ShortGPT: Layers in Large Language Models are More Redundant Than You Expect

Xin Men[1,*], **Mingyu Xu**[1,*], **Qingyu Zhang**[2,3,*], **Qianhao Yuan**[2,3,*],
Bingning Wang[1,†], **Hongyu Lin**[2,†], **Yaojie Lu**[2], **Xianpei Han**[2], **Weipeng Chen**[1]
[1]Baichuan Inc.
[2]Chinese Information Processing Laboratory, Institute of Software, Chinese Academy of Sciences
[3]University of Chinese Academy of Sciences

Abstract

As Large Language Models (LLMs) continue to advance, their computational overhead has increased significantly. In this study, we identify notable redundancy across the layers of LLMs, where some layers contribute minimally to the overall network functionality. To quantify this, we introduce a metric called Block Influence (BI), which measures the importance of each layer based on the similarity between its input and output. Based on the observation of layer redundancy, we propose straightforward pruning methods for different tasks: ShortGPT for multiple-choice tasks and ShortGPT-gen for generative tasks. They prune redundant layers based on their BI scores. Our methods demonstrate superior performance over previous pruning methods. The ability to achieve better results through simple layer pruning, as opposed to more complex pruning techniques, suggests a high degree of redundancy across layers. We hope this work will contribute to future research for improving LLM efficiency. The code is publicly available at https://github.com/icip-cas/ShortGPT.

1 Introduction

The field of large language models (LLMs) has witnessed rapid development recently, with LLMs achieving impressive performance across various domains. Guided by the scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022), modern LLMs require significant computational resources, creating substantial barriers to their practical use.

To mitigate the computational demands of large models, techniques for improving model efficiency have become a critical area. These techniques are generally divided into quantization (Liu et al., 2021; Gholami et al., 2022; Dettmers et al., 2022, 2024) and pruning(LeCun et al., 1989; Han et al., 2015; Frantar and Alistarh, 2023). Quantization reduces the precision of model parameters, but its effectiveness often requires specific hardware support. In contrast, pruning reduces the participating networks to decrease the model's computation, offering a more flexible approach. Despite its advantages, many existing pruning methods are complex. For example, some require gradient information (Ma et al., 2024), which limits their practicality.

In this paper, we focus on layer redundancy in LLMs and propose methods to improve LLM efficiency. We introduce **Block Influence (BI)**, a metric that quantifies how much hidden states change after passing through each layer, providing a direct measure of layers' importance. Leveraging BI, we propose simple yet effective pruning methods for multiple-choice and generative tasks. For multiple-choice tasks, we propose **ShortGPT**, which identifies and removes redundant layers with lower BI scores, significantly reducing model size without sacrificing much performance. For generative tasks, we propose a dynamic pruning method, **ShortGPT-gen**, where the input tokens skip the same redundant layers as in ShortGPT while the generated token pass through all layers to resolve the accumulated errors of ShortGPT during generation.

To evaluate our methods, we conduct evaluation across comprehensive benchmarks. Our experiments reveal that our methods exhibit a smaller performance decrement than previous methods. For instance, ShortGPT removes 10 layers (25% of the total 40 layers) from the LLaMA2-13B model, resulting in only a slight drop in performance on the MMLU benchmark (Hendrycks et al., 2020), from 55.0 to 52.2. ShortGPT-gen skips 10 layers of LLaMA2-13B, leading to only a minimal decrease in performance on the GSM8K (Cobbe et al., 2021) benchmark without requiring any training, from 28.89 to 26.91. Our findings highlight substantial redundancy in current LLMs and suggest potential avenues for improving the efficiency

[*] Equal contribution.
[†] Corresponding authors. daniel@baichuan-inc.com, hongyu@iscas.ac.cn

Findings of the Association for Computational Linguistics: ACL 2025, pages 20192–20204

of models by reducing inherent redundancy in the future. Moreover, our methods are orthogonal to quantization methods, meaning it can be combined with quantization techniques to further reduce the computational overhead of LLMs.

Our main contributions are as follows:

- We analyze the redundancy in large language models (LLMs) and reveal their significant redundancy at the layer level. This finding inspires us to improve LLM efficiency by pruning redundant layers.
- We propose a metric called Block Influence (BI) as an indicator of layer importance. Based on BI, we propose layer pruning methods: ShortGPT for multiple-choice tasks and ShortGPT-gen for generative tasks.
- Our ShortGPT maintains approximately 90% performance for multiple-choice tasks while reducing about 25% of parameters, outperforming previous methods. ShortGPT-gen maintains more than 90% performance for generative tasks in a training-free manner when skipping about 25% of the layers.

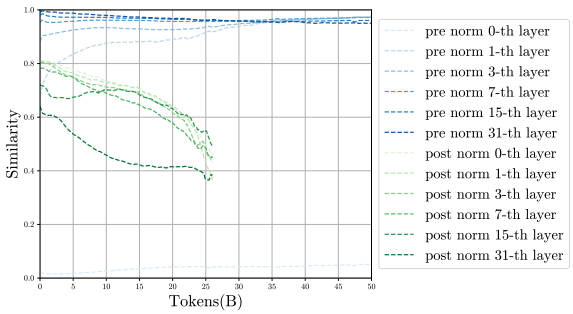


Figure 1: Cosine similarity between a layer's input and output during training. The horizontal axis (X-axis) represents the number of training tokens, and the vertical axis (Y-axis) depicts the similarity. Notably, the model employing post-norm (green) exhibits divergence after approximately ∼26B tokens of training. The training settings are provided in Appendix E.

2 Motivation

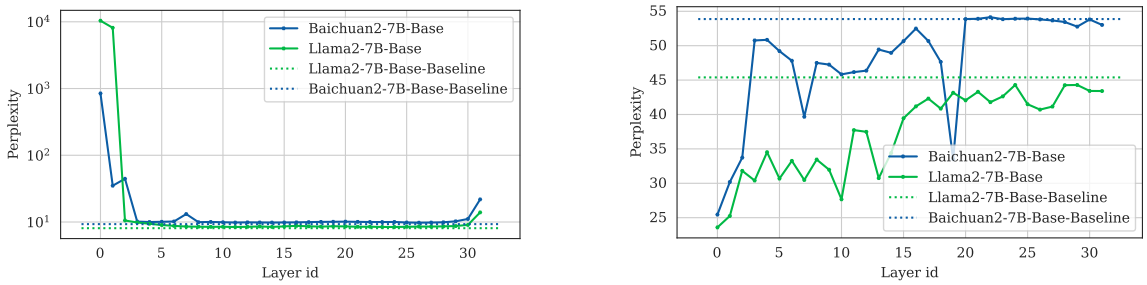
2.1 Background

LLMs are primarily based on the Transformer architecture (Vaswani et al., 2017), with the pre-norm configuration being commonly adopted, as in models like LLaMA (Touvron et al., 2023). The pre-norm configuration, where layer normalization is applied before the self-attention and feed-forward networks, offers several advantages such as faster convergence, improved training stability, and better scalability for deeper networks (Xiong et al., 2020; Liu et al., 2020; Wang et al., 2024). Due to these benefits, the pre-norm approach has been adopted even in non-transformer models, such as Mamba (Gu and Dao, 2023) and RWKV (Peng et al., 2023). For the sake of simplicity in descriptions, our analysis primarily focuses on Transformers, though we extend our experiments to non-Transformer structures in Section 4.4.

However, we observe that when pre-norm is adopted, the similarity between the input and output of transformer layers tends to be higher, as illustrated in Figure 1. This high similarity indicates that certain layers induce minimal changes to the hidden states, suggesting they contribute little to the model's overall function. It suggests that the deep layers of the model with pre-norm might not play a critical role in the overall function, and that **the layers in large language models could be more redundant than expected**.

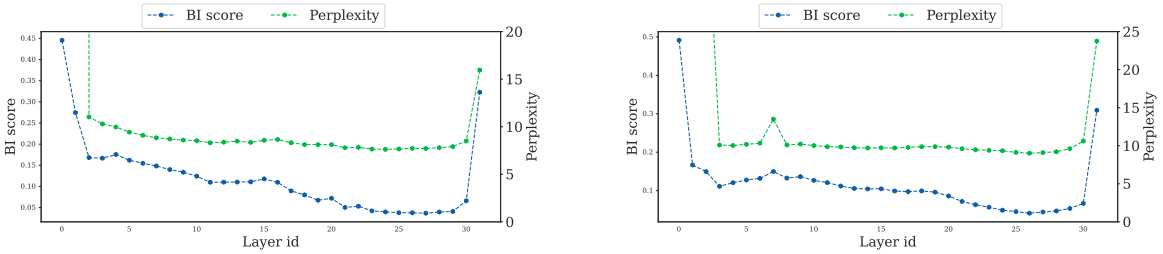
2.2 Layer redundancy

As discussed in the previous subsection, we speculate that LLMs exhibit layer redundancy. To verify this, we assess the performance degradation caused by removing individual layers of two popular models, Llama2-7B (Touvron et al., 2023), an English-based LLMs, and Baichuan2-7B (Yang et al., 2023) which mainly focuses on Chinese. Figure 2 confirms our speculation, where some layers do not play a crucial role in LLMs, causing little degradation when omitting them individually. Moreover, this redundancy is primarily manifested in the middle to later layers, with the initial layers and the last layer often being more critical. Notably, we found the last layer to be particularly important, aligning with previous works (Ma et al., 2024; Namburi et al., 2023; Mitchell et al., 2022). We posit that this discrepancy arises because the final FFN functions as part of the token classifier and should be considered in conjunction with the language model head. To verify this, we conduct further investigation, detailed in Table 1. The results show that within the last layer, the FFN is crucial, while the attention module is less significant, supporting our interpretation of the final layer's importance.



(a) Perpelxity (b) MMLU

Figure 2: Performance of removing certain layer from LLMs. We can see that certain layers are redundant, and their removal results in minimal performance degradation. For perplexity calculation, we randomly select 10 text segments of 1k length from each piece of data in PG19 (Rae et al., 2019).



(a) Llama2 7B (b) Baichuan2 7B

Figure 3: The BI score of a layer and the PPL after removing the layer.

Delete	Perplexity
None	7.60
The whole last layer	13.37
Attention of the last layer	7.65
FFN of the last layer	12.35

Table 1: Ablation of removing FFN and Attention of Llama2-7B-Base. We sample 100 instances from PG19 (Rae et al., 2019) to calculate perplexity.

3 Methodology

In this section, we begin by introducing Block Influence (BI), a novel metric designed to assess the hidden states transformation of each layer. Leveraging BI, we then detail our methods for different tasks, i.e. ShortGPT for multiple-choice tasks and ShortGPT-gen for generative tasks.

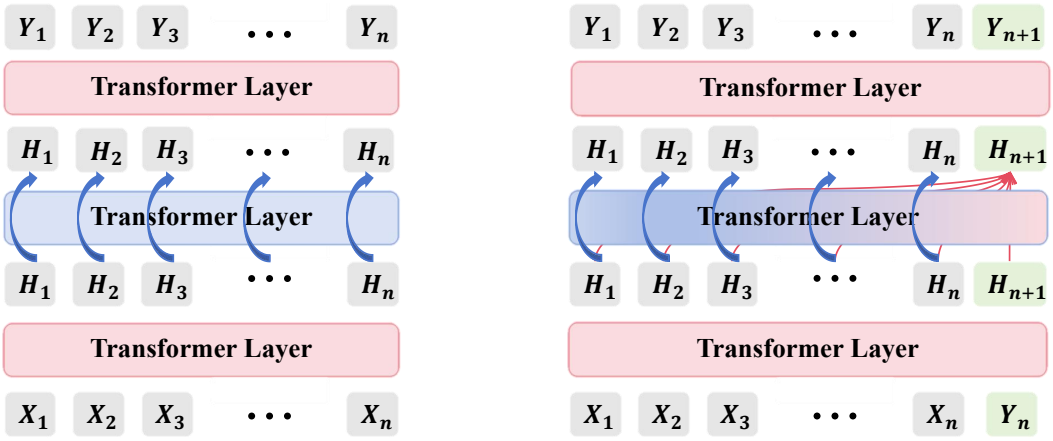
3.1 Layer importance

As outlined above, the layers of LLMs exhibit redundancy. To capture the degree of layer redundancy, we introduce a new metric, Block Influence (BI), to measure the degree of transformation performed by each layer. The BI score of i^{th} layer can be calculated as follows:

$$\mathrm{BI}_i = 1 - \mathbb{E}_{X,t}\frac{X_{i,t}^T X_{i+1,t}}{||X_{i,t}||_2||X_{i+1,t}||_2}, \quad (1)$$

where $X_{i,t}$ means the t^{th} row of hidden states of the i^{th} layer. Lower BI score implies that X_i and X_{i+1} exhibit higher cosine similarity, suggesting that the layer makes minimal transformations to the hidden states and is therefore less important. We plot the BI scores of a single layer and the PPL after removing it separately, as shown in Figure 3. The results demonstrate a positive correlation between the BI score and the importance of a layer.

Why use cosine similarity instead of other similarities? As mentioned above, most LLMs apply layer normalization within their modules. Consequently, the magnitude of hidden states become less important for the output than their direction. Besides cosine similarity, there are some other metrics to measure the similarity between two vectors, e.g. Euclidean distance. However, they are sensitive to the magnitude of vectors and can introduce bias. In contrast, cosine similarity is agnostic to the magnitude, focusing on the direction of vectors.



(a) ShortGPT (b) ShortGPT-gen

Figure 4: Overview of ShortGPT for multiple-choice tasks and ShortGPT-gen for generative tasks. $\{X_1, ..., X_n\}$ represent the input tokens, $\{Y_1, ..., Y_{n+1}\}$ represent the output tokens, and $\{H_1, ..., H_{n+1}\}$ represent the hidden states of tokens.

3.2 Multiple-Choice Task

In typical multi-choice tasks, each choice is concatented with the question to form sentences, and the perplexity of each sentence is computed to select the answer. Since LLMs function as series of transformations applied to hidden states across their layers and we can determine the importance of each layer, we propose a straightforward pruning method: ShortGPT, as illustrated in Figure 4a. We remove certain layers in LLMs based on BI. First of all, we construct a calibration set, which is a set of text samples such as PG19 (Rae et al., 2019). Then we collect the hidden states of each layer during inference on these samples. Next, we calculate the BI scores based on the collected hidden states. Finally, we sort layers in ascending order according to BI, and remove the layers with the lower BI scores. The number of layers to be removed can vary to trade off efficiency and performance.

3.3 Generative Task

Although ShortGPT demonstrates strong capabilities in multiple-choice tasks (see Table 2), it leads to a significant performance drop in generative tasks such as GSM8K (Cobbe et al., 2021), like other pruning methods such as LLMPruner (Ma et al., 2024) and SliceGPT (Ashkboos et al., 2024). We attribute this performance drop to the accumulated errors during generation, compared to multiple-choice tasks. To address this issue, we propose a training-free dynamic pruning method for generative tasks, ShortGPT-gen, as illustrated in Figure 4b. In ShortGPT-gen, the input tokens skip the same redundant layers as in ShortGPT, and the generated tokens pass through all layers. Since the hidden states of input tokens remain unchanged in the pruned layers, we use their output hidden states of the preceding unpruned layers to obtain keys and values during decoding.

4 Experiments

4.1 Experimental Setup

Models. To validate the effectiveness of our methods, we conduct experiments on existing popular open-source language models, including Llama2-7B (Touvron et al., 2023), Llama2-13B, Baichuan2-7B, and Baichuan2-13B. They are all large language models based on the decoder-only Transformer architecture. LLaMA2 was trained on more than 2 trillion tokens. Baichuan2 was mainly trained in Chinese and its 13-Billion model replaced the RoPE (Su et al., 2024) positional embedding with ALiBi (Press et al., 2021).

Benchmarks. In order to evaluate the changes in the ability of large language models before and after pruning, we conduct comprehensive evaluation from five aspect of multiple-choice tasks: **Reasoning**: CMNLI (Li et al., 2024), HellaSwag (HeSw) (Zellers et al., 2019), PIQA (Bisk et al., 2020). **Language**: CHID (Zheng et al., 2019). **Knowledge**: CommonSenseQA (CoQA) (Reddy et al., 2019), BoolQ (Clark et al., 2019). **Examination**: MMLU (Hendrycks et al., 2020), CMMLU (Li et al., 2024). **Understanding**: Race-High/Middle (H/M) (Lai et al., 2017), C3 (Sun et al., 2020) and PG19 (Rae et al., 2019). For generative tasks,

LLM	Method	Ratio	Benchmarks											Avg.	Per.
			CMNLI	HeSw	PIQA	CHID	CoQA	BoolQ	Race-H	Race-M	C3	MMLU	CMMLU		
Llama2 7B	Dense	0.0%	32.99	71.26	77.91	41.66	64.62	71.62	35.71	34.19	43.56	45.39	32.92	50.17	100.00
	LLMPruner	27.0%	**34.33**	**56.46**	**71.22**	25.25	42.51	55.20	22.56	22.35	25.64	23.33	25.25	36.74	73.23
	SliceGPT	26.4%	31.70	50.27	66.21	20.79	41.36	38.32	21.07	21.66	**39.78**	28.92	25.37	35.04	69.84
	LaCo	27.1%	34.43	55.69	69.80	**36.14**	45.70	64.07	22.61	23.61	39.67	26.45	25.24	40.31	80.35
	ShortGPT	27.1%	32.95	53.02	66.43	24.68	**47.99**	**74.71**	**32.25**	**35.17**	39.62	**43.96**	**32.25**	**43.91**	**87.52**
Llama2 13B	Dense	0.0%	32.99	74.78	79.71	47.35	66.91	82.39	57.95	60.38	47.51	55.00	38.40	58.49	100.00
	LLMPruner	24.4%	**33.03**	**67.76**	**76.66**	35.64	50.86	56.42	22.47	22.08	32.33	25.21	24.71	40.65	69.50
	SliceGPT	23.6%	29.82	55.71	69.04	19.31	47.26	37.86	23.41	24.03	41.92	37.14	25.79	37.39	63.93
	LaCo	24.6%	32.86	64.39	63.20	**40.10**	52.66	**63.98**	54.49	56.55	44.93	45.93	32.62	50.16	85.76
	ShortGPT	24.6%	33.00	66.64	73.45	36.61	**58.64**	62.48	**58.35**	**60.17**	**46.90**	**54.69**	**38.38**	**53.57**	**91.59**
Baichuan2 7B	Dense	0.0%	33.37	67.56	76.17	85.56	63.14	74.10	52.63	51.04	64.55	53.87	56.95	61.72	100.00
	LLMPruner	24.2%	32.28	53.66	**71.82**	69.80	**47.83**	61.19	21.96	22.28	41.64	24.93	25.69	43.01	69.69
	SliceGPT	22.2%	32.07	25.29	50.33	14.85	19.57	39.30	23.53	22.49	26.58	25.18	25.25	27.68	44.85
	LaCo	24.2%	33.00	52.28	68.50	**76.24**	47.26	56.15	28.99	27.72	50.85	31.53	31.24	45.80	74.21
	ShortGPT	24.2%	**33.30**	**56.96**	67.68	65.63	46.70	**67.83**	**53.26**	**46.76**	**56.33**	**45.77**	**47.87**	**53.46**	**86.62**
Baichuan2 13B	Dense	0.0%	33.21	71.10	78.07	86.51	65.60	77.89	67.27	68.94	65.64	59.50	61.30	66.82	100.00
	LLMPruner	24.3%	**33.80**	53.57	**71.82**	72.77	38.82	56.54	21.17	21.61	39.89	23.19	25.18	41.67	62.36
	SliceGPT	22.8%	32.07	25.85	51.03	10.40	18.02	37.83	21.56	21.52	24.99	22.95	25.26	26.50	39.66
	LaCo	24.7%	33.03	**60.71**	68.88	76.73	**55.45**	62.35	**56.92**	**57.80**	**61.10**	51.35	53.65	58.00	86.80
	ShortGPT	24.7%	32.81	60.55	**71.60**	**80.17**	54.30	**62.54**	55.77	56.41	60.16	**52.11**	**58.86**	**58.66**	**87.79**

Table 2: Comparison of pruning methods on multiple-choice natural language benchmarks. The last column reports the relative performance retention.

we use three benchmarks, including Xsum (Hasan et al., 2021), GSM8K (Cobbe et al., 2021) and StrategyQA (Geva et al., 2021).

Baselines. To evaluate the effectiveness of our methods, we compared several structured pruning methods for large language models, including:

1) LLMPruner (Ma et al., 2024), which adopts structured pruning that selectively removes non-critical coupled structures based on gradient information, maximally preserving the majority of the LLM's functionality. LLMPruner applies post training to the pruned model, but for fair comparison, we do not apply post training to it.

2) SliceGPT (Ashkboos et al., 2024), which is a post-training scheme that replaces each weight matrix with a smaller matrix, reducing the embedding dimension of the network. Specifically, it applies PCA to the hidden representation from shallow to deep layers, and incorporates the dimension reduction matrix into the existing network.

3) LaCo (Yang et al., 2024), which is a pruning method for large language models based on reducing layers. LaCo gradually merges similar layers from deep to shallow and sets a threshold to avoid continuously merging too many layers.

4.2 Main Results

We conduct comparative experiments against baselines commonly employed in large language models. Considering the current structured pruning methods generally use reduce ratios no more than 30% (Ma et al., 2024; Ashkboos et al., 2024; Yang et al., 2024; Namburi et al., 2023; Michel et al., 2019; Gordon et al., 2020), we perform experiments with approximately 25% of the layers pruned. We use PG19 as the calibration dataset for layer importance and perplexity calculation. We list the removed layers and investigate the effects of calibration datasets in Appendix A.

LLM	Method	Benchmarks			Avg.	Per.
		Xsum	GSM8K	StrategyQA		
Llama2 7B	Dense	18.54	15.69	51.53	28.59	100.00
	LLMPruner	11.51	0.61	44.20	18.77	65.65
	SliceGPT	4.89	3.34	45.70	17.98	62.89
	LaCo*	13.01	1.44	39.08	17.84	62.40
	ShortGPT	11.13	1.60	30.26	14.33	50.12
	ShortGPT-gen	**14.88**	**14.71**	**48.95**	**26.18**	**91.57**
Llama2 13B	Dense	22.19	28.89	63.58	38.22	100.00
	LLMPruner	19.17	1.90	43.70	21.59	56.49
	SliceGPT	5.27	1.90	38.30	15.16	39.67
	LaCo*	15.18	2.58	43.88	20.55	53.77
	ShortGPT	17.59	2.35	46.00	21.98	57.51
	ShortGPT-gen	**21.76**	**26.91**	**62.70**	**37.12**	**97.12**

Table 3: Evaluation on generative benchmarks. "*" indicates our reproduced results.

The experimental results of the multiple-choice tasks are presented in Table 2. The evaluation on the generative benchmarks are shown in Table 3. Additional experiments exploring different parameter reduction proportions will be discussed in the subsequent sections.

The results demonstrate that our pruning methods surpass the baseline methods, maintaining most of the models' capabilities. Furthermore, we note that the approach of reducing layers (ShortGPT/LaCo) outperforms the method of reducing the embedding dimensions (LLMPruner/SliceGPT) in multiple-choice tasks. Further experimental anal-

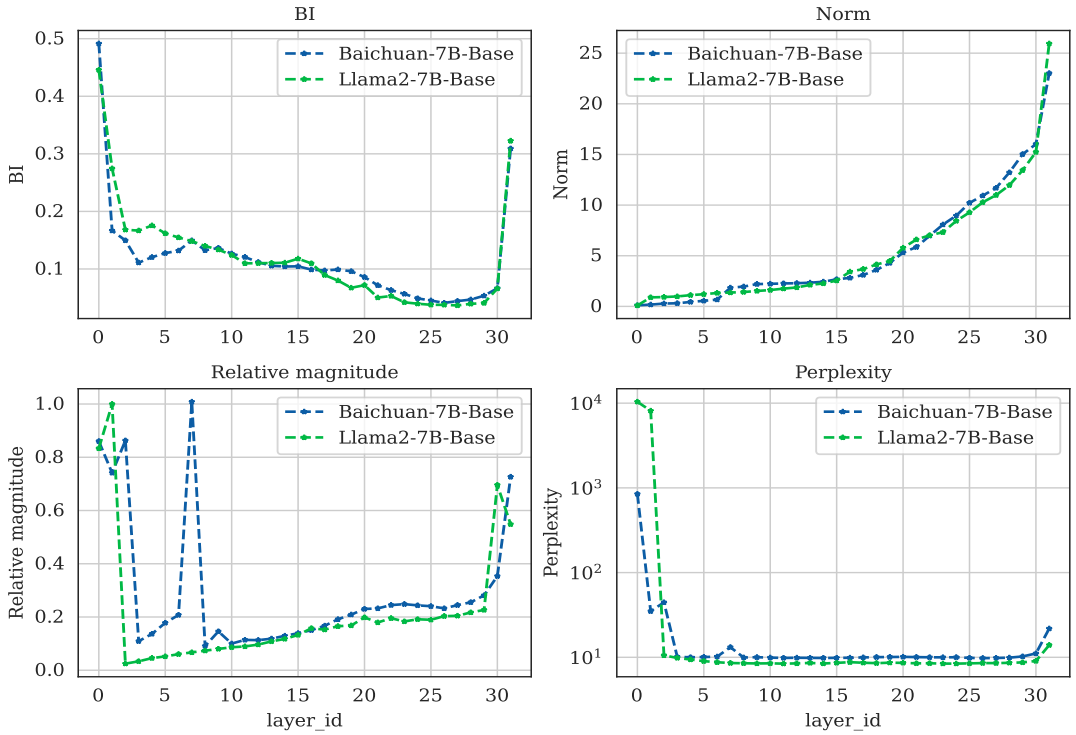


Figure 5: Comparison of different importance metrics on ShortGPT. Perplexity is calculated by removing each single layer, other metrics are calculated by hidden states of each layer.

ysis will be presented in the ensuing sections.

In order to make a more fair comparison with LLMPruner and SliceGPT, we compared them with the same settings as their original papers in Appendix C. These experiments further demonstrate that ShortGPT achieves superior performance compared to the other pruning methods.

The results show that coarse-grained pruning methods, such as removing layers, often outperform fine-grained methods like SliceGPT or LLM-Pruner. We speculate that the reason is that the LLM is actually very robust, as shown in Figure 2, removing any deep layer individually actually has very little impact on the final output, which means it is difficult to define the importance of a finer-grained module and perform pruning.

4.3 Varying metric and pruning ratio

The core principle of our methods is to rank layers by their importance and remove the less significant ones. The choice of importance metric significantly influences the outcome. In this section, we define and compare several different importance metrics:

- **Sequential**: The importance is directly proportional to the sequence order, with shallower layers being less important. This can be implemented by assigning the negative value of each layer's index as its importance metric.
- **Norm/Reverse-order**: This metric posits that importance is inversely proportional to the sequence order. It assigns higher importance scores to the shallower layers. This method gives the same order as measuring importance by hidden states norm as Figure 5 shows.
- **Relative Magnitude**: Proposed in (Samragh et al., 2023), this metric assumes layers with larger $||\frac{f(x)}{x+f(x)}||$ are of higher importance, where f is the layer transformation function.
- **BI**: we calculate the BI score mentioned in Section 3.1 as importance metric.

Figure 5 shows the different metrics. We observe that shallower layers in the LLM are more crucial than deeper ones. Figure 6 shows the results of removing layers by different metrics, demonstrating that our proposed BI outperforms other metrics. The method of Relative Magnitude is highly competitive, indicating that relative values can also reflect the importance to some extent. It is worth noting that the hidden states norm seems to be a good metric when only considering the MMLU benchmark, but the perplexity is relatively poor.

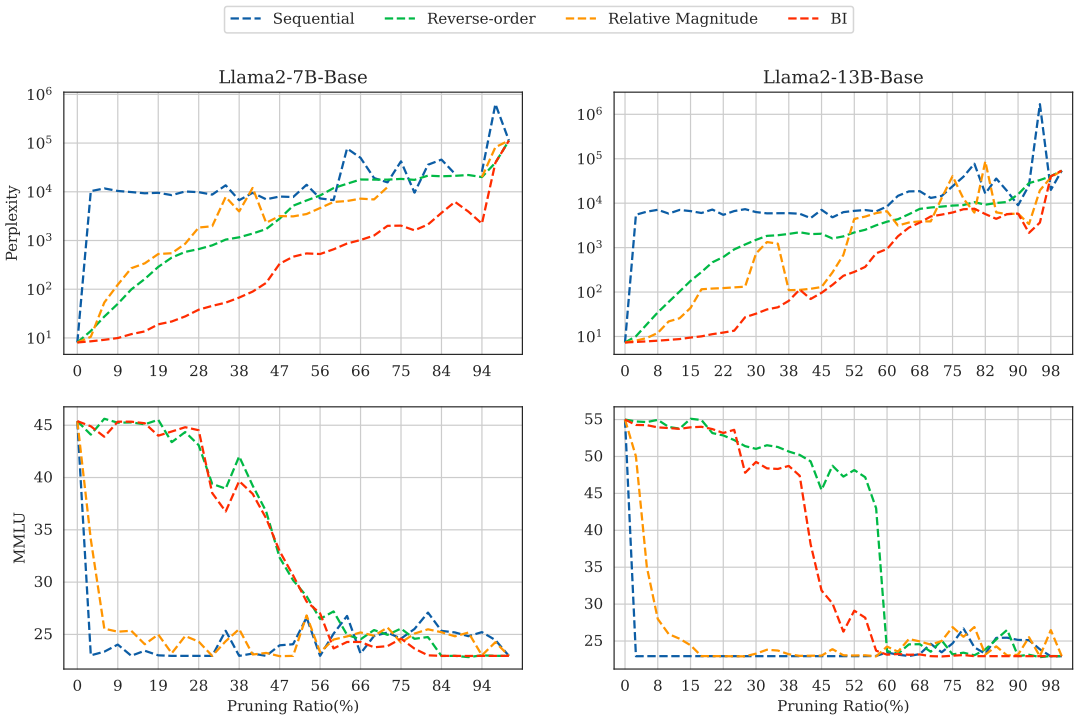


Figure 6: Performance of ShortGPT on MMLU and perplexity when we prune by different metrics, with increasing pruning ratio. We can see that as the pruning ratio increases, the performance of the model declines.

Model	Ratio	CMNLI	HeSw	PIQA	CHID	CoQA	BoolQ	Race-H	Race-M	C3	MMLU	CMMLU	Avg.	Per.
Mamba 2.8B	0.0%	35.97	61.84	75.52	35.56	56.35	60.67	24.90	25.30	42.08	26.29	25.32	42.71	100.00
	10.9%	32.95	59.71	73.01	32.52	52.66	51.41	24.27	25.21	41.10	26.01	25.00	40.35	94.47
	20.3%	31.29	55.69	69.64	29.12	48.32	62.20	23.61	23.61	41.59	25.69	25.37	39.65	92.84
	25.0%	29.96	52.38	68.77	26.02	44.96	62.20	23.67	23.26	40.71	24.32	24.89	38.29	90.37
	31.3%	28.25	47.02	64.91	21.38	44.96	62.17	21.87	22.77	40.44	24.48	24.77	36.64	85.79
RWKV 7B	0.0%	32.07	65.98	77.09	85.36	62.65	62.72	38.56	45.47	57.97	31.85	28.54	53.48	100.00
	9.4%	32.60	56.41	73.94	78.12	49.55	62.35	25.90	25.77	54.68	27.29	25.03	46.51	86.97
	18.8%	32.11	49.47	71.55	65.63	40.54	61.19	22.04	23.75	49.15	26.35	25.00	42.43	79.34
	25.0%	32.41	39.73	65.13	52.60	29.65	60.92	22.56	21.59	41.86	25.52	25.08	37.91	70.89
	28.1%	33.11	32.22	60.01	32.47	28.34	60.85	22.27	21.31	37.81	25.64	25.15	34.47	64.45

Table 4: ShortGPT on Mamba and RWKV.

We further validated the effects of different pruning ratios on ShortGPT. Experiments are conducted on the Llama2 and Baichuan2 models, observing the perplexity and MMLU. The results for Llama2, as shown in Figure 6, demonstrate that the models' performance generally declines as the pruning ratio increases. However, we observe a notable phenomenon: the MMLU score exhibits a sharp drop at a specific layer. This sudden decrease suggests the presence of certain critical layers within the network that play a particularly important role in maintaining performance. Similar patterns are observed in the Baichuan2 models, as illustrated in Appendix B.

4.4 Redundancy on non-transformer LLMs

To determine whether the observed layer redundancy is specific to Transformers, we extend our investigation to include two popular non-Transformer models, RWKV-7B (Peng et al., 2023) and Mamba-2.8B (Gu and Dao, 2023). Our experiments reveal that these models also exhibit resilience to layer removal, maintaining performance despite the elimination of certain layers. Table 4 shows that ShortGPT is applicable and effective for Mamba and RWKV models, suggesting that the redundancy is universal across current LLMs.

4.5 Post training to restore performance

To mitigate the performance loss resulting from layer removal in ShortGPT, we explore post-

Method	Ratio	CMNLI	HeSw	PIQA	CHID	CoQA	BoolQ	Race-H	Race-M	C3	MMLU	CMMLU	Avg.
Dense	0.0%	32.99	71.26	77.91	41.66	64.62	71.62	35.71	34.19	43.56	45.39	32.92	50.17
ShortGPT	27.1%	32.95	53.02	66.43	24.68	47.99	74.41	32.25	35.17	39.62	43.96	32.25	43.88
ShortGPT+post-train	24.0%	32.99	54.83	68.12	31.82	58.32	72.36	34.18	34.68	40.37	44.47	32.73	45.90

Table 5: Post training Llama2-7B to restore performance.

Method	Ratio/Layer	Perplexity	MMLU	Throughput (speed up)
Baseline	0.0%/32	8.03	43.17	4331.23 Token/s (1.00x)
ShortGPT	3.1%/31	8.37	42.88	4399.31 Token/s (1.02x)
	9.4%/29	9.44	42.31	4602.26 Token/s (1.06x)
	12.5%/28	10.24	41.62	4680.68 Token/s (1.08x)
	15.6%/27	11.42	43.17	4756.94 Token/s (1.10x)
	25.0%/24	22.29	41.68	5045.59 Token/s (1.16x)
	27.1%/23	40.78	43.35	5146.99 Token/s (1.19x)

Table 6: ShortGPT on Llama2-7B-Base-GPTQ.

Method	MMLU	CMMLU
Llama2-7B-Baseline	45.4	32.9
4-bit quantization	44.9	32.5
ShortGPT (27.1%)	44.0	32.3
4-bit quantization then ShortGPT	42.4	31.0
ShortGPT then 4-bit quantization	41.2	30.5

Table 7: Performance comparison of different methods

training strategies inspired by (Chen et al., 2024). Our approach consists of two key steps: 1) Replacement: We substitute the removed layers with lightweight Multi-Layer Perceptrons (MLPs). 2) Retraining: We subsequently retrain the modified model. The results in Table 5 demonstrate the potential of post training to recover performance loss. Appendix D shows the post training settings.

4.6 Orthogonal to Quantization

In this section, we show that ShortGPT is orthogonal to quantization. We apply ShortGPT to Llama2-7B quantized by GPTQ algorithm. Table 6 shows that ShortGPT is compatible with quantization, and applying it to the quantized model can further improve its efficiency. In addition, we compare the performance of the model applied ShortGPT before and after quantization. The results shown in Table 7 further indicates that quantization and ShortGPT are orthogonal operations.

5 Related works

Pruning: pruning (LeCun et al., 1989; Han et al., 2015) aims to reduce model redundancy, including static pruning and dynamic pruning. Static pruning (Dong et al., 2017; Fang et al., 2023) removes model components based on various metrics. LLM-Pruner (Ma et al., 2024) removes structures according to gradient information. LaCo (Yang et al., 2024) uses layer merging. Dynamic pruning (Tang et al., 2021; Hua et al., 2019; Gao et al., 2019) preserves the entire model and accelerates models by skipping unimportant components. Dynamic pruning methods typically do not perform fine-tuning or retraining (Cheng et al., 2024). Ada-Infer (Fan et al., 2024) introduces early-exit to stop forward propagation at intermediate layers, but its performance on generative benchmarks such as GSM8K decreases to nearly zero. In our methods, ShortGPT is a static pruning method, while ShortGPT-gen is a dynamic pruning method. Notably, ShortGPT-gen preserves most of the performance of dense models on generative tasks in a training-free manner.

Quantization: quantization (Liu et al., 2021; Gholami et al., 2022; Dettmers et al., 2022, 2024) can save computational costs of deep learning models. It converts models' floating-point weights into integers or other discrete forms. LUT-GEMM (Park et al., 2022) quantifies weights and optimizes matrix multiplication in LLMs using BCQ format. SPQR (Dettmers et al., 2023) identifies and isolates abnormal weights, stores them with higher accuracy and converts others into 3-4 bits. Our pruning methods and quantization method are orthogonal.

Model redundancy: researchers have long noticed significant redundancy in nonlinear models (Catchpole and Morgan, 1997). Recently, Transformer models have been widely applied. Researchers have also studied their redundancy. In (Bian et al., 2021), researchers analyzed redundancy in attention mechanisms, where similar redundancy patterns are observed among attention heads. In (Dalvi et al., 2020), researchers dissect BERT (Devlin et al., 2018) and XLNet (Yang et al., 2019), studying how much redundancy they exhibit at representation level and the fine-grained neuron level. However, the redundancy of decoder-only LLMs still needs to be explored.

6 Conclusion

In this work, we uncovered the significant layer-wise redundancy of LLMs, Our research demonstrates that certain layers contribute minimally to

overall network functionality and can be removed without substantially compromising model performance. Based on our observation, we introduce Block influence to quantify the importance of each layer and propose simple and straightforward pruning methods for multiple-choice tasks and generative tasks. Our experiments demonstrates that it is possible to maintain approximately 90% of a LLM's performance while reducing the computational requirements in approximately 25% of the LLM's layers. Our work suggests potential avenues for improving the efficiency of models by reducing their inherent redundancy.

7 Limitations

Although our methods demonstrates strong competitiveness compared to current methods, there are some limitations that must be acknowledged. The negative effect of ShortGPT is significant on certain generative tasks, which creates barriers to its practical use. When we remove 25% layers from Llama2-7B or Baichuan2-7B, the performance on GSM8K deceases to nearly zero. In contrast, ShortGPT-gen preserves most of models' performance on generative tasks, but it does not reduce the sizes of models.

Acknowledgments

We sincerely thank the reviewers for their insightful comments and valuable suggestions. This work was supported by Beijing Natural Science Foundation (L243006), Beijing Municipal Science and Technology Project (Nos. Z231100010323002), the Natural Science Foundation of China (No. 62476265, 62306303).

References

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*.

Yuchen Bian, Jiaji Huang, Xingyu Cai, Jiahong Yuan, and Kenneth Church. 2021. On attention redundancy: A comprehensive study. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 930–945.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, pages 7432–7439.

Edward A Catchpole and Byron JT Morgan. 1997. Detecting parameter redundancy. *Biometrika*, 84(1):187–196.

Xiaodong Chen, Yuxuan Hu, and Jing Zhang. 2024. Compressing large language models by streamlining the unimportant layer. *arXiv preprint arXiv:2403.19135*.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. 2024. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *arXiv preprint arXiv:2308.06767*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Fahim Dalvi, Hassan Sajjad, Nadir Durrani, and Yonatan Belinkov. 2020. Analyzing redundancy in pretrained transformer models. *arXiv preprint arXiv:2004.04010*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. 2017. More is less: A more complicated network with less inference complexity. *arXiv preprint arXiv:1703.08651*.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*.

Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. *arXiv preprint arXiv:2301.12900*.

Elias Frantar and Dan Alistarh. 2023. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*.

Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. 2019. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *arXiv preprint arXiv:2101.02235*.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC.

Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Tahmid Hasan, Abhik Bhattacharjee, Md Saiful Islam, Kazi Mubasshir, Yuan-Fang Li, Yong-Bin Kang, M Sohel Rahman, and Rifat Shahriyar. 2021. Xl-sum: Large-scale multilingual abstractive summarization for 44 languages. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4693–4703.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G. Edward Suh. 2019. Channel gating neural networks. *arXiv preprint arXiv:1805.12549*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. 2024. Cmmlu: Measuring massive multitask language understanding in chinese. *arXiv preprint arXiv:2306.09212*.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763.

Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. 2021. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems*, 34:28092–28103.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2024. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*.

Satya Sai Srinath Namburi, Makesh Sreedhar, Srinath Srinivasan, and Frederic Sala. 2023. The cost of compression: Investigating the impact of compression on parametric knowledge in language models. *arXiv preprint arXiv:2312.00960*.

Gunho Park, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2022. nuqmm: Quantized matmul for efficient inference of large-scale generative language models. *arXiv preprint arXiv:2206.09557*.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al.

2023. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*.

Ofir Press, Noah A Smith, and Mike Lewis. 2021. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*.

Siva Reddy, Danqi Chen, and Christopher D Manning. 2019. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

Mohammad Samragh, Mehrdad Farajtabar, Sachin Mehta, Raviteja Vemulapalli, Fartash Faghri, Devang Naik, Oncel Tuzel, and Mohammad Rastegari. 2023. Weight subcloning: direct initialization of transformers using larger pretrained ones. *arXiv preprint arXiv:2312.09299*.

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

Kai Sun, Dian Yu, Dong Yu, and Claire Cardie. 2020. Investigating prior knowledge for challenging chinese machine reading comprehension. *Transactions of the Association for Computational Linguistics*, 8:141–155.

Yehui Tang, Yunhe Wang, Yixing Xu, Yiping Deng, Chao Xu, Dacheng Tao, and Chang Xu. 2021. Manifold regularized dynamic network pruning. *arXiv preprint arXiv:2103.05861*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. 2024. Deepnet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Mingyang Zhang, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, Bohan Zhuang, et al. 2023. Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*.

Chujie Zheng, Minlie Huang, and Aixin Sun. 2019. Chid: A large-scale chinese idiom dataset for cloze test. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 778–787.

A Detailed Strategies for Layer Removal

We list different layer removal strategies in Table 8. The removed layers in Table 2 and Table 3 are listed in Table 9. To investigate the effects of calibration datasets, we also list the removed layers of Llama2 models when using MMLU as calibration dataset in Table 10. We can observe that the calibrations datasets hardly affect the removed layers, demonstrating the robust of our BI metric.

B ShortGPT on Baichuan2 models

The effects of different pruning ratios on Baichuan2 models' performance are shown in Figure 7.

C A Fair comparison with SliceGPT and LLMPruner

For a fair comparison with LLMPruner and SliceGPT, we do the same experiments as in the original papers of LLMPruner and SliceGPT. The results are provided in Table 11 and Table 12. We take the same settings as the corresponding paper. The results demonstrate that our method is highly competitive.

D Setup for post training

Table 13 shows the post training settings.

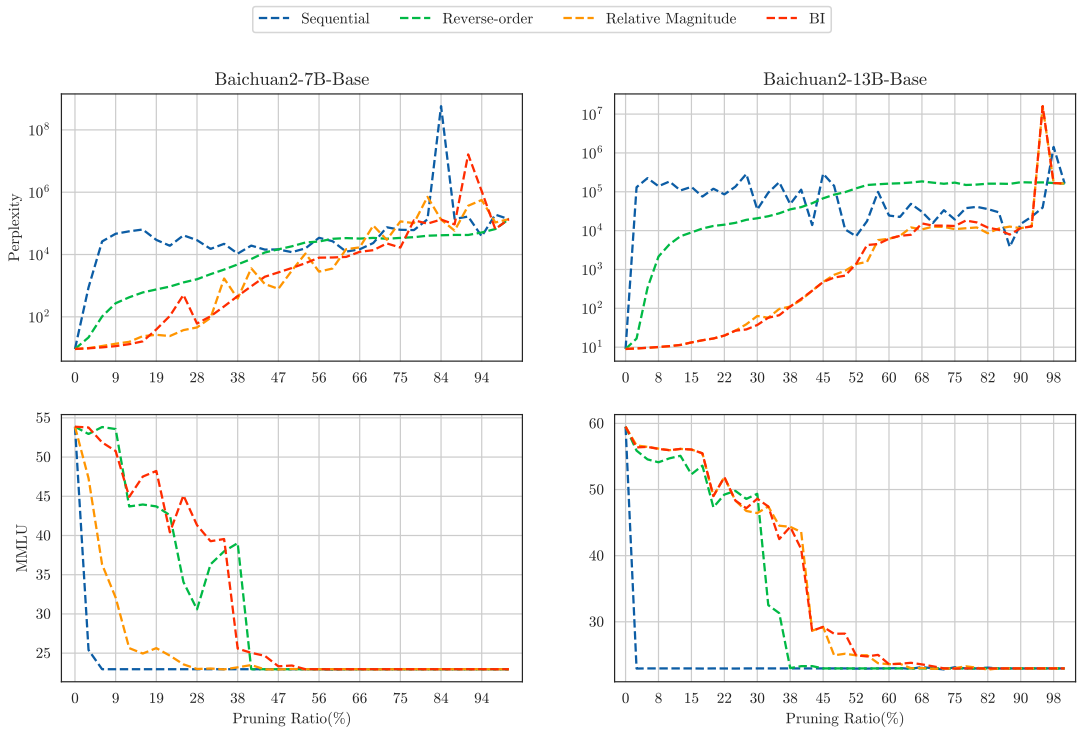


Figure 7: Pruning by different pruning ratios on Baichuan2 models.

Strategy	Description
Sequential	Layers are removed sequentially from the beginning of the model. The process starts with layer 0 and progressively includes more layers for removal (e.g., {0}, {0, 1}, . . .).
Reverse-order	This strategy involves starting from the model's final layer and progressively removing layers in reverse order (e.g., {-1}, {-1, -2}, . . .).
Relative Magnitude	Layers are removed in ascending order based on their Relative Magnitude values. The process accumulates layers from those with the smallest to the largest values, mirroring the sequential strategy's accumulation method.
BI (Block Influence)	A similar accumulation approach as the Sequential strategy, but layers are ordered and removed according to their BI scores, starting from the lowest and moving to the highest.

Table 8: Strategies for Layer Removal in Models.

Model	Removed Layers
Llama2-7B	21, 22, 23, 24, 25, 26, 27, 28, 29
Llama2-13B	26, 27, 28, 29, 30, 31, 32, 33, 34, 35
Baichuan2-7B	22, 23, 24, 25, 26, 27, 28, 29, 30
Baichuan2-13B	26, 27, 28, 29, 30, 31, 32, 33, 34, 35

Table 9: Removed Layers for Benchmark Models, using PG19 as calibration dataset.

Model	Removed Layers
Llama2-7B	21, 22, 23, 24, 25, 26, 27, 28, 29
Llama2-13B	26, 27, 28, 29, 30, 31, 32, 33, 34, 35

Table 10: Removed Layers for Llama2 models, using MMLU as calibration dataset.

E Setup for training post-norm model and pre-norm model

Table 14 lists the specific training settings for pre norm and post norm in Section 2.1.

Model	Pruning ratio	Method	BoolQ	PIQA	Hellaswag	Winogrande	Arc-e	Arc-c	OBQA	Avg.
Llama 7B	Ratio=0%	Baseline	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
	Ratio=20%	LLMPruner	59.39	75.57	65.34	61.33	59.18	37.12	39.80	56.82
	Ratio=21.9 %	ShortGPT	68.26	72.28	61.70	63.77	60.22	39.00	41.60	58.12
Llama 13B	Ratio=0%	Baseline	68.47	78.89	76.24	70.09	74.58	44.54	42.00	64.97
	Ratio=20%	LLMPruner	67.68	77.15	73.41	65.11	68.35	38.40	42.40	61.79
	Ratio=20%	ShortGPT	68.41	76.36	72.90	67.40	68.62	39.20	41.00	61.98

Table 11: Comparison between ShortGPT and LLMPruner. The Table is corresponding to the Table 1 of LLM-Pruner (Zhang et al., 2023).

Model	Pruning ratio	Method	PIQA	Hellaswag	Winogrande	Arc-e	Arc-c	Avg.
Llama2 7B	0%	Baseline	79.11	75.99	69.06	74.58	46.25	69.00
	20%	SliceGPT	71.87	58.10	63.04	69.87	43.09	61.19
	25%	SliceGPT	68.55	58.10	62.04	57.46	35.07	56.24
	30%	SliceGPT	66.10	52.69	56.82	35.07	56.82	53.50
	21.9%	ShortGPT	72.76	66.39	66.27	59.39	39.85	60.93
	25%	ShortGPT	70.53	62.68	64.70	58.39	39.51	59.16
	31.6%	ShortGPT	67.87	62.19	64.38	56.57	40.86	58.37
Llama2 13B	0%	Baseline	80.47	79.39	72.22	77.48	49.23	71.76
	20%	SliceGPT	71.87	69.38	63.04	69.87	43.09	63.45
	25%	SliceGPT	68.55	67.48	58.10	62.50	37.88	58.90
	30%	SliceGPT	66.10	65.11	52.69	56.82	35.07	55.16
	20%	ShortGPT	76.95	74.67	71.14	69.56	45.63	67.59
	25%	ShortGPT	74.39	71.65	70.98	67.09	43.93	65.61
	30%	ShortGPT	72.11	71.93	67.19	61.09	40.88	62.64
Llama2 70B	0%	Baseline	82.70	83.84	77.98	80.98	57.34	76.57
	20%	SliceGPT	76.61	72.98	74.92	80.51	55.20	72.04
	25%	SliceGPT	74.92	68.74	74.92	77.90	51.71	69.64
	30%	SliceGPT	72.31	63.69	73.40	51.71	47.61	61.74
	20%	ShortGPT	76.02	78.87	71.69	76.02	52.95	71.11
	25%	ShortGPT	73.20	76.72	71.85	73.20	49.90	68.97
	30%	ShortGPT	74.44	75.31	72.33	74.44	49.22	69.15

Table 12: Comparison between ShortGPT and SliceGPT. The Table is corresponding to the Table 7 of SliceGPT (Ashkboos et al., 2024).

Parameter	**Value**
Global Batch Size	2048
Sequence length	4096
Precision	bf16
Learning Rate Scheduler	cosine
Max Learning Rate	2e-5
Min Learning Rate	1e-5
Warm-up steps	3000
Training Tokens	50B
Weight Decay	0.1
Adam Beta1	0.9
Adam Beta2	0.98
Gradient Clip	1.0

Table 13: Post training parameters.

Parameter	**Value**
Global Batch Size	2048
Sequence length	4096
Precision	bf16
Learning Rate Scheduler	cosine
Max Learning Rate	4e-4
Min Learning Rate	5e-5
Warm-up steps	3000
Training Tokens	200B
Weight Decay	0.1
Adam Beta1	0.9
Adam Beta2	0.98
Gradient Clip	1.0
Tokenizer	Llama2
Layers	32
Hidden state	2048
Attention heads	32
Head dim	64
FFN size	5504
Activation function	Silu

Table 14: Training parameters.