BlockPruner: Fine-grained Pruning for Large Language Models

Anonymous ACL submission

Abstract

With the rapid growth in the size and complexity of large language models (LLMs), the costs associated with their training and inference have escalated significantly. Research indicates that certain layers in LLMs harbor substantial redundancy, and pruning these layers has minimal impact on the overall performance. While various layer pruning methods have been developed based on this insight, they generally overlook the finer-grained redundancies within the layers themselves. In this paper, we delve deeper into the architecture of LLMs and demonstrate that finer-grained pruning can be achieved by targeting redundancies in multi-head attention (MHA) and multi-layer perceptron (MLP) blocks. We propose a novel, training-free structured pruning approach called BlockPruner. Unlike existing layer pruning methods, BlockPruner segments each Transformer layer into MHA and MLP blocks. It then assesses the importance of these blocks using perplexity measures and applies a heuristic search for iterative pruning. We applied BlockPruner to LLMs of various sizes and architectures and validated its performance across a wide range of downstream tasks. Experimental results show that BlockPruner achieves more granular and effective pruning compared to state-of-the-art baselines.

1 Introduction

Large language models (LLMs) (Zhao et al., 2023; Minaee et al., 2024) have demonstrated outstanding performance across a diverse array of natural language processing tasks. However, their growing size and complexity have led to substantial computational demands and increased memory usage, creating obstacles for deployment in resource-constrained environments. Model compression techniques (Gao et al., 2020; Li et al., 2023; Wang et al., 2024) have emerged as a promising solution to address the challenges of deploying large, computationally intensive models. These techniques

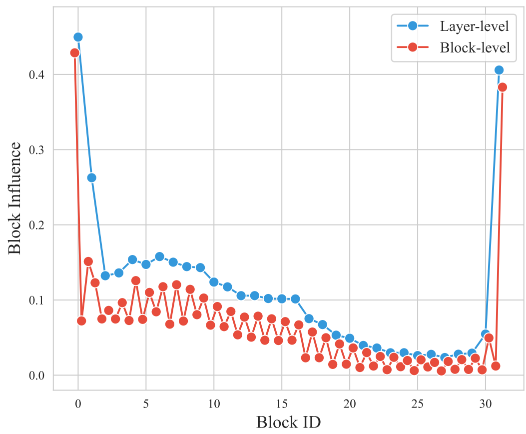

Figure 1: Block Influence (BI) scores (Men et al., 2024) for the Llama2-7B model (Touvron et al., 2023b) computed at both layer and block levels, where blocks/layers with lower BI scores indicate less importance. The model has 32 Transformer layers, each containing one MHA and one MLP block, totaling 64 blocks. Block-level BI scores are generally lower than layer-level scores, indicating finer-grained redundancies.

aim to transform large models into more compact versions that require less storage and execute with lower latency, while minimizing performance degradation. Model compression methods typically involve knowledge distillation (Huang et al., 2022; Gu et al., 2024), quantization (Yao et al., 2022; Dettmers et al., 2023), and pruning (van der Ouderaa et al., 2024; Ashkboos et al., 2024). In this study, we primarily focus on pruning, a technique that can be combined with these other methods to achieve more effective and efficient compression.

Recent research on layer redundancy has shown that LLMs contain a substantial number of redundant layers (Yang et al., 2024; Men et al., 2024; Chen et al., 2024). Removing these layers does not severely impact the model's performance. To quantify this redundancy, researchers have investigated various similarity-based measurement methods and developed corresponding pruning strategies, including layer merging (Yang et al., 2024) and layer removal (Men et al., 2024). These methods not only maintain the original width of the model architecture and avoid introducing additional structures, but also demonstrate superior performance. Furthermore, Gromov et al. (2024) posited that this observed redundancy may be intrinsically linked to the residual structure (He et al., 2016) inherent in the Transformer architecture. Building on this intuition and recognizing that Transformer layers can be further subdivided into smaller residual blocks, namely multi-head attention (MHA) and multi-layer perceptron (MLP)[1], we hypothesize that fine-grained block redundancies could exist within LLMs. Consequently, we conducted a preliminary experiment to assess the significance of blocks at varying granularities. Specifically, we sampled 32 instances from the Alpaca dataset (Taori et al., 2023) and employed the Block Influence (BI) metric (Men et al., 2024) to evaluate blocks at layer and block levels, as depicted in Figure 1. The results reveal that block-level BI scores are generally lower than layer-level BI scores, indicating that fine-grained redundancies at the block level are more significant within the model.

Building on these findings, we argue that finer-grained pruning can be effectively implemented in LLMs. Therefore, we introduce BlockPruner, a novel, training-free structured pruning approach. Unlike existing methods that focus on entire layers, BlockPruner segments each Transformer layer into multi-head attention (MHA) and multi-layer perceptron (MLP) blocks. It then evaluates the importance of these blocks using perplexity measures and applies a heuristic search for iterative pruning.

To validate the effectiveness of our method, we applied BlockPruner to six LLMs of varying sizes and architectures, and evaluated their performance using five representative benchmarks. Our experimental results demonstrate that BlockPruner provides more granular and effective pruning compared to state-of-the-art baselines. Additionally, we performed a series of analytical experiments to investigate the impact of block type, block importance metrics, and data on pruning effectiveness. Our findings confirm that LLMs contain substantial redundancies at the block level compared to the layer level, demonstrating that fine-grained pruning is more effective and appropriate than layer-based approaches for compressing these models.

[1]In this work, unless otherwise specified, we refer to a block as one of the two sublayers: MHA or MLP.

2 Related Work

Pruning is a well-established technique to compress and accelerate neural networks by removing superfluous weights or structures within models. Pruning methods can be broadly categorized into unstructured pruning and structured pruning.

Unstructured pruning. Unstructured pruning targets individual weights, eliminating redundant connections in neural networks by setting the corresponding weights to zero. For instance, SparseGPT (Frantar and Alistarh, 2023) formulates pruning as a layer-wise sparse regression problem, approximately solving it via a sequence of efficient Hessian updates and weight reconstructions. Wanda (Sun et al., 2024) computes the importance score of each weight based on the product of the magnitude of each weight and the norm of the corresponding input activation, identifying and removing weights with lower importance scores. OWL (Yin et al., 2024) identifies the correlation between pruning efficacy and the retention ratio of outliers, assigning different sparsity ratios to each layer based on the observed outlier ratio. RIA (Zhang et al., 2024b) introduces a metric that considers both weight and activation information, utilizing a permutation strategy for the input channels of weight matrices to enhance pruning performance. BESA (Xu et al., 2024) adopts a layer-wise pruning strategy, independently pruning each Transformer layer to minimize the reconstruction error between the outputs of pruned and dense Transformer layers, which avoids accumulating errors across layers.

Structured pruning. Structured pruning focuses on broader network structures, such as neurons, attention heads, or even entire modules. LLM-Pruner (Ma et al., 2023) utilizes gradient information to identify interdependent structures within LLMs, pruning the least important groups and subsequently using Low-Rank Adaptation (LoRA) (Hu et al., 2022) to restore the performance of pruned models. LoRAPrune (Zhang et al., 2023) estimates the importance of pre-trained weights using LoRA gradients, iteratively removing redundant channels in the weight matrices and recovering the pruned models' performance through fine-tuning. Sheared-LLaMA (Xia et al., 2024) learns a set of pruning masks to extract a sub-network with the specified target structure from the source model, employing a dynamic batch loading algorithm to adjust the data proportion of each domain based on the loss reduction rate in different domains. SliceGPT (Ashkboos et al., 2024) introduces the concept of computational invariance, achieving compression by removing rows or columns corresponding to smaller principal components in the weight matrix. LaCo (Yang et al., 2024) proposes a concise layer pruning approach, reducing model size by merging layers while maintaining the overall model structure. ShortGPT (Men et al., 2024) introduces a metric for measuring layer importance, achieving model compression by removing redundant layers.

Although unstructured pruning can maintain performance at higher pruning ratios, it often requires additional hardware or library support, making model acceleration impractical. Current structured pruning methods typically require retraining the model after pruning to avoid performance collapse. While layer pruning techniques like LaCo eliminate the need for additional retraining, their disregard for fine-grained block redundancy makes it challenging to avoid significant performance loss.

Concurrent and independent of our research, FINERCUT (Zhang et al., 2024a) also presents a fine-grained block pruning algorithm. However, their study does not delve into the rationale behind treating Transformer layers as two distinct sublayers for pruning purposes. In contrast, we began by conducting preliminary experiments that unveiled the fine-grained block redundancy within Transformer models. This discovery led us to propose the concept of minimal residual blocks. Furthermore, we explored how pruning various types of blocks affects the model's performance. Furthermore, FINERCUT evaluates block importance by comparing the similarity between the output logits of the initial and pruned models. However, this metric may not adequately ensure that the pruned model generates coherent and semantically correct text, as it overlooks semantic information. In contrast, our approach measures block importance using the perplexity of the pruned model, which better reflects the fluency and quality of its output.

3 Methodology

The proposed fine-grained block pruning method (BlockPruner) is depicted in Figure 3. It begins by decomposing each Transformer layer into two minimal residual blocks (§3.1). We then evaluate the importance of each block using our proposed block importance metric (§3.2). Finally, we iteratively prune the block with the lowest importance (§3.3).

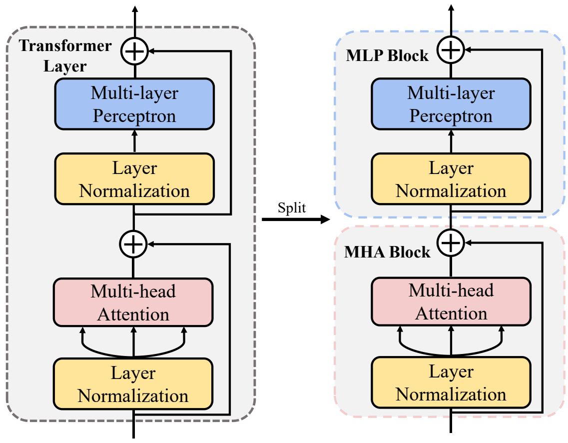

Figure 2: Illustration depicting that a Transformer layer can be subdivided into two residual blocks.

3.1 Minimal Residual Block

Most contemporary LLMs (Brown et al., 2020; Touvron et al., 2023a,b) are built upon the GPT architecture (Radford et al., 2019), which constitutes a decoder-only model comprising multiple Transformer layers, an embedding layer, and a language model head. As depicted in Figure 2, each Transformer layer can be decomposed into two primary residual blocks: the multi-head attention (MHA) block and the multi-layer perceptron (MLP) block.

Formally, consider the input hidden states of the ith Transformer layer, denoted as $X_{i-1} \in \mathbb{R}^{n \times d}$, where n represents the length of the input sequence, and d represents the hidden layer dimension of the model. The computational process within the ith Transformer layer can be represented as follows:

$$X'_i = \mathrm{MHA}(\mathrm{LN}(X_{i-1})) + X_{i-1}, \quad (1)$$

$$X_i = \mathrm{MLP}(\mathrm{LN}(X'_i)) + X'_i. \quad (2)$$

Here, $\mathrm{LN}(\cdot)$ denotes the layer normalization module and $X'_i \in \mathbb{R}^{n \times d}$ represents the intermediate hidden states after the MHA block.

Equations (1) and (2) indicate that both types of residual blocks can be abstracted into a same computational formula. Therefore, intuitively treating them as independent layers for pruning appears reasonable, which will be further validated by our subsequent experimental results.

3.2 Block Importance

While previous layer pruning methods (Men et al., 2024; Chen et al., 2024) rely solely on the similarity between layer inputs and outputs to measure layer importance, we contend that this approach considers only the local influence of the layer while

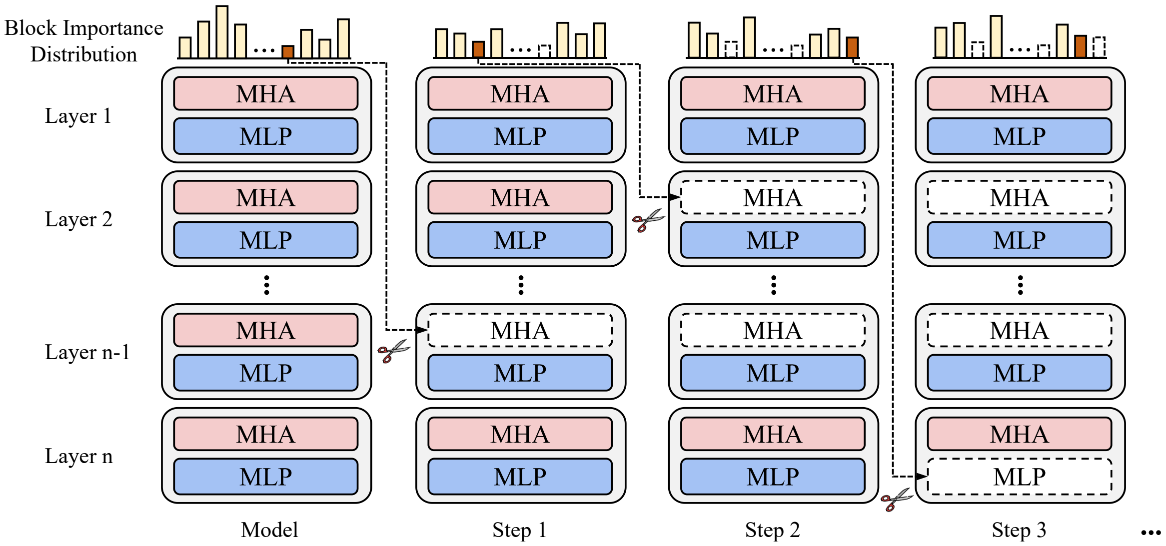

Figure 3: Overview of our BlockPruner. We iteratively calculate the importance score for each block (MHA or MLP) to obtain the block importance distribution, and subsequently remove the block with the lowest importance.

neglecting its role in the overall model's output. To address the drawback, we introduce *perplexity* as a measure of block importance. Specifically, we determine the importance score of each block by masking it and then computing the perplexity of the new model on a given dataset. Intuitively, a block with the lowest importance score indicates that its removal results in minimal performance degradation. This method more effectively captures each block's overall impact on the model's performance, thereby more accurately reflecting its significance.

Mathematically, perplexity is defined as the exponential of the average negative log-likelihood of a sequence of words. Given a sequence of words $w_1, \ldots, w_n$ and a language model that predicts the probability $p_\theta(w_i|w_{<i})$ for each word w_i, the perplexity PPL is calculated as:

$$\mathrm{PPL} = \exp(-\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(w_i|w_{<i})), \quad (3)$$

where $p_\theta(w_i|w_{<i})$ denotes the probability of word w_i given the preceding words in the sequence.

3.3 Iterative Search for Block Pruning

Unlike existing layer pruning techniques, which indiscriminately remove entire Transformer layers, we propose a novel fine-grained pruning strategy. This strategy selectively prunes multi-head attention (MHA) or multi-layer perceptron (MLP) blocks based on their defined importance. By employing this finer-grained pruning approach, we aim to better preserve the critical components and capabilities of the model while aggressively removing the less significant blocks.

For an LLM $\mathcal{M}$ with L layers, we first divide them into $2L$ blocks, consisting of multi-head attention (MHA) and multi-layer perceptron (MLP) blocks. Then, we perform iterative pruning search on a calibration dataset $\mathcal{C}$ to sequentially prune K blocks. The steps are outlined as follows:

Step 1: Mask Block. For each block B_i (MHA or MLP) in $\mathcal{M}$, we generate a modified model $\hat{\mathcal{M}}$ by masking out this block.

Step 2: Calculate Importance. We compute the perplexity P_i for the modified model $\hat{\mathcal{M}}$ on the calibration dataset $\mathcal{C}$ as the importance score for the masked block B_i.

Step 3: Sort and Prune. After computing the importance scores for all blocks, we sort these scores and remove the block with the lowest importance score from $\mathcal{M}$ to create a new model.

Step 4: Iterate. The aforementioned steps are iteratively repeated until K blocks are removed.

By iteratively removing the blocks with the lowest importance scores, we aim to prune the LLM while minimizing performance degradation on the calibration dataset $\mathcal{C}$. This fine-grained block pruning approach provides a more targeted method for pruning LLMs compared to traditional layer-level pruning techniques, thereby facilitating more efficient model compression while better preserving the model's performance. The detailed procedure for this pruning process is outlined in Algorithm 1.

Algorithm 1 Iterative Block Pruning

Input: Model $\mathcal{M}$ with L layers, calibration dataset $\mathcal{C}$, number of blocks to remove K
Output: Pruned model $\mathcal{M}^*$

$\mathcal{M}_0 \leftarrow \mathcal{M}$
Split the model $\mathcal{M}_0$ into $2L$ blocks
for $j = 1$ to K **do**
  for $i = 1$ to $2L - j + 1$ **do**
    Create model $\hat{\mathcal{M}}$ by masking block B_i;
    Compute the perplexity P_i for $\hat{\mathcal{M}}$ on the calibration dataset $\mathcal{C}$;
  end for
  Sort the blocks based on their perplexities;
  Remove the block with the lowest perplexity from $\mathcal{M}_{j-1}$ and obtain $\mathcal{M}_j$;
end for
$\mathcal{M}^* \leftarrow \mathcal{M}_K$
return Pruned model $\mathcal{M}^*$

4 Experiments

In this section, we first introduce the experimental setups and then present the main results.

4.1 Experimental Setups

Models. To validate the widespread effectiveness of our pruning method, we experiment with three series of models: Llama2 (Touvron et al., 2023b), Baichuan2 (Yang et al., 2023), and Qwen1.5 (Bai et al., 2023). These models share analogous architectures as described in equations (1) and (2). Due to computational constraints, we employ 7B and 13B models for Llama2 and Baichuan2, respectively, and 7B and 14B models for Qwen1.5.

Baselines. We compare our method with several state-of-the-art structured pruning methods. The specific baseline methods include **SliceGPT** (Ashkboos et al., 2024), **LaCo** (Yang et al., 2024), **ShortGPT** (Men et al., 2024), and **Relative Magnitude** (Samragh et al., 2023; Men et al., 2024). SliceGPT achieves pruning by removing rows or columns corresponding to smaller principal components in the weight matrix. LaCo merges model layers from deep to shallow, using model output representations to calculate thresholds to avoid over-merging. ShortGPT eliminates redundant layers by calculating Block Influence. Relative Magnitude (RM) uses $||\frac{f(x)}{x+f(x)}||$ as an importance metric for layers, where $f(.)$ represents the non-residual part of the Transformer layer, and employs the same pruning method as ShortGPT. For SliceGPT, we used the official implementation[2]. For LaCo, we implemented it based on their code and controlled the number of pruned layers by adjusting the merging threshold. For ShortGPT and RM, we reproduced the results based on their paper descriptions.

Data and GPUs. In our main experiment, we utilize the Alpaca dataset (Taori et al., 2023) to calculate importance scores. For our method, we employ only 256 samples to compute perplexity, and we discuss the influence of varying sample sizes in Section 5.4. Moreover, we observe that ShortGPT and Relative Magnitude are not sensitive to different numbers of samples. The effect of sample size on ShortGPT and Relative Magnitude is detailed in Appendix E. Nevertheless, we used the same number of samples as our method for consistency. All experiments in this study are conducted using two RTX 4090 GPUs.

Evaluations. Following SliceGPT, we use LM Evaluation Harness (Gao et al., 2023) for evaluation and validation on five well-known benchmarks: PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), ARC-e and ARC-c (Clark et al., 2018). We also utilize Wikitext2 dataset (Merity et al., 2016) for evaluating the perplexity after pruning. More comprehensive details of can be found in Appendix C.

4.2 Main Results

Previous structured pruning methods typically prune less than 30% of the parameters. Therefore, in our main experiments, we controlled the pruning ratio within this range. Since it is challenging to achieve identical pruning ratios across different methods and models, we select the closest available pruning ratios for comparison. The experimental results are presented in Table 1.

As shown in the results, our BlockPruner method significantly outperforms previous structured pruning baselines in terms of average performance and achieves the best results across most benchmarks, even though the pruning ratios in our method are slightly higher than that of baselines. We also observe that Llama2-13B maintains better performance at higher pruning ratios compared to Llama2-7B, with Baichuan2 and Qwen1.5 exhibiting similar behavior. This suggests that as the model scale grows, so does the number of redundant blocks, allowing for more pruning space.

[2]As SliceGPT's official code does not support Baichuan2 and Qwen1.5, we only employ it on the Llama2 series models.

Model	Method	Ratio (%)	PPL (↓)	PIQA	WinoGrande	HellaSwag	ARC-e	ARC-c	Avg. Score
Llama2-7B	Dense	0	5.47	79.05	69.06	75.99	74.54	46.16	68.96
	SliceGPT	21.45	30.74	72.42	59.91	56.04	**63.64**	37.12	57.83
	LaCo	21.02	50.39	68.34	60.46	54.08	55.39	35.84	54.82
	RM	21.02	676.80	54.46	49.25	29.22	34.43	22.53	37.98
	ShortGPT	21.02	18.45	70.24	**65.90**	62.63	56.06	36.09	58.18
	BlockPruner	21.99	**11.51**	**74.21**	62.43	**65.87**	61.07	**37.29**	**60.17**
Llama2-13B	Dense	0	4.89	80.52	72.14	79.36	77.36	49.23	71.72
	SliceGPT	21.52	23.95	74.32	65.59	60.71	**68.52**	**42.41**	62.31
	LaCo	24.37	13.97	72.42	59.27	60.44	54.34	34.56	56.21
	RM	24.37	10.08	73.72	66.61	66.80	66.12	41.98	63.05
	ShortGPT	24.37	20.06	72.74	**70.80**	67.80	60.35	41.30	62.60
	BlockPruner	25.12	**8.16**	**76.93**	66.30	**72.20**	65.82	41.38	**64.53**
Baichuan2-7B	Dense	0	6.04	77.48	68.27	72.18	72.98	42.75	66.73
	LaCo	21.57	26.46	68.28	58.56	51.50	52.90	28.50	51.95
	RM	21.57	189.78	59.96	52.33	30.87	38.17	23.63	40.99
	ShortGPT	21.57	31.05	63.71	**62.67**	50.01	47.31	30.72	50.88
	BlockPruner	22.45	**15.38**	**69.75**	61.48	**58.09**	**58.08**	**33.02**	**56.08**
Baichuan2-13B	Dense	0	6.66	78.84	70.40	75.23	74.07	47.70	69.25
	LaCo	22.68	27.07	70.89	58.01	54.00	57.11	32.94	54.59
	RM	22.68	17.70	68.99	67.88	63.78	57.49	37.54	59.14
	ShortGPT	22.68	20.69	69.31	**68.27**	61.71	56.52	36.69	58.50
	BlockPruner	24.19	**15.36**	**71.44**	64.01	**64.20**	**59.81**	**37.88**	**59.47**
Qwen1.5-7B	Dense	0	7.95	79.22	66.46	76.92	62.16	42.66	65.48
	LaCo	20.97	39.23	70.40	58.64	56.35	46.89	32.85	53.03
	RM	20.97	2026.31	67.36	49.88	42.00	**54.17**	28.58	48.40
	ShortGPT	20.97	49.88	69.53	**62.12**	58.87	43.60	32.17	53.26
	BlockPruner	21.83	**20.58**	**71.71**	55.56	**59.31**	53.70	**33.28**	**54.71**
Qwen1.5-14B	Dense	0	7.44	79.87	70.56	79.41	68.48	47.01	69.07
	LaCo	22.25	16.32	71.55	58.33	60.16	53.70	34.04	55.56
	RM	22.25	55.99	67.08	53.28	42.08	50.72	29.01	48.43
	ShortGPT	22.25	1237.21	58.60	55.96	36.16	38.09	34.81	44.72
	BlockPruner	23.72	**15.67**	**75.24**	**61.48**	**66.92**	**59.51**	**39.08**	**60.45**

Table 1: Zero-shot downstream task performance of various models using different pruning methods. "Dense" represents the original, unpruned models. "PPL" means the perplexity on Wikitext2. All evaluations are conducted using the same configuration to ensure comparability.

Furthermore, it's noteworthy that models exhibiting lower perplexity on the Wikitext2 dataset generally outperform those with higher perplexity on the same dataset. This underscores the potential of perplexity as a metric reflecting model performance. Notably, despite our method conducting pruning searches on the Alpaca dataset, it achieves lower perplexity on the Wikitext2 dataset.

Finally, we observe that while approaches such as ShortGPT and Relative Magnitude result in a significant decline in model performance across different tasks, BlockPruner stands out by avoiding such drastic reductions. This suggests that our proposed block pruning method effectively mitigates performance degradation during the pruning process. Due to space constraints, we have moved the details of pruning baselines and comparisons across various pruning ratios to Appendix F.

5 Analyses

5.1 Ablation Study

To assess the influence of various key operations within the proposed pruning algorithm on its performance, we undertake a thorough ablation study across six models. In particular, we first drop the iterative search procedure and directly remove blocks with the lowest importance scores. Then, we substitute the fine-grained block pruning with a coarser-grained layer pruning approach. The results of these experiments are shown in Table 2.

The experimental findings highlight that solely relying on the perplexity metric without incorporating a search component can result in subpar pruning results and even performance deterioration. This phenomenon may stem from the intrinsic nature of perplexity, which, unlike other importance metrics focusing solely on local block influence, is inherently influenced by the interaction among multiple blocks due to its derivation from the model's output calculation. While perplexity aids in identifying redundant blocks within the model, it doesn't directly yield an optimal pruning sequence.

Furthermore, pruning at the layer level rather than the fine-grained block level yields less robust performance. This observation indicates that the model contains fine-grained redundancies, and seg-

Model	Method	Ratio (%)	Avg. Score
Llama2-7B	**BlockPruner**	21.99	**60.17**
	- search	20.95	55.89 (-7.11%)
	- block	21.02	58.63 (-2.56%)
Llama-2-13B	**BlockPruner**	25.12	**64.53**
	- search	25.08	58.58 (-9.21%)
	- block	24.37	62.91 (-2.51%)
Baichuan2-7B	**BlockPruner**	22.45	**56.08**
	- search	22.39	38.81 (-30.80%)
	- block	21.57	54.76 (-2.36%)
Baichuan2-13B	**BlockPruner**	24.19	**59.47**
	- search	24.19	55.95 (-5.92%)
	- block	24.95	58.22 (-2.10%)
Qwen1.5-7B	**BlockPruner**	21.83	**54.71**
	- search	20.90	37.72 (-31.06%)
	- block	20.97	52.66 (-3.75%)
Qwen1.5-14B	**BlockPruner**	23.72	**60.45**
	- search	22.98	40.80 (-32.51%)
	- block	22.25	60.10 (-0.58%)

Table 2: Average score of ablation study of BlockPruner on downstream tasks. "- search" indicates dropping the iterative search procedure and directly removing blocks with the lowest importance score. "- block" means we substitute the fine-grained block pruning with a coarser-grained layer pruning approach.

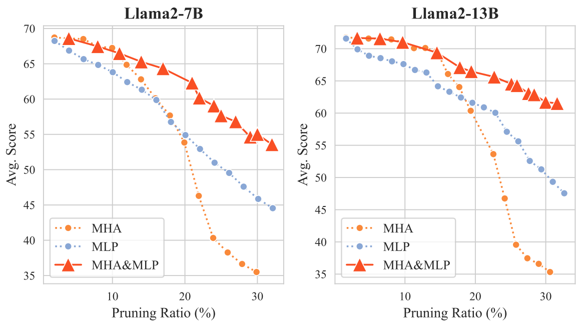

Figure 4: The impact of pruning MHA and MLP individually on model performance. "MHA&MLP" represents the original BlockPruner algorithm.

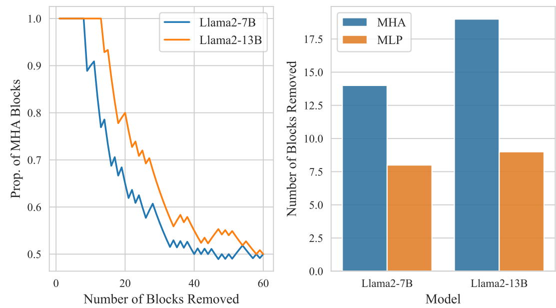

Figure 5: **Left**: The proportion of MHA blocks removed during the pruning process, relative to the total number of removed blocks. **Right**: The number of different blocks removed from models at a pruning ratio of 30%.

menting layers into smaller blocks for pruning allows for more efficient removal of this redundancy, thereby better preserving the model's capabilities.

5.2 Redundancies Between MHA and MLP

To investigate the significance and roles of the MHA and MLP modules in modern LLMs, we conduct pruning experiments focusing exclusively on MHA or MLP blocks. We apply this pruning strategy to two models of varying sizes, Llama2-7B and Llama2-13B, while keeping the pruning ratios below 33%. The results illustrated in Figure 4 reveal several notable observations.

Before reaching a pruning ratio of 17%, pruning only the MHA blocks results in less performance loss compared to pruning MLP blocks and even matches the performance of mixed pruning. This indicates that MHA modules in LLMs may possess greater redundancy than initially anticipated, whereas MLP modules are relatively less redundant. However, when the pruning ratio surpasses 17%, further pruning of MHA blocks leads to a sharp decline in performance. This trend suggests that as pruning advances, the redundant MHA blocks are progressively removed, leaving only the crucial MHA blocks. Moreover, in the larger model, the sharp decline in performance occurs at higher pruning ratios. This observation is consistent with our previous findings, suggesting that larger models contain more redundant blocks. Such redundancy may stem from factors like insufficient training, resulting in higher initial redundancy.

We also examine the proportion of MHA blocks removed during pruning. Specifically, we present the number of MHA and MLP blocks removed at different pruning stages. In Figure 5 (left), we set the number of removed blocks to 60. In Figure 5 (right), the models have 22 and 28 blocks removed, respectively, maintaining a pruning ratio of 30%.

The results in Figure 5 (left) for both models reveal a consistent tendency to initially remove only MHA blocks. As the pruning process progresses and more blocks are removed, the proportion of MHA blocks being pruned follows a zigzag downward trend. Notably, the curve for Llama2-13B shifts to the right compared to Llama2-7B, suggesting that the larger model contains more redundant MHA blocks. This is further emphasized in Figure 5 (right), where, at the same pruning ratio, Llama2-13B prunes more MHA blocks than Llama2-7B. Additionally, given that our pruning method tends to remove more MHA blocks at equivalent pruning ratios, it can significantly reduce the usage of the key-value (KV) cache (Pope et al., 2023) in MHA, which potentially accelerate the inference process.

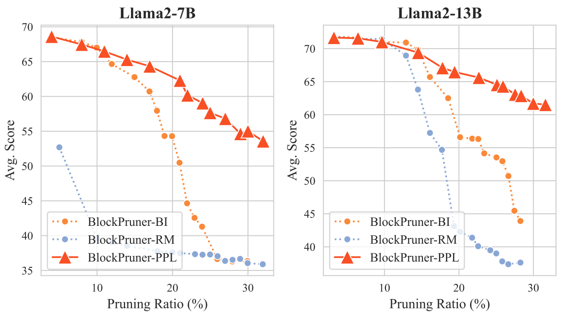

Figure 6: The impact of different block importance metrics on the pruning performance of BlockPruner

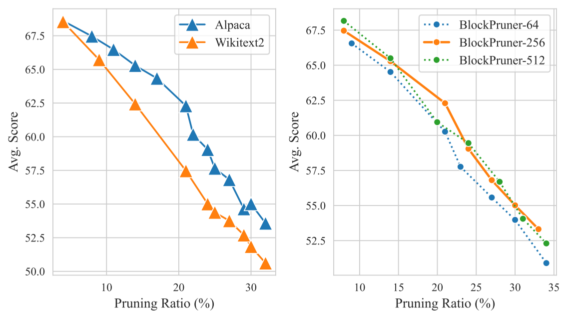

Figure 7: **Left**: The performance of BlockPruner on the Alpaca and Wikitext2 datasets using a calibration dataset of 256 samples. **Right**: Impact of sample sizes on BlockPruner's performance on Alpaca, with the numbers indicating the sample sizes used.

5.3 Perplexity for Block Redundancy

In this section, we explore the impact of different block importance metrics. Generally, Block Influence (BI) and Relative Magnitude (RM) measure the importance of a block based solely on its input and output hidden states, thereby reflecting the block's local influence. In contrast, perplexity is derived from the model's output representations and thus can better measure a block's overall influence.

However, as indicated in the ablation study, using perplexity without the iterative search procedure leads to a significant decline in performance. This suggests that perplexity alone is not an effective block importance metric. Instead, it is better suited for dynamic pruning algorithms that offer greater flexibility compared to static algorithms.

As illustrated in Figure 6, when BI and RM are applied in dynamic pruning algorithms, they sometimes achieve performance comparable to perplexity at lower pruning ratios. However, as the pruning ratio increases, their limitations become evident, resulting in a sharp decline in model performance. This suggests that these local metrics do not adequately capture the impact of different blocks on the model's overall performance.

In summary, perplexity leverages global information to effectively measure block redundancy, especially when used with a dynamic pruning strategy. This combination captures the complex interactions among blocks. In contrast, local metrics like Block Importance (BI) and Relative Magnitude (RM) are useful in specific scenarios but don't reflect the overall contribution of blocks to the model, particularly at higher pruning ratios.

5.4 Impact of Data on Pruning

In the work on SliceGPT (Ashkboos et al., 2024), the authors also used the Wikitext2 (Merity et al., 2016) and Alpaca (Taori et al., 2023) datasets for pruning experiments. They observed that the Alpaca dataset often yielded better pruning results. In our study, we obtain similar findings. As shown in Figure 7 (left), when pruning Llama2-7B, the performance across different pruning ratios is significantly higher when using the Alpaca dataset compared to Wikitext2. We hypothesize that this may be due to the Alpaca dataset being an instruction-following dataset, which is more closely aligned with downstream tasks. This suggests that the choice of dataset has a significant impact on the final pruning performance of the model.

To determine the appropriate sample size and analyze its impact on the pruning performance of BlockPruner, we extract varying numbers of instances from the Alpaca dataset and conduct pruning experiments using Llama2-7B. The results presented in Figure 7 (right) indicate that increasing the sample size beyond 256 yields no significant improvement in the pruning effect of BlockPruner. Therefore, we set the number of samples to 256.

6 Conclusion

In this work, we introduce BlockPruner, a novel structured pruning approach for efficiently pruning large language models (LLMs). BlockPruner decomposes Transformer layers into two minimal residual blocks and employs a new block importance metric along with a pruning search algorithm to iteratively remove redundant blocks. Extensive experiments across various models show that our method outperforms other baselines in post-pruning performance. Our findings highlight the presence of fine-grained block redundancy in LLMs and reveal significant differences in redundancy levels among different block types. We hope our work contributes to a deeper understanding of the importance of various blocks within LLMs.

Limitations

Our current work has three potential limitations. First, while perplexity serves as a useful indicator of block importance, it may not be the optimal metric. Second, while our proposed pruning search algorithm is effective, other combinatorial optimization algorithms might identify superior pruning sequences. Lastly, due to constraints in computational resources, we did not apply our method to prune larger models. Nevertheless, our approach is highly scalable and readily adaptable for pruning larger models in future research.

Ethics Statement

The aim of this study is to provide a generalizable pruning method for large language models. All models and datasets used in our experiments are publicly accessible and do not contain any private information. We strictly adhere to the usage policies of these resources and utilize them solely for research purposes.

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. In *The Twelfth International Conference on Learning Representations*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

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*, volume 34, pages 7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. 2020. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1899–1908.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. 2024. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yukun Huang, Yanda Chen, Zhou Yu, and Kathleen McKeown. 2022. In-context learning distillation: Transferring few-shot learning ability of pre-trained language models. *arXiv preprint arXiv:2212.10670*.

Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Losparse: Structured compression of large language models based on low-rank and sparse approximation. In *International Conference on Machine Learning*, pages 20336–20350. PMLR.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-pruner: On the structural pruning of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5.

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

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*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Tycho F. A. van der Ouderaa, Markus Nagel, Mart Van Baalen, and Tijmen Blankevoort. 2024. The LLM surgeon. In *The Twelfth International Conference on Learning Representations*.

Wenxiao Wang, Wei Chen, Yicong Luo, Yongliu Long, Zhengkai Lin, Liye Zhang, Binbin Lin, Deng Cai, and Xiaofei He. 2024. Model compression and efficient inference for large language models: A survey. *arXiv preprint arXiv:2402.09748*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*.

Peng Xu, Wenqi Shao, Mengzhao Chen, Shitao Tang, Kaipeng Zhang, Peng Gao, Fengwei An, Yu Qiao, and Ping Luo. 2024. BESA: Pruning large language models with blockwise parameter-efficient sparsity allocation. In *The Twelfth International Conference on Learning Representations*.

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*.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems*, volume 35, pages 27168–27183. Curran Associates, Inc.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Mykola Pechenizkiy, Yi Liang, Zhangyang Wang, and Shiwei Liu. 2024. Outlier weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

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*.

Yang Zhang, Yawei Li, Xinpeng Wang, Qianli Shen, Barbara Plank, Bernd Bischl, Mina Rezaei, and Kenji Kawaguchi. 2024a. Finercut: Finer-grained interpretable layer pruning for large language models. *arXiv preprint arXiv:2405.18218*.

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. 2024b. Plug-and-play: An efficient post-training pruning method for large language models. In *The Twelfth International Conference on Learning Representations*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

A Details of Implementations

In this section, we detail our experimental setup. We sampled from the Alpaca dataset with a fixed random seed of 42. For SliceGPT, we followed the original paper's configuration, using 1024 samples, a sparsity ratio set at 30%, and a maximum sequence length of 2048. For ShortGPT, RM, and BlockPruner, we sampled 256 samples from the dataset, with the same maximum sequence length of 2048. For LaCo, we adjusted the merging threshold using the provided code and data to achieve the corresponding pruning ratio.

B Details of Datasets

B.1 Pruning Datasets

Alpaca (Taori et al., 2023) is a general instruction-following dataset containing 52,000 questions. Each sample comprises three fields: instruction, input, and response. We selected 10% of the dataset and utilized 256 samples for the main experiments. Perplexity calculation was performed uniformly across all text in the samples without differentiation between fields.

B.2 Evaluation Datasets

All downstream task datasets were partitioned and evaluated using the default configuration of LM Evaluation Harness.

Wikitext-2 (Merity et al., 2016) is a collection of over 100 million tokens extracted from verified Good and Featured articles on Wikipedia. This dataset is commonly used to measure the quality of a model's text generation. We employed samples from the pre-split test set for calculating perplexity.

PIQA (Bisk et al., 2020) is a dataset designed to evaluate natural language models' understanding of physical commonsense. It employs a multiple-choice format where the model selects the most appropriate solution from two options given a goal.

WinoGrande (Sakaguchi et al., 2021) is an extensive dataset to evaluate models' commonsense reasoning capabilities. It comprises 44,000 questions. The dataset features fill-in-the-blank tasks with binary options, aiming to select the correct option for a given sentence that requires commonsense reasoning.

HellaSwag (Zellers et al., 2019) is also a dataset designed to assess models' commonsense reasoning abilities, specifically to highlight the limitations of current models in handling commonsense natural language reasoning tasks. Despite being trivial for humans (with >95% accuracy), the dataset presents significant difficulties for models. The evaluation is conducted using four-way multiple-choice questions.

ARC (Clark et al., 2018) dataset comprises 7,787 multiple-choice science exam questions sourced from various origins. Each question typically offers four answer options. These questions are categorized into two distinct difficulty sets: 2,590 questions for Challenge Set and 5,197 for Easy Set.

Model	Method	Ratio(%)	Avg.Score
Llama2-7B	SliceGPT	21.45	57.93
	SliceGPT*	21.45	57.83
Llama2-13B	SliceGPT	21.52	62.34
	SliceGPT*	21.52	62.31

Table 3: Comparison of average performance on downstream tasks between the official SliceGPT results and our reproduced results (indicated by "∗" for our results).

C Details of Evaluations

To ensure a fair and comprehensive comparison, we employed the same version of the LM Evaluation Harness as used in the SliceGPT experiments and obtained evaluation scores under identical experimental configurations. These scores closely match those reported in the SliceGPT paper, as detailed in Table 3. For consistency, we present our reproduced results in the main experiments.

To evaluate the performance of pruned models on downstream tasks, we utilized five multiple-choice QA datasets: PIQA, WinoGrande, HellaSwag, ARC-e, and ARC-c. Additionally, to assess text generation quality, we calculated perplexity using the test set of the Wikitext2 dataset. For the downstream task evaluations, we adhered to

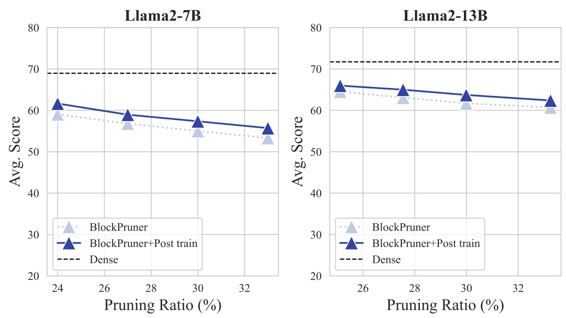

Figure 8: Average score of BlockPruner with varying pruning ratios before and after post-training.

the default evaluation parameters and zero-shot settings, with a batch size set to 1. For perplexity calculations, the maximum text length was set to 2048, maintaining a batch size of 1 as well.

D Post-training after Pruning

We sampled 8,000 instances from the Alpaca dataset and conducted post-training on the pruned Llama2-7B and Llama2-13B models obtained via BlockPruner using LoRA. All linear layers, excluding the embedding layer and the language model head, were trained. The LoRA rank and LoRA α parameters were set to 32 and 10, respectively, with a learning rate of 2e-4 and a batch size of 1. Additionally, we configured the gradient accumulation steps to 4 and employed a linear learning rate scheduler. We controlled the pruning ratios within the range of 24% to 33%. The results are shown in Figure 8. It can be seen that after training, our models showed further improvement at different pruning ratios. The Llama2-7B and Llama2-13B models recovered to 89% and 92% of the performance of the unpruned models, respectively, when pruned by approximately 1/4.

E Sensitivity to Sample Size

ShortGPT uses Block Influence as the importance metric for layers, while RM uses Relative Magnitude. The former calculates the similarity between the input and output hidden states of a layer, while the latter utilizes the input and the non-residual part of the output. In our preliminary experiments, we found that these two metrics are not sensitive to sample size. We sampled different numbers of instances from the test set of the Alpaca dataset to observe their impact on these metrics, and the results are shown in Figure 9. We can see that all the lines almost overlap, indicating that Block Influence and Relative Magnitude are not sensitive to the sample size. We speculate that this may be due to the limited information provided by the changes in the input and output of a single layer.

F Varying Pruning Ratios

To broadly demonstrate the superiority of our method, we present the pruning effects of BlockPruner, ShortGPT, and Relative Magnitude on six representative large models at different pruning ratios. As depicted in Figure 10, our method effectively minimizes performance loss throughout the pruning process, avoiding any sudden drops in performance. In contrast, RM exhibits significant instability and is prone to performance collapse. ShortGPT performs relatively well, but in the pruning experiments on Qwen1.5-14B, it also leads to severe performance degradation at higher pruning ratios. Overall, the advantages of our method become more pronounced as both model size and pruning ratio increase.

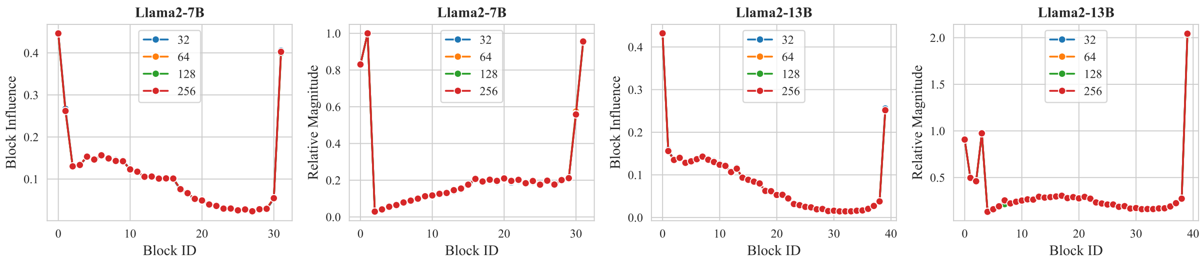

Figure 9: The changes in Block Influence and Relative Magnitude of the model under different sample sizes.

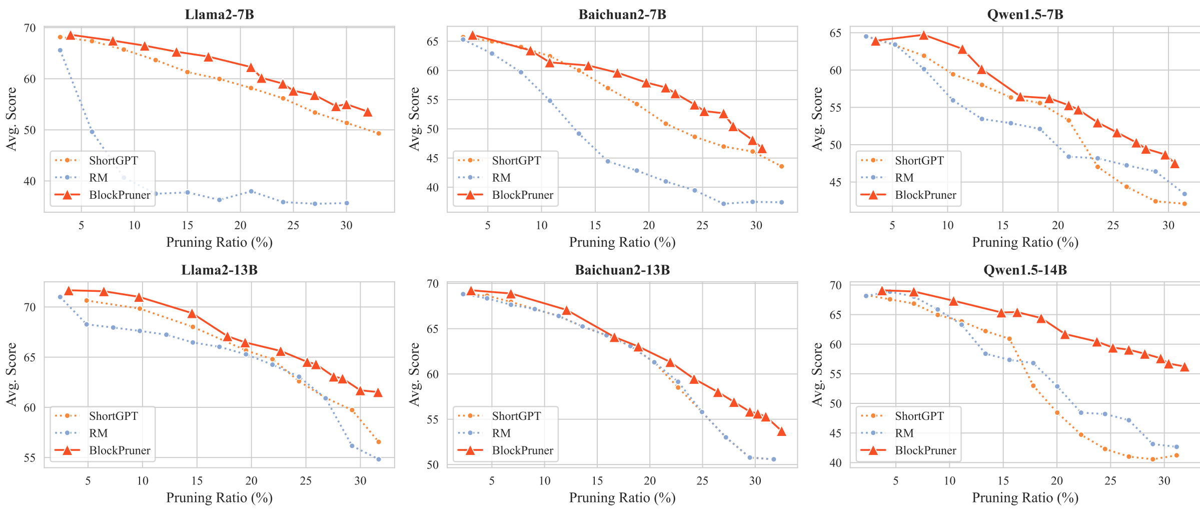

Figure 10: Average score of BlockPruner with varying pruning ratios compared with ShortGPT and RM.