

FlexiGPT: Pruning and Extending Large Language Models with Low-Rank Weight Sharing

James Seale Smith¹ Chi-Heng Lin¹ Shikhar Tuli¹ Haris Jeelani¹
Shangqian Gao² Yilin Shen¹ Hongxia Jin¹ Yen-Chang Hsu¹
¹Samsung Research America, ²Florida State University

Abstract

The rapid proliferation of large language models (LLMs) in natural language processing (NLP) has created a critical need for techniques that enable efficient deployment on memory-constrained devices without compromising performance. We present a method to prune LLMs that selectively prunes model blocks based on an importance score and replaces them with a low-parameter replacement strategy. Specifically, we propose a principled metric to replace each pruned block using a weight-sharing mechanism that leverages unpruned counterparts from the model and block-specific low-rank adapters. Furthermore, we facilitate the learning of these replacement blocks with output feature normalization and an adapter initialization scheme built on low-rank SVD reconstructions. Empirical evaluations demonstrate substantial performance gains over existing methods, achieving state-of-the-art performance on 5/6 benchmarks for a compression rate of 30% and 6/6 benchmarks for a compression rate of 40%. We also demonstrate that our approach can extend smaller models, boosting performance on 6/6 benchmarks using only $\approx 0.3\%$ tokens of extended training with minimal additional parameter costs.

1 Introduction

The widespread adoption of LLMs has revolutionized NLP applications, driving significant advancements in areas such as virtual assistants, automated customer support, and real-time language translation (Minaee et al., 2024; Naveed et al., 2023). However, deploying these models on memory-constrained devices, such as smartphones and edge devices, remains a formidable challenge due to their substantial parameter sizes and computational demands (Hadi et al., 2023; Raiaan et al., 2024). This paper addresses this challenge by presenting a novel approach that targets *parameter efficiency* to make LLMs more suitable for on-device applications with minimal performance compromises.

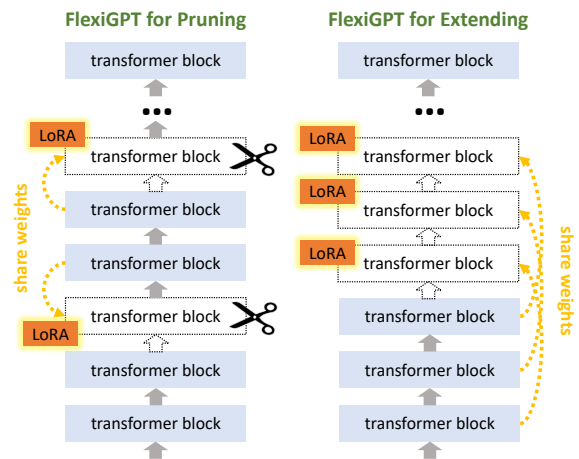


Figure 1: FlexiGPT is used for two settings: (1) pruning a model to reduce parameters with minimal performance cost *or* (2) extending a model to increase performance with minimal parameter cost. **Left:** For pruning models (setting 1), we prune entire blocks and replace them using weight sharing and learned adapters. **Right:** For extending models (setting 2), we repeat block patterns in the model using weight sharing and learned adapters.

Parameter efficiency is particularly critical as it directly impacts the feasibility of deploying LLMs on devices with limited memory and storage resources. Recent model pruning techniques, such as SliceGPT (Ashkboos et al., 2024), LLM Surgeon (van der Ouderaa et al., 2024), LLM-Pruner (Ma et al., 2023), LaCo (Yang et al., 2024), and ShortGPT (Men et al., 2024), reduce the number of parameters but often result in significant performance degradation with minimal recovery after pruning. This gap in existing techniques underscores the need for an end-to-end pruning method that not only reduces the model size but also facilitates performance recovery. *In this work, we propose to recover performance by utilizing existing weights within the model.*

Specifically, we introduce a comprehensive pruning strategy combined with an innovative

weight sharing technique and Low-Rank Adapters (LoRA) (Hu et al., 2021), facilitating efficient parameter usage while preserving the model’s performance. We begin by pruning model blocks based on ShortGPT’s Block Influence (BI) score (Men et al., 2024). To replace the pruned blocks, we introduce a low-parameter weight-sharing mechanism that leverages existing block modules within the model and incorporates block-specific LoRA parameters, ensuring the selected replacement blocks have high similarity to the pruned blocks while maintaining block diversity. Furthermore, we introduce a novel method to initialize the LoRA adapters in weight-sharing blocks, setting them to be the low-rank difference between the pruned block and the weight-shared replacement block. This initialization minimizes initial disruptions and facilitates smoother model adaptation. Finally, we incorporate *output* feature normalization for pruned blocks to ensure a smooth transition and adaptation, allowing the model to gradually learn and stabilize its performance over time.

Empirical evaluations of our method, which we refer to as *FlexiGPT*, demonstrate substantial performance gains over existing methods. Specifically, we achieve state-of-the-art performance on 5/6 benchmarks for a compression rate of 30% and 6/6 benchmarks for a compression rate of 40% for the popular LLaMA-2 7B model (Touvron et al., 2023). As visualized in Figure 1, our proposed technique not only effectively prunes large models for on-device deployment but *also extends smaller models*, improving their performance at minimal additional parameter costs. Specifically, our method shows that a 22-layer TinyLLaMA (Zhang et al., 2024) model can be extended with repeated blocks, boosting performance on 6/6 benchmarks using only $\approx 0.3\%$ tokens of extended training with minimal additional parameter costs. *In summary, we make the following contributions:*

1. We develop a weight-sharing technique using adapters and low-rank SVD reconstructions to replace pruned blocks effectively.
2. We apply output normalization to maintain stability and enable gradual learning post-pruning.
3. We propose a method for extending smaller models by repeating layers with unique adapters and normalization parameters.
4. We achieve significant empirical performance gains, achieving state-of-the-art performance on several benchmarks for a variety of models.

2 Background and Related Work

Pruning - Pruning selectively removes less important parameters, reducing model size and computational complexity while maintaining performance. Its greatest benefit lies in optimizing LLMs for deployment in resource-constrained environments such as mobile devices, facilitating faster inference. Several works have been proposed to reduce the size of LLMs by pruning model structures. LLM-Pruner (Ma et al., 2023) removes unimportant coupled structures and the importance is calculated from Taylor expansions. SliceGPT (Ashkboos et al., 2024) applies orthogonal projections to the feature maps and then it performs pruning in the projected space. LLM Surgeon (van der Ouderaa et al., 2024) periodically updates model weights and structures, resulting in a higher cost compared to other methods. Besides reducing the width of the model, ShortGPT (Men et al., 2024) is proposed to remove blocks by using Block Influence scores, and LaCo (Yang et al., 2024) is proposed to collapse layers. Existing pruning methods primarily focus on removing redundant model weights, often neglecting the loss of model capacity. Our approach addresses this limitation by sharing weights from the pruned model to restore its capacity.

PEFT - Parameter-Efficient Fine-Tuning (PEFT) methods aim to mitigate the extensive computational and memory demands of fine-tuning large models by focusing on a smaller subset of parameters. One prominent category of PEFT methods is *adapters*, which involves adding trainable modules to the existing frozen layers of the model (He et al., 2021; Houlsby et al., 2019). Another significant category is *prompt* methods, which augment the initial input sequence with additional trainable vectors known as prompts. This technique focuses on fine-tuning these added tokens rather than the entire model, as demonstrated in works such as Lester et al. (2021); Liu et al. (2021).

Recently, LoRA (Hu et al., 2021) has emerged as the most efficient and highest-performing PEFT approach. LoRA introduces the use of low-rank matrices to adjust model weights efficiently, merging with pre-trained weights before inference to maintain the model’s operational speed. Building on this, DoRA (Liu et al., 2024a) decomposes pre-trained weights into magnitude and direction components for fine-tuning, focusing on fine-tuning the directional components using LoRA. *Our research extends these low-rank PEFT methods by incorpo-*

rating LoRA and normalization for efficient weight sharing. Similar to DoRA, our method involves a normalization stage; however, we normalize in the *feature-space* instead of the *weight-space*.

Weight-Sharing - For GPT LLMs, Dehghani et al. (2018) proposed to share all the layers with a dynamic halting mechanism to improve accuracy on the downstream tasks. However, it requires the number of parameters of the base layer (unshared) to match the number of parameters of all layers of vanilla transformers (Vaswani et al., 2017). Subformer (Reid et al., 2021) applies a sandwich-like method of parameter sharing where only middle layers are shared but it does not use any adapter. (Takase and Kiyono, 2021) developed efficient cyclic sharing patterns to increase the accuracy, however their sharing patterns are mainly based on ablation studies. MobileLLM (Liu et al., 2024b) proposed sub-billion parameter architectures for mobile devices and adopted immediate block-wise weight sharing for further accuracy improvement. However, they do not use any pruning and adapters. Cao et al. (2024) introduced matching functions to develop head-wise shareable attention in a principled fashion. Although they use pretrained weights for faster convergence, their matching functions are only applied to share weights among multiple heads of the same layer.

SVD - The Singular Value Decomposition (SVD) of a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ is, $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix of singular values. SVD is widely used to obtain a low-rank representation of \mathbf{W} by selecting the k most significant singular values and their corresponding singular vectors, where $k < \min(m, n)$. Hence, the low-rank representation of \mathbf{W} is given as: $\mathbf{W}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$, where $\mathbf{U}_k \in \mathbb{R}^{m \times k}$, $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$, and $\mathbf{V}_k \in \mathbb{R}^{n \times k}$, have lower dimensions than \mathbf{U} , $\mathbf{\Sigma}$, \mathbf{V} , respectively. SVD has been applied for model compression (Denton et al., 2014; Hsu et al., 2022) and is closely related to LoRA methods for reducing fine-tuning overhead.

3 FlexiGPT

In this section, we detail our approach, FlexiGPT (Figure 3), which prunes and extends LLMs using LoRA adapters, weight sharing techniques, and output feature normalization. Our method focuses on achieving parameter efficiency while minimizing performance degradation, particularly for memory-

constrained devices.

Our method is based on the transformer architecture (Vaswani et al., 2017), which consists of Multi-Head Self-Attention (MHSA) and Multi-Layer Perceptron (MLP) layers. However, our approach is not constrained to this architecture. In general, we refer to blocks (MHSA+MLP), layers (MHSA or MLP), and weights (denoted as W), as our method affects the weights in a uniform manner.

3.1 Pruning Strategy

Our pruning strategy aims to identify and remove blocks that minimally impact the model’s performance. To achieve this, we leverage the Short-GPT (Men et al., 2024) Block Influence (BI) score, which has been shown to effectively measure the importance of each block. The Block Influence (BI) score (Men et al., 2024) BI_i for a block i is defined as follows:

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

where $X_{i,t}$ denotes the t^{th} row of X_i , and X_i represents the hidden states matrix at block i , with dimensions $T \times d$, where T is the sequence length and d is the hidden dimension. This score captures the extent to which each block transforms its input, with higher scores indicating more significant changes. We calculate the BI score for each block in our model using the validation MiniPile (Kaddour, 2023) subset of the Pile dataset (Gao et al., 2020), and prune the blocks with the lowest BI scores.

We tried other criteria to select blocks for pruning that considered a block’s replaceability by another block in the model. However, we found that the BI score results in higher performance on downstream tasks. Our intuition is that the BI score prunes blocks deeper along the model’s depth in a sequence, leaving much of the model intact and in the same order, which may explain how it retains strong downstream performance.

3.2 Selection of Weight Sharing Bases

To replace pruned blocks, we aim to find similar unpruned blocks in the model which, when paired with adapters, can recover much of the performance lost after pruning. We aim to select each pruned block’s weight sharing ‘base’ by identifying *similar* unpruned weights. However, a naïve approach such

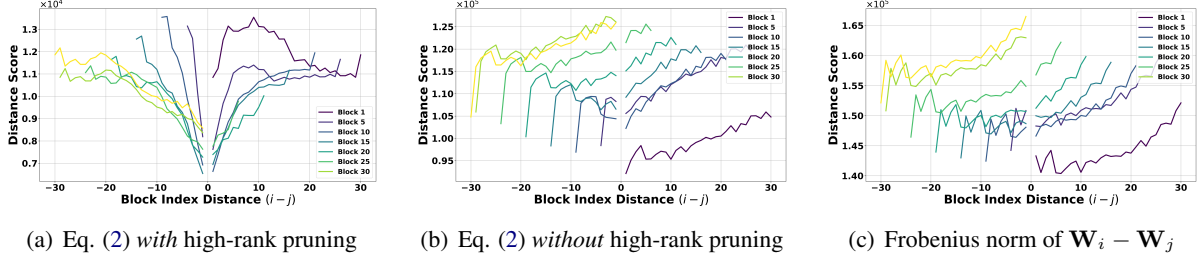


Figure 2: Comparison of block distance score versus block index distance ($i - j$) for different metrics. (a) Using the proposed metric in Eq. (2) with high-rank pruning, showing that closer blocks score lower (better), matching our intuition that weights close in the model have similar function. (b) Ablation of high-rank pruning, where there is no clear trend except that blocks closer to 0 are lower and those closer to 31 are higher. (c) Simple Frobenius norm, showing a similar lack of clear trend as in (b). We found that using the score in (a) as the weight-sharing selection metric results in a much higher performing model compared to using the scores in (b) and (c).

as the Frobenius norm in the weight space often results in suboptimal selections. Specifically, we find that *all blocks ‘choose’ a single block*, whereas intuition suggests a diverse selection of base blocks would work better¹.

Instead, we employ a selection metric based on *low-rank SVD reconstructions* to achieve a more effective and intuitive solution. Utilizing low-rank approximations, namely $\hat{\mathbf{W}}_i$ and $\hat{\mathbf{W}}_j$, instead of directly using \mathbf{W}_i and \mathbf{W}_j helps avoid the pitfall where all pruned blocks are replaced by a single block. We believe high-rank elimination is beneficial because low-rank approximations capture the most significant components of the weights, thereby simplifying the process of identifying suitable replacements by eliminating high-rank ‘noise’. Our method reveals that blocks nearest to the pruned blocks in the model tend to have the lowest scores, indicating higher similarity.

The distance metric $d(\mathbf{W}_i, \mathbf{W}_j)$ for selecting the replacement block is defined as:

$$d(\mathbf{W}_i, \mathbf{W}_j) = \left\| \hat{\mathbf{W}}_i - \left(\hat{\mathbf{W}}_j + \Delta_{i-j} \right) \right\|_F \quad (2)$$

where:

- $\hat{\mathbf{W}}_i$ and $\hat{\mathbf{W}}_j$ are the low-rank SVD reconstructions of \mathbf{W}_i and \mathbf{W}_j , respectively, using the first r ranks.
- $\Delta_{i-j} \triangleq (\mathbf{U}_{i-j} \boldsymbol{\Sigma}_{i-j})[1 : r](\mathbf{V}_{i-j}[1 : r])^T$ is the rank- r approximation of the difference $\hat{\mathbf{W}}_i - \hat{\mathbf{W}}_j$. We used a rank of $r = 256$.
- $\| \cdot \|_F$ denotes the Frobenius norm.

¹We empirically demonstrate this in the experiments section in the first row of Table 4

These low-rank approximations $\hat{\mathbf{W}}_i$ and $\hat{\mathbf{W}}_j$ are obtained via:

$$\hat{\mathbf{W}}_i = (\mathbf{U}_i \boldsymbol{\Sigma}_i)[1 : r](\mathbf{V}_i[1 : r])^T \quad (3)$$

$$\hat{\mathbf{W}}_j = (\mathbf{U}_j \boldsymbol{\Sigma}_j)[1 : r](\mathbf{V}_j[1 : r])^T \quad (4)$$

Finally, for each pruned block i , we select its base for weight sharing as the candidate block j with the minimum score in (2):

$$j = \underset{j' \neq i}{\operatorname{argmin}} d(\mathbf{W}_i, \mathbf{W}_{j'}) \quad (5)$$

This approach is highly intuitive, as proximal blocks are naturally more alike. In Figure 2(a), the proposed metric with high-rank pruning, Eq. (2), shows that blocks closer in the model score lower (better), confirming our intuition that proximate blocks have similar functions. Figures 2(b) and 2(c), which respectively ablate high-rank pruning and use simple Frobenius norm, lack this clear trend, and furthermore we found that they result in significantly weaker models. An alternative version of this Figure is available in the Appendix, where the x-axis is candidate block index j instead of block index distance ($i - j$).

3.3 Output Normalization

We apply layer normalization (Ba et al., 2016) to the output of each MHSA and MLP layer in the weight-sharing layers, specifically to the previously pruned blocks. This normalization is applied across the hidden state dimension and is initialized to a small value set by a hyperparameter, allowing the model to gradually learn and adjust the output magnitudes over time. The normalized output \mathbf{h}_{norm} is defined as:

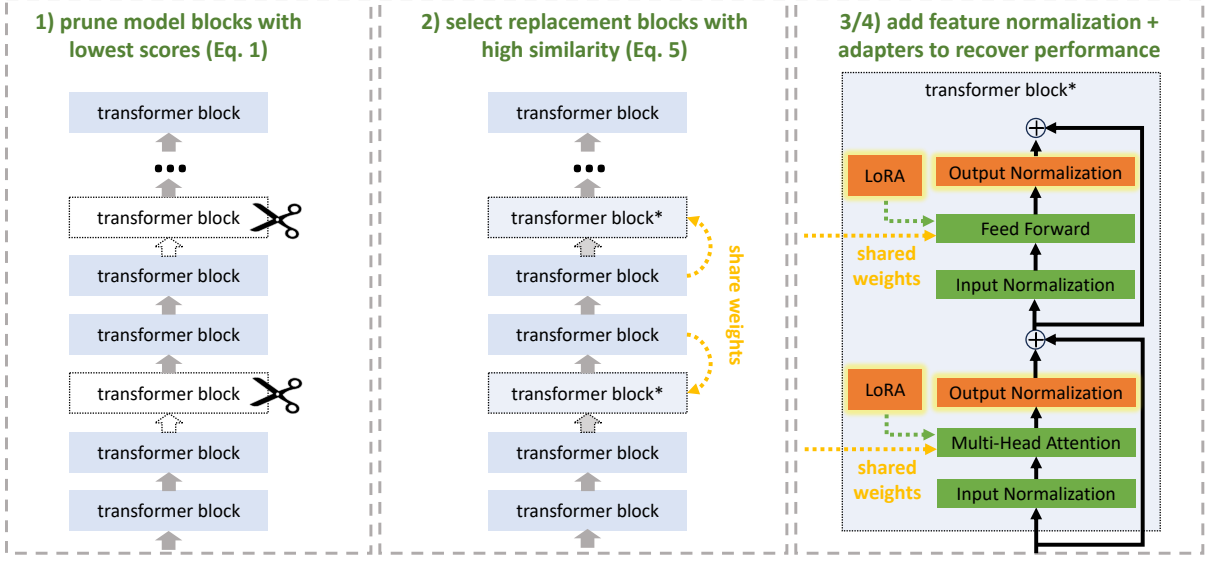


Figure 3: Overview of the FlexiGPT pruning process. **Left:** We prune model blocks with the lowest scores based on (1). **Center:** We select replacement blocks with high similarity using (5). **Right:** We add feature normalization and learn adapters to recover performance.

$$\mathbf{h}_{norm} = \frac{\mathbf{h} - \mu(\mathbf{h})}{\sigma(\mathbf{h})} \times \gamma \quad (6)$$

where:

- \mathbf{h} is the output of the layer before normalization.
- $\mu(\mathbf{h})$ and $\sigma(\mathbf{h})$ are the mean and standard deviation of \mathbf{h} , respectively.
- γ is a learnable scaling weight of the same dimension as the model hidden state size.

This approach is akin to initializing the B matrix in LoRA (Hu et al., 2021) such that $\Delta W = BA$ is zero at the beginning of training. This similarity arises because both methods aim to minimize initial disruptions to the model and allow gradual learning. In LoRA, initializing $\Delta W = BA$ to zero helps avoid high initial loss, ensuring smoother training. Similarly, by initializing the hidden states of weight-shared blocks to small values, we avoid significant jumps in PPL at the start of training. As shown in Table 4, this approach is crucial for maintaining low PPL post-pruning and ensuring stable model performance during fine-tuning.

3.4 Adapters and Initialization

We employ LoRA to facilitate weight sharing for the pruned blocks, providing a parameter-efficient mechanism to adjust the weights of the replaced

blocks. The LoRA adapters consist of two low-rank matrices, A and B , inserted into the linear transformations of the shared weights in the model, effectively increasing the expressive capacity of these blocks despite the weight-sharing constraint². The weights of the adapters are initialized using the SVD between the pruned block and its replacement block, as described in the selection of weight-sharing bases. Specifically, we decompose the difference between the pruned block \mathbf{W}_i and the replacement block \mathbf{W}_j into low-rank matrices:

$$\mathbf{W}_i - \mathbf{W}_j = \mathbf{U}_{i-j} \mathbf{\Sigma}_{i-j} \mathbf{V}_{i-j}^T \quad (7)$$

The adapter matrices A and B are then initialized as:

$$A = (\mathbf{U}_{i-j} \mathbf{\Sigma}_{i-j})[1 : r], B = (\mathbf{V}_{i-j}[1 : r])^T \quad (8)$$

where:

- $(\mathbf{U}_{i-j} \mathbf{\Sigma}_{i-j})[1 : r]$ is the product of the left singular vectors and the diagonal matrix of singular values, indexed to take the first r columns.
- $(\mathbf{V}_{i-j}[1 : r])^T$ is the transposed matrix containing the first r columns of the right singular vectors.

²We also ‘unlock’ the shared weight during training.

Our method requires a small amount of post-pruning fine-tuning to fully recover performance, which is discussed in Section 4. However, we generally observe that the post-prune PPL is indicative of which method will finish with a lower PPL. In Table 4, we see the effect of output normalization and LoRA initialization on post-prune PPL. While the SVD initialization is of smaller yet significant importance to our method, the output normalization, initialized to a small value to minimize initial disruptions and allow gradual learning, is crucially important. This is evident from the drastic increase in post-prune PPL when output normalization is ablated. The combination of SVD initialization and carefully tuned output normalization ensures that our method maintains low perplexity and stable performance during the fine-tuning phase.

3.5 Model Extension

In addition to pruning, FlexiGPT can also be used to extend smaller models, such as a 22-layer TinyL-LaMA (Zhang et al., 2024). In this second setting, we repeat blocks in a sequence determined by hyperparameter indexes that denote the start and end of the repetition. For instance, we might repeat layers indexed 3 through 18. Each repeated block has unique LoRA adapters and normalization parameters, and we apply output normalization to repeated blocks after the first repetition. We explore two repetition patterns: (i) **block**: each block is repeated a specified number of times, and (ii) **sequential**: the entire sequence of blocks is repeated in a specified manner. This method allows for efficient extension of smaller models, improving their performance while introducing minimal parameter overhead.

4 Model Compression with FlexiGPT

4.1 Setup

Models - We evaluated our method using LLaMA-2 7B (Touvron et al., 2023), OPT 1.3B and 6.7B (Zhang et al., 2022), and LLaMA-3 8B (AI@Meta, 2024), focusing on these due to their widespread adoption by the community.

Frameworks and resources - Our implementations were done using PyTorch, leveraging FSDP and FP-16 mixed training for efficiency. Experiments were conducted on 4 NVIDIA A100 80GB GPUs, and we utilized the Hugging Face Transformers library for model handling and training. Detailed configurations and additional resources are provided in the Appendix.

Datasets and Benchmarks - We used 1B tokens from the SlimPajama (Soboleva et al., 2023) pre-training dataset for post-prune recovery. For zero-shot performance evaluations, we use the ARC-e, ARC-c (Clark et al., 2018), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), and HellaSwag (Zellers et al., 2019) zero-shot benchmarks, utilizing the LM Evaluation Harness (Gao et al., 2021). For perplexity performance evaluations, we use the validation MiniPile (Kaddour, 2023) subset of the Pile dataset (Gao et al., 2020)³.

Baselines - We compared our method against several baselines, including LLM Surgeon (van der Ouderaa et al., 2024), SliceGPT (Ashkboos et al., 2024), ShortGPT (Men et al., 2024), and ShortGPT + LoRA (an improved version of ShortGPT for a fair comparison with our method). LLM Surgeon and SliceGPT are presented for additional context for experiment results which overlapped with our setting (we use the original results presented in their papers), whereas we implement ShortGPT from scratch for a direct comparison in our setting. LLM-Pruner (Ma et al., 2023) and LaCo (Yang et al., 2024) are not included in our tables as ShortGPT has been found to outperform both methods.

4.2 Results

Main Results - Table 1 summarizes the perplexity (PPL) and zero-shot task performance of various pruning methods on the Llama-2 7B model. Our method, FlexiGPT, shows the lowest PPL of 6.55 at a 30% pruning ratio, outperforming both ShortGPT and ShortGPT + LoRA. In terms of zero-shot task performance, FlexiGPT achieves the highest scores in ARC-c (38.62%), PIQA (74.12%), WinoGrande (66.78%), and HellaSwag (69.02%), with an average performance of 62.68%. This represents a significant improvement over the other methods, demonstrating the effectiveness of our approach. For the 40% pruning ratio, similar trends are observed as FlexiGPT consistently shows superior performance over other methods, achieving the *highest score in every benchmark task*.

Table 2 summarizes the perplexity (PPL) and zero-shot task performance of various pruning methods on the Llama-3 8B model, and Table 3 summarizes the perplexity (PPL) of various pruning methods on the OPT 6.7B and OPT 1.3B models (Zhang et al., 2022). These trends also align with those seen in the Llama-2 7B models, further

³We avoid the SlimPajama validation set to avoid giving an unfair advantage to methods trained on the this dataset.

Table 1: Perplexity (PPL) and zero-shot task performance of compressed Llama-2 7B models. * indicates the model underwent recovery training for 1B tokens after pruning using the SlimPajamas dataset (Soboleva et al., 2023). The results for SliceGPT (Ashkboos et al., 2024) and LLM Surgeon (van der Ouderaa et al., 2024) are taken from their papers. Two variants of results are given for LLM Surgeon which correspond to pruning with Wikitext-2 (Merity et al., 2016) and C4 (Raffel et al., 2019).

Method	Ratio	PPL	ARC-e	ARC-c	PIQA	WinoG.	HellaS.	Average
Unpruned	0.0%	5.11	74.58%	46.25%	79.11%	69.14%	76.00%	69.02%
SliceGPT	30%	N/A	51.77%	31.23%	63.55%	61.33%	49.62%	51.50%
LLM Surgeon (C4)	30%	N/A	62.16%	34.47%	72.85%	56.83%	58.11%	56.88%
LLM Surgeon (Wikitext-2)	30%	N/A	63.09%	36.69%	73.56%	61.09%	60.72%	59.03%
ShortGPT	30%	22.76	48.61%	32.68%	64.42%	64.33%	56.15%	53.24%
ShortGPT + LoRA*	30%	6.71	62.50%	37.54%	73.17%	66.61%	68.19%	61.40%
FlexiGPT*	30%	6.55	62.84%	38.62%	74.12%	66.78%	69.02%	62.68%
LLM Surgeon (C4)	40%	N/A	51.56%	27.99%	68.93%	55.64%	48.10%	50.44%
LLM Surgeon (Wikitext-2)	40%	N/A	52.31%	30.29%	69.26%	54.38%	48.04%	50.86%
ShortGPT	40%	42.69	41.29%	30.03%	60.17%	60.54%	43.72%	47.15%
ShortGPT + LoRA*	40%	7.69	55.85%	33.11%	70.51%	65.27%	62.02%	57.35%
FlexiGPT*	40%	7.35	57.03%	33.62%	71.44%	66.61%	63.22%	58.38%

Table 2: Perplexity (PPL) and zero-shot task performance of compressed Llama-3 8B models. * indicates the model underwent recovery training for 1B tokens after pruning using the SlimPajamas dataset (Soboleva et al., 2023).

Method	Ratio	PPL	ARC-e	ARC-c	PIQA	WinoG.	HellaS.	Average
Unpruned	0.0%	6.30	77.69%	55.33%	80.79%	72.85%	79.17%	73.17%
ShortGPT	30%	1.4e4	38.80%	31.83%	60.83%	57.93%	31.62%	44.20%
FlexiGPT*	30%	8.67	64.02%	41.21%	74.76%	70.09%	69.12%	63.85%
ShortGPT	40%	9.1e4	36.99%	30.20%	58.60%	54.85%	30.72%	42.27%
FlexiGPT*	40%	10.25	55.60%	37.88%	69.31%	66.14%	59.60%	57.70%

Table 3: Perplexity (PPL) of compressed OPT models. * indicates the model underwent recovery training for 1B tokens after pruning using the SlimPajamas dataset (Soboleva et al., 2023).

Method	Ratio	OPT 6.7B	OPT 1.3B
Unpruned	0.0%	7.46	9.29
ShortGPT	30%	8.61e2	6.26e2
ShortGPT + FT*	30%	8.66	11.04
FlexiGPT*	30%	8.39	10.81
ShortGPT	40%	2.38e3	1.19e3
ShortGPT + FT*	40%	10.12	13.25
FlexiGPT*	40%	9.18	11.54

validating the robustness of our method across different model sizes and pruning ratios. We note that Llama-3 8B is much more sensitive to pruning compared to Llama-2 7B, which *underscores the need for post-pruning recovery* such as our weight-sharing and adapters scheme.

Ablation Results - Table 4 presents the results of our ablation studies, highlighting the importance of each component in our pruning method. Removing the weight-sharing score, output normalization, or LoRA initialization leads to higher PPL, confirm-

Table 4: Ablation Perplexity (PPL) of 30% compressed Llama-2 7B models. The models underwent recovery training for 1B tokens after pruning using the SlimPajamas dataset (Soboleva et al., 2023). We include post-prune PPL (denoted as Start PPL) to show the effect of output feature normalization and adapter initialization on starting PPL.

Method	Start PPL	PPL
Ablate high-rank prune (3.2)	22.54	6.77
Ablate output norm. (3.3)	8648.94	6.68
Ablate LoRA init. (3.4)	19.69	6.63
Full Method	21.82	6.55

ing that each component contributes to the overall effectiveness of our approach.

Analysis - Computation and Throughput - Table 6 shows the normalized computation costs and throughputs for running our method compared to ShortGPT (Men et al., 2024) and the unpruned Llama-2 7B model on a single A100 GPU. The unpruned model serves as the baseline with 100% computation time and throughput. Our method incurs a marginal increase in compute cost compared to the unpruned model but achieves a reduction

Table 5: Perplexity (PPL) and zero-shot task performance of extended TinyLlama 1.1B models. All models underwent continued pre-training on 10B tokens from the SlimPajamas dataset (Soboleva et al., 2023).

Method	Layers	PPL	ARC-e	ARC-c	PIQA	WinoG.	HellaS.	Average
Base	22	6.84	55.34%	30.11%	73.29%	59.11%	59.20%	55.41%
FlexiGPT (Block)	36	6.73	56.90%	31.48%	73.23%	59.28%	59.77%	56.13%
FlexiGPT (Sequential)	36	6.76	56.94%	30.72%	73.78%	57.85%	59.32%	55.72%

Table 6: Normalized computation costs and throughputs for 1xA100 running FlexiGPT vs ShortGPT vs Unpruned on the Llama-2 7B model.

Method	Norm. Time	Throughput
Unpruned	100.0%	100.0%
ShortGPT	65.4%	152.8%
FlexiGPT	105.1%	95.1%

in the number of stored parameters by approximately 30%. Although our method is slower than ShortGPT, this is expected, as our approach involves replacing the pruned blocks with weight-sharing techniques. However, as shown in Table 1, our method offers significant performance gains over ShortGPT. These gains come at the expense of compute savings but are crucial for on-device applications that cannot tolerate the performance drop associated with methods like ShortGPT. Our method strikes a balance between computational efficiency and high performance, making it suitable for memory-constrained environments where performance is a critical factor.

In order to increase computational efficiency, we implemented a simple self-speculative decoding where the drafting stage uses FlexiGPT without the weight-sharing replacement layers (i.e., the same architecture as ShortGPT), and the verification stage uses the full FlexiGPT model. Importantly, no extra parameters or heads are needed, and our full model performance is retained. We achieved the same outputs as our model with a speedup of 30.11% compared to our naïve FlexiGPT decoding. We note that the speedup can be improved by combining our self-speculative decoding with other methods such as Medusa (Cai et al., 2024), Jacobi decoding (Santilli et al., 2023), or speculative decoding (Leviathan et al., 2023) with a smaller, separate model.

5 Model Extension with FlexiGPT

5.1 Setup

In the previous section, we showed that FlexiGPT is a powerful solution for *pruning and recovering*

LLMs. In this section, we show that FlexiGPT can also be used to extend an off-the-shelf LLM and *introduce performance gains with marginal parameter overhead*. We evaluated our method for model extension using TinyLLaMA (Zhang et al., 2024) due to its suitability for demonstrating the effectiveness of our approach in extending smaller models. The resources, framework, datasets, and benchmarks are the same as the previous section.

5.2 Results

Main Results - Table 5 shows the perplexity (PPL) and zero-shot task performance of extended TinyLlama 1.1B models after continued pre-training on 1B tokens from the SlimPajamas dataset. The base model with 22 layers serves as our baseline.

Our method, FlexiGPT, was evaluated with two extension strategies: Block and Sequential. Both strategies extend the model to 36 layers. FlexiGPT (Block) achieves the lowest PPL of 6.73, compared to the base model’s 6.84, indicating a more efficient model. In terms of zero-shot task performance, FlexiGPT (Block) consistently outperforms the base model across most tasks, with notable improvements in ARC-e (56.90% vs. 55.34%), ARC-c (31.48% vs. 30.11%), and HellaSwag (59.77% vs. 59.20%). FlexiGPT (Sequential) also shows competitive results with a PPL of 6.76. It achieves the highest performance in ARC-e (56.94%) and PIQA (73.78%) among the extended models. While it slightly underperforms compared to FlexiGPT (Block) in ARC-c and HellaSwag, its overall average performance of 55.72% still surpasses the baseline. While the downstream task accuracy margins are not as large as the last section, these results are highly significant in that *we are able to boost performance on all tasks using only 10B training tokens for a model which as already been trained on 30T tokens ($\approx 0.3\%$ extended training)*.

Analysis - Computation and Throughput - Table 7 compares the normalized computation costs and throughputs for running our method against TinyLlama 1.1B on a single A100 GPU. The base model serves as the baseline with 100% computa-

Table 7: Normalized computation costs and throughputs for 1xA100 running FlexiGPT vs Unpruned on the TinyLlama 1.1B model.

Method	Norm. Time	Throughput
Base	100.0%	100.0%
FlexiGPT	139.1%	71.9%

tion time and throughput. As expected, our method introduces an increased computation cost due to the extended effective length of our model, which is over 50% longer. However, these costs can be mitigated through strategies such as speculative decoding (Leviathan et al., 2023) or early-exit (Chen et al., 2023; Elhoushi et al., 2024; Pan et al., 2024), where the model is only extended when encountering particularly difficult tasks or data, effectively reducing the overall computation burden.

6 Conclusion

In this paper, we presented an approach to pruning and extending LLMs using LoRA and weight-sharing techniques. Our method targets memory-constrained devices by selectively pruning model blocks based on an importance score and replacing them with a low-parameter replacement strategy. Empirical evaluations show substantial performance gains over existing methods, highlighting our technique’s effectiveness. Furthermore, our approach can extend smaller models, achieving significant performance improvements with minimal additional parameters. This work paves the way for more accessible and efficient on-device NLP applications, leveraging our novel combination of pruning, weight-sharing, and parameter-efficient adapters, thereby bringing the power of LLMs to a broader range of memory-constrained devices and use cases.

7 Limitations

Our method, while effective in achieving parameter efficiency, does not provide gains in computational efficiency. The focus is primarily on reducing the model size for memory-constrained environments, which means that the computational load remains similar to the unpruned model during inference. Additionally, our approach involves a small post-pruning recovery phase where the model undergoes fine-tuning to regain performance. While this phase is crucial for restoring performance, it does require additional computational resources and time.

Our study was limited to evaluating three popular models, which may not cover the full spectrum of LLM architectures. However, the principles of our method are broadly applicable, and we have no reason to believe the results would not extrapolate to other models with similar architectures. Future work could involve testing our method on a wider variety of models to further validate its generalizability.

8 Broader Impact

Our method emphasizes parameter efficiency over computation efficiency, making it particularly valuable for on-device settings where memory and storage constraints are critical. By reducing the model size without significantly impacting performance, our approach enables the deployment of powerful LLMs on devices with limited resources, such as smartphones and edge devices. This can democratize access to advanced NLP capabilities, bringing sophisticated language understanding and generation tools to a broader range of users and applications.

Furthermore, our method can be used in conjunction with faster models, deploying the pruned model only for more complex tasks. This hybrid approach can virtually eliminate the computation cost on average while boosting performance for difficult tasks, requiring minimal parameter overhead. This flexibility in deployment can lead to more efficient and effective use of LLMs in various real-world applications.

9 Potential Risks

While our work is designed to move LLMs to on-device settings, thereby increasing security and data privacy, there are some potential risks. One risk is that our method involves a small post-training phase, unlike many one-shot pruning methods. This post-training phase could contribute to environmental impact as it requires additional compute, albeit to a smaller extent compared to the initial training of LLMs. Additionally, the ability to deploy LLMs on a wider range of devices could inadvertently lead to increased surveillance. Lastly, while our method emphasizes parameter efficiency, it does not address computational efficiency during inference, which might still pose challenges for extremely resource-constrained environments.

References

- AI@Meta. 2024. Llama 3 model card. <https://github.com/meta-llama/llama3/blob/main>.
- Saleh Ashkboos, Maximilian L Croci, Marcelo Genari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicept: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*.
- Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.
- 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.
- Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.
- Zouying Cao, Yifei Yang, and Hai Zhao. 2024. Head-wise shareable attention for large language models. *arXiv preprint arXiv:2402.11819*.
- Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. *arXiv preprint arXiv:2312.04916*.
- 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*.
- Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.
- Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27.
- Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.
- Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.
- Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, page 8.
- Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. 2023. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints*.
- Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.
- Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112*.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Jean Kaddour. 2023. The minipile challenge for data-efficient language models. *arXiv preprint arXiv:2304.08442*.
- Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.
- Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024a. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.
- Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.
- Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. 2024b. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*.

- Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.
- 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](#). *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*.
- Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.
- Xuchen Pan, Yanxi Chen, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024. Ee-tuning: An economical yet scalable solution for tuning early-exit large language models. *arXiv preprint arXiv:2402.00518*.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. [Exploring the limits of transfer learning with a unified text-to-text transformer](#). *arXiv e-prints*.
- Mohaimenul Azam Khan Raiaan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunos Ali, and Sami Azam. 2024. A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*.
- Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. 2021. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. *arXiv preprint arXiv:2101.00234*.
- 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.
- Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*.
- Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. Slimpajama: A 627b token cleaned and deduplicated version of redpajama. <https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama>.
- Sho Takase and Shun Kiyono. 2021. Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Tycho FA van der Ouderaa, Markus Nagel, Mart Van Baalen, and Tijmen Blankevoort. 2024. The llm surgeon. In *The Twelfth International Conference on Learning Representations*.
- 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.
- Yifei Yang, Zouying Cao, and Hai Zhao. 2024. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*.
- 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*.
- Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

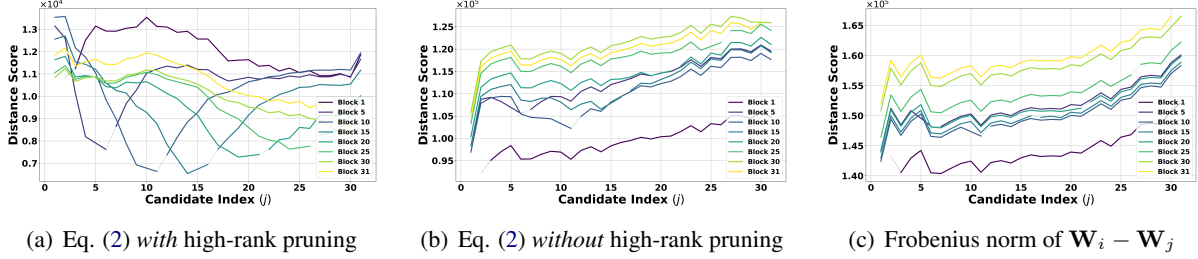


Figure 4: Comparison of block distance score versus candidate block index j for different metrics. Dotted lines represent where candidate block index j is equal to pruning block index i , which is not a valid candidate. (a) Using the proposed metric in Eq. (2) with high-rank pruning, showing that closer blocks score lower (better), matching our intuition that weights close in the model have similar function. (b) Ablation of high-rank pruning, where there is no clear trend except that blocks closer to 0 are lower and those closer to 31 are higher. (c) Simple Frobenius norm, showing a similar lack of clear trend as in (b). We found that using the score in (a) as the weight-sharing selection metric results in a much higher performing model compared to using the scores in (b) and (c).

Appendix

A Additional Experimental Details

Our implementations were carried out using PyTorch, utilizing Fully Sharded Data Parallel (FSDP) and FP-16 mixed precision training for enhanced efficiency. The experiments were conducted on a setup comprising 4 NVIDIA A100 80GB GPUs and required ≈ 192 gpu hours per experiment. While we only report a single run per result, we evaluate on several models and several tasks. For model handling and training, we employed the Hugging Face Transformers library. We used a learning rate of 0.004 with a cosine learning rate decay schedule, with a batch size of 2 per GPU and a total batch size of 480 achieved through gradient accumulation. The SlimPajamas dataset (Soboleva et al., 2023) train set was used, with 1B tokens dedicated to pruning experiments and 10B tokens for model extension experiments due to the faster processing speeds of the models. The LoRA rank utilized was 256. Compared to ShortGPT, our method incurs a 3.67% relative increase in total parameters for the main experiment setting of Table 1.

B Additional Method Analysis

In Figure 4, we show an alternative version of Figure 2 where the x-axis is candidate block index j instead of block index distance $(i - j)$. The purpose of this figure is to give an additional way to visualize the metrics which better highlights the issue in Figures 2(b) and 2(c) where all blocks i choose candidate block $j = 0$.

C Licenses of Datasets and Models

We used 1B tokens from the SlimPajama (Soboleva et al., 2023) pre-training dataset for post-prune recovery. For zero-shot performance evaluations, we used the ARC-e, ARC-c (Clark et al., 2018), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), and HellaSwag (Zellers et al., 2019) zero-shot benchmarks, utilizing the LM Evaluation Harness (Gao et al., 2021). For perplexity performance evaluations, we used the validation MiniPile (Kaddour, 2023) subset of the Pile dataset (Gao et al., 2020). We confirmed that the data that was used does not contain any information that names or uniquely identifies individual people or offensive content by checking their distribution sources. All datasets use the English language. For the pruning experiments, we evaluated our method using LLaMA-2 7B (Touvron et al., 2023), OPT 1.3B and 6.7B (Zhang et al., 2022), and LLaMA-3 8B (AI@Meta, 2024), focusing on these due to their widespread adoption by the community. For the model extension experiments, we evaluated our method for model extension using TinyL-LaMA (Zhang et al., 2024) due to its suitability for demonstrating the effectiveness of our approach in extending smaller models.

The licenses for the datasets and models used in this paper are as follows:

- **SlimPajama:** Apache License 2.0
- **ARC:** CC BY-SA 4.0
- **PIQA:** Academic Free License 3.0
- **HellaSwag:** MIT License

- **WinoGrande**: Apache License 2.0
- **MiniPile**: MIT License
- **LLaMA**: Meta LLaMA Community License Agreement
- **OPT**: OPT License Agreement
- **TinyLLaMA**: Apache License 2.0

We used the datasets and models purely for scientific research purposes to create this paper, which is within the scope of their licenses and intended uses.