

YaRN: Efficient Context Window Extension of Large Language Models

Bowen Peng^{1*}Jeffrey Quesnelle^{1†}Honglu Fan²³Enrico Shippole[‡]¹Nous Research²EleutherAI³University of Geneva

Abstract

Rotary Position Embeddings (RoPE) have been shown to effectively encode positional information in transformer-based language models. However, these models fail to generalize past the sequence length they were trained on. We present YaRN (Yet another RoPE extension method), a compute-efficient method to extend the context window of such models, requiring **10x less tokens and 2.5x less training steps than previous methods**. Using YaRN, we show that LLaMA models can **effectively utilize and extrapolate to context lengths much longer than their original pre-training would allow**, while also surpassing previous the state-of-the-art at context window extension. In addition, we demonstrate that YaRN exhibits the capability to extrapolate beyond the limited context of a fine-tuning dataset. The models fine-tuned using YaRN has been made available and reproduced online up to 128k context length at <https://github.com/jquesnelle/yarn>.

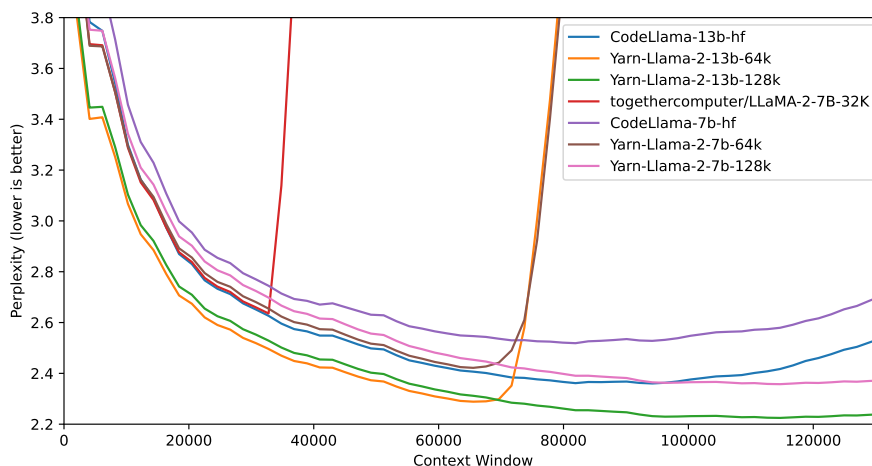


Figure 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

*Reddit: /u/bloc97 GitHub: bloc97

†Reddit: /u/emozilla X: @theemozilla GitHub: jquesnelle

‡X: @EnricoShippole GitHub: conceptofmind

1 Introduction

Transformer-based Large Language Models[40] (LLMs) have become the near-ubiquitous choice for many natural language processing (NLP) tasks where long-range abilities such as *in-context learning* (ICL) has been crucial. In performing the NLP tasks, the maximal length of the sequences (the *context window*) determined by its training processes has been one of the major limits of a pretrained LLM. Being able to dynamically extend the context window via a small amount of fine-tuning (or without fine-tuning) has become more and more desirable. To this end, the position encodings of transformers are the center of the discussions.

The original Transformer architecture used an absolute sinusoidal position encoding, which was later improved to a learnable absolute position encoding [15]. Since then, relative positional encoding schemes [32] have further increased the performance of Transformers. Currently, the most popular relative positional encodings are *T5 Relative Bias* [30], *RoPE* [34], *XPos* [35], and *ALiBi* [27].

One reoccurring limitation with positional encodings is the inability to generalize past the context window seen during training. While some methods such as *ALiBi* are able to do limited generalization, none are able to generalize to sequences significantly longer than their pre-trained length [22].

Some works have been done to overcome such limitation. [9] and concurrently [21] proposed to extend the context length by slightly modifying RoPE via Position Interpolation (PI) and fine-tuning on a small amount of data. As an alternative, [6] proposed the "NTK-aware" interpolation by taking the loss of high frequency into account. Since then, two improvements of the "NTK-aware" interpolation have been proposed, with different emphasis:

- the "Dynamic NTK" interpolation method [14] for pre-trained models without fine-tuning.
- the "NTK-by-parts" interpolation method [7] which performs the best when fine-tuned on a small amount of longer-context data.

The "NTK-aware" interpolation and the "Dynamic NTK" interpolation have already seen their presence in the open-source models such as *Code Llama* [31] (using "NTK-aware" interpolation) and *Qwen 7B* [2] (using "Dynamic NTK").

In this paper, in addition to making a complete account of the previous unpublished works on the "NTK-aware", the "Dynamic NTK" and the "NTK-by-part" interpolations, we present *YaRN* (Yet another RoPE extension method), an improved method to efficiently extend the context window of models trained with Rotary Position Embeddings (RoPE) including the LLaMA [38], the GPT-NeoX [5], and the PaLM [10] families of models.

YaRN reaches state-of-the-art performances in context window extensions after fine-tuning on less than $\sim 0.1\%$ of the original pre-training data. In the meantime, by combining with the inference-time technique called *Dynamic Scaling*, the *Dynamic-YaRN* allows for more than 2x context window extension without any fine-tuning.

2 Background and Related Work

2.1 Rotary Position Embeddings

The basis of our work is the Rotary Position Embedding (RoPE) introduced in [34]. We work on a hidden layer where the set of hidden neurons are denoted by D . Given a sequence of vectors $\mathbf{x}_1, \dots, \mathbf{x}_L \in \mathbb{R}^{|D|}$, following the notation of [34], the attention layer first converts the vectors into the query vectors and the key vectors:

$$\mathbf{q}_m = f_q(\mathbf{x}_m, m) \in \mathbb{R}^{|D|}, \mathbf{k}_n = f_k(\mathbf{x}_n, n) \in \mathbb{R}^{|D|}. \quad (1)$$

Next, the attention weights are calculated as

$$\text{softmax}\left(\frac{\mathbf{q}_m^T \mathbf{k}_n}{\sqrt{|D|}}\right), \quad (2)$$

where $\mathbf{q}_m, \mathbf{k}_n$ are considered as column vectors so that $\mathbf{q}_m^T \mathbf{k}_n$ is simply the Euclidean inner product. In RoPE, we first assume that $|D|$ is even and identify the embedding space and the hidden states as

complex vector spaces:

$$\mathbb{R}^{|D|} \cong \mathbb{C}^{|D|/2}$$

where the inner product $\mathbf{q}^T \mathbf{k}$ becomes the real part of the standard Hermitian inner product $\text{Re}(\mathbf{q}^* \mathbf{k})$. More specifically, the isomorphisms interleave the real part and the complex part

$$((\mathbf{x}_m)_1, \dots, (\mathbf{x}_m)_{|D|}) \mapsto ((\mathbf{x}_m)_1 + i(\mathbf{x}_m)_2, \dots, ((\mathbf{x}_m)_{|D|-1} + i(\mathbf{x}_m)_{|D|})), \quad (3)$$

$$((\mathbf{q}_m)_1, \dots, (\mathbf{q}_m)_{|D|}) \mapsto ((\mathbf{q}_m)_1 + i(\mathbf{q}_m)_2, \dots, ((\mathbf{q}_m)_{|D|-1} + i(\mathbf{q}_m)_{|D|})). \quad (4)$$

To convert embeddings $\mathbf{x}_m, \mathbf{x}_n$ into query and key vectors, we are first given \mathbb{R} -linear operators

$$\mathbf{W}_q, \mathbf{W}_k : \mathbb{R}^{|D|} \rightarrow \mathbb{R}^{|D|}.$$

In complex coordinates, the functions f_q, f_k are given by

$$f_q(\mathbf{x}_m, m) = e^{im\theta} \mathbf{W}_q \mathbf{x}_m, \quad f_k(\mathbf{x}_n, n) = e^{in\theta} \mathbf{W}_k \mathbf{x}_n, \quad (5)$$

where $\theta = \text{diag}(\theta_1, \dots, \theta_{|D|/2})$ is the diagonal matrix with $\theta_d = b^{-2d/|D|}$ and $b = 10000$. This way, RoPE associates each (complex-valued) hidden neuron with a separate frequency θ_d . The benefit of doing so is that the dot product between the query vector and the key vector only depends on the relative distance $m - n$ as follows


$$\langle f_q(\mathbf{x}_m, m), f_k(\mathbf{x}_n, n) \rangle_{\mathbb{R}} \quad (6)$$

$$= \text{Re}(\langle f_q(\mathbf{x}_m, m), f_k(\mathbf{x}_n, n) \rangle_{\mathbb{C}}) \quad (7)$$

$$= \text{Re}(\mathbf{x}_m^* \mathbf{W}_q^* \mathbf{W}_k \mathbf{x}_n e^{i\theta(m-n)}) \quad (8)$$

$$= g(\mathbf{x}_m, \mathbf{x}_n, m - n). \quad (9)$$

In real coordinates, the RoPE can be written using the following function



$$f_{\mathbf{W}}(\mathbf{x}_m, m, \theta_d) = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \dots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & \cos m\theta_l & -\sin m\theta_l \\ 0 & 0 & 0 & 0 & \dots & \sin m\theta_l & \cos m\theta_l \end{pmatrix} \mathbf{W} \mathbf{x}_m,$$

so that

$$f_q = f_{\mathbf{W}_q}, \quad f_k = f_{\mathbf{W}_k}.$$

2.2 Position Interpolation

As language models are usually pre-trained with a fixed context length, it is natural to ask how to extend the context length by fine-tuning on relatively less amount of data. For language models using RoPE as the position embedding, Chen et al. [9], and concurrently kaiokendev [21] proposed the Position Interpolation (PI) to extend the context length beyond the pre-trained limit. While a direct extrapolation does not perform well on sequences w_1, \dots, w_L with L larger than the pre-trained limit, they discovered that interpolating the position indicies within the pre-trained limit works well with the help of a small amount of fine-tuning. Specifically, given a pre-trained language model with RoPE, they modify the RoPE by

$$f'_{\mathbf{W}}(\mathbf{x}_m, m, \theta_d) = f_{\mathbf{W}}\left(\mathbf{x}_m, \frac{mL}{L'}, \theta_d\right), \quad (10)$$

where $L' > L$ is a new context window beyond the pre-trained limit. With the original pre-trained model plus the modified RoPE formula, they fine-tuned the language model further on several orders of magnitude fewer tokens (a few billion in Chen et al. [9]) and successfully achieved context window extension.

2.3 Additional Notation

The ratio between the extended context length and the original context length has been of special importance, and we introduce the notation s defined by

$$s = \frac{L'}{L}, \quad (11)$$

and we call s the *scale factor*.

We also rewrite and simplify Eq. 10 into the following general form:

$$f'_{\mathbf{W}}(\mathbf{x}_m, m, \theta_d) = f_{\mathbf{W}}(\mathbf{x}_m, g(m), h(\theta_d)), \quad (12)$$

where $g(m), h(\theta_d)$ are method-dependent functions. For PI, we have $g(m) = m/s, h(\theta_d) = \theta_d$. In the subsequent sections, when we introduce a new interpolation method, we sometimes only specify the functions $g(m)$ and $h(\theta_d)$.

Additionally, we define λ_d as the *wavelength* of the RoPE embedding at d -th hidden dimension:

★ Wavelength \rightarrow dependent and defined w.r.t d

$$\lambda_d = \frac{2\pi}{\theta_d} = 2\pi b^{\frac{2d}{|D|}}. \quad (13)$$

The wavelength describes the length of tokens needed in order for the RoPE embedding at dimension d to perform a full rotation (2π).

Given that some interpolation methods (eg. PI) do not care about the wavelength of the dimensions, we will refer to those methods as "blind" interpolation methods, while others do (eg. YaRN), which we will classify as "targeted" interpolation methods.

Blind \rightarrow Interpolation where wavelength is not changed

2.4 Related work

ReRoPE [33] also aims to extend the context size of existing models pre-trained with RoPE, and claims "infinite" context length without needing any fine-tuning. This claim is backed by a monotonically decreasing loss with increasing context length up to 16k on the Llama 2 13B model. It achieves context extension by modifying the attention mechanism and thus is not purely an embedding interpolation method. Since it is currently not compatible with Flash Attention 2 [13] and requires two attention passes during inference, we do not consider it for comparison.

Concurrently with our work, LM-Infinite [16] proposes similar ideas to YaRN, but focuses on "on-the-fly" length generalization for non-fine-tuned models. Since they also modify the attention mechanism of the models, it is not an embedding interpolation method and is not immediately compatible with Flash Attention 2.

3 Methodology

Whereas PI stretches all RoPE dimensions equally, we find that the theoretical interpolation bound described by PI [9] is insufficient at predicting the complex dynamics between RoPE and the LLM's internal embeddings. In the following subsections, we describe the main issues with PI we have individually identified and solved, so as to give the readers the context, origin and justifications of each method which we use in concert to obtain the full YaRN method.

3.1 Loss of High Frequency information - "NTK-aware" interpolation

If we look at RoPE only from an information encoding perspective, it was shown in [36], using Neural Tangent Kernel (NTK) theory, that deep neural networks have trouble learning high frequency information if the input dimension is low and the corresponding embeddings lack high frequency components. Here we can see the similarities: a token's positional information is one-dimensional, and RoPE expands it to an n -dimensional complex vector embedding.

★

RoPE closely resembles Fourier Features [36] in many aspects, as it is possible to define RoPE as a special 1D case of a Fourier Feature. Stretching the RoPE embeddings indiscriminately results in the loss of important high frequency details which the network needs in order to resolve tokens that are both very similar and very close together (the rotation describing the smallest distance needs to not be too small for the network to be able to detect it).

We hypothesise that the slight increase of perplexity for short context sizes after fine-tuning on larger context sizes seen in PI [9] might be related to this problem. Under ideal circumstances, there is no reason that fine-tuning on larger context sizes should degrade the performance of smaller context sizes.

In order to resolve the problem of losing high frequency information when interpolating the RoPE embeddings, the "NTK-aware" interpolation was developed in [6]. Instead of scaling every dimension of RoPE equally by a factor s , we spread out the interpolation pressure across multiple dimensions by scaling high frequencies less and low frequencies more. One can obtain such a transformation in many ways, but the simplest would be to perform a base change on the value of θ .

More precisely, following the notations set out in Section 2.3, we define the "NTK-aware" interpolation scheme as follows (see the Appendix A.1 for the details of the deduction).

Definition 1 The "NTK-aware" interpolation is a modification of RoPE by using Eq. 12 with the following functions.

$$g(m) = m \quad (14)$$

$$h(\theta_d) = b'^{-2d/|D|} \quad (15)$$

where

$$b' = b \cdot s^{\frac{|D|}{|D|-2}} \quad (16)$$

Given the results from [6], this method performs much better at extending the context size of non-fine-tuned models compared to PI [9]. However, one major disadvantage of this method is that given it is not just an interpolation scheme, some dimensions are slightly extrapolated to "out-of-bound" values, thus fine-tuning with "NTK-aware" interpolation [6] yields inferior results to PI [9]. Furthermore, due to the "out-of-bound" values, the theoretical scale factor s does not accurately describe the true context extension scale. In practice, the scale value s has to be set higher than the expected scale for a given context length extension.

We note that shortly before the release of this article, Code Llama [31] was released and uses "NTK-aware" scaling by manually scaling the base b to $1M$.

3.2 Loss of Relative Local Distances - "NTK-by-parts" interpolation

In the case of blind interpolation methods like PI and "NTK-aware" interpolation, we treat all the RoPE hidden dimensions equally (as in they have the same effect on the network). However, there are strong clues that point us towards the need for targeted interpolation methods.

In this section, we think heavily in terms of the wavelengths λ_d defined in Eq. 13 in the formula of RoPE. For simplicity, we omit the subscript d in λ_d and the reader is encouraged to think about λ as the wavelength of an arbitrary periodic function.

One interesting observation of RoPE embeddings is that given a context size L , there are some dimensions d where the wavelength is longer than the maximum context length seen during pretraining ($\lambda > L$), this suggests that some dimensions' embeddings might not be distributed evenly in the rotational domain. In such cases, we presume having all unique position pairs implies that the absolute positional information remains intact. On the contrary, when the wavelength is short, only relative positional information is accessible to the network.

Moreover, when we stretch all the RoPE dimensions either by a scale s or using a base change b' , all tokens become closer to each other, as the dot product of two vectors rotated by a lesser amount is bigger. This scaling severely impairs a LLM's ability to understand small and local relationships between its internal embeddings. We hypothesize that such compression leads to the model being confused on the positional order of close-by tokens, and consequently harming the model's abilities.

In order to remedy this issue, given the two previous observations that we have found, we choose not to interpolate the higher frequency dimensions at all while always interpolating the lower frequency dimensions. In particular,

- if the wavelength λ is much smaller than the context size L , we do not interpolate;
- if the wavelength λ is equal to or bigger than the context size L , we want to only interpolate and avoid any extrapolation (unlike the previous "NTK-aware" method);
- dimensions in-between can have a bit of both, similar to the "NTK-aware" interpolation.

As a result, it is more convenient to introduce the ratio $r = \frac{L}{\lambda}$ between the original context size L and the wavelength λ . In the d -th hidden state, the ratio r depends on d in the following way:

$$r(d) = \frac{L}{\lambda_d} = \frac{L}{2\pi b' \frac{2d}{|D|}}$$

Using b' \Rightarrow extension of NTK aware

In order to define the boundary of the different interpolation strategies as above, we introduce two extra parameters α, β . All hidden dimensions d where $r(d) < \alpha$ are those where we linearly interpolate by a scale s (exactly like PI, avoiding any extrapolation), and the d where $r(d) > \beta$ are those where we do not interpolate at all. Define the ramp function γ to be

$$\gamma(r) = \begin{cases} 0, & \text{if } r < \alpha \\ 1, & \text{if } r > \beta \\ \frac{r - \alpha}{\beta - \alpha}, & \text{otherwise.} \end{cases}$$

$\lambda_d \rightarrow 0 \rightarrow r \uparrow \rightarrow 1$
 $\lambda_d \rightarrow L \rightarrow r \downarrow \rightarrow 0$
 $\frac{\partial d}{s}$

With the help of the ramp function, the "NTK-by-parts" method can be described as follows.

Definition 2 The "NTK-by-parts" interpolation is a modification of RoPE by using Eq. 12 with the following functions⁴.

$$g(m) = m \quad (19)$$

$$h(\theta_d) = \left(1 - \gamma(r(d))\right) \frac{\theta_d}{s} + \gamma(r(d)) \theta_d. \quad (20)$$

The values of α and β should be tuned on a case-by-case basis. For example, we have found experimentally that for the Llama family of models, good values for α and β are $\alpha = 1$ and $\beta = 32$.

Using the techniques described in this section, a variant of the resulting method was released under the name "NTK-by-parts" interpolation [7]. This improved method performs better than the previous PI [9] and "NTK-aware" 3.1 interpolation methods, both with non-fine-tuned models and with fine-tuned models, as shown in [7].

3.3 Dynamic Scaling - "Dynamic NTK" interpolation

In a lot of use cases, multiple forward-passes are performed with varying sequence lengths from 1 to the maximal context size. A typical example is the autoregressive generation where the sequence lengths increment by 1 after each step. There are two ways of applying an interpolation method that uses a scale factor s (including PI, "NTK-aware" and "NTK-by-parts"):

1. Throughout the whole inference cycle, the embedding layer is fixed including the scale factor $s = L'/L$ where L' is the fixed number of extended context size.
2. In each forward-pass, the position embedding updates the scale factor $s = \max(1, l'/L)$ where l' is the sequence length of the current sequence.

The problem of (1) is that the model may experience a performance discount at a length less than L and an abrupt degradation when the sequence length is longer than L' . But by doing Dynamic

⁴The interpolation by linear ramp on h may have alternatives, such as a harmonic mean over θ_d/s and θ_d converted from a linear interpolation on wavelengths. The choice of h here was for the simplicity of implementation, but both would work.

Better than PI and NTK aware both with and without finetune.

normal PI issue

Known \rightarrow Short content length issue

Scaling as (2), it allows the model to gracefully degrade instead of immediately breaking when hitting the trained context limit L' . We call this inference-time method the Dynamic Scaling method. When it is combined with "NTK-awared" interpolation, we call it "Dynamic NTK" interpolation. It first appeared in public as a reddit post in [14].

One notable fact is that the "Dynamic NTK" interpolation works exceptionally well on models pre-trained on L without any finetuning ($L' = L$). This is supported by the experiment in Appendix B.3.

Often in the repeated forward-passes, the kv-caching [8] is applied so that we can reuse the previous key-value vectors and improve the overall efficiency. We point out that in some implementations when the RoPE embeddings are cached, some care has to be taken in order to modify it for Dynamic Scaling with kv-caching. The correct implementation should cache the kv-embeddings before applying RoPE, as the RoPE embedding of every token changes when s changes.

3.4 YaRN

In addition to the previous interpolation techniques, we also observe that introducing a temperature t on the logits before the attention softmax has a uniform impact on perplexity regardless of the data sample and the token position over the extended context window (See Appendix A.2). More precisely, instead of Eq. 2, we modify the computation of attention weights into

$$\text{softmax} \left(\frac{\mathbf{q}_m^T \mathbf{k}_n}{t \sqrt{|D|}} \right). \quad (21)$$

The reparametrization of RoPE as a set of 2D matrices has a clear benefit on the implementation of this attention scaling: we can instead use a "length scaling" trick which scales both \mathbf{q}_m and \mathbf{k}_n by a constant factor $\sqrt{1/t}$ by simply scaling the complex RoPE embeddings by the same amount. With this, YaRN can effectively alter the attention mechanism without modifying its code. Furthermore, it has zero overhead during both inference and training, as RoPE embeddings are generated in advance and are reused for all forward passes. Combining it with the "NTK-by-parts" interpolation, we have the YaRN method.

Definition 3 By the "YaRN method", we refer to a combination of the attention scaling in Eq. 21 and the "NTK-by-parts" interpolation introduced in Section 3.2.

For LLaMA and Llama 2 models, we recommend the following values:

$$\sqrt{\frac{1}{t}} = 0.1 \ln(s) + 1. \quad (22)$$

The equation above is found by fitting $\sqrt{1/t}$ at the lowest perplexity against the scale extension by various factors s using the "NTK-by-parts" method (Section 3.2) on LLaMA 7b, 13b, 33b and 65b models without fine-tuning. We note that the same values of t also apply fairly well to Llama 2 models (7b, 13b and 70b). It suggests that the property of increased entropy and the temperature constant t may have certain degree of "universality" and may be generalizable across some models and training data.

The YaRN method combines all our findings and surpasses all previous methods in both fine-tuned and non-fine-tuned scenarios. Thanks to its low footprint, YaRN allows for direct compatibility with libraries that modify the attention mechanism such as Flash Attention 2 [13].

4 Experiments

We show that YaRN successfully achieves context window extension of language models using RoPE as its position embedding. Moreover, this result is achieved with only 400 training steps, representing approximately 0.1% of the model's original pre-training corpus, a 10x reduction from Rozière et al. [31] and 2.5x reduction in training steps from Chen et al. [9], making it highly compute-efficient for training with no additional inference costs. We calculate the perplexity of long documents and score

on established benchmarks to evaluate the resulting models, finding that **they surpass all other context window extension methods**.

We broadly followed the training and evaluation procedures as outlined in [9].

4.1 Training

For training, we extended the Llama 2 [39] 7B and 13B parameter models. No changes were made to the LLaMA model architecture other than the calculation of the embedding frequencies as described in 3.4 with $s = 16$ and $s = 32$.

We used a learning rate of 2×10^{-5} with **no weight decay** and a **linear warmup of 20 steps** along with AdamW [24] $\beta_1 = 0.9$ and $\beta_2 = 0.95$. For $s = 16$ we fine-tuned for 400 steps with global batch size 64 using PyTorch [26] Fully Sharded Data Parallelism [42] and Flash Attention 2 [13] on the PG19 dataset [29] chunked into 64k segments bookended with the BOS and EOS token. For $s = 32$ we followed the same procedure, but started from the finished $s = 16$ checkpoint and trained for an additional 200 steps.

*Incremental improvement
For $s = 32$, start with $s = 16$*

4.2 Extrapolation and Transfer Learning

In Code Llama [31], a dataset with 16k context was used with a scale factor set to $s \approx 88.6$, which corresponds to a context size of 355k. They show that the **network extrapolates up to 100k context without ever seeing those context sizes during training**. Similar to 3.1 and Rozière et al. [31], YaRN also supports training with a higher scale factor s than the length of the dataset. Due to compute constraints, we test only $s = 32$ by further fine-tuning the $s = 16$ model for 200 steps using the same dataset with 64k context.

Since orig = 4096

We show in 4.3.1 that the $s = 32$ model successfully extrapolates up to 128k context using only 64k context during training. Unlike previous "blind" interpolation methods, **YaRN is much more efficient at transfer learning when increasing the scale s** . This demonstrates successful transfer learning from $s = 16$ to $s = 32$ **without the network needing to relearn the interpolated embeddings**, as the $s = 32$ model is equivalent to the $s = 16$ model across the entire context size, despite only being trained on $s = 32$ for 200 steps.

4.3 Evaluation

The evaluations focus on three aspects:

1. the perplexity scores of fine-tuned models with extended context window,
2. the passkey retrieval task on fine-tuned models,
3. the common LLM benchmark results of fine-tuned models,

4.3.1 Long Sequence Language Modeling

To evaluate the long sequence language modeling performances, we use the **GovReport [18]** and **Proof-pile [4]** datasets both of which contain many long sequence samples. For all evaluations, the test splits of both datasets were used exclusively. All **perplexity evaluations** were calculated using the **sliding window method** from Press et al. [27] with $S = 256$.

Firstly, we evaluated how the model performed as the context window increased. We selected 10 random samples from Proof-pile with at least 128k tokens each and evaluated the perplexity of each of these samples when truncated at 2k steps from a sequence length of 2k tokens through 128k tokens.

Evaluate sliding window ppl from 2k to 128k

Table 1 shows a side-by-side comparison of Llama-2 model extended from 4096 to 8192 context length via PI (LLongMA-2 7b⁵), "NTK-aware" and YaRN. Note that PI and "NTK-aware" models were trained using the methodology in Chen et al. [9], while **YaRN used the same methodology but 2.5x less training steps and data**, as described in 4.

⁵LLongMA-2 7b [28] is fine-tuned from Llama-2 7b, trained at 8k context length with PI using the RedPajama dataset [12].

Extension Method	Trained Tokens	Context Window	Evaluation Context Window Size				
			2048	4096	6144	8192	10240
PI ($s = 2$)	1B	8k	3.92	3.51	3.51	3.34	8.07
NTK ($\theta = 20k$)	1B	8k	4.20	3.75	3.74	3.59	6.24
YaRN ($s = 2$)	400M	8k	3.91	3.50	3.51	3.35	6.04

Table 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents over Llama-2 extended via PI, NTK and YaRN

We further evaluated YaRN at the scale factor $s = 16, 32$ and compared them against a few open-source models fine-tuned from Llama-2 and extended to more than 32k context window such as Together.ai [37] and "NTK-aware" Code Llama [31]. The results are summarized in Table 2 (with a more detailed plot in Figure 1).

Model Size	Model Name	Context Window	Extension Method	Evaluation Context Window Size				
				8192	32768	65536	98304	131072
7B	Together	32k	PI	3.50	2.64	$> 10^2$	$> 10^3$	$> 10^4$
7B	Code Llama	100k	NTK	3.71	2.74	2.55	2.54	2.71
7B	YaRN ($s = 16$)	64k	YaRN	3.51	2.65	2.42	$> 10^1$	$> 10^1$
7B	YaRN ($s = 32$)	128k	YaRN	3.56	2.70	2.45	2.36	2.37
13B	Code Llama	100k	NTK	3.54	2.63	2.41	2.37	2.54
13B	YaRN ($s = 16$)	64k	YaRN	3.25	2.50	2.29	$> 10^1$	$> 10^1$
13B	YaRN ($s = 32$)	128k	YaRN	3.29	2.53	2.31	2.23	2.24

Table 2: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

We observe that the model exhibits strong performance across the entire targeted context size, with YaRN interpolation being the first method to successfully extend the effective context size of Llama 2 to 128k. Of particular note are the YaRN ($s = 32$) models, which show continued declining perplexity through 128k, despite the fine-tuning data being limited to 64k tokens in length, demonstrating that the model is able to generalize to unseen context lengths.

Furthermore, in Appendix B.1, we show the results of the average perplexity on 50 untruncated GovReport documents with at least 16k tokens per sample evaluated on the setting of 32k maximal context window without Dynamic Scaling in Table 4. Similar to the Proof-pile results, the GovReport results show that fine-tuning with YaRN achieves good performance on long sequences.

4.3.2 Passkey Retrieval

The passkey retrieval task as defined in [25] measures a model’s ability to retrieve a simple passkey (i.e., a five-digit number) from amongst a large amount of otherwise meaningless text. For our evaluation of the models, we performed 10 iterations of the passkey retrieval task with the passkey placed at a random location uniformly distributed across the evaluation context window on different context window sizes ranging from 8k to 128k. Both 7b and 13b models fine-tuned using YaRN at 128k context size passes the passkey retrieval task with very high accuracy ($> 99\%$) within the entire context window size. We show detailed results in Appendix B.2.

4.3.3 Standardized Benchmarks

The Hugging Face Open LLM Leaderboard [19] compares a multitude of LLMs across a standardized set of four public benchmarks. Specifically, we use 25-shot ARC-Challenge [11], 10-shot HellaSwag [41], 5-shot MMLU [17], and 0-shot TruthfulQA [23].

To test the degradation of model performance under context extension, we evaluated our models using this suite and compared it to established scores for the Llama 2 baselines as well as publicly available PI and "NTK-aware" models. The results are summarized in Table 3.

Model Size	Model Name	Context Window	Extension Method	ARC-c	Hellaswag	MMLU	TruthfulQA
7B	Llama 2	4k	None	53.1	77.8	43.8	39.0
7B	Together	32k	PI	47.6	76.1	43.3	39.2
7B	Code Llama	100k	NTK	39.9	60.8	31.1	37.8
7B	YaRN ($s = 16$)	64k	YaRN	52.3	78.8	42.5	38.2
7B	YaRN ($s = 32$)	128k	YaRN	52.1	78.4	41.7	37.3
13B	Llama 2	4k	None	59.4	82.1	55.8	37.4
13B	Code Llama	100k	NTK	40.9	63.4	32.8	43.8
13B	YaRN ($s = 16$)	64k	YaRN	58.1	82.3	52.8	37.8
13B	YaRN ($s = 32$)	128k	YaRN	58.0	82.2	51.9	37.3

Table 3: Performance of context window extensions methods on the Hugging Face Open LLM benchmark suite compared with original Llama 2 baselines

We observe that there is minimal performance degradation between the YaRN models and their respective Llama 2 baselines. We also observe that there was on average a 0.49% drop in scores between the YaRN $s = 16$ and $s = 32$ models. From this we conclude that the the iterative extension from 64k to 128k results in negligible performance loss.

5 Conclusion

In conclusion, we have shown that YaRN improves upon all existing RoPE interpolation methods and can act as a drop-in replacement to PI, with no downsides and minimal implementation effort. The fine-tuned models preserve their original abilities on multiple benchmarks while being able to attend to a very large context size. Furthermore, YaRN allows efficient extrapolation with fine-tuning on shorter datasets and can take advantage of transfer learning for faster convergence, both of which are crucial under compute-constrained scenarios. Finally, we have shown the effectiveness of extrapolation with YaRN where it is able to "train short, and test long".

6 Reproducibility

To aid in reproducibility, we provide, as supplementary material, the entirety of the code used to train the YaRN models in Table 2, as well as the evaluation code that produced Figure 1 and Tables 1, 2, 3, 4, and 5. The code also contains implementations of various extension methods referenced throughout the paper. For training YaRN, we used the publicly available PG19 dataset [29] tokenized to 64k tokens.

References

- [1] Mistralite. URL <https://huggingface.co/amazon/MistralLite>.
- [2] Introducing Qwen-7B: Open foundation and human-aligned models (of the state-of-the-arts). URL https://github.com/QwenLM/Qwen-7B/blob/main/tech_memo.md.
- [3] Long-data collections. URL <https://huggingface.co/datasets/togethercomputer/Long-Data-Collections>.
- [4] Z. Azerbayev, E. Ayers, , and B. Piotrowski. Proof-pile, 2022. URL <https://github.com/zhangir-azerbayev/proof-pile>.
- [5] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang, M. Pieler, U. S. Prashanth, S. Purohit, L. Reynolds, J. Tow, B. Wang, and S. Weinbach. GPT-NeoX-20B: An open-source autoregressive language model, 2022. arXiv: 2204.06745.
- [6] bloc97. NTK-Aware Scaled RoPE allows LLaMA models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation., 2023. URL https://www.reddit.com/r/LocalLLaMA/comments/141z7j5/ntkaware_scaled_rope_allows_llama_models_to_have/.
- [7] bloc97. Add NTK-Aware interpolation "by parts" correction, 2023. URL <https://github.com/jquesnelle/scaled-rope/pull/1>.
- [8] C. Chen. Transformer Inference Arithmetic, 2022. URL <https://kipp.ly/blog/transformer-inference-arithmetic/>.
- [9] S. Chen, S. Wong, L. Chen, and Y. Tian. Extending context window of large language models via positional interpolation, 2023. arXiv: 2306.15595.
- [10] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. PaLM: Scaling language modeling with pathways, 2022. arXiv: 2204.02311.
- [11] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try ARC, the AI2 Reasoning Challenge, 2018. arXiv: 1803.05457.
- [12] T. Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023. URL <https://github.com/togethercomputer/RedPajama-Data>.
- [13] T. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. arXiv: 2307.08691.

- [14] emozilla. Dynamically Scaled RoPE further increases performance of long context LLaMA with zero fine-tuning, 2023. URL https://www.reddit.com/r/LocalLLaMA/comments/14mrgpr/dynamically_scaled_rope_further_increases/.
- [15] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning, 2017. arXiv: 1705.03122.
- [16] C. Han, Q. Wang, W. Xiong, Y. Chen, H. Ji, and S. Wang. LM-Infinite: Simple on-the-fly length generalization for large language models, 2023. arXiv: 2308.16137.
- [17] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [18] L. Huang, S. Cao, N. Parulian, H. Ji, and L. Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436. Association for Computational Linguistics, June 2021.
- [19] Hugging Face. Open LLM Leaderboard, 2023. URL https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.
- [20] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.
- [21] kaiokendev. Things I’m learning while training superhot., 2023. URL <https://kaiokendev.github.io/til#extending-context-to-8k>.
- [22] A. Kazemnejad, I. Padhi, K. N. Ramamurthy, P. Das, and S. Reddy. The impact of positional encoding on length generalization in transformers, 2023. arXiv: 2305.19466.
- [23] S. Lin, J. Hilton, and O. Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, May 2022.
- [24] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.
- [25] A. Mohtashami and M. Jaggi. Landmark attention: Random-access infinite context length for transformers, 2023. arXiv: 2305.16300.
- [26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019.
- [27] O. Press, N. Smith, and M. Lewis. Train Short, Test Long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022.
- [28] J. Quesnelle, E. Shippole, and "Kaiokendev". Llongma: Scaling rotary embeddings through linear positional interpolation. <https://huggingface.co/conceptofmind/LLongMA-2-7b/>, 2023.
- [29] J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.
- [30] A. Roberts, C. Raffel, K. Lee, M. Matena, N. Shazeer, P. J. Liu, S. Narang, W. Li, and Y. Zhou. Exploring the limits of transfer learning with a unified text-to-text transformer. Technical report, Google, 2019.

- [31] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code Llama: Open foundation models for code, 2023. arXiv: 2308.12950.
- [32] P. Shaw, J. Uszkoreit, and A. 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.
- [33] J. Su. Rectified rotary position embeddings. <https://github.com/bojone/rerope>, 2023.
- [34] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. RoFormer: Enhanced transformer with rotary position embedding, 2022. arXiv: 2104.09864.
- [35] Y. Sun, L. Dong, B. Patra, S. Ma, S. Huang, A. Benhaim, V. Chaudhary, X. Song, and F. Wei. A length-extrapolatable transformer, 2022. arXiv: 2212.10554.
- [36] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS’20*, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.
- [37] Together.ai. LLaMA-2-7B-32K, 2023. URL <https://huggingface.co/togethercomputer/LLaMA-2-7B-32K>.
- [38] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. LLaMA: Open and efficient foundation language models, 2023. arXiv: 2302.13971.
- [39] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.
- [40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [41] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.
- [42] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, B. Nguyen, G. Chauhan, Y. Hao, and S. Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel, 2023. arXiv: 2304.11277.

A Additional details on interpolation methods

A.1 Short notes on the deduction of "NTK-aware" interpolation

In Section 3.1, we introduce a change of basis from b to b' in the definition of "NTK-aware" interpolation method. Here is a short note on its mathematical deduction.

Recall that our goal is to spread out the interpolation pressure across the hidden dimensions using a base-change instead of scaling the frequencies by a fixed factor s . The property we want to guarantee is that: The lowest frequency needs to be scaled as much as linear positional scaling and the highest frequency to stay constant.

We introduce a new base b' such that the last dimension matches the wavelength of linear interpolation with a scale factor s . Since the original RoPE method skips odd dimensions in order to concatenate both $\cos(\frac{2\pi x}{\lambda})$ and $\sin(\frac{2\pi x}{\lambda})$ components into a single embedding, the last dimension $d \in D$ is $|D| - 2$.

The new base b' can be chosen so that

$$b'^{\frac{|D|-2}{|D|}} = s \cdot b^{\frac{|D|-2}{|D|}}. \quad (23)$$

Solving for b' yields

$$b' = b \cdot s^{\frac{|D|}{|D|-2}}. \quad (24)$$

A.2 The impact of pre-softmax scaling of YaRN on perplexity

In Section 3.4, we mention the impact of the factor t inside the softmax computation of attention weights. Here we fix 896 16k-token documents from RedPajama [12]⁶, and calculate their perplexity scores with different scaling $1/\sqrt{t}$. The result is in Figure 2. For comparison, recall that our recommended factor in this case ($s = 8$) is given by the following.

$$\sqrt{\frac{1}{t}} = 0.1 \ln(s) + 1 \approx 1.208. \quad (25)$$

Goal → Make new lowest freq (using b') equal to what old lowest freq would be under PI

New freq at last dimension using base b'
 $\Theta'_{\text{last}} = (b')^{-2d_{\text{last}}/|D|}$

Old freq at last dimension under simple PI would be scaled by $1/s$

$$\Theta_{\text{last}} = \frac{\Theta_{\text{last}}}{s} = \frac{b^{-2d_{\text{last}}/|D|}}{s}$$

$$\Rightarrow \frac{b^{-2d_{\text{last}}/|D|}}{s} = b'^{-2d_{\text{last}}/|D|}$$

$$\text{And } d_{\text{last}} = |D| - 2$$

$$\Rightarrow b'^{2d_{\text{last}}/|D|} = s \cdot b^{2d_{\text{last}}/|D|}$$

⁶We choose RedPajama because it is the open-source dataset closest to the training dataset of LLaMA as far as we are aware of.

To show the impact of the factor $1/\sqrt{t}$ on different token positions, we cut each 16k-token document into chunks of 2048 tokens, and further plot the mean perplexity change comparing to $t = 1$ in percentages

$$\frac{\text{ppl}(t) - \text{ppl}(t = 1)}{\text{ppl}(t = 1)} \quad (26)$$

of each chunk. The plot is shown in Figure 3.

To further demonstrate the best values of t across all samples over different token positions, we plot the sample counts with minimal perplexity at a given $1/\sqrt{t}$ for each of the 8 position segments over the 16k-token range in Figure 4.

We observe that:

- for a suitable t , a sample may obtain better perplexity scores across the extended context window;
- the best value of t is mostly consistent across different samples and different positions.

We remark that this finding is consistent for different values of s and the best value of t follows our recommended formula (Eq. 22) closely.

B Additional tables and charts

B.1 GovReport evaluations

In Section 4.3.1, we mention the evaluation on GovReport documents. The evaluation results are detailed in Table 4 below.

B.2 Passkey Retrieval

Here we can observe that the lowest perplexity point alone does not provide a comprehensive depiction on the "effective context size" that an LLM can attend to. While the Code Llama 13b model exhibits increasing perplexity above 100k context lengths, it was still able to accurately retrieve the passkey at a context length of 128k. This suggest that while the output of Code Llama might start to degrade in quality above 100k context size, it is still able to maintain strong retrieval capabilities.

In addition, as YaRN with $s = 32$ was trained for 200 more steps than YaRN with $s = 16$ while having a higher passkey accuracy with similar perplexity, we hypothesize that perplexity may not be a great indicator of whether an LLM is able to attend to all tokens and does not exhaustively determine long context performance. This also suggests that the YaRN models with $s = 16$ might be relatively undertrained for the passkey retrieval task.

B.3 Dynamic scaling on models without any fine-tuning

We first recall from Section 3.3 that the Dynamic Scaling technique is an inference-time technique that dynamically update the factor s in interpolation methods such as PI, "NTK-by-parts" and YaRN. We choose the original Llama 2, fix a sample in GovReport and calculate its perplexity on a sliding window of 256 tokens using RoPE, Dynamic-PI and Dynamic-YaRN. Since the original maximal context length of Llama 2 is 4096, we observe that Dynamic Scaling effectively extend the inference length and Dynamic-YaRN achieves better performance than Dynamic-PI. The resulting chart is in Figure 5.

We see that

- Dynamic Scaling effectively prevents the blow-up of perplexity score beyond pretrained context window;
- Dynamic-YaRN outperforms Dynamic-PI in terms of long-range perplexity on pretrained Llama-2 without any finetuning.

attention pre-softmax scaling vs final perplexity over 56 16k-token documents
($s=8$)

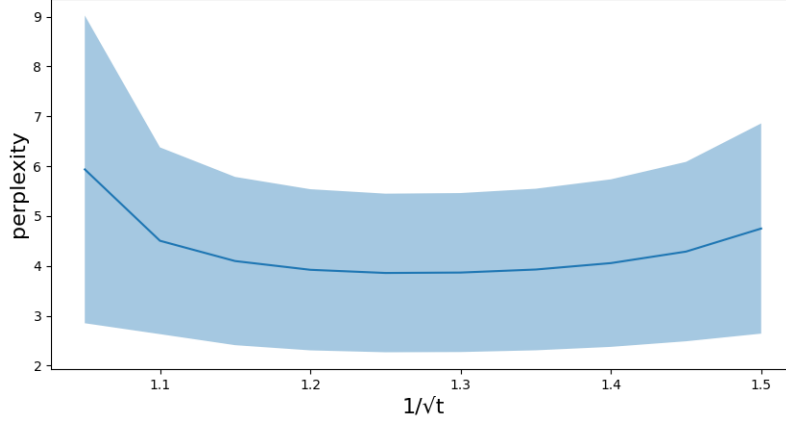


Figure 2: Fix $s = 8$, compare the LLaMA 7b perplexity on 896 16k-token documents over different scaling $1/\sqrt{t}$. The shaded area represents 1 standard deviation (68%).

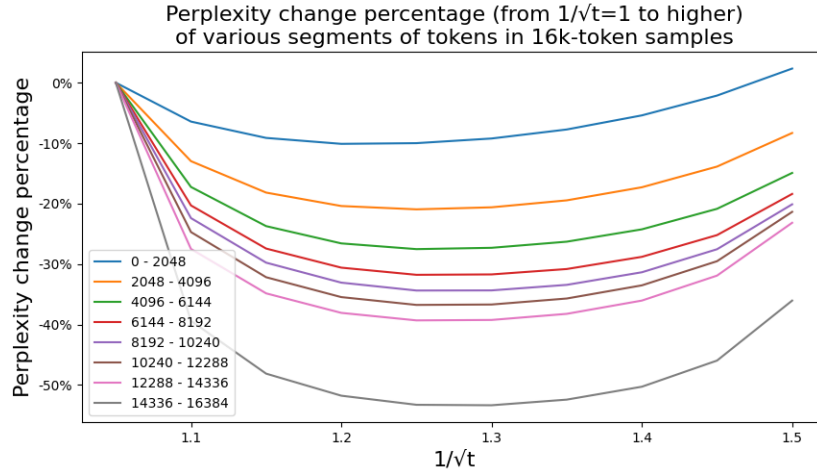


Figure 3: Fix $s = 8$, compare the mean of perplexity change percentages $\frac{\text{ppl}(t) - \text{ppl}(t=1)}{\text{ppl}(t=1)}$ at different segments of token positions on 896 16k-token documents over different scaling $1/\sqrt{t}$.

Model Size	Model Name	Context Window	Extension Method	Perplexity
7B	Together	32k	PI	3.67
7B	Code Llama	100k	NTK	4.44
7B	YaRN ($s = 16$)	64k	YaRN	3.59
7B	YaRN ($s = 32$)	128k	YaRN	3.64
13B	Code Llama	100k	NTK	4.22
13B	YaRN ($s = 16$)	64k	YaRN	3.35
13B	YaRN ($s = 32$)	128k	YaRN	3.39

Table 4: Sliding window perplexity ($S = 256$) of 50 long GovReport documents with a fixed context window size of 32k

The sample counts with minimal perplexity at a given $1/\sqrt{t}$
(different lines represent different position segments at the 16k-token samples)

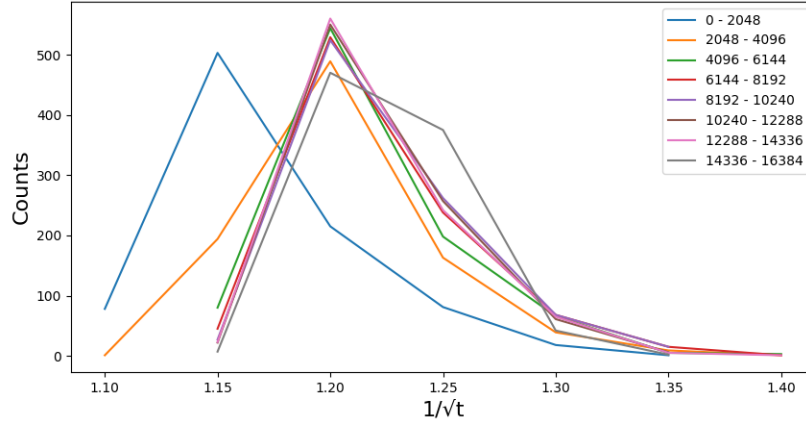


Figure 4: The sample counts (out of the 896 samples) with minimal perplexity at a given $1/\sqrt{t}$ for a given segment of token positions over the 16k-token range.

Model Size	Model Name	Scaling Factor (s)	Context Window	Training Data Context	Extension Method	Passkey Context	Passkey Accuracy
7B	Together	4	32k	32k	PI	32k	100%
7B	Code Llama	88.6	100k	16k	NTK	112k	94.3%
7B	YaRN	16	64k	64k	YaRN	64k	96.3%
7B	YaRN	32	128k	64k	YaRN	128k	99.4%
13B	Code Llama	88.6	100k	16k	NTK	128k	99.4%
13B	YaRN	16	64k	64k	YaRN	64k	97.5%
13B	YaRN	32	128k	64k	YaRN	128k	99.4%

Table 5: Passkey retrieval performance of various models. The passkey context denotes the maximum tested context window size where the accuracy of passkey retrieval was $\geq 80\%$, and the passkey accuracy is the average accuracy of passkey retrieval on all context sizes tested that were smaller or equal than the passkey context size.

The comparison between RoPE, Dynamic-PI and Dynamic-YaRN on Llama-2 7b

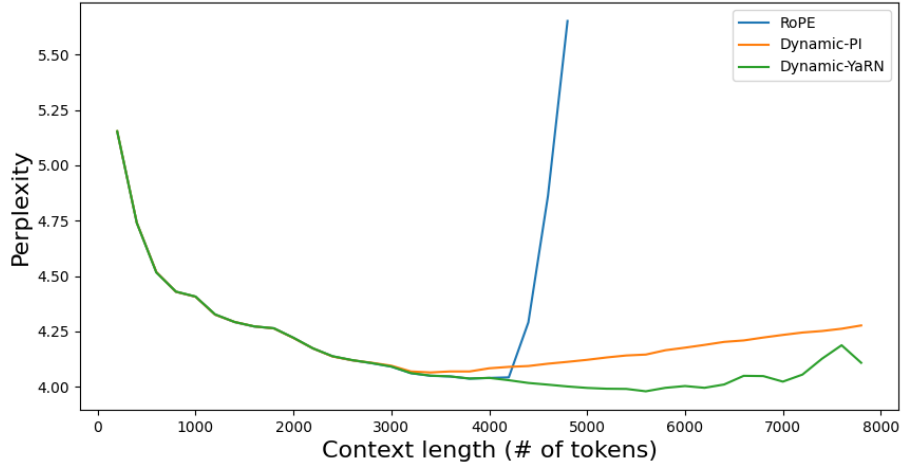


Figure 5: The comparison between RoPE, Dynamic-PI and Dynamic-YaRN using Llama 2 on a long GovReport sample. This model has not been finetuned for long context.

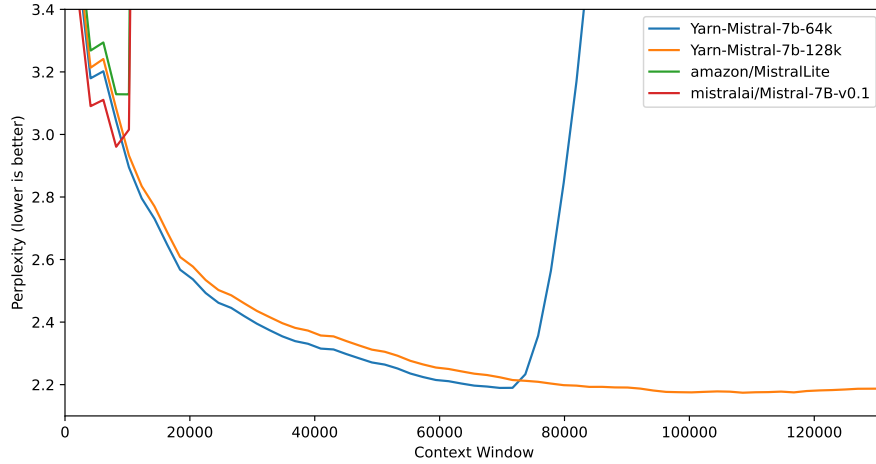


Figure 6: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

B.4 Mistral

We additionally extended the Mistral 7B v0.1 model [20], which broadly follows the Llama architecture. For Mistral we trained a 64k context window model ($s = 8$) for 1000 steps using 16k sequence lengths with a constant learning rate of 1×10^{-6} . The model’s sliding window attention size was set to the context window size, effectively disabling sliding window attention. We then trained for an additional 500 steps at $s = 16$ to arrive at a 128k context window model. The training data was a mix of the pre-train and fine-tune splits of Together Computer’s Long-Data Collections [3].

We evaluated the models following the same procedure as described in 4.3.1, comparing against the base v0.1 model and MistralLite [1], an NTK-aware ($\theta = 1M$) version of v0.1. The results (Figure 6 and Table 6) were consistent with those of the Llama family of models.

Model Size	Model Name	Context Window	Extension Method	Evaluation Context Window Size				
				4096	8192	16384	65536	131072
7B	Mistral v0.1	8k	-	3.09	2.96	36.8	$> 10^3$	$> 10^3$
7B	MistralLite	16k	NTK	3.26	3.13	47.3	$> 10^3$	$> 10^3$
7B	YaRN ($s = 8$)	64k	YaRN	3.18	3.04	2.65	2.20	57.4
7B	YaRN ($s = 16$)	128k	YaRN	3.21	3.08	2.68	2.24	2.19

Table 6: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size