HiRoPE: Length Extrapolation for Code Models Using Hierarchical Position

Kechi Zhang, Ge Li[*], Huangzhao Zhang, Zhi Jin[*]
Key Lab of High Confidence Software Technology (PKU), Ministry of Education
School of Computer Science, Peking University, China
{zhangkechi,lige,zhang_hz,zhijin}@pku.edu.cn

Abstract

Addressing the limitation of context length in large language models for code-related tasks is the primary focus of this paper. Existing LLMs are constrained by their pre-trained context lengths, leading to performance issues in handling long complex code sequences. Inspired by how human programmers navigate code, we introduce Hierarchical Rotary Position Embedding (HiRoPE), a novel approach that enhances the traditional rotary position embedding into a hierarchical format based on the hierarchical structure of source code. HiRoPE offers easy integration into existing LLMs without extra training costs. Our method is extensively evaluated with various LLMs, demonstrating stable performance in tasks such as language modeling and long code completion. We also introduce a new long code understanding task with real-world code projects, in hopes of promoting further development in this code-related field. Theoretically and experimentally, we find that HiRoPE also addresses the out-of-distribution issue in position encoding. Our HiRoPE significantly expands the context length capabilities of LLMs, enabling inference at lengths exponentially greater than the training length.

1 Introduction

Large language models (LLMs) such as LLaMA-2 (Touvron et al., 2023b), and CodeLLaMA (Rozière et al., 2023) have achieved significant performances in code-related tasks. These Transformer-based models excel in code comprehension and generation but face a notable challenge: the limitation of maximum context length. LLMs are typically pre-trained with a context length ranging from 2k to 16k tokens, which often proves insufficient for complex, extended source code. Exceeding this length limitation during inference may lead to performance degradation for these code models, particularly in tasks like project-level code completion or long code generation.

[*]Corresponding authors.

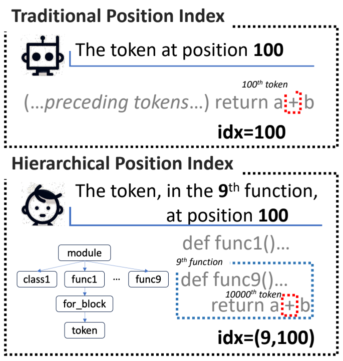
Traditional Position Index
The token at position 100
100th token
(...preceding tokens...) return a+b
idx=100
Hierarchical Position Index
The token, in the 9th function, at position 100
module
class1
func1
...
func9
for_block
token
def func1()...
9th function
def func9()...
10000th token
return a+b
idx=(9,100)

Figure 1: Illustration of the hierarchical position in source code, such as function-level and token-level positions. We also show a simplified abstract syntax tree of the code in the bottom left corner.

Various methods have been developed to extend the context window of LLMs. Some approaches involve fine-tuning on extensive texts (Xiong et al., 2023; Chen et al., 2023b,a; Peng et al., 2023), which can be resource-intensive and potentially lead to overfitting and loss of performance on shorter sequences. There are also some training-free methods (Xiao et al., 2023; Han et al., 2023; Ding et al., 2023). However, these methods usually use window attention rely on local information, and ignore the long dependency in code. It is essential to incorporate *certain structural characteristics* of the code into position encoding to efficiently model these long-distance code dependencies.

Our work diverges from these methods by focusing on the hierarchical information of source code in position encoding, inspired by how human programmers navigate code. Traditional positional encoding uses token counts for positioning, and treats code as plain text. However, human programmers often use hierarchical information in the code, representing positions in the code efficiently through multi-level hierarchical positions. We propose a hierarchical position approach that identifies token positions within specific levels, such as functions or statements. Figure 1 shows the comparison of the traditional position and our hierarchical position. It is clear that the hierarchical positional encoding, benefiting from the full utilization of structural information in the code, can more conveniently locate positional information within long code sequences. This method could more effectively model long dependencies in source code.

Following such inspirations, we introduce a novel approach, Hierarchical Rotary Position Embedding (HiRoPE), which enhances the popular rotary position embedding (RoPE) (Su et al., 2024) into a hierarchical format. HiRoPE differentiates itself by extracting hierarchical information from the source code and splitting the RoPE dimension to represent different hierarchical levels. It simultaneously models token-level relative location and higher-level relative location information. We also add a window mechanism to ensure stability with short texts, aligning with traditional positional encoding.

HiRoPE is a plug-and-play solution, easily integrated into existing LLMs without additional training costs. Our extensive experiments with popular LLMs on tasks like language modeling and token completion in long code contexts (CodeParrot, 2022) demonstrate its effectiveness. We compare HiRoPE with existing length extrapolation methods using long code benchmarks such as LCC (Guo et al., 2023) and RepoBench (Liu et al., 2023a). We also introduce a new long code understanding task named code symbol understanding with real-world code libraries. Theoretically and experimentally, we find that HiRoPE effectively addresses the out-of-distribution issue (Liu et al., 2023b) in position encoding. Our HiRoPE significantly expands the context length capabilities of LLMs, enabling inference at lengths exponentially greater than the training length. We believe our work with HiRoPE not only addresses a critical length limitation in LLM applications but also opens new avenues for long-structured data modeling research.

In summary, we make the following main contributions:

- We propose Hierarchical RoPE (HiRoPE), enhancing the traditional rotary position embedding into a hierarchical format based on the hierarchical structure of source code, providing improved extrapolation capabilities.
- We conducted comprehensive experiments with LLMs on various long code tasks involving language modeling and code completion. We also introduce a new long code understanding task with real-world code projects, in hopes of promoting further development in this code-related field.
- We demonstrate that HiRoPE effectively addresses the out-of-distribution issue in position encoding, enabling inference at lengths exponentially greater than the training length.

2 Preliminary

We first introduce rotary position embedding in Transformer in Section 2.1. While existing work usually regards source code as plain text for modeling, we will also introduce the ignored hierarchical information in source code in Section 2.2.

2.1 Rotary Position Embedding in Transformer

Transformer models require explicit positional information to be injected, typically in the form of positional encodings, to represent the order of inputs. Recently, the rotary position embedding (RoPE) (Su et al., 2024) has become one of the most popular and elegant position encoding strategies and is adopted by various LLMs (Touvron et al., 2023a,b; Rozière et al., 2023). The main point of the RoPE method is using absolute position encoding to show relative position information. Formally, given a position index $m \in [0, L)$ and an embedding vector $\mathbf{x} := [x_0, x_1, \ldots, x_{d-1}]^\top$, where d is the dimension of the attention head, RoPE defines a complex function $\mathbf{f}(\mathbf{x}, m)$ as follows:

$$\begin{aligned}\mathbf{f}(\mathbf{x}, m) = [(x_0 + \mathrm{i}x_1)e^{\mathrm{i}m\theta_0}, (x_2 + \mathrm{i}x_3)e^{\mathrm{i}m\theta_1}, \ldots,\\ (x_{d-2} + \mathrm{i}x_{d-1})e^{\mathrm{i}m\theta_{d/2-1}}]^\top\end{aligned} \tag{1}$$

where $\mathrm{i} := \sqrt{-1}$ is the imaginary unit and $\theta_j = 10000^{-2j/d}$.

With RoPE, the self-attention score can be calculated as:

$$\begin{aligned}&a(m, n) = \mathrm{Re}\langle \mathbf{f}(\mathbf{q}, m), \mathbf{f}(\mathbf{k}, n)\rangle\\ &= \mathrm{Re}\left[\sum_{j=0}^{d/2-1} (q_{2j} + \mathrm{i}q_{2j+1})(k_{2j} - \mathrm{i}k_{2j+1})e^{\mathrm{i}(m-n)\theta_j}\right]\\ &= \sum_{j=0}^{d/2-1} [(q_{2j}k_{2j} + q_{2j+1}k_{2j+1})\cos((m-n)\theta_j)\\ &\quad + (q_{2j}k_{2j+1} - q_{2j+1}k_{2j})\sin((m-n)\theta_j)]\\ &=: a(m-n)\end{aligned} \tag{2}$$

Here $\mathrm{Re}\langle\dots\rangle$ denotes the real part function for a complex number. $\mathbf{q}$ and $\mathbf{k}$ are the query and key vector for a specific attention head. At each layer, RoPE is applied on both query and key embeddings for computing attention scores. We can observe that the calculated attention score is only dependent on relative position $m-n$ through trigonometric functions, which reflects the core of RoPE that uses the absolute position to represent the relative distance. Existing studies show that when dealing with long plain text, the RoPE will meet O.O.D issues where the value of $m-n$ during inference is unseen (Liu et al., 2023b), leading to poor performances.

2.2 Hierarchical Position in Source Code

Most LLMs treat source code as plain text, processing it as if it were ordinary natural language. However, it is essential to take the structural information of code into mind. Source code can be transformed into abstract syntax trees, and these tree structures contain rich hierarchical position information. For example, the code snippets usually can be split into several class or function units, and each class/function contains various types of code blocks and statements. Figure 1 shows an illustration of the simplified abstract syntax tree for a code snippet in the bottom left corner. This higher-level positional information contains rich semantics of source code, making it easy for human programmers to locate and refer to different semantic parts. Therefore, in many existing program representation tasks, this hierarchical information plays a very important role (Allamanis et al., 2018; Zhang et al., 2023). However, for today's large language models, this high-level hierarchical positional information is almost ignored. In this paper, we try to incorporate this hierarchical information into the position encoding method.

3 Hierarchical RoPE

In this paper, we propose HiRoPE, a hierarchical rotary position encoding for source code modeling. Our proposed HiRoPE requires two modified stages: ❶ **Hierarchical Format**: We first take the step to transfer the existing rotary position embedding into a hierarchical format. We verify our approach theoretically and find that the hierarchical format can bring stable extrapolation ability for RoPE. ❷ **Window Mechanism**. To ensure performance stability without further training, we also add a window mechanism, so that when dealing with short texts, our proposed method is consistent with the original positional encoding. An illustration of our HiRoPE is shown in Figure 2.

3.1 Hierarchical format

Unlike previous work that encodes the information of each position as a number index $m \in [0, L)$, we use a h-dimensional vector to represent the hierarchical position index from high-level to low-level, where h is a hyperparameter that indicates how many levels of information we consider in the encoding. We begin with a simple case that we set $h = 2$ and each token position can be represented as (m_1, m_2). We use the higher and lower dimensions in RoPE respectively to represent these two hierarchical position indexes, so the Equation 1 can be rewritten as:

$$\begin{aligned}
&\mathbf{f}'(\mathbf{x}, m_1, m_2) = \\
&[(x_0 + \mathrm{i}x_1)e^{\mathrm{i}m_1\theta_0}, \dots, (x_{d_s-2} + \mathrm{i}x_{d_s-1})e^{\mathrm{i}m_1\theta_{d_s/2-1}}, \\
&(x_{d_s} + \mathrm{i}x_{d_s+1})e^{\mathrm{i}m_2\theta_{d_s/2}}, \dots, (x_{d-2} + \mathrm{i}x_{d-1})e^{\mathrm{i}m_2\theta_{d/2-1}}]^\top
\end{aligned} \tag{3}$$

There are a total of d dimensions in RoPE, and we use the lower d_s dimensions to represent the hierarchical index of m_1, and the remaining dimensions to represent m_2. When we apply it to self-attention, we can get a new calculation of the attention score:

$$\begin{aligned}
&\mathrm{hierarchicalAttn} = \mathrm{Re}\langle\mathbf{f}(\mathbf{q}, m_1, m_2), \mathbf{f}(\mathbf{k}, n_1, n_2)\rangle \\
&= \sum_{j=0}^{d_s/2-1} [(q_{2j}k_{2j} + q_{2j+1}k_{2j+1})\cos((m_1 - n_1)\theta_j) \\
&\quad + (q_{2j}k_{2j+1} - q_{2j+1}k_{2j})\sin((m_1 - n_1)\theta_j)] \\
&\quad + \sum_{j=d_s/2+1}^{d/2-1} [(q_{2j}k_{2j} + q_{2j+1}k_{2j+1})\cos((m_2 - n_2)\theta_j) \\
&\quad + (q_{2j}k_{2j+1} - q_{2j+1}k_{2j})\sin((m_2 - n_2)\theta_j)] \\
&=: a'(m_1 - n_1, m_2 - n_2)
\end{aligned} \tag{4}$$

The attention score $a'(\dots)$ we ultimately obtained through the inner product is quite elegant. It includes the relative position distance of various hierarchical levels, and this form can be similarly extended to the representation of more hierarchical positional structures, as shown in Equation 5. Specifically, when $h = 1$, it remains the same as the original RoPE.

$$\begin{aligned}
&\mathrm{hierarchicalAttn_h} \\
&= \mathrm{Re}\langle\mathbf{f}(\mathbf{q}, m_1, \dots, m_h), \mathbf{f}(\mathbf{k}, n_1, \dots, n_h))\rangle \\
&=: a'(m_1 - m_2, \dots, m_h - n_h)
\end{aligned} \tag{5}$$

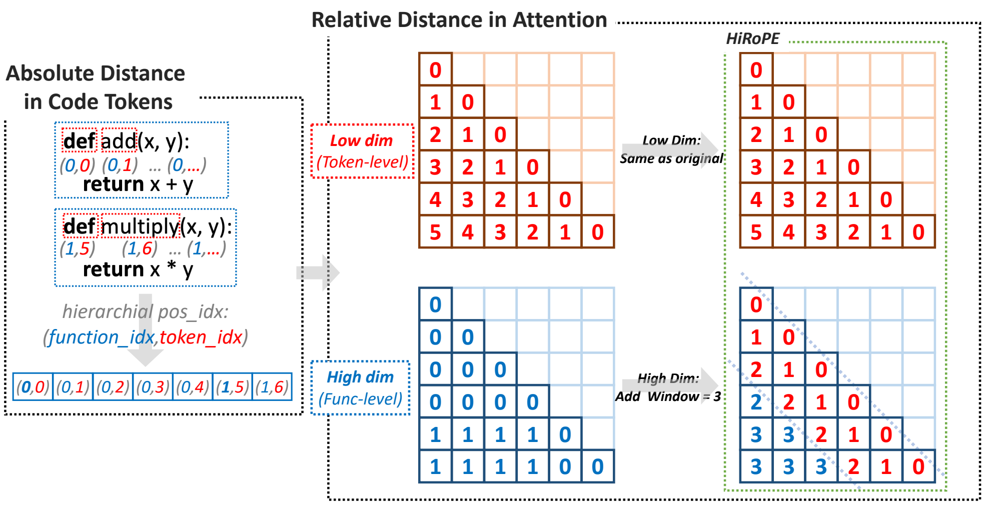

Figure 2: Overview of our HiRoPE. We transfer the existing position encoding method into a hierarchical format (*i.e.*,, function-level and token-level) and apply it across different dimensions. We also add a window mechanism to ensure performance stability (in this figure we set L_{window} to 3).

It indicates that the RoPE has the potential to be transformed into a hierarchical form, and we can use the hierarchical position index in the source code for this new form of RoPE, as shown in the left part of Figure 2.

The original RoPE sets θ_j to $10000^{-2j/d}$, which means that the lower the dimension, the higher its frequency, and the more emphasis on modeling the relative position information of short distances (Pal et al., 2023). Therefore, in our hierarchical format, we use those low dimensions to represent token-level information, and high dimensions to represent higher-level hierarchical information, such as the function level or statement level in source code.

3.2 Window Mechanism

To ensure performance stability without further training, we follow existing extrapolation methods (Xiao et al., 2023; Han et al., 2023) and add a window mechanism. Specifically, when dealing with short code snippets, we believe that existing LLMs have already mastered the ability to model these short semantic structures from vast pre-training code datasets. Therefore, when calculating the attention score, we directly use the original RoPE for those parts that are shorter than a specific length L_{window}. And for those long context parts that the distance is larger than L_{window}, we transfer them to the new hierarchical format by adding $L_{window} - 1$ to each high-level distance. Our subsequent experiments have proved that even without any additional training, this window mechanism can bring strong stability, making the model's performance applicable in various scenarios with arbitrary input lengths. An illustration of the final HiRoPE is shown in the right part of Figure 2.

Task	Dataset		Avg. Length	Samples
Long Code Language Modeling	CodeParrot	0-2048	1031.46	100
		2048-4096	3667.76	100
		4096-8192	7074.57	100
		8192-16384	14353.94	100
Long Text Language Modeling	ReRoPE-eval		21367.55	200
Code Symbol Understanding	Real-world Code Project		12976.89	56
Long Code Completion	LCC		17855.73	300
	RepoBench		21103.42	300

Table 1: Statistics of the evaluation datasets

4 Experiment Setup

In this section, we aim to answer the following research questions through a series of experiments. Details of the evaluation dataset statistics are shown in Table 1.

RQ1. How is the language modeling capability of HiRoPE on long code sequences? We evaluate HiRoPE's language modeling ability on ***CodeParrot-valid*** dataset (CodeParrot, 2022) in Section 5.1.

RQ2. How is the language modeling capability of HiRoPE on long natural language sequences? The natural language lacks the explicit hierarchical structure information found in code, so we have made some modifications: we set every 128 tokens as a segment, and encode it as higher-level position information. We use the evaluation

	LLaMA-2	Sheared LLaMA	TinyLLaMA	Vicuna
Para.	7B	1.3B	1.1B	7B
$L_{pretrain}$	4096	4096	2048	2048
Vocab Size	32000	32000	32000	32000
Hidden Size	4096	2048	2048	4096
Attention Head	32	16	32	32
RoPE Dim	128	128	64	128

Table 2: Statistics of base LLMs

dataset from ***ReRoPE-eval*** [1] (Su, 2023) in Section 5.2. It is a dataset curated from Common Crawl (Crawl, 2023), refined by length-based selection criteria.

RQ3. How does HiRoPE perform in understanding real-world, long-code projects? To evaluate the effect of the method in real long-code scenarios, we design a new evaluation task on real code projects: ***Code Symbol Understanding*** in Section 5.3. Given a long code file, the model is required to output all the function names and class names defined in it. We extract long code files from popular open-sourced code repositories, especially those newly updated code projects to avoid data leakage. Details of these code projects are shown in Table 3.

RQ4. How does HiRoPE perform on existing benchmarks for long code completion? We further perform the evaluation using two long code completion benchmarks: ***LCC*** (Guo et al., 2023) and ***RepoBench*** (Liu et al., 2023a) in Section 5.4.

RQ5. What is the impact of various settings in HiRoPE? To demonstrate that each setting in the design of our HiRoPE works, we carry out extensive ablation studies that include the dimensions' split settings, the window mechanism, and the high-level segment split strategy in Section 5.5.

4.1 Base LLMs

The models used include LLaMA-2 (7B) (Touvron et al., 2023b), Sheared-LLaMA (1.3B) (Xia et al., 2023), TinyLlama (1.1B) (Zhang et al., 2024), and Vicuna (7B) (Chiang et al., 2023). This model choice is driven by their widespread use and popularity, as well as the constraints of our computing capabilities. Details are provided in Table 2.

4.2 Baselines

Considering training on long context sequences is resource-intensive and time-consuming, we focus on those popular length extrapolation methods without training, including NTK (bloc97, 2023), ReRoPE (Su, 2023) and Self-Extend (Jin et al., 2024). ReRoPE sets a window when calculating the relative distance, and clips those distances outside the window. Self-Extend uses group attention with the floor operation to calculate relative distances. These methods have shown impressive performance on long context language modeling. We also make comparisons with the original RoPE method, which is denoted as "origin" in the first row of Table 4, 5 and 6.

4.3 Inference Settings

In our experiments, our HiRoPE uses a two-layer hierarchy, accounting for the position index at the token and function/class levels of the source code based on *tree-sitter* (Brunsfeld et al., 2024). For long context in natural language, we make some modifications and set every 128 tokens as a higher-level segment. We set the split dimension half of the total: $d_s = 0.5 * d_{total}$, and choose a window length: $L_{window} = 512$. We keep the hyperparameters the same for those state-of-the-art baselines for a fair comparison. We use greedy search decoding for generation. We use 4 A6000 GPUs for all experiments.

4.4 Details of Code Symbol Understanding task

In RQ3, we design a new task: Code Symbol Understanding. Given a long code context, the model is required to output all the function names and class names defined in it. We extract long code files from popular open-sourced code repositories, especially those newly updated code projects to avoid data leakage. We construct a static analysis tool to get the abstract syntax tree of each code file, and then get all defined function and class names in it as the ground-truth output symbols. Details of these code projects are shown in Table 3. We show the number of files we extracted from each project as well as the average length of these code files. We also show the average number of symbols and their location statistics in the table.

In Figure 3 we show an illustration of our proposed new task. We replace the *input_code* part with each code file, and use the task prompt to guide models to extract and output all defined function and class names in input code. We also show an example output in the figure.

[1] `https://github.com/bojone/rerope/blob/main/samples_15k.jsonl`

Github Repo	File Nums	File Length	Avg. Symbols	Min. Symbol Loc	Max. Symbol Loc
ddbourgin/numpy-ml	2	14055	15	927	10845
gradio-app/gradio	1	12938	11	272	12265
huggingface/accelerate	4	12814.5	11.5	379	12500
huggingface/diffusers	4	13324.75	16.2	276	8347
huggingface/optimum	1	13491	11	580	12868
huggingface/peft	1	15457	11	514	8525
huggingface/transformers	18	12919.4	12.7	239	14514
langchain-ai/langchain/	2	14195	11.5	285	10790
numpy/numpy	6	11158.8	14	100	12026
tensorflow/tensorflow	17	13191.7	13.4	285	14920
Total	**56**	**12976.89**	**13.17**	**100**	**14920**

Table 3: Statistics of Code Symbol Understanding task. We show the number of files we extracted from each project as well as the average length of these code files. We also show the average number of symbols and their location statistics in the table.

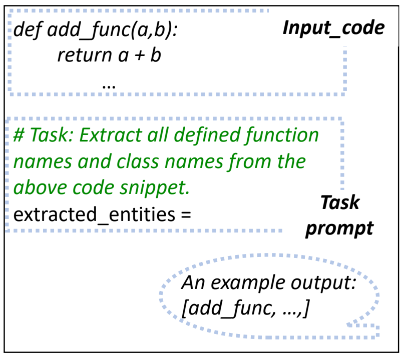

Figure 3: Illustration of Code Symbol Understanding task. We use the task prompt to guide models to extract and output all defined function and class names in input code.

5 Results and Analyses

5.1 Long Code Language Modeling

Language modeling is the most fundamental and the least requirement for a LLM. We evaluate HiRoPE's language modeling ability on CodeParrot-valid dataset. We divide the original dataset into different length intervals. The experiment results are shown in Table 4. A smaller *loss* and a smaller *ppl* indicate a stronger language modeling capacity of the corresponding model. Conversely, a larger *acc* suggests a stronger capability of the respective model in code completion on the given dataset. The *origin* indicates that we directly use the original setting of the model to evaluate.

Experiments show that original LLMs perform badly on the long code language modeling task. Even when the length slightly exceeds the pre-training length, the ppl of all models exceed 45, demonstrating their essential lack of modeling and understanding capabilities for longer codes. When we apply length extrapolation methods, all methods can reduce the loss and ppl into an acceptable range for those long source codes. Specifically, for ultra-long code sequences (length over 8192), HiRoPE achieves the best results in all metrics and under all settings. This fully demonstrates the advantages of our HiRoPE in modeling long sequence codes. Our method also shows generalization abilities. The four models evaluated have differences in model parameters, pre-training data, and pre-training length, yet our method has shown very good results on all these models.

It is worth noting that our method does not impair the model's performance on shorter code. We noted that some popular length extension methods, such as NTK, can impair the performance on short code. Thanks to our window mechanism, our method stays on par with the baseline model on shorter datasets (length 0-2048) and even surpasses the baseline on some metrics. HiRoPE demonstrates consistently excellent language modeling capabilities in various code length scenarios.

5.2 Long Text Language Modeling

In addition to testing the ability of language modeling on long code, we also evaluate its effects on long natural language texts. We set every 128 tokens as a high-level segment. We use the ReRoPE-eval dataset, and results are shown in Table 5. In addition to calculating metrics on all tokens (refer to "all_..." in the table), we also record metrics on the last 2048 tokens of each data (refer to "last_...").

We find that our HiRoPE can also achieve significant improvement. HiRoPE achieves the best results on almost all metrics. Another interesting observation is that, given sufficient context, the model can utilize this contextual information to perform better when generating later tokens. That is to say, the metrics of "last_..." should be better than those of "all_...". However, we observe that

			Dataset Length:	0-2048			2048-4096			4096-8192			8192-16384		
	Para.	$L_{pretrain}$		loss ↓	ppl ↓	acc ↑	loss ↓	ppl ↓	acc ↑	loss ↓	ppl ↓	acc ↑	loss ↓	ppl ↓	acc ↑
LLaMA-2	7B	4096	origin	**0.8579**	**2.3583**	**0.8065**	0.9820	2.6698	0.7551	nan	nan	nan	nan	nan	nan
			NTK	1.1107	3.0365	0.7472	1.0469	2.8488	0.7414	0.9858	2.6800	0.7729	1.0181	2.7678	0.7630
			ReRoPE	0.8593	2.3615	0.8054	0.7278	2.0705	0.8121	0.6633	1.9411	0.8411	0.7187	2.0518	0.8243
			Self-Extend	0.8588	2.3604	0.8055	0.7209	2.0562	0.8147	0.6519	1.9192	0.8441	0.6983	2.0103	0.8290
			HiRoPE	0.8586	2.3598	0.8060	**0.7185**	**2.0514**	**0.8153**	**0.6482**	**1.9121**	**0.8452**	**0.6821**	**1.9780**	**0.8332**
ShearedLLaMA	1.3B	4096	origin	**1.2874**	**3.6235**	**0.7341**	1.3103	3.7074	0.7019	3.8381	46.4375	0.4866	5.6958	297.6160	0.3211
			NTK	1.6242	5.0744	0.6607	1.5047	4.5029	0.6619	1.3708	3.9383	0.7015	1.3657	3.9183	0.6964
			ReRoPE	1.2897	3.6316	0.7332	1.0497	2.8567	0.7560	0.9699	2.6376	0.7846	1.0044	2.7303	0.7716
			Self-Extend	1.2892	3.6300	0.7337	1.0428	2.8371	0.7586	0.9568	2.6034	0.7874	0.9804	2.6656	0.7768
			HiRoPE	1.2888	3.6285	0.7338	**1.0382**	**2.8242**	**0.7600**	**0.9514**	**2.5894**	**0.7885**	**0.9660**	**2.6273**	**0.7811**
TinyLlama	1.1B	2048	origin	1.0788	2.9410	0.7594	4.1732	64.9235	0.4506	6.6603	780.8009	0.2630	7.9938	2962.5881	0.1682
			NTK	1.1837	3.2664	0.7405	1.0952	2.9899	0.7256	0.9719	2.6430	0.7733	1.0021	2.7240	0.7626
			ReRoPE	0.9703	2.6388	0.7877	0.8251	2.2821	0.7905	0.7685	2.1565	0.8210	0.8275	2.2877	0.8040
			Self-Extend	**0.9698**	**2.6375**	0.7856	**0.8123**	**2.2530**	0.7931	**0.7577**	**2.1333**	**0.8235**	0.8119	2.2521	0.8073
			HiRoPE	0.9743	2.6493	**0.7881**	0.8268	2.2861	**0.7981**	0.7683	2.1562	0.8208	**0.8040**	**2.2345**	**0.8094**
Vicuna	7B	2048	origin	1.1787	3.2502	0.7551	4.6046	99.9449	0.4473	7.7207	2254.4730	0.2597	9.9449	20846.3927	0.1601
			NTK	1.3417	3.8255	0.7150	1.2587	3.5208	0.7068	1.1809	3.2573	0.7344	1.1912	3.2911	0.7285
			ReRoPE	1.0716	2.9201	0.7800	0.8760	2.4012	**0.7912**	0.8138	2.2566	0.8182	0.8580	2.3585	0.8023
			Self-Extend	1.0710	2.9183	**0.7802**	0.8735	2.3953	0.7891	**0.7988**	**2.2228**	**0.8220**	0.8351	2.3049	0.8066
			HiRoPE	**1.0707**	**2.9174**	0.7799	**0.8724**	**2.3926**	0.7903	0.8002	2.2261	0.8213	**0.8314**	**2.2965**	**0.8080**

Table 4: Language Modeling Ability on CodeParrot-valid dataset. “nan” indicates that the model performs significantly poor on the given setting.

		last loss ↓	last ppl ↓	last acc ↑	all loss ↓	all ppl ↓	all acc ↑
TinyLlama	origin	9.3083	>1000	0.0194	7.3486	>1000	0.1466
	NTK	2.1338	8.4468	0.5553	2.2745	9.7234	0.53
	ReRoPE	1.8448	6.327	0.6008	1.903	6.7062	0.5867
	Self-Extend	1.7904	5.9916	0.6089	1.8749	6.5201	0.5908
	HiRoPE	**1.7829**	**5.9474**	**0.6102**	**1.8717**	**6.4991**	**0.5913**
SheardLLaMA	origin	8.5027	>1000	0.0412	5.5237	250.5	0.274
	NTK	2.3596	10.5863	0.5159	2.4056	11.0853	0.5059
	ReRoPE	1.7952	6.0205	0.605	1.8514	6.3687	0.5932
	Self-Extend	1.7662	5.8486	0.6105	1.8348	6.264	**0.5966**
	HiRoPE	**1.7622**	**5.8252**	**0.6113**	**1.8332**	**6.2536**	0.5963

Table 5: Language Modeling Ability on ReRoPE-eval dataset. In addition to calculating metrics on all tokens (refer to "all_..."), we also record metrics on the last 2048 tokens of each data (refer to "last_...").

for the original model, the situation is contrary to this. We attribute this to the fact that the original model's ability to model long sequence languages is so poor that it can't utilize that distant contextual information at all. Our HiRoPE can significantly improve the model's ability to handle long codes and textual data, without requiring any training, reflecting its practical value.

5.3 Code Symbol Understanding

To evaluate the effect of the method in real long-code scenarios, we have designed an evaluation task for real code projects: Code Symbol Understanding. Given a long code context, the model is required to output all the function names and class names defined in it. These pre-defined functions and classes are reused frequently in actual code development. For code models, understanding which functions and classes are defined in the code project is a basic capability requirement. This task is inspired by the popular "Needle in a Haystack" synthetic evaluation (gkamradt, 2023), but our code symbol understanding task is more realistic and code-related. Task examples are shown in Figure 3. We use *recall* as the evaluation metric.

		Code Symbol Understanding ***(Recall*** ↑***)***	Long Code Completion ***(Edit Sim*** ↑***)***					
			LCC			RepoBench		
			0-4k	4k-8k	>8k	0-4k	4k-8k	>8k
LLaMA-2	origin	0.0012	54.5	4.36	4.08	8.29	6.79	6.59
	ReRoPE	0.0837	65.83	67.43	63.22	**52.82**	47.85	45.38
	HiRoPE	**0.0911**	**66.61**	**69.93**	**65.38**	52.20	**53.30**	**51.24**
SheardLLaMA	origin	0.0067	27.17	3.33	2.52	4.39	2.94	2.35
	ReRoPE	0.0743	35.56	36.1	36.91	34.03	37.37	33.44
	HiRoPE	**0.0809**	**46.13**	**51.67**	**46.33**	**40.17**	**39.98**	**39.52**
TinyLLaMA	origin	0.0067	17.29	5.45	6.28	7.91	7.53	7.07
	ReRoPE	0.1214	**49.22**	**57.20**	**53.11**	37.72	40.50	39.38
	HiRoPE	**0.1415**	35.17	42.83	49.92	**42.48**	**43.53**	**39.82**
Vicuna	origin	0.0067	18.47	2.56	2.76	3.67	2.49	2.32
	ReRoPE	0.0636	57.95	59.73	58.52	**42.78**	**43.65**	45.23
	HiRoPE	**0.0721**	**63.42**	**62.01**	**64.42**	37.10	42.30	**45.93**

Table 6: Performance on Code Symbol Understanding and Long Code Completion.

Our experiments in Table 6 have proven that this seemingly simple task is extremely difficult for LLMs. We also evaluate this task using *GPT-3.5-16k* (GPT-3.5, 2023) and find that its result is only 0.72. All these LLMs are not ideal in this more realistic code symbol understanding task. Our HiRoPE has been improved from the perspective of positional encoding, enabling the model to perceive structural hierarchy changes in the code, thus achieving relatively good results. Compared to the original models, our HiRoPE can achieve almost a hundredfold improvement on average across four models. We release our dataset in hopes of promoting further development in this code-related field.

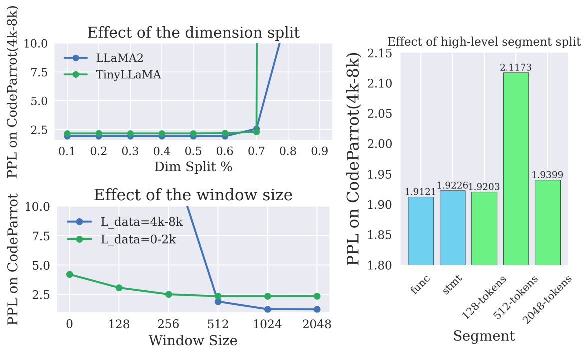

Figure 4: Ablation Studies including the settings of the dimension split, the window mechanism, and the high-level segment split strategy.

5.4 Long Code Completion

We further perform the evaluation using two real-world long code completion benchmarks: LCC and RepoBench. Given a long code context, the model is required to generate the complete next line of code. We follow the experiment settings in Longbench-E (Bai et al., 2023) and use *edit similarity* as metrics. Results are shown in Table 6.

Our HiRoPE also achieves stable improvements on this long code-related task. The input code context is filled with various predefined functions and classes. Our method can effectively sense these contents, handle these complex dependencies, and successfully use those functions that are defined far away during generation. Under two dataset scenarios, our method consistently outperforms other baseline settings. The results reflect the practicality and generalization ability of our method.

5.5 Ablation Study

In our experiments, ❶ we set the split dimension half of the total $d_s = 0.5 * d_{total}$ and ❶ choose a window length ❷ $L_{window} = 512$. ❸ Our hierarchical position includes both the token and function/class levels of the source code. We further carry out extensive ablation studies including these settings and results are shown in Figure 4.

We choose different dimension splitting ratios to observe the performance of LLaMA-2 and TinyLLaMA in terms of ppl on CodeParrot [4k-8k]. When the split ratio is 1, the HiRoPE degenerates into the original model. Specifically, both models show significant fluctuations in ppl between ratios of $[0.6, 0.7]$. We will explore the reason in Section 6.1.

We change the window length to observe the performance of LLaMA-2 on CodeParrot [0-2k] and [4k-8k]. For shorter code data, we observe a decreasing trend in ppl as the window size increases. This validates that the window mechanism can allow the model to retain its original computational mechanisms and better handle short-distance dependencies. For longer code data, ppl behaves anomalously when the window size is very small. This also indicates that window mechanisms play a key role in modeling long-distance dependencies.

We change the high-level segment split strategy. In addition to dividing the hierarchy at the func-level, we also try to split at the code statement level as well as implementing a strategy of splitting continuous n-tokens as a high-level segment (n = 128, 512, 1024). Experiments show that dividing at the function level achieves the best results. The semantics within a function are relatively similar, while the semantics between functions usually vary greatly. It is necessary to divide long code sequences into levels according to functions and classes.

6 Discussion

6.1 Mitigating Out-of-Domain Issues in Long Contexts

Existing work (Liu et al., 2023b) shows that LLMs fail on input out of the pretraining context window because of the Out-Of-Distribution (O.O.D) issues. We take the inspiration to explain it from a cyclical perspective. In RoPE, each dimension is composed of trigonometric functions, and its period can be denoted as $T_j = \frac{2\pi}{\theta_j} = 2\pi * 10000^{\frac{2j}{d}}$. Only those dimensions that have been completely trained within the pre-training length can be considered as a reliable part for extrapolation, others would encounter O.O.D issues when dealing with problems of extrapolation. We can then get those reliable dimensions by calculating $T_j < L_{pretrain}$. The calculated dim split is 0.70 and 0.63 for the four models in our experiments (Table 7). We are surprised to find that it is similar to the ratio we obtained in the ablation study in Figure 4. Our HiRoPE uses those high dimensions to represent higher-level position index information, and properly applies them to smaller input numbers, thus mitigating O.O.D issues in long code contexts.

The traditional RoPE uses a number m as the position index to represent position information. Due to the O.O.D problem in high dimensions, its reliable range is $\{m \in [0, L_{pretrain}]\}$. In our HiRoPE, we use a two-layer hierarchy as (m_1, m_2) and

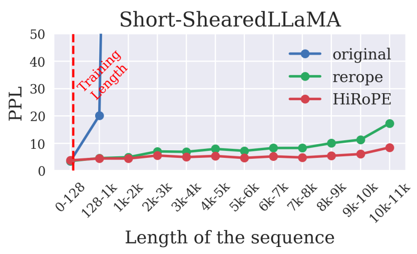

Figure 5: Performance of Short-ShearedLLaMA on CodeParrot dataset. The training length is set to 128. The results suggest our method has the potential to extrapolate code models at an exponential length.

the reliable range is $\{m_1 \in [0, L_{pretrain}], m_2 \in [0, L_{pretrain}]\}$. It proves that under ideal circumstances, HiRoPE can effectively extrapolate to the length of L^h in an exponential ratio, where h is the hierarchy layer. We attempt to explore the upper limit of our HiRoPE's extrapolation performance in the experiment in the next Section 6.2.

6.2 Upper Limit of HiRoPE's Performance

Due to computational resource constraints, we made the following modifications based on Section 5.1: ❶ Firstly, in order to obtain a base model of suitable length, we designed a training strategy to obtain a shorter context length LLM, named **ShortLLM**: We use the position interpolation (Chen et al., 2023b) method ***reversely*** to reduce the input length of some mainstream models to $L_{short} = 128$ at a smaller training cost. Specifically, given a position index m in the original RoPE, we use the new index $\alpha_{short} * m$ to replace it as shown in Table 8. We fine-tune short models for 1000 steps. ❷ We resample the CodeParrot-valid dataset, further refining it into smaller distance ranges, each range containing up to 50 test samples. The results are shown in Figure 5 and 6.

Our trained ShortLLM successfully demonstrates the expected performance: the performance drastically decreases after surpassing the training length $L_{short} = 128$. We then apply our HiRoPE as well as the baseline ReRoPE. Our HiRoPE demonstrates a more stable trend on long code sequences, even at the position close to L_{short}^2. It suggests our method has the potential to extrapolate code models **at an exponential length**.

7 Related Work

Existing large language models are originally trained with fixed context sizes. When dealing with longer sequences, the model's performance may decrease quite drastically. Recent studies have explored ways to expand the context length. Position Interpolation (Chen et al., 2023b) linearly down-scales the input position indices to match the original context window size of LLMs with several training steps. Similar studies (Chen et al., 2023a; Peng et al., 2023; Chen et al., 2023c; Guo et al., 2023) also require fine-tuning. However, these methods need additional tuning in longer contexts or face a disastrous collapse after the extrapolation bound. There are also some approaches without training. Some studies use window attention to clip the long sequences such as (Xiao et al., 2023; Han et al., 2023; Ding et al., 2023)). However, these methods rely on local information and may not effectively expand the context window, struggling with long dependencies in code. Recently, some methods have explored modifying the relative distance to extend the extrapolation length (bloc97, 2023; Su, 2023; Jin et al., 2024) and focus on the natural language text. We pursue the research line of training-free methods and propose considering the structural information of the code when modeling the position. Our work is inspired by the hierarchical embedding in our former work (Zhang et al., 2023). We notice hierarchical positions are used in some concurrent work (He et al., 2024; Lu et al., 2024) in NLP and CV domains. We expand the traditional RoPE method into a hierarchical format and prove the effectiveness of our HiRoPE through theoretical derivations and practical experiments.

.

8 Conclusion

We propose HiRoPE, a training-free solution to the context length limitation in LLMs for long code modeling. We integrate the hierarchical structure of source code into position encoding of LLMs. Experiments demonstrate that HiRoPE achieves stable improvements on diverse code-related tasks. Our work not only addresses a critical limitation in LLM applications but also opens new avenues for long structured data modeling research.

9 Acknowledgments

This research is supported by the National Key R&D Program under Grant No. 2023YFB4503801, the National Natural Science Foundation of China under Grant No. 62072007, 62192733, 61832009, 62192730, and the Major Program (JD) of Hubei Province (No.2023BAA024).

Limitations

There are several limitations to our work that we aim to address:

Firstly, constrained by computational resources, we choose models below 7B for experiments. The four models evaluated have differences in model parameters, pre-training data, and pre-training length, yet our method has shown very good results on all these models. We will attempt to conduct experiments on models with larger parameters and more complex structures to promote the development of the LLM community.

Next, our discussion on the upper limit of the HiRoPE's performance tends to lean towards theoretical derivation. We have designed a set of Short-LLM experiments to prove our conclusions. It suggests our method has the potential to extrapolate code models at an exponential length, so for the LLaMA-2 model with $L_{pretrain} = 4096$, we can **theoretically** extrapolate its length to $L_{pretrain}^2 \approx 16,000,000$. We are not clear whether some settings will implicitly affect the performance of the model. We will continue to explore the robustness of this experimental idea and try to explore the maximum performance of our method on real LLMs.

References

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to represent programs with graphs. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *CoRR*, abs/2308.14508.

bloc97. 2023. Ntk-aware scaled rope allows llama models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation. `https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/`.

Max Brunsfeld, Andrew Hlynskyi, Amaan Qureshi, Patrick Thomson, Josh Vera, and et al. 2024. tree-sitter/tree-sitter: v0.21.0-pre-release-1.

Guanzheng Chen, Xin Li, Zaiqiao Meng, Shangsong Liang, and Lidong Bing. 2023a. CLEX: continuous length extrapolation for large language models. *CoRR*, abs/2310.16450.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023b. Extending context window of large language models via positional interpolation. *CoRR*, abs/2306.15595.

Yuhan Chen, Ang Lv, Ting-En Lin, Changyu Chen, Yuchuan Wu, Fei Huang, Yongbin Li, and Rui Yan. 2023c. Fortify the shortest stave in attention: Enhancing context awareness of large language models for effective tool use. *CoRR*, abs/2312.04455.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

CodeParrot. 2022. https://huggingface.co/codeparrot.

Common Crawl. 2023. Common crawl. `https://commoncrawl.org/`.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. 2023. Longnet: Scaling transformers to 1, 000, 000, 000 tokens. *CoRR*, abs/2307.02486.

gkamradt. 2023. Needle in a haystack - pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack/tree/main`.

GPT-3.5. 2023. `https://platform.openai.com/docs/models/gpt-3-5`.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian J. McAuley. 2023. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, pages 12098–12107.

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

Zhenyu He, Guhao Feng, Shengjie Luo, Kai Yang, Di He, Jingjing Xu, Zhi Zhang, Hongxia Yang, and Liwei Wang. 2024. Two stones hit one bird: Bilevel positional encoding for better length extrapolation. *CoRR*, abs/2401.16421.

Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. 2024. LLM maybe longlm: Self-extend LLM context window without tuning. *CoRR*, abs/2401.01325.

Tianyang Liu, Canwen Xu, and Julian J. McAuley. 2023a. Repobench: Benchmarking repository-level code auto-completion systems. *CoRR*, abs/2306.03091.

Xiaoran Liu, Hang Yan, Shuo Zhang, Chenxin An, Xipeng Qiu, and Dahua Lin. 2023b. Scaling laws of rope-based extrapolation. *CoRR*, abs/2310.05209.

Zeyu Lu, Zidong Wang, Di Huang, Chengyue Wu, Xihui Liu, Wanli Ouyang, and Lei Bai. 2024. Fit: Flexible vision transformer for diffusion model. *CoRR*, abs/2402.12376.

Arka Pal, Deep Karkhanis, Manley Roberts, Samuel Dooley, Arvind Sundararajan, and Siddartha Naidu. 2023. Giraffe: Adventures in expanding context lengths in llms. *CoRR*, abs/2308.10882.

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

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

Jianlin Su. 2023. Rectified rotary position embeddings. `https://github.com/bojone/rerope`.

Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, and et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *CoRR*, abs/2310.06694.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *CoRR*, abs/2309.17453.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. 2023. Effective long-context scaling of foundation models. *CoRR*, abs/2309.16039.

Kechi Zhang, Zhuo Li, Zhi Jin, and Ge Li. 2023. Implant global and local hierarchy information to sequence based code representation models. In *31st IEEE/ACM International Conference on Program Comprehension, ICPC 2023, Melbourne, Australia, May 15-16, 2023*, pages 157–168.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *CoRR*, abs/2401.02385.

A Details of ShortLLM experiments

A.1 Theoretical calculation for O.O.D issues

According to the analysis in Section 6.1, we can calculate the reliable dimensions for each model as:

$$Split\% = log_{10000} \frac{L_{pretrain}}{2\pi} \tag{6}$$

The theoretical calculation results are shown in Table 7.

	$L_{pretrain}$	RoPE Dim	Split %	Split Dim
LLaMA-2	4096	128	0.70	90.05
ShearedLLaMA	4096	128	0.70	90.05
TinyLLaMA	2048	64	0.63	40.21
Vicuna	2048	128	0.63	80.42

Table 7: Theoretical calculation of the reliable extrapolation dimension.

A.2 ShortLLM Training

In order to obtain the ShortLLM of suitable length, we use the position interpolation (Chen et al., 2023b) method ***reversely***. Given a position index m in the original RoPE, we use the new index $\alpha_{short} * m$ to replace it as shown in Table 8. We sample the CodeParrot-train dataset and filter the data length less than L_{short}. We set the global batch size to 64 and fine-tune the models for 1000 steps.

	Para.	$L_{pretrain}$	L_{short}	α_{short}
LLaMA-2	7B	4096	128	32
ShearedLLaMA	1.3B	4096	128	32
TinyLLaMA	1.1B	2048	128	16
Vicuna	7B	2048	128	16

Table 8: Model Statistics for ShortLLM experiments.

A.3 ShortLLM Performances

After training, we apply our HiRoPE to these ShortLLM models. Figure 6 shows the experiment results of the ShortLLM. We can observe that for all models, Our HiRoPE can maintain good stability as the length of the input code increases. Even if the baseline ReRoPE method gradually becomes unstable under some experimental settings (such as on Short-TinyLLaMA), our method can resist these performance declines.

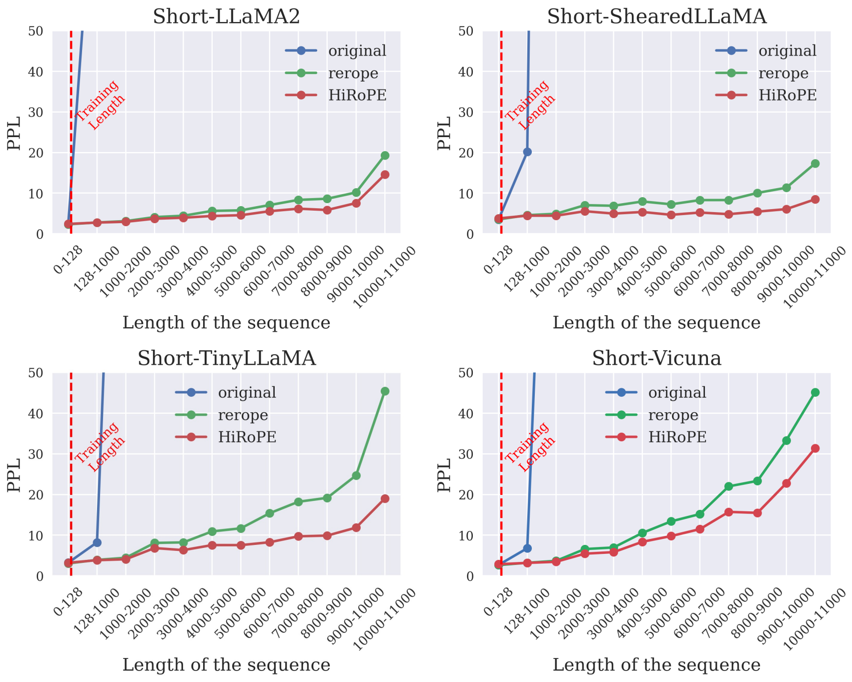

Figure 6: Performance of ShortLLMs on CodeParrot dataset. The training length is set to 128.