m - n$. It is to facilitate the extraction of global information instead of enforcing the windowed-attention effect [Raffel et al., 2020b; Press et al., 2022a; Chi et al., 2022, 2023b]. M will then be added to the original self-attention matrix, creating the proposed Sliding-Dilated-Attention effect. The output of $\text{SA}^{(l)}$ will be transformed by the positional-independent $\text{FFN}^{(l)}$ to produce $o^{(l)}_{i\in[T]}$.

[1]Whenever T is not divisible by C, we pad the input sequence such that its length is a multiple of C.

The case of $C = 2$ is used as a possible construction of Theorem 1 in Liu et al. [2023a]. However, their focus is not on length extrapolation, hence lacking the below two proposed modifications.

6.2.2 Adaptive-Depth and Weight-Sharing

Since our Sliding-Dilated-Attention limits the number of accessible tokens at a time, we need an adaptive depth $\bar{L} = \log_C T$ so that the final output can utilize every single piece of input information. However, when $T_{ex} > T_{tr}$, the depth during inference will be higher than that during training. The simplest way to solve this challenge without further parameter updating is to perform Weight-Sharing across layers. To account for the possible performance loss due to Weight-Sharing, we first thicken the model by K times, resulting in a total number of $K \cdot \bar{L}$ layers. Next, we share the weights across the $K \cdot \bar{L}$ layers in the following way for $k \in [K]$:

$$\begin{aligned}\text{SA}^{(l \cdot K+k)} &= \text{SA}^{(k)} \quad \text{for } l \in [\bar{L}] \\ \text{FFN}^{(l \cdot K+k)} &= \text{FFN}^{(k)} \quad \text{for } l \in [\bar{L}]\end{aligned}$$

It can be equivalently interpreted as stacking more SA and FFN components within every Transformer layer, and the same thickened layer is reused $\bar{L}$ times. This layer thickening design is only used in the natural language modeling experiments in § 6.5.

6.2.3 Where is the WM Notion?

Instead of instantiating WM along the temporal dimension as the combination of scratchpad and recency biases, RegularGPT limits the amount of information along the depth dimension. As we have seen, the idea of breaking T units into several chunks limits the amount of accessible information at each layer, thereby enabling the WM notion. A similar argument was made by Yogatama et al. [2021] in a sense that they categorized Longformer [Beltagy et al., 2020], a Transformer variant with local attention pattern, as a model of working memory. Finally, thanks to modern accelerators such as GPU, all chunks at a layer can be processed concurrently, and this further makes RegularGPT more favorable over the scratchpad and recency biases approach.

6.2.4 Complexity Analysis

The sparse attention pattern of RegularGPT suggests it might enjoy the same speedup provided by sparsified Transformers. The complexity of our model is $O(TCK \log_C T)$ where TC is the complexity of each self-attention module and $K \log_C T$ is the total number of layers. On the other hand, the vanilla Transformer follows $O(T^2 L)$. To illustrate the possible speedup, if $T = 512$ and $C = 128$, then $512 \cdot 128 \cdot K \log_{128} 512 < 512^2 L$ when $K < \frac{512L}{128 \log_{128} 512} \approx 3.11L$. Namely, as long as $K < 3L$, our model is likely to be more efficient than a vanilla Transformer.

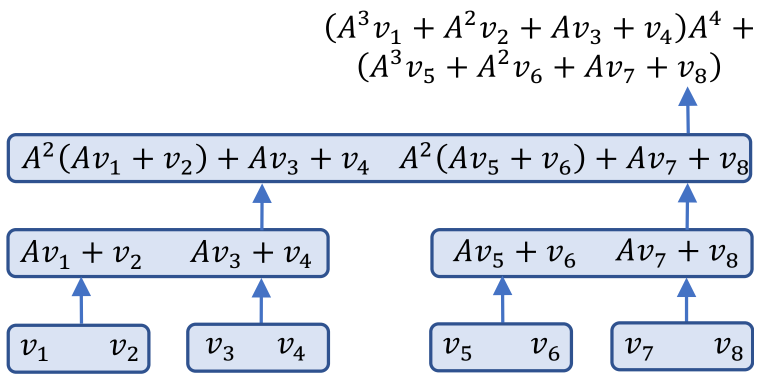

$(A^3v_1 + A^2v_2 + Av_3 + v_4)A^4 +$
$(A^3v_5 + A^2v_6 + Av_7 + v_8)$
$A^2(Av_1 + v_2) + Av_3 + v_4$
$A^2(Av_5 + v_6) + Av_7 + v_8$
$Av_1 + v_2$
$Av_3 + v_4$
$Av_5 + v_6$
$Av_7 + v_8$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8

Figure 6.2: This is the parallel scan algorithm that can accelerate a linear RNN. In this example, we visualize the routing path for computing x_8. Blocks at the same layer can be computed in parallel on GPUs.

6.3 Connection to Prior Work

Sliding-Dilated-Attention This special attention pattern dates back to pre-Transformer era such as Wavenet [van den Oord et al., 2016] with dilated convolution. It can also be viewed as a special form of Longformer attention pattern with systematic dilation [Beltagy et al., 2020].[2] Limiting the range of attention in lower layers of a Transformer is also corroborated in Rae and Razavi [2020a], where they find such design does not deteriorate the performance.

Adaptive-Depth and Weight-Sharing ALBERT [Lan et al., 2020] and Universal Transformer [Dehghani et al., 2019] share the parameters across layers. The weight sharing design makes them compatible with the idea of Adaptive Computation Time [Graves et al., 2014] and Dynamic Halting [Dehghani et al., 2019; Elbayad et al., 2020], which allocate different computational budget depending on the complexity of tasks [Simoulin and Crabbé, 2021; Csordás et al., 2022]. However, they lack the special Sliding-Dilated-Attention design that is necessary for ruling out naive solutions.

Linear RNN Given $x_0 = 0 \in \mathbb{R}^N$ and the input vectors $u_1 \cdots u_T$, a linear RNN [Orvieto et al., 2023b] for $k \in [T]$ can be written as:

$$\begin{aligned} x_k = Ax_{k-1} + Bu_k &= \sum_{j=0}^{k-1} A^j B u_{k-j} \\ &= \sum_{j=0}^{k-1} A^j v_{k-j}, \end{aligned}$$

where we set $v_{k-j} = Bu_{k-j}$. The operation can be accelerated by the parallel scan algorithm that permits efficient cumulative sum [Ladner and Fischer, 1980; Blelloch, 1990; Lakshmivarahan

[2]The original Longformer also adopts dilated attention on a few heads at higher layers but without the systematic pattern used in this chapter.

and Dhall, 1994; Martin and Cundy, 2018; Liu et al., 2023a; Smith et al., 2023b]. As we can see in Figure 6.2, the routing path specified by the parallel scan algorithm is the same as our Sliding-Dilated-Attention illustrated in Figure 6.1.

Task	RNN	Transformer	RegularGPT	
			$C = 2$	$C = 3$
1) Deletang et al.				
Even Pairs	100.0 / 100.0	99.7 / 73.2	100.0 / 89.3	100.0 / 96.6
Modular Arithmetic	100.0 / 100.0	21.9 / 20.3	96.4 / 82.6	21.2 / 20.5
Parity Check	100.0 / 98.9	52.3 / 50.1	100.0 / 100.0	100.0 / 88.7
Cycle Navigation	100.0 / 100.0	21.7 / 20.6	100.0 / 100.0	100.0 / 78.6
2) Bhattamishra et al.				
D_2	100.0 / 100.0	100.0 / 80.1	100.0 / 100.0	99.8 / 96.5
D_3	100.0 / 100.0	100.0 / 77.8	100.0 / 99.7	98.6 / 93.0
D_4	100.0 / 100.0	100.0 / 82.6	100.0 / 98.7	97.7 / 91.6
D_{12}	100.0 / 100.0	100.0 / 80.3	100.0 / 99.8	94.1 / 90.4
Tomita 3	100.0 / 100.0	100.0 / 94.4	100.0 / 99.7	100.0 / 99.9
Tomita 4	100.0 / 100.0	100.0 / 70.0	100.0 / 99.8	100.0 / 99.3
Tomita 5	100.0 / 100.0	74.5 / 74.5	100.0 / 99.8	98.2 / 84.1
Tomita 6	100.0 / 100.0	50.0 / 50.0	100.0 / 98.5	100.0 / 65.7

Table 6.1: Length generalization results on regular languages (Max/Avg). All models in the first section (Deletang et al.) are trained on sequences of length 40. The reported numbers are the average of length extrapolation results from 41 to 500. Each result is an average over 3 seeds. All models in the second section (Bhattamishra et al.) are trained on sequences of length 50. The reported numbers are the average of length extrapolation results from 51 to 100. Each result is an average over 3 seeds. Please refer to § 6.9.1 for the detailed hyperparameters.

6.4 Regular Language Experiments

6.4.1 Task Descriptions

We focus on the four tasks in section 1) [Deletang et al., 2023] of Table 6.1 as they will also be used in our analysis in § 6.6. For tasks in section 2), please refer to Bhattamishra et al. [2020a] for details.

Even Pairs A model needs to predict whether the total count of "ab" and "ba" pairs is even. In the example of "aabba", there is one "ab" and one "ba", resulting in a total count of 2, which is even. This task is equivalent to checking whether the first and last characters in a string are identical.

Modular Arithmetic Given a sequence of numbers in {0, 1, 2, 3, 4} and operations in {+, -, ·}, a model needs to compute the result modulo 5. For example, $x = 1 + 2 - 4$ evaluates to $y = 4$.

Parity Check A model needs to compute whether the number of bs in a given binary string is even. For example, the sequence x = aaabba contains 2 bs, which is even.

Cycle Navigation Given a sequence of movements on a cycle of length 5, a model needs to compute the end position. The possible movements are STAY, INCREASE, DECREASE encoded as {0, 1, 2}. The agent always starts at position 0. For example, 010211 means the agent stops at position $2 = 0 + 1 + 0 - 1 + 1 + 1$.

6.4.2 Language Transduction and Extrapolation

First, we want to know if endowing a Transformer with the notion of WM really improves its length extrapolation capability on regular languages. We test RegularGPT and all the baselines on two sets of regular languages from prior work [Deletang et al., 2023; Bhattamishra et al., 2020a].[3] Prior work often reports the maximum score across different hyperparameter settings and random seeds because their goal is to know if a model can extrapolate *at all*. We additionally report the average scores since we want to know if the model can consistently obtain good performance. The baseline models we compare against are an RNN and vanilla Transformer with Transformer-XL style relative positional embedding [Dai et al., 2019a]. Table 6.1 shows that RegularGPT with $C = 2$ acheives similar performance as an RNN and substantially outperforms a vanilla Transformer.

6.4.3 The Effect of Chunk Size C

We vary the chunk size C of RegularGPT to see its impact on the performance. The motivation for using a larger C is to reduce the number of layers (i.e., $\bar{L} = \log_C T$ decreases in C) and increase the degree of parallelization. However, in Table 6.1, a larger C seems to pose a challenge to RegularGPT on the Modular Arithmetic task. Modular Arithmetic is a hard task with far more states and complicated state transitions. Increasing C is likely to increase the task difficulty by composing more state transitions at once. We will have an in-depth discussion of the theoretical reasons in § 6.6.

6.4.4 Robust to Probability Changes

Other than the length extrapolation experiment, we alter the probability of sampling 1s of PARITY, i.e., set $\mathbb{P}(\sigma_i) \neq 0.5$. The results in Table 6.2 show that RegularGPT is robust to different sampling probabilities, indicating its successful modeling of the underlying regular language grammar. In contrast, a vanilla Transformer model struggles to achieve good performance even for the same

[3]Our implementation is based on the codebase of Deletang et al. [2023] at: `https://github.com/deepmind/neural_networks_chomsky_hierarchy`. We additionally implement the regular languages in the second section of Table 6.1.

Settings	**Probability** $\mathbb{P}(\sigma_i = 1)$				
	0.1	0.3	0.5	0.7	0.9
1) Same Length					
RegularGPT	100	100	100	100	100
RNN	100	100	100	100	100
Transformer	98.4	99.8	99.6	97.8	77.2
2) Extrapolation					
RegularGPT	100	100	100	100	100
RNN	100	100	100	100	100
Transformer	50.1	49.7	50.3	49.9	50.0

Table 6.2: We alter the probability $\mathbb{P}(\sigma_i = 1)$ used to sample 1s of PARITY. The same length setting is 40. The extrapolation setting is from 41 to 500. Each entry is an average over 3 seeds.

length setting, again validating the fact that it only finds the naive-summation solution as discussed in § 6.1.2.

T_{ex}	KERPLE	T5	ALiBi	RegularGPT (C/K)				
				32 / 6	64 / 6	128 / 6	128 / 12	256 / 6
512	24.71	24.50	24.53	32.06	30.17	28.80	26.37	27.90
1024	24.42	24.38	24.90	32.03	30.30	28.94	26.91	34.38
2048	24.21	25.01	25.08	791.74	30.56	29.14	27.08	34.85
4096	24.53	28.91	25.08	812.00	30.80	29.25	27.28	35.11
8192	24.74	39.08	25.08	818.49	1175.91	29.41	27.39	35.42

Table 6.3: Natural language extrapolation results on OpenWebText2. The training length is 512. The numbers are averaged over three random seeds. Please refer to § 6.9.2 for the detailed hyperparameters. The numbers for KERPLE, T5, and ALiBi deviate from the ones reported in Table 4.3 since the implementation was based on nanoGPT instead of GPT-Neox. nanoGPT is more flexible compared to GPT-Neox, making it a more suitable option when it comes to implementing the depth-wise recursive computations of RegularGPT.

6.5 Natural Language Experiments

Given that RegularGPT has been battle-tested on the main experiment of regular languages, we now shift gear to benchmark its performance in the natural language scenario. Given a model trained on sequences of length T_{tr}, we test it on much longer sequences of length $T_{ex} \gg T_{tr}$ during inference, and the goal is to observe similar perplexities. To optimize efficiency, we employ a random selection process to extract 1,000 chunks, each with T_{ex} tokens from the testing set. Subsequently, we calculate the average perplexity of the last tokens within these chunks to ensure each of them has $T_{ex} - 1$ tokens as the context, thereby avoiding the issue of early token curse [Press et al., 2022a; Chi

T_{ex}	KERPLE	T5	ALiBi	RegularGPT (C/K)				
				32 / 6	64 / 6	128 / 6	128 / 12	256 / 6
512	2.66	2.59	2.68	3.96	3.80	3.63	3.22	3.17
1024	2.61	2.53	2.88	3.87	3.76	3.49	3.24	3.38
2048	2.61	2.55	2.94	4.37	3.78	3.50	3.25	3.37
4096	2.67	2.78	2.94	4.33	3.79	3.50	3.26	3.38
8192	2.70	3.29	2.94	4.28	4.14	3.50	3.25	3.37

Table 6.4: Natural language extrapolation results on GitHub. The training length is 512. The numbers are averaged over three random seeds. Please refer to § 6.9.2 for the detailed hyperparameters. The numbers for KERPLE, T5, and ALiBi deviate from the ones reported in Table 4.3 since the implementation was based on nanoGPT instead of GPT-Neox. nanoGPT is more flexible compared to GPT-Neox, making it a more suitable option when it comes to implementing the depth-wise recursive computations of RegularGPT.

et al., 2023b]. We compare our model against the existing methods that are known to demonstrate the ability of length extrapolation including T5 [Raffel et al., 2020b], ALiBi [Press et al., 2022a], and KERPLE [Chi et al., 2022].[4] To counteract the loss of expressive power due to weight sharing, we thicken each layer of RegularGPT to K as detailed in § 6.2.

In Table 6.3, we first observe exploding perplexities for $C = 32$ after $T_{ex} \geq 2048$. RegularGPT might only learn to model $\lceil \log_{32} 512 \rceil = 2$ layers during training, hence it fails to recursively model more than $32^2 = 1024$ tokens during inference. This is validated by $C = 64$ since this time it is able to extrapolate until $64^{\lceil \log_{64} 512 \rceil} = 4096$. While the above argument seems to suggest large C, setting $C = 256$ also deteriorates the performance. This might be due to the limited number of chunks $(512/256 = 2)$ and r_i's (in Eq. (6.1)) observed at the second layer, making the learning of r_i's harder. We observe similar trends in Table 6.4, where we apply the same RegularGPT model on the GitHub dataset. Overall, C is a hyperparameter that needs to be carefully decided for RegularGPT on natural languages. We also observe that 128/12 performs better than 128/6, implying RegularGPT's performance could be improved by stacking more layers to counteract the performance loss due to Weight-Sharing.

Overall, Tables 6.3 and 6.4 show different sequence lengths that RegularGPT favors. In particular, RegularGPT performs the best when the chunk size is set to 128 on OpenWebText2, whereas a chunk size of 256 gives us the best performance on GitHub. This is reasonable as OpenWebText2 comes mainly from Internet forums such as Reddit with mostly short contextual dependencies.

It is worth noting that 128/12 performs relatively well and is close to previous methods designed specifically for the task of natural language extrapolation. We will analyze its inner workings in depth in Figure 6.4 and § 6.6, in which we find that RegularGPT learns the similar local receptive field as prior work, which is likely the key to its successful natural language extrapolation performance.

[4]We use the nanoGPT codebase: `https://github.com/karpathy/nanoGPT`, and the OpenWebText2 dataset: `https://huggingface.co/datasets/the_pile_openwebtext2`.

6.6 Discussion and Analysis

6.6.1 Regular Language and Finite State Semiautomaton

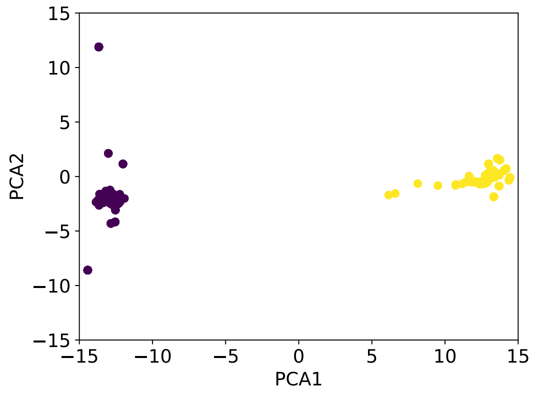
Regular language is the type of formal language recognized by an FSA [Chomsky, 1956a], which is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite non-empty set of states, Σ is a finite non-empty set of symbols, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a transition function; $F \subseteq Q$ is a set of final states. However, some of our tasks are better modeled by a finite-state transducer (FST) as discussed in § 6.1.1. To underpin both FSA and FST, we consider a semiautomation $\mathcal{A} = (Q, \Sigma, \delta)$ (i.e., an FSA without q_0 and F) and establish its connection to a Transformer model.

Let $\sigma_{a:b}$ be the sequence from position a (inclusive) to b (exclusive) out of a length T input sequence (i.e., $0 \leq a < b \leq T$). We define $\mathcal{A}(\sigma_{a:b}) : Q \to Q$ as the $(b-a)$-step state transition relation after receiving $\sigma_{a:b}$.

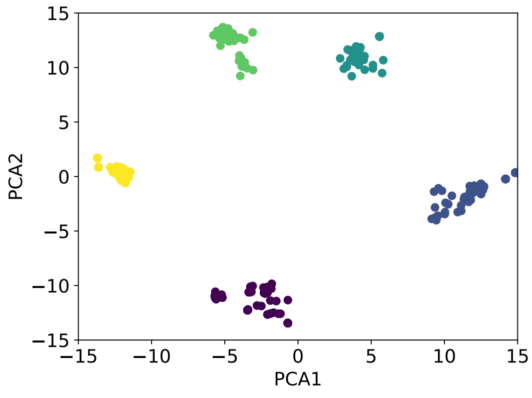
$$\mathcal{A}(\sigma_{a:b}) = \delta(\cdot|\sigma_{b-1}) \circ \cdots \circ \delta(\cdot|\sigma_a),$$

where $f(\cdot) \circ g(\cdot) = f(g(\cdot))$ denotes function composition. With abuse of notation, we define $\mathcal{A}_q(\sigma_{a:b}) \in Q$ as the state after receiving $\sigma_{a:b}$ if starting at $q \in Q$

$$\mathcal{A}_q(\sigma_{a:b}) = \delta(\cdot|\sigma_{b-1}) \circ \cdots \circ \delta(\cdot|\sigma_a), q_0 = q.$$



(a) PARITY.



(b) Cycle Navigation.

Figure 6.3: Clustering of FFN output vectors across all layers via PCA on the tasks of PARITY and Cycle Navigation.

6.6.2 Modeling Transition Composition

We want to show that the layers of RegularGPT with chunk size $C = 2$ can model the composition of two transition functions:

$$\mathcal{A}(\sigma_{a:b}) = \mathcal{A}(\sigma_{i:b}) \circ \mathcal{A}(\sigma_{a:i}) \text{ for } i \in [a+1, \ldots, b).$$

This way, the regular language problem can be solved recursively using the construction outlined in § 6.2 and Figure 6.1. To formalize the statement, we first observe that $\mathcal{A}(\sigma_{a:b})$, $\mathcal{A}(\sigma_{a:i})$, and $\mathcal{A}(\sigma_{i:b})$ can be represented in $\mathbb{R}^{|Q|^2}$:

$$\mathcal{A}(\sigma_{a:b}) = \begin{bmatrix} \text{OneHot}_{|Q|}(\mathcal{A}_{q_0}(\sigma_{a:b})) \\ \text{OneHot}_{|Q|}(\mathcal{A}_{q_1}(\sigma_{a:b})) \\ \cdots \\ \text{OneHot}_{|Q|}(\mathcal{A}_{q_{|Q|-1}}(\sigma_{a:b})) \end{bmatrix} \in \mathbb{R}^{|Q|^2}, \tag{6.2}$$

where $\text{OneHot}_{|Q|}(i)$ is a one-hot vector of length $|Q|$ with the i-th index being 1.

The next step is to mix $\mathcal{A}(\sigma_{a:i})$ and $\mathcal{A}(\sigma_{i:b})$ together and get $\mathcal{A}(\sigma_{a:b})$. We show in Lemma 6.1 that a 2-layer ReLU network can learn (and so can a Transformer layer) the composition. The proof of Lemma 6.1 is deferred to § 6.9.3.

Lemma 6.1 (Approximation for Binary Matrix Product)**.** *Let $A, B \in \{0,1\}^{n\times n}$ be binary matrices of dimension $n \times n$. Then, there exists a two-layer ReLU network such that*

$$f_{mlp}([Flat(A), Flat(B)]) = Flat(AB),$$

where $Flat(X)_{(i-1)n+j} = X_{i,j}$ for $i, j \in [n]$ is the operation that flattens a matrix into a vector.

Now, we can relate Lemma 6.1 to the FFN layers in RegularGPT. Following § 6.2, when chuck size $C = 2$ and thickness $K = 1$, the output vector $o_i^{(l)}$ depends on input sequence $\sigma_{i-2^{l+1}+1:i+1}$. Also, $o_i^{(l)}$ is computed from $o_{i-2^l}^{(l-1)}$ and $o_i^{(l-1)}$, which depend on input sequences $\sigma_{i-2^{l+1}+1:i-2^l+1}$ and $\sigma_{i-2^l+1:i+1}$, respectively. This observation implies that $o_i^{(l)}$ likely models the transition function $\mathcal{A}(\sigma_{i-2^{l+1}+1:i+1})$, which we denote as $o_i^{(l)} \sim \mathcal{A}(\sigma_{i-2^{l+1}+1:i+1})$. We will verify this assumption in § 6.6.3.

If $o_i^{(l)} \sim \mathcal{A}(\sigma_{i-2^{l+1}+1:i+1})$ is true, Lemma 6.1 implies that RegularGPT's FFN models the transition function composition. This is immediate by setting $o_{i-2^l}^{(l-1)} \sim \text{Flat}(\mathcal{A}(\sigma_{i-2^{l+1}+1:i-2^l+1}))$, $o_i^{(l-1)} \sim \text{Flat}(\mathcal{A}(\sigma_{i-2^l+1:i+1}))$ and recognizing the fact that function composition is a matrix product under the representation of Eq. (6.2).

The next step is to explain the use of self-attention layers in RegularGPT. Although Lemma 6.1 has established a composition, it is unclear how the transitions are concatenated in the first place (i.e., $[\text{Flat}(A), \text{Flat}(B)]$). With a two-head self-attention and the learnable relative positional scalars, it is possible to adjust them so that the attention output contains the concatenated information $[\text{Flat}(A), \text{Flat}(B)]$.

Recall in Eq. (6.1), each head has a different set of scalars r_i's. One concrete construction for concatenation is setting $r_0 = 0$ and the remaining $-\infty$ for the first head; $r_1 = 0$ and the remaining $-\infty$ for the second head. In other words, each head is only responsible for capturing one state transition. After the multi-head self-attention operation, we obtain the concatenation of two state transitions.

Finally, when the prediction head reads out the answer, the operation is equivalent to a mapping from $\mathcal{A}(\sigma_{0:T}) \in \mathbb{R}^{|Q|\times|Q|}$ to $\mathcal{A}_{q_0}(\sigma_{0:T}) = \mathcal{A}(\sigma_{0:T}) \circ q_0 \in \mathbb{R}^{|Q|}$. Since we assume that $o_{T-1}^{(l)}$ models $\mathcal{A}(\sigma_{0:T})$, the transduction readout is performed by a linear map on $o_{T-1}^{(l)}$ as $W_o o_{T-1}^{(l)}$. Our tree-structured construction also guarantees that the final answer could be derived using $\log_2 T$ layers.

6.6.3 Verification of Transition Modeling

To verify whether our model learns the dynamics of a semiautomaton, we perform a clustering experiment to demystify the FFN output representations on the tasks of PARITY and Cycle Navigation. The two tasks are chosen as we can easily derive their state transition functions. For example, there are only two state transitions in PARITY:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and five state transitions in Cycle Navigation[5]:

$$\begin{bmatrix} \text{OneHot}_5((0+k)\text{mod } 5) \\ \text{OneHot}_5((1+k)\text{mod } 5) \\ \text{OneHot}_5((2+k)\text{mod } 5) \\ \text{OneHot}_5((3+k)\text{mod } 5) \\ \text{OneHot}_5((4+k)\text{mod } 5) \end{bmatrix}, \quad \text{for } k \in [0, ..., 4].$$

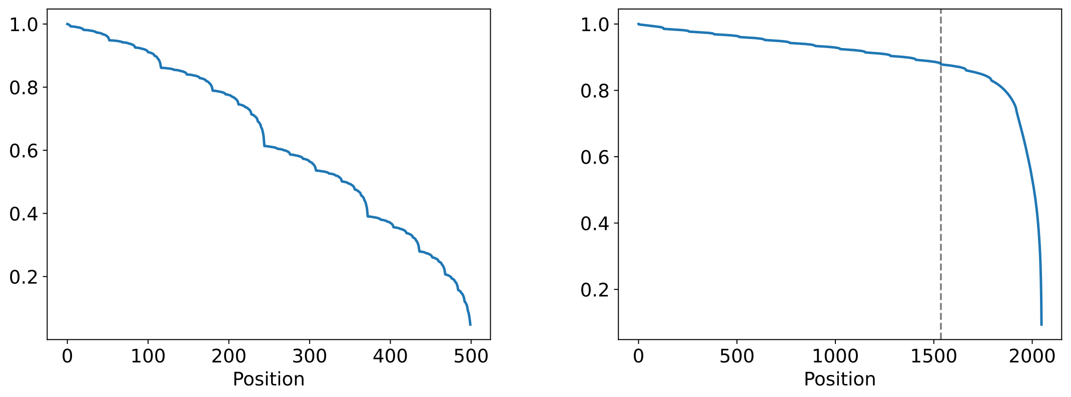
e.g., $k = 2$ gives $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$.

Given a testing input sequence of length 500 that is much longer than the training length 40, we extract the output $o_i^{(l)}$ of all layers l, perform dimension reduction using PCA, and plot the dimension-reduced points on a 2D plane. Ideally, we want to see a limited number of clusters across all layers, indicating the model learns to capture the state transition function. As we can see in Figure 6.3, PARITY has 2 clusters and Cycle Navigation has 5 clusters. The clear clustering effect demonstrates RegularGPT's correct learning of state transition functions. This is in contrast to the naive-summation approach learned by a vanilla Transformer as shown in Figure B.4 of Deletang et al. [2023].

6.6.4 Receptive Field Analysis

We resort to the gradient analysis tool [Chi et al., 2023b] to inspect the receptive field of RegularGPT on regular and natural languages. It computes a cumulative sum of the gradient norms starting from the most recent token to the earliest one. A large magnitude of slope at a position means the most recent token has a high dependency on that position. Ideally, we would like to see the receptive field covering the whole input sequence for the case of regular languages because every single bit in the input sequence is important for the final results. This is equivalent to a slanted line going from the lower right to the upper left, which is validated in Figure 6.4a. As for natural language, we discover something interesting in Figure 6.4b in that RegularGPT settles on the local windowed-attention

[5]Cycle Navigation (§ 6.4.1) has 3 one-step transitions (i.e., $|\mathcal{A}(\sigma_{a:a+1})| = 3$). Composing these transitions yields 5 different multi-step state transitions (i.e., $|\mathcal{A}(\sigma_{a:b})| = 5$ if $b - a \geq 5$).



(a) Regular Language - PARITY (b) Natural Language - OpenWebText2

Figure 6.4: Receptive field of RegularGPT via the cumulative gradient analysis tool [Chi et al., 2023b].

pattern as those enforced manually in prior work [Press et al., 2022a; Chi et al., 2022, 2023b]. This suggests the task of natural language modeling mostly needs only local context to achieve good performance, which aligns with the common belief.

6.6.5 RegularGPT Exploits Natural Language Structures

The recursive composition nature of RegularGPT can be likened to how humans process long context texts. We break it down following inherent structures such as chapters and paragraphs, then the final outcome is derived by recursively merging partial information. Taking summarization as an example, prior work [Wu et al., 2021b] also hinges on the same recursive decomposition idea to summarize inputs as long as books.

6.7 Conclusion

This chapter introduces RegularGPT, a novel variant of the Transformer architecture inspired by the notion of working memory that can effectively model regular languages with high efficiency. Theoretical explanations and accompanying clustering visualizations are presented to illustrate how RegularGPT captures the essence of regular languages. Moreover, RegularGPT is evaluated on the task of natural language length extrapolation, revealing its intriguing rediscovery of the local windowed attention effect previously observed in related research. Notably, RegularGPT establishes profound connections with various existing architectures, thereby laying the groundwork for the development of future Transformer models that facilitate efficient algorithmic reasoning and length extrapolation.

6.8 Limitations

Currently, this thesis sets the chunk size C of RegularGPT to a constant. Can we make the chunk size more flexible? A flexible and data-driven C could further boost its performance in natural languages, as they often demonstrate diverse patterns unlike regular languages underpinned by simple grammars. This might also improve the performance of RegularGPT when $C \neq 128$.

Naively applying RegularGPT to instruction following tasks such as QA might be suboptimal, since the question or instruction will be undesirably merged with the input context. One potential solution is to modify RegularGPT into an encoder-decoder architecture, where the encoder keeps the current design, and the decoder is trained to predict continuations. The modified architecture allows practitioners to put the question or instruction at the beginning of the decoder, so that it can stand out from the long encoder input context.

6.9 Proofs and Experimental Details

6.9.1 Hyperparameters for the Regular Language Experiments

We report the hyperpamaters used in the regular language experiments (Table 6.1) in Table 6.5.

6.9.2 Hyperparameters for the Natural Language Experiments

# Layers	$\log_C T$
Hidden Size	256
# Attention Heads	8
Train Seq. Len.	40 or 50
# Trainable Params.	4.3 M
Optimizer	Adam (lr 1e-4, 3e-4, 5e-4)
Batch Size	128
Train Steps	100,000
Precision	float32
Dataset	Regular Languages

Table 6.5: Hyperparameters for the regular language experiments.

We report the hyperpamaters used in the natural language experiments (Table 6.3) in Table 6.6.

6.9.3 Proof of Lemma 6.1

Lemma 6.1 (Approximation for Binary Matrix Product)**.** *Let $A, B \in \{0,1\}^{n\times n}$ be binary matrices of dimension $n \times n$. Then, there exists a two-layer ReLU network such that*

$$f_{mlp}([Flat(A), Flat(B)]) = Flat(AB),$$

# Layers	$K \log_C T$
Hidden Size	768
# Attention Heads	12
Train Seq. Len.	512
# Trainable Params.	81M ($K = 6$) or 123M ($K = 12$)
Optimizer	Adam (lr 6e-4)
Batch Size	32
Train Steps	50,000
Precision	bfloat16
Dataset	OpenWebText2

Table 6.6: Hyperparameters for the natural language experiments.

where Flat$(X)_{(i-1)n+j} = X_{i,j}$ *for* $i, j \in [1, ..., n]$ *is the operation that flattens a matrix into a vector.*

Proof. Observe that a ReLU operation can perfectly approximate the multiplication of two binary scalars:

$$\text{ReLU}(a + b - 1) = a \cdot b, \quad \text{for } a, b \in \{0, 1\}.$$

The binary matrix product AB is composed of n^3 binary scalar products of the form:

$$\begin{aligned} A_{ik}B_{kj} &= x_{(i-1)n+k}x_{(n+k-1)n+j} \\ \text{for} \quad & i, j, k \in [1, .., n], \end{aligned}$$

where $x = [\text{Flat}(A), \text{Flat}(B)]$ is the concatenated flattened input. Our goal is to construct two neural network layers. The first layer computes all n^3 binary scalar products. The second layer sums these products into the form of matrix product; i.e., $\sum_{k=1}^{n} A_{ik}B_{kj}$.

The first layer's binary weight matrix $W^{(1)} \in \{0, 1\}^{2n^2 \times n^3}$ is constructed as:

$$\begin{aligned} &\text{For } z \in [1, ..., 2n^2], \;\; i, j, k \in [1, ..., n], \\ &W^{(1)}_{z,(i-1)n^2+(j-1)n+k} = \\ &\begin{cases} 1 & \text{if } z = (i-1)n + k \text{ or } (n + k - 1)n + j \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{6.3}$$

Then, the first layer computes all n^3 binary scalar products as follows:

$$\begin{aligned} &\text{ReLU}\Big([\text{Flat}(A), \text{Flat}(B)]W^{(1)} - \mathbb{1}^{\top}_{n^3}\Big)_{(i-1)n^2+(j-1)n+k} \\ &= A_{ik}B_{kj} \quad \text{for } i, j, k \in [1, ..., n]. \end{aligned}$$

To sum these n^3 products into n^2 results, the second layer's binary weight matrix $W^{(2)} \in \{0, 1\}^{n^3 \times n^2}$

is constructed as:

$$W^{(2)} = I_{n^2} \otimes \mathbb{1}_n = \begin{bmatrix} \mathbb{1}_n & 0_n & 0_n & \dots & 0_n \\ 0_n & \mathbb{1}_n & 0_n & \dots & 0_n \\ \vdots & & & & \vdots \\ 0_n & \dots & & 0_n & \mathbb{1}_n \end{bmatrix}$$
$$\in \{0,1\}^{n^3 \times n^2},$$

where I_{n^2} is an $n^2 \times n^2$ identity matrix, $\otimes$ is the Kronecker product, 0_n is an n-dimensional column vector of all zeros, and $\mathbb{1}_n$ is an n-dimensional column vector of all ones. We arrive at a two-layer ReLU network that perfectly approximates the multiplication of two binary matrices:

$$\begin{aligned} & f_{\text{mlp}}([\text{Flat}(A), \text{Flat}(B)]) \\ &= \text{ReLU}\left([\text{Flat}(A), \text{Flat}(B)] W^{(1)} - \mathbb{1}_{n^3}^\top\right) W^{(2)} \\ &= \text{Flat}(AB). \end{aligned}$$

□

6.9.4 Illustration of Lemma 6.1

6.9.5 Illustration of the Binary Weight Matrices

We illustrate $W^{(1)}$ and $W^{(2)}$ of Lemma 6.1 as follows:

```
import numpy as np

def get_W1(n):
  n2 = n*n
  W1 = np.zeros((2*n*n, n**3), dtype=int)
  for i in range(n):
    for j in range(n):
      for k in range(n):
        W1[i*n+k, i*n2+j*n+k] = 1
        W1[n2+k*n+j, i*n2+j*n+k] = 1
  return W1

def get_W2(n):
  eye = np.eye(n*n, dtype=int)
  ones = np.ones((n,1), dtype=int)
  W2 = np.kron(eye, ones)
  return W2
```

get_W1(2) gives:

```
[[1 0 1 0 0 0 0 0]
 [0 1 0 1 0 0 0 0]
 [0  0  0  0  1  0  1  0]
 [0  0  0  0  0  1  0  1]
 [1  0  0  0  1  0  0  0]
 [0  0  1  0  0  0  1  0]
 [0  1  0  0  0  1  0  0]
 [0  0  0  1  0  0  0  1]]
```

get_W2(2) gives:

```
[[1  0  0  0]
 [1  0  0  0]
 [0  1  0  0]
 [0  1  0  0]
 [0  0  1  0]
 [0  0  1  0]
 [0  0  0  1]
 [0  0  0  1]]
```

6.9.6 An Illustrative Example for $n = 2$

Suppose the input matrices are:

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

The concatenated flattened input becomes:

$$x = [\text{Flat}(A), \text{Flat}(B)] = [1\ 0\ 1\ 0\ 0\ 1\ 1\ 0].$$

Then, Lemma 6.1 is verified as follows:

$$\begin{aligned} &\text{ReLU}\left(xW^{(1)} - \mathbb{1}_{n^3}^\top\right) W^{(2)} \\ =&\text{ReLU}\left([1\ 1\ 2\ 0\ 1\ 1\ 2\ 0] - 1\right) W^{(2)} \\ =&[0\ 0\ 1\ 0\ 0\ 0\ 1\ 0]W^{(2)} \\ =&[0\ 1\ 0\ 1] \\ =&\text{Flat}\left(\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}\right) = \text{Flat}\left(AB\right). \end{aligned}$$

Here is the Python code for the above example:

```
A = np.array([[1,0],[1,0]]).reshape(-1)
B = np.array([[0,1],[1,0]]).reshape(-1)
x = np.concatenate([A, B]).reshape(1,-1)
W1 = get_W1(2)
W2 = get_W2(2)
flat_AB = np.maximum(x @ W1 -1,0) @ W2
```

Chapter 7

Conclusion and Future Work

7.1 Summary of Contributions

Sequence length has a significant impact on transformer model computations; it is quadratic w.r.t input sequence length. This is a known weakness of such models and makes it difficult for them to model longer sequences, which is important for many tasks. This thesis presents an analysis of sequence properties and then uses it to propose several new techniques that address the consequences of length.

Chapter 2 establishes the foundational role of positional embeddings in Transformers. The shrinking variance effect, while crucial for indicating positions, fails to differentiate positions effectively as sequence length extends, underscoring the need for positional embeddings such as ALiBi, a state-of-the-art relative positional embedding design. **Chapter 3** therefore generalizes ALiBi via the help of conditionally positive definite kernels. The resulting KERPLE relative positional embedding design is the first to truly make use of longer-than-training sequence information during inference.

With a plethora of length-extrapolated positional embeddings proposed, it becomes imperative for the research community to monitor the progress of developed techniques. **Chapter 4** addresses the needs by quantitatively measuring the empirical receptive field size of a Transformer using the proposed gradient analysis tool. In addition, the tool also helps elucidate the failure of length extrapolation: the explosion of empirical receptive field size. This finding prompts us to revisit the earliest proposed Sinusoidal positional embedding and transform it into an extrapolatable variant named Sandwich.

This thesis then recognizes the constraints of previously explored recency-biased positional embeddings and pivots toward a broader array of applications. **Chapter 5** proposes two attention alignment strategies, optimized through softmax temperature scaling, enhancing task performance where dynamic and flexible access to contextual information is crucial, such as in retrieval, multi-document QA, and code completion. **Chapter 6** finally extends the exploration to scenarios demanding global context awareness, exemplified by the PARITY task. This thesis proposes a new Transformer language model, RegularGPT, that achieves length-extrapolatable performance via local input composition. Its three critical design choices, Weight-Sharing, Adaptive-Depth, and Sliding-Window-Attention, jointly enables global context understanding and more robust extrapolation performance on regular and natural languages.

7.2 Recipe for Practitioners

Depending on the actual amount of context information required by a task, this thesis provides the following recipe and the corresponding inference cost of a length L sequence for practitioners.

- For tasks that require recent tokens, use the proposed KERPLE positional embedding design (generalized ALiBi). Alternatively, one can follow the receptive field alignment principle to design a new type of positional embeddings that suits their needs. Since this thesis maintains the original Transformer self-attention computation and only modifies the construction of relative positional embeddings, the inference complexity stays the same, which is L^2. Note that this line of research requires pretraining a Transformer language model from scratch, which can be costly.
- For tasks that require flexible access to parts of the whole input context, use T5 with the proposed softmax temperature scaling strategy. The inference cost is also L^2. Note that there are several large T5-based pre-trained models available such as Flan-T5-XXL, obviating the need to pretrain a new Transformer language model from scratch.
- For tasks that require global context understanding, use the proposed RegularGPT to combine local and chunked primitives. The inference complexity is $LC\log_C L$, where C is the chunk size of RegularGPT. As long as $C\log_C L < L$, RegularGPT will be more efficient than a vanilla Transformer. This line of research also requires the pre-training of a model from scratch.

Although task dependent, the methods proposed in this thesis in general can achieve *better* performance on sequences 10x longer than the training sequence length (512$\sim$768). For tasks such as long-context retrieval in § 5.3, the proposed method can give nearly perfect performance on sequences up to 30k on an A6000 GPU. We can expect the same level of performance on even longer sequences, provided that there is sufficient GPU memory. The relevant information is detailed in the chapters that present the different techniques.

7.3 Comparison to Recent Long Context Language Models

Apart from pure Transformer-based language models, there has recently been a surge in linear recurrent neural networks (LRNN) [Gu et al., 2022; Gupta et al., 2022; Hasani et al., 2023; Smith et al., 2023a; Orvieto et al., 2023a; Peng et al., 2023a] due to their constant inference cost. There is no free lunch: evidence shows that pure LRNNs have a hard time representing basic arithmetic operations [Chi et al., 2024a] and recalling early information in long sequences, hindering their in-context learning and retrieval capabilities [Emami et al., 2021; Wen et al., 2024; Jelassi et al., 2024]. Our proposed length-extrapolated Transformers suffer less from the aforementioned issues as they still maintain the original self-attention mechanism and are further equipped with greatly enhanced long context processing power.

To combine the best of both worlds, recent models have begun to adopt a hybrid architecture that blends the self-attention mechanism with LRNNs. This trend is manifested by the release of several large pre-trained LRNNs such as Griffin [De et al., 2024], Jamba [Lieber et al., 2024], Zamba[1], and Megalodon [Ma et al., 2024]. We foresee that the proposed length extrapolation improvements can function as an extension to these models, particularly on their self-attention components, further bolstering their long context processing capabilities.

7.4 Future Work

Each chapter of this thesis paves the way for exciting future work. We outline several concrete ideas below:

- **Chapter 2** reveals that the shrinking variance effect can be a hidden source of positional information for a Transformer language model on its own, without explicit positional embeddings. Variance is unlikely to be the only useful statistic; there might be other geometric information [Song and Zhong, 2023] encoded in the model. Such discovery will deepen our understanding of Transformer's inner workings and possibly lead to better length extrapolation performance.
- **Chapter 3** experiments with training a Transformer language model equipped with KERPLE from scratch. However, to preserve computational resources, KERPLE might also be used as a post-hoc length extending technique during the fine-tuning stage. There is evidence that suggests combining ALiBi [Press et al., 2022a], a special case of KERPLE, with position interpolation [Al-Khateeb et al., 2023] helps boost the length extrapolation performance. We believe using the more general KERPLE might lead to further improvements.
- **Chapter 4** hypothesizes that the log-decaying trend imposed on input contexts is helpful for the length extrapolation performance of T5, Sandwich, and KERPLE. While this hypothesis was corroborated by some follow-up work [Li et al., 2024], little is still known as to why log-decay is more effective than other decaying trends. Mechanistically analyzing the compositional effect of self-attention and log-decay might help elucidate the underlying dynamics [Han et al., 2023].
- **Chapter 5** focuses on extending the input context length of the T5 models. Considering that rotary [Su et al., 2021a] is widely adopted nowadays, devising a rotaty-compatible temperature scaling method will make our findings more impactful. Some concurrent work by others has scratched the surface of this idea [Peng et al., 2023b] by adjusting rotary's softmax temperature followed by long-context fine-tuning; however, a) they did not explain the underlying dynamics of temperature scaling, and b) long-context fine-tuning is a costly process. The method proposed in this thesis might shed light on potential remedies.
- **Chapter 6** delves into the realm of regular languages, leaving aside the more complex formal languages in the Chomsky hierarchy. Adapting the Transformer architecture to integrate

[1]`https://www.zyphra.com/zamba`

memory modules, e.g. a stack, could potentially enhance the model's ability to extrapolate longer sequences, particularly in dealing with complex formal languages. Although efforts have been made to train Transformers to grasp context-free languages [Allen-Zhu and Li, 2023], this aspect of length extrapolation remains largely unexplored. We consider this to be an essential test to assess the efficacy of grammar learning.

We hope that these ideas can be explored in the future and that they yield new insights into the design of Transformer language models.

Bibliography

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Eryn J Adams, Anh T Nguyen, and Nelson Cowan. Theories of working memory: Differences in definition, degree of modularity, role of attention, and purpose. *Language, speech, and hearing services in schools*, 49(3):340–355, 2018.

Salesforce AI. Long sequence modeling with xgen: A 7b llm trained on 8k input sequence length, 2023. URL `https://blog.salesforceairesearch.com/xgen`.

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.19. URL `https://aclanthology.org/2020.emnlp-main.19`.

Faisal Al-Khateeb, Nolan Dey, Daria Soboleva, and Joel Hestness. Position interpolation improves alibi extrapolation. *arXiv preprint arXiv:2310.13017*, 2023.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, context-free grammar. *arXiv preprint arXiv:2305.13673*, 2023.

Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Shivanshu Purohit, Tri Songz, Wang Phil, and Samuel Weinbach. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch, 8 2021. URL `https://www.github.com/eleutherai/gpt-neox`.

Cem Anil, Yuhuai Wu, Anders Johan Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Venkatesh Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=zSkYVeX7bC4`.

Anthropic. The claude 3 model family: Opus, sonnet, haiku. 2024.

Andre Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. doi: 10.23915/distill.00021. https://distill.pub/2019/computing-receptive-fields.

Kristijan Armeni, Christopher Honey, and Tal Linzen. Characterizing verbatim short-term memory in neural language models. In *Proceedings of the 26th Conference on Computational Natural Language Learning (CoNLL)*, pages 405–424, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL `https://aclanthology.org/2022.conll-1.28`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Alan Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.

Alan D Baddeley and Graham Hitch. Working memory. In *Psychology of learning and motivation*, volume 8, pages 47–89. Elsevier, 1974.

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

Christian Berg, Jens Peter Reus Christensen, and Paul Ressel. *Harmonic analysis on semigroups: theory of positive definite and related functions*, volume 100. Springer, 1984.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online, November 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.576. URL `https://aclanthology.org/2020.emnlp-main.576`.

Satwik Bhattamishra, Arkil Patel, and Navin Goyal. On the computational power of transformers and its implications in sequence modeling. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 455–475, Online, November 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.conll-1.37. URL `https://aclanthology.org/2020.conll-1.37`.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

Sid Black, Stella Biderman, Alex Andonian, Quentin Anthony, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Jason Phang, Michael Pieler, Shivanshu Purohit, Tri Songz, Phil Wang, and Samuel Weinbach. GPT-NeoX: Large scale autoregressive language modeling in pytorch, 2021. URL `http://github.com/eleutherai/gpt-neox`.

Guy E Blelloch. Prefix sums and their applications. *School of Computer Science, Carnegie Mellon University Pittsburgh, PA, USA*, 1990.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on* *machine learning*, pages 2206–2240. PMLR, 2022.

Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/32bb90e8976aab5298d5da10fe66f21d-Paper.pdf`.

Guanzheng Chen, Xin Li, Zaiqiao Meng, Shangsong Liang, and Lidong Bing. CLEX: Continuous length extrapolation for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=wXpSidPpc5`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a.

Pu-Chin Chen, Henry Tsai, Srinadh Bhojanapalli, Hyung Won Chung, Yin-Wen Chang, and Chun-Sung Ferng. A simple and effective positional encoding for transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2974–2988, 2021b.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

Yifan Chen, Qi Zeng, Heng Ji, and Yun Yang. Skyformer: Remodel self-attention with gaussian kernel and nystr\" om method. *Advances in Neural Information Processing Systems*, 34, 2021c.

Ta-Chung Chi, Ting-Han Fan, Peter J Ramadge, and Alexander Rudnicky. Kerple: Kernelized relative positional embedding for length extrapolation. *Advances in Neural Information Processing Systems*, 35:8386–8399, 2022.

Ta-Chung Chi, Ting-Han Fan, Li-Wei Chen, Alexander Rudnicky, and Peter Ramadge. Latent positional information is in the self-attention variance of transformer language models without positional embeddings. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1183–1193, 2023a.

Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, and Peter Ramadge. Dissecting transformer length extrapolation via the lens of receptive field analysis. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13522–13537, 2023b.

Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, and Peter Ramadge. Transformer working memory enables regular language reasoning and natural language length extrapolation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5972–5984, 2023c.

Ta-Chung Chi, Ting-Han Fan, and Alexander Rudnicky. Advancing regular language reasoning in linear recurrent neural networks. In *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024a.

Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, and Peter Ramadge. Attention alignment and flexible positional embeddings improve transformer length extrapolation. In *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024b.

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7654–7664, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.527. URL `https://aclanthology.org/2022.acl-long.527`.

N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956a. doi: 10.1109/TIT.1956.1056813.

Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956b.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4828. URL `https://aclanthology.org/W19-4828`.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020.

Nelson Cowan. *Attention and memory: An integrated framework*. Oxford University Press, 1998.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The neural data router: Adaptive control flow in transformers improves systematic generalization. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=KBQP4A_J1K`.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017. doi: 10.1109/ICCV.2017.89.

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*, pages 2978–2988, Florence, Italy, July 2019a. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL `https://aclanthology.org/P19-1285`.

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*, pages 2978–2988, Florence, Italy, July 2019b. Association for Computational Linguistics.

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

Wietse de Vries, Andreas van Cranenburgh, and Malvina Nissim. What's so special about BERT's layers? a closer look at the NLP pipeline in monolingual and multilingual models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4339–4350, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.389. URL `https://aclanthology.org/2020.findings-emnlp.389`.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyzdRiR9Y7`.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A Ortega. Neural networks and the chomsky hierarchy. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=WbxHAzkeQcn`.

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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019a. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019b. Association for Computational Linguistics.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556, 2004.

Adele Diamond. Executive functions. *Annual review of psychology*, 64:135–168, 2013.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm:

General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335, 2022.

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. Position Information in Transformers: An Overview. *Computational Linguistics*, pages 1–31, 07 2022.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJg7KhVKPH`.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Melikasadat Emami, Mojtaba Sahraee-Ardakan, Parthe Pandit, Sundeep Rangan, and Alyson K Fletcher. Implicit bias of linear rnns. In *International Conference on Machine Learning*, pages 2982–2992. PMLR, 2021.

K Anders Ericsson and Walter Kintsch. Long-term working memory. *Psychological review*, 102(2): 211, 1995.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020a.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020b.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1243–1252. JMLR.org, 2017.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=uYLFoz1vlAC`.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, 2023. URL `https://api.semanticscholar.org/CorpusID:259262301`.

Ankit Gupta and Jonathan Berant. GMAT: global memory augmentation for transformers. *CoRR*, abs/2006.03274, 2020. URL `https://arxiv.org/abs/2006.03274`.

Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=RjS0j6tsSrf`.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938.

PMLR, 2020.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020. doi: 10.1162/tacl_a_00306. URL `https://aclanthology.org/2020.tacl-1.11`.

Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=g4OTKRKfS7R`.

Adi Haviv, Ori Ram, Ofir Press, Peter Izsak, and Omer Levy. Transformer language models without positional encodings still learn positional information. *arXiv preprint arXiv:2203.16634*, 2022.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. {DEBERTA}: {DECODING}-{enhanced} {bert} {with} {disentangled} {attention}. In *International Conference on Learning Representations*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. In *International Conference on Learning Representations*, 2019.

Zhiheng Huang, Davis Liang, Peng Xu, and Bing Xiang. Improve transformer models with better relative position embeddings. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3327–3335, Online, November 2020. Association for Computational Linguistics.

Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. Language modeling with deep transformers. In *INTERSPEECH*, 2019.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.74. URL `https://aclanthology.org/2021.eacl-main.74`.

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

M I Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986. 5 1986. URL `https://www.osti.gov/biblio/6910294`.

Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020a.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 13–18 Jul 2020b.

Guolin Ke, Di He, and Tie-Yan Liu. Rethinking positional encoding in language pre-training. In *International Conference on Learning Representations*, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL `http://arxiv.org/abs/1412.6980`. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgNKkHtvB`.

Shun Kiyono, Sosuke Kobayashi, Jun Suzuki, and Kentaro Inui. SHAPE: Shifted absolute position embedding for transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3309–3321, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Richard E Ladner and Michael J Fischer. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4):831–838, 1980.

Sivaramakrishnan Lakshmivarahan and Sudarshan K Dhall. *Parallel computing using the prefix problem*. Oxford University Press, 1994.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1eA7AEtvS`.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/lee19d.html`.

Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Biocomputing 2002*, pages 564–575. World Scientific, 2001.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL `https://aclanthology.org/2021.emnlp-main.243`.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can opensource llms truly promise on context length, 2023. URL `https://lmsys.org/blog/2023-06-29-longchat/`.

Shanda Li, Chong You, Guru Guruganesh, Joshua Ainslie, Santiago Ontanon, Manzil Zaheer, Sumit Sanghai, Yiming Yang, Sanjiv Kumar, and Srinadh Bhojanapalli. Functional interpolation for relative positions improves long context transformers. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=rR03qFesqk`.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Tatiana Likhomanenko, Qiantong Xu, Gabriel Synnaeve, Ronan Collobert, and Alex Rogozhnikov. Cape: Encoding relative positions with continuous augmented positional embeddings. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 16079–16092. Curran Associates, Inc., 2021.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *International Conference on Learning Representations*, 2023a. URL `https://openreview.net/forum?id=De4FYqjFueZ`.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023b.

Xiaoran Liu, Hang Yan, Chenxin An, Xipeng Qiu, and Dahua Lin. Scaling laws of roPE-based extrapolation. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=JO7k0SJ5V6`.

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. *arXiv preprint arXiv:1907.11692*, 2019.

Shengjie Luo, Shanda Li, Tianle Cai, Di He, Dinglan Peng, Shuxin Zheng, Guolin Ke, Liwei Wang, and Tie-Yan Liu. Stable, fast and accurate: Kernelized attention with relative positional encoding. *Advances in Neural Information Processing Systems*, 34, 2021a.

Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4905–4913, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Ziyang Luo, Artur Kulmizev, and Xiaoxi Mao. Positional artefacts propagate through masked language model embeddings. 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)*, pages 5312–5327, Online, August 2021b. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.413. URL `https://aclanthology.org/2021.acl-long.413`.

Xuezhe Ma, Xiaomeng Yang, Wenhan Xiong, Beidi Chen, Lili Yu, Hao Zhang, Jonathan May, Luke Zettlemoyer, Omer Levy, and Chunting Zhou. Megalodon: Efficient llm pretraining and inference with unlimited context length. *arXiv preprint arXiv:2404.08801*, 2024.

Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=HyUNwulC-`.

Sebastian Mika, Bernhard Schölkopf, Alex Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel pca and de-noising in feature spaces. *Advances in neural information processing systems*, 11, 1998.

Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239, 2012. doi: 10.1109/SLT.2012.6424228.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

Akira Miyake, Priti Shah, et al. *Models of working memory*. Cambridge: Cambridge University Press, 1999.

Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. *arXiv preprint arXiv:2305.16300*, 2023.

Aida Nematzadeh, Sebastian Ruder, and Dani Yogatama. On memory in human and artificial language processing systems. In *Proceedings of ICLR Workshop on Bridging AI and Cognitive Science*, 2020.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2022. URL https://openreview.net/forum?id=iedYJm92o0a.

Klaus Oberauer. Access to information in working memory: exploring the focus of attention. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 28(3):411, 2002.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. URL http://arxiv.org/abs/1609.03499. cite arxiv:1609.03499.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 26670–26698. PMLR, 23–29 Jul 2023a. URL https://proceedings.mlr.press/v202/orvieto23a.html.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. *arXiv preprint arXiv:2303.06349*, 2023b.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. Rwkv: Řřinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023a.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023b.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *International Conference on Learning Representations*, 2021.

Ofir Press. The use case for relative position embeddings. 2022. URL https://ofir.io/The-Use-Case-for-Relative-Position-Embeddings/.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=R8sQPpGCv0.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022b.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. In *International Conference on Learning Representations*, 2022.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019b.

Jack Rae and Ali Razavi. Do transformers need deep long-range memory? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7524–7529, Online, July 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.672. URL `https://aclanthology.org/2020.acl-main.672`.

Jack Rae and Ali Razavi. Do transformers need deep long-range memory? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7524–7529, 2020b.

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, 2020a.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020b.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=R-616EWWKF5`.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

Veselin Raychev, Pavol Bielik, and Martin Vechev. Probabilistic model for code with decision trees. *ACM SIGPLAN Notices*, 51(10):731–747, 2016.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Eric Michael Smith, Y-Lan Boureau, and Jason Weston. Recipes for building an open-domain chatbot. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 300–325, Online, April 2021. Association for Computational Linguistics.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022a.

Teven Le Scao, Thomas Wang, Daniel Hesslow, Lucile Saulnier, Stas Bekman, M Saiful Bari, Stella Biderman, Hady Elsahar, Jason Phang, Ofir Press, Colin Raffel, Victor Sanh, Sheng Shen, Lintang Sutawika, Jaesung Tae, Zheng Xin Yong, Julien Launay, and Iz Beltagy. What language model to train if you have one million GPU hours? In *Challenges & Perspectives in Creating Large Language Models*, 2022b. URL `https://openreview.net/forum?id=rI7BL3fHIZq`.

Isaac J Schoenberg. Metric spaces and positive definite functions. *Transactions of the American Mathematical Society*, 44(3):522–536, 1938.

Bernhard Schölkopf. The kernel trick for distances. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

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)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Antoine Simoulin and Benoit Crabbé. How many layers and why? An analysis of the model depth in transformers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 221–228, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-srw.23. URL `https://aclanthology.org/2021.acl-srw.23`.

Koustuv Sinha, Robin Jia, Dieuwke Hupkes, Joelle Pineau, Adina Williams, and Douwe Kiela. Masked language modeling and the distributional hypothesis: Order word matters pre-training for little. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2888–2913, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.230. URL `https://aclanthology.org/2021.emnlp-main.230`.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=Ai8Hw3AXqks.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=Ai8Hw3AXqks.

Jiajun Song and Yiqiao Zhong. Uncovering hidden geometry in transformers via disentangling position and context. *arXiv preprint arXiv:2310.04861*, 2023.

Jianlin Su. Scaling attention via the lens of entropy invariance, Dec 2021. URL https://spaces.ac.cn/archives/8823.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021a.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021b.

Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1433–1443, 2020.

Hiroyuki Takeda, Sina Farsiu, and Peyman Milanfar. Kernel regression for image processing and reconstruction. *IEEE Transactions on image processing*, 16(2):349–366, 2007.

The MosaicML NLP Team. Introducing mpt-7b: A new standard for open-source, commercially usable llms. *https://www.mosaicml.com/blog/mpt-30b*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4344–4353, Hong Kong, China, November 2019a. Association for Computational Linguistics. doi: 10.18653/v1/D19-1443. URL https://aclanthology.org/D19-1443.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4344–4353, Hong Kong, China, November 2019b. Association for Computational Linguistics.

Josef Valvoda, Naomi Saphra, Jonathan Rawski, Adina Williams, and Ryan Cotterell. Benchmarking compositionality with formal languages. In *Proceedings of the 29th International Conference on*

Computational Linguistics, pages 6007–6018, 2022.

Äaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Arxiv*, 2016. URL `https://arxiv.org/abs/1609.03499`.

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. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017a. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017b.

Ivan Vulić, Edoardo Maria Ponti, Robert Litschko, Goran Glavaš, and Anna Korhonen. Probing pretrained language models for lexical semantics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7222–7240, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.586. URL `https://aclanthology.org/2020.emnlp-main.586`.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=_VjQlMeSB_J`.

Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. Rnns are not transformers (yet): The key bottleneck on in-context retrieval. *arXiv preprint arXiv:2402.18510*, 2024.

Ulme Wennberg and Gustav Eje Henter. The case for translation-invariant self-attention in transformer-based language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 130–140, Online, August 2021. Association for Computational Linguistics.

Martin B Wilk and Ram Gnanadesikan. Probability plotting methods for the analysis for the analysis of data. *Biometrika*, 55(1):1–17, 1968.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi 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 Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

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*, pages 2059–2068, Online, June 2021a. Association for Computational Linguistics.

Jeff Wu, Long Ouyang, Daniel M Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. Recursively summarizing books with human feedback. *arXiv preprint arXiv:2109.10862*, 2021b.

Kan Wu, Houwen Peng, Minghao Chen, Jianlong Fu, and Hongyang Chao. Rethinking and improving relative position encoding for vision transformer. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10013–10021, 2021c.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, 2020.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nystöm-based algorithm for approximating self-attention. In *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, volume 35, page 14138. NIH Public Access, 2021.

Hang Yan, Bocao Deng, Xiaonan Li, and Xipeng Qiu. Tener: adapting transformer encoder for named entity recognition. *arXiv preprint arXiv:1911.04474*, 2019.

Baosong Yang, Longyue Wang, Derek F. Wong, Lidia S. Chao, and Zhaopeng Tu. Assessing the ability of self-attention networks to learn word order. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3635–3644, Florence, Italy, July 2019a. Association for Computational Linguistics. doi: 10.18653/v1/P19-1354. URL `https://aclanthology.org/P19-1354`.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. 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)*, pages 3770–3785, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.292. URL `https://aclanthology.org/2021.acl-long.292`.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. Adaptive Semiparametric Language Models. *Transactions of the Association for Computational Linguistics*, 9:362–373, 04

2021. ISSN 2307-387X. doi: 10.1162/tacl_a_00371. URL https://doi.org/10.1162/tacl_a_00371.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.07122.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *CoRR*, abs/2007.14062, 2020. URL https://arxiv.org/abs/2007.14062.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization, 2014. URL http://arxiv.org/abs/1409.2329. cite arxiv:1409.2329.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.