RankAdaptor: Hierarchical Rank Allocation for Efficient Fine-Tuning Pruned LLMs via Performance Model

Changhai Zhou[*1,2], **Shijie Han**[*1,3], **Lining Yang**[4],
Yuhua Zhou[5], **Xu Cheng**[6], **Yibin Wang**[1,2], **Hongguang Li**[†1]

[1]JF SmartInvest Holdings Ltd, [2]Fudan University, [3]Columbia University,
[4]King's College London, [5]Zhejiang University, [6]Wuhan University

Abstract

The efficient compression of large language models (LLMs) has become increasingly popular. However, recovering the performance of compressed LLMs remains a major challenge. The current practice in LLM compression entails the implementation of structural pruning, complemented by a recovery phase that leverages the Low-Rank Adaptation (LoRA) algorithm. Structural pruning's uneven modification of model architecture, coupled with standard LoRA's fixed configuration allocation across layers in an online pipeline, leads to suboptimal performance in various downstream tasks for pruned models. To address this challenge, we introduce RankAdaptor, a hierarchical rank allocation method that enables efficient fine-tuning of pruned LLMs according to layerwise specific recovery requirements. We employ a performance model that conducts offline meta-learning and online incremental learning to explore optimal rank values for each layer. Comprehensive experiments on popular benchmarks show that RankAdaptor consistently outperforms state-of-the-art methods across a variety of pruning settings and LLM architectures, with improvements ranging from 0.7% to 5.5%.

1 Introduction

In recent years, large language models (LLMs) have provided innovative solutions across various natural language processing (NLP) tasks, such as machine translation (Zhang et al., 2023a; Sato et al., 2020; Aycock and Bawden, 2024), sentiment analysis (Zhang et al., 2023b; Deng et al., 2023), and speech recognition (Min and Wang, 2023; Fathullah et al., 2024).

However, the exceptional performance of LLMs comes at the cost of a massive number of parameters and high-end hardware resources. Current compression techniques like pruning (Ma et al., 2023; Xia et al., 2023; Santacroce et al., 2023; Frantar and Alistarh, 2023), quantization (Shao et al., 2023; Lee et al., 2023), and distillation (Gu et al., 2023; Tan et al., 2023) have been explored. Compressed LLMs typically require fine-tuning to recover their original performance. Therefore, designing an efficient algorithm for compressed LLMs to achieve optimal performance on downstream tasks has become a pioneering direction.

Method	BoolQ	PIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg Diff
AdaLoRA	61.90	76.31	67.82	62.57	**64.02**	36.70	40.40	**0.56**
LoRA	**63.30**	**76.82**	**68.68**	**63.38**	63.76	**37.11**	**40.60**	

Table 1: Zero-shot performance comparison between AdaLoRA (Zhang et al., 2023d) and LoRA (Hu et al., 2021). 'Bold' indicates better performance. 'Avg Diff' represents the average performance difference between AdaLoRA and LoRA across all benchmarks. The results are reported in percentage (%).

Among compression techniques, structural pruning is a popular one that removes redundant weight connections to reduce model size and computational requirements. It primarily involves two stages: (1) pruning based on architectural importance and (2) recovery using efficient fine-tuning. While research has primarily focused on the initial pruning stage, the equally crucial recovery stage has been understudied. Existing approaches often rely on standard LoRA (Hu et al., 2021) for recovering pruned models, applying a general rank configuration across all layers. However, this approach overlooks the inherent structural irregularities introduced by pruning. Therefore, a one-size-fits-all rank configuration may not optimally meet the unique needs of each layer, potentially affecting downstream performance.

Among the various LoRA variants, AdaLoRA (Zhang et al., 2023d) proposes an importance-based adaptive rank allocation method. It dynamically adjusts the rank configurations for each layer by continuously estimating the model structure's impor-

[*] Equal contribution: zhouch23@m.fudan.edu.cn & sh4460@columbia.edu

[†] Corresponding author: harvey2@mail.ustc.edu.cn

Findings of the Association for Computational Linguistics: NAACL 2025, pages 5796–5810
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

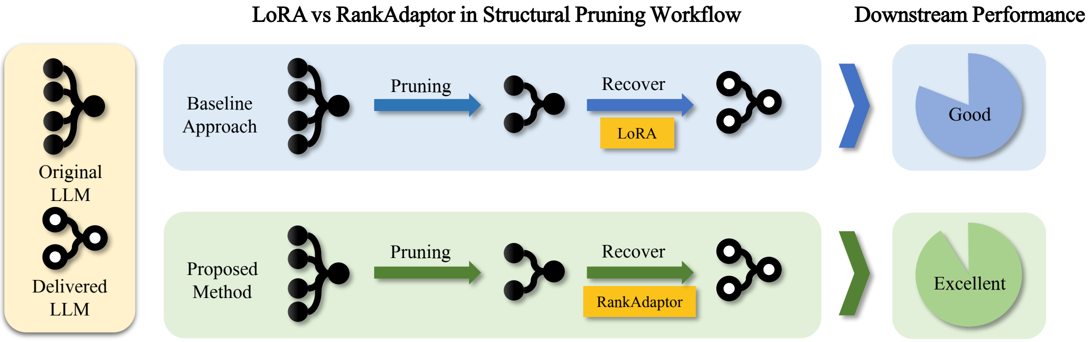

Figure 1: Illustration of the process of pruning and recovery. The baseline approach is detailed in Section 2, and the proposed method is described in Section 3.

tance through SVD parameterization during fine-tuning. Despite AdaLoRA's proven advantages in fine-tuning original LLMs, its effectiveness in recovering accuracy for irregularly pruned models falls short of standard LoRA. We conducted experiments using a 20% pruned LLaMA-7B model, comparing LoRA with rank=8 and AdaLoRA for performance recovery. As shown in Table 1, AdaLoRA's performance across seven tasks averages 0.56% lower than LoRA. The suboptimal performance of AdaLoRA may be attributed to its difficulty in identifying the highly complex structures of pruned models during its online rank adjustment process. Given this observation, we propose adopting a static rank allocation strategy for each layer during the recovery stage of pruned models.

To achieve this goal, we propose RankAdaptor, an algorithm that leverages a performance model to statically determine the optimal rank configuration for each layer in the recovery stage for the pruned models. Our contributions are as follows:

1. We point out a critical bottleneck in pruned LLM recovery: existing fine-tuning approaches fail to address the unique requirements of pruned models' complex structures.

2. We introduce RankAdaptor, a tailored fine-tuning strategy specifically designed for recovering pruned models. Our approach employs a performance model that combines offline meta-learning with online incremental learning to efficiently explore optimal hierarchical rank configuration.

3. Extensive experimentation has demonstrated that RankAdaptor consistently outperforms the state-of-the-art method across a range of pruning configurations and LLM architectures, with improvements ranging from 0.7% to 5.5%.

2 Background and Motivation

There are numerous compression methods for LLMs and our work focuses on pruning. The entire pruning process for LLMs primarily consists of two main stages: (1) Pruning based on structural importance, and (2) Recovery using efficient fine-tuning, typically with LoRA.

Pruning Stage. This stage involves identifying and removing less important structures within the LLM. The process begins by establishing structural dependencies among neurons. A neuron N_j is considered dependent on neuron N_i if:

$$\begin{aligned} N_j \in \mathrm{Out}(N_i) \wedge \mathrm{Deg}^-(N_j) = 1 \\ \Rightarrow N_j \text{ is dependent on } N_i \end{aligned} \tag{1}$$

where $\mathrm{Deg}^-(N_j)$ represents the in-degree of N_j. This dependency means that if N_i is pruned, N_j must also be pruned. The process identifies and groups dependent neurons, forming clusters of interconnected structures.

The importance of each group is then assessed using a Taylor expansion-based formula:

$$\begin{aligned} I_{\mathbf{W}_\mathrm{i}^\mathrm{k}} \approx \Bigg| \frac{\partial \mathcal{L}}{\partial \mathbf{W}_\mathrm{i}^\mathrm{k}} \mathbf{W}_\mathrm{i}^\mathrm{k} - \frac{1}{2} \sum_{j=1}^{N} \left(\frac{\partial \mathcal{L}_j}{\partial \mathbf{W}_\mathrm{i}^\mathrm{k}} \mathbf{W}_\mathrm{i}^\mathrm{k} \right)^2 \\ + \mathcal{O}(\|\mathbf{W}_\mathrm{i}^\mathrm{k}\|^3) \Bigg| \end{aligned} \tag{2}$$

where $\mathbf{W}_\mathrm{i}^\mathrm{k}$ is the k-th parameter in structure $\mathbf{W}_\mathrm{i}$, and $\mathcal{L}$ is the loss for next-token predictions. The groups are then ranked by importance, and those with lower significance are pruned based on a predefined ratio.

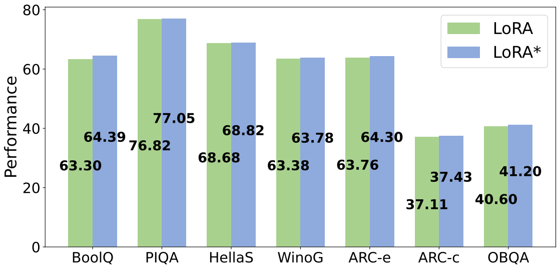

Figure 2: Performance of benchmarks for the different fine-tuning configurations. LoRA denotes using fixed ranks for different layers, whereas LoRA* indicates using different rank configurations. The results are reported in percentage (%).

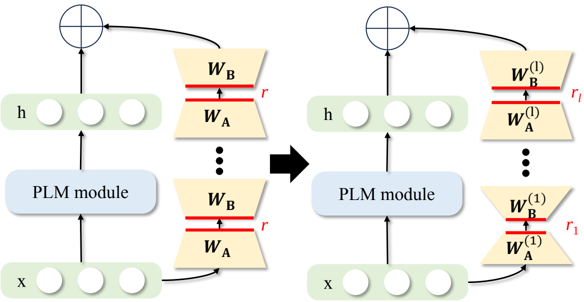

Figure 3: Hierarchical weight matrices decomposition: same rank in LoRA (left) versus hierarchical different ranks in RankAdaptor (right).

Recovery Stage. After pruning, the model's performance is recovered using efficient fine-tuning, typically through LoRA. In LoRA, for each layer of the pruned LLM, the weight update matrix $\Delta\mathbf{W}$ is decomposed into the product of two low-rank matrices $\mathbf{A}$ and $\mathbf{B}$:

$$\begin{aligned} f(x) &= (\mathbf{W} + \Delta\mathbf{W})\mathbf{X} + \mathbf{b} \\ &= (\mathbf{W}\mathbf{X} + \mathbf{b}) + (\mathbf{A}\mathbf{B})\mathbf{X} \end{aligned} \tag{3}$$

where $\mathbf{A} \in \mathbb{R}^{d\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times d}$, with r being the rank, typically fixed across all layers. Only $\Delta\mathbf{W}$ (i.e., $\mathbf{AB}$) is updated during fine-tuning, while the original weight matrix $\mathbf{W}$ remains frozen. This approach significantly reduces the number of trainable parameters and computational cost from d^2 to $2dr$, as r is usually much smaller than d.

A Motivating Example. The uneven distribution of importance within LLMs' internal architecture (Zhang et al., 2023d), coupled with importance-based pruning criteria, leads to non-uniform pruning across layers. This results in a highly complex structure for the pruned LLM. While standard LoRA with fixed rank configurations offers some recovery, it falls short in addressing the specific recovery needs of differently pruned layers, leading to suboptimal performance.

Studies (Vaswani et al., 2017; Zhao et al., 2024) indicate that bottom layers in LLMs capture more semantic information, making them more powerful. Based on this insight, we explore two approaches for an LLaMA-7B model with a 20% pruning ratio. Standard LoRA applies a fixed rank configuration of 8 across all layers, and LoRA* assigns increasing rank configuration from the bottom to the top layers. Specifically, in LoRA*, layers 1–8 use rank 4, layers 9–16 use rank 6, layers 17–24 use rank 10, and layers 25–32 use rank 12.

The exploration in Figure 2 demonstrates the efficacy of different rank allocations for recovering pruned LLMs. LoRA with fixed rank configurations demonstrates general accuracy since its inability to meet the specific recovery requirements of different layers results in performance inferior to LoRA* across most tasks. LoRA* achieves superior recovery performance by gradually adjusting rank configurations for each layer during fine-tuning.

Through this motivating example, we have demonstrated the effectiveness of using different rank configurations across layers for fine-tuning pruned models. However, determining the optimal rank configuration allocation for each layer remains a challenge. In the next section, we propose a method that uses a performance model to find the best combination of rank configuration for each layer.

3 Methodology

In this section, we propose RankAdaptor, a hierarchical rank allocation tailored for fine-tuning pruned LLMs. The visible comparison from LoRA can be found in Figure 3.

3.1 Problem Definition

As mentioned in Section 2, using a globally fixed rank r in LoRA during the recovery stage can lead to suboptimal performance. While AdaLoRA attempts to dynamically adjust rank configurations for different layers during recovery, it is designed for unpruned models and proves unsuitable for pruned models. In addition, testing all combinations in the solution space S is impractical for LLMs due to the vast number of layers and potential rank configurations. With n rank candidates and l layers, the number of combinations n^l be-comes astronomically large, rendering exhaustive evaluation unfeasible.

Problem Formulation. Given a pruned model PL and a collection of rank configurations for all layers R_{H}, our overall objective is to identify the optimal rank set R_{H}^{*} that maximizes the recovery performance. This can be expressed as:

$$R_{\mathrm{H}}^{*} = \underset{R_{\mathrm{H}} \in S}{\arg\max}\, \mathcal{P}(recover(PL, R_{\mathrm{H}})), \quad (4)$$

where $\mathcal{P}(recover(PL, R_{\mathrm{H}}))$ represents the actual performance of recovering PL with R_{H}. However, finding R_{H}^{*} directly poses significant challenges. To address these, we propose an efficient method utilizing a predictive performance model, detailed in Section 3.2.

Training Objective. Let $\mathcal{Q}(R_{\mathrm{H}})$ denote the recovered performance predicted by our performance model, approximating $\mathcal{P}(recover(PL, R_{\mathrm{H}}))$. If we can develop a model that takes R_{H} as input and directly produces a performance prediction closely approximating the actual performance, we can efficiently explore a wide range of R_{H} configurations at minimal cost and select the optimal one.

Consequently, we can formulate our training objective to focus on training a reliable performance prediction model, which means minimizing the discrepancy between the actual performance and the predicted performance:

$$\min \left[\mathcal{P}(recover(PL, R_{\mathrm{H}})) - \mathcal{Q}(R_{\mathrm{H}})\right]^{2}. \quad (5)$$

3.2 Performance Model.

A performance model is constructed to estimate the performance of the recovered model fine-tuned by R_{H} on downstream tasks. Input R_{H} to obtain $\mathcal{Q}(R_{\mathrm{H}})$, which is a predicted configuration of $\mathcal{P}(recover(PL, R_{\mathrm{H}}))$.

Model Architecture Our performance model is inspired by the MLP architecture, featuring a simple and efficient structure that minimizes overhead during forward inference and backward propagation. We define the performance model as an MLP network comprising five fully connected layers. The input layer accepts R_{H}, representing the fine-tuning rank configurations of the pruned LLM as an l-dimensional vector, where l is the number of layers in the LLM. Each hidden layer has a dimension D_i where $i = 1, 2, 3$, which can be adjusted as needed. The final output layer employs a linear activation function to generate a single scalar value representing the predicted performance score.

Model Integration. The performance model operates in two distinct phases.

1. **Offline meta-learning:** Before actual fine-tuning, we pre-train the performance model on multiple diverse datasets. This meta-learning approach endows the model with the ability to quickly adapt to new tasks and datasets, providing a foundation of generalized knowledge about rank configuration performance across various scenarios.

2. **Online incremental learning:** Once a specific downstream task is identified, the performance model is integrated into the RankAdaptor workflow. This phase enables rapid and accurate performance estimation for a large set of candidate rank configurations on the specific downstream task. By incrementally updating its knowledge based on task-specific data, the performance model refines its predictions to better align with the unique characteristics of the target task.

3.3 RankAdaptor

Overview. Combined performance model, we propose RankAdaptor, a learning-based algorithm, as shown in Figure 4, to allocate rank configuration for each layer. There are three important phases in our design: Initialization, Iteration, and Convergence. The first phase involves meta-learning pretraining of the performance model using multiple datasets. The function of his phase is to equip the model with fundamental learning abilities and generalization capabilities. The subsequent iterative phase focuses on incrementally enhancing the model's predictive power through continuous learning on specific tasks. In the final convergence stage, the performance model is utilized to predict performance for a large number of candidate rank configurations from the S. This process enables the selection of the optimal configuration that demonstrates superior performance on downstream tasks.

Initialization Phase. At the beginning of RankAdaptor, a lightweight performance model offline using meta-learning techniques is introduced. This process involves randomly selecting multiple configurations of R_{H} from the solution space S, with a different set of R_{H} being chosen for each step. The actual performance $\mathcal{P}(\mathrm{recover}(PL, R_{\mathrm{H}}))$

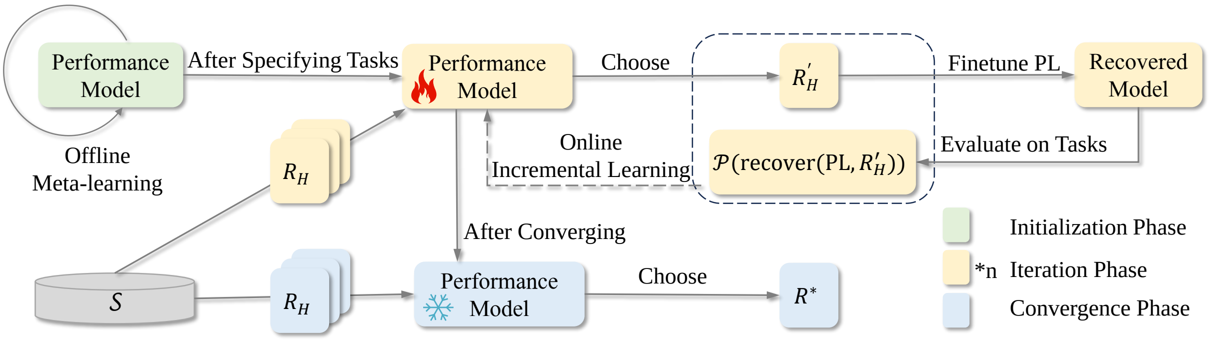

Figure 4: RankAdaptor Workflow: Through three phases (Initialization-Iteration-Convergence), find the optimal hierarchical rank configuration for recovering pruned LLM.

of these configurations across various datasets is used as a training set to update the performance model, serving as a foundation for subsequent optimization steps.

Iteration Phase. After the initialization phase, we enter the iteration phase. Here, many R_{H} samples are randomly drawn from S at each iteration. The performance model with the learning ability gained from the last phase predicts their corresponding performance $\mathcal{Q}(R_{\text{H}})$. Based on these predictions, we identify the optimal R'_{H} for the current iteration, which is expected to yield the best accuracy on the downstream task. We then real fine-tune the PL using R'_{H} and evaluate its actual performance $\mathcal{P}(\text{recover}(PL, R'_{\text{H}}))$ on tasks. This performance data $(R'_{\text{H}}, \mathcal{P}(\text{recover}(PL, R'_{\text{H}}))$ is fed back into the performance model, enabling continuous improvement of its predictive accuracy through successive iterations.

Convergence Phase. The process continues until the discrepancy between predicted performance $\mathcal{Q}(R_{\text{H}})$ and actual performance $\mathcal{P}(recover(PL, R_{\text{H}}))$ falls within a predetermined threshold. At this point, the performance model is considered converged. RankAdaptor can then efficiently identify the optimal R^*_{H} from many R_{H} in S that maximizes the actual performance metrics of the pruned model.

4 Experiments

4.1 Experimental Setup

LLMs and Benchmarks. To demonstrate the effectiveness of RankAdaptor, we test it on three open-source LLMs: LLaMA-7B[1] (Touvron et al., 2023), LLaMA-13B[2] (Touvron et al., 2023) and Vicuna-7B[3] (Zheng et al., 2024). We conduct these LLMs on zero-shot classification tests for commonsense reasoning datasets, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy (Clark et al., 2018), ARC-challenge (Clark et al., 2018), and OpenbookQA (Mihaylov et al., 2018).

Baseline and Configuration. We employ both LLM-Pruner (Ma et al., 2023) and Shortened LLaMA (Taylor+) (Kim et al., 2024) as pruning stage operations, which allow us to validate our method's effectiveness across different pruning strategies. We apply AdaLoRA (Zhang et al., 2023d) and LoRA (Hu et al., 2021) as recovery methods compared with our RankAdaptor.

Previous research has identified that specific layers of LLaMA-7B, Vicuna-7B, and LLaMA-13B are crucial to the models' architecture and should remain unpruned (Ma et al., 2023). Thus, we prune only layers 5-30 of LLaMA-7B and Vicuna-7B, and layers 5-36 of LLaMA-13B to achieve the predefined global pruning rate. Specifically, we prune 25%, 32%, 38%, and 63.5% of the middle layers to attain global pruning rates of 20%, 25%, 30%, and 50%, respectively. For the unpruned layers, we maintain their rank configurations consistent with those of standard LoRA.

Implementation Details. Our implementation utilizes PyTorch 2.1.2, Transformers 4.41.0, and PEFT 0.6.0 libraries, running on CUDA 12.4. The hardware setup consists of an NVIDIA A800 GPU with 80GB memory, operating on Ubuntu. The MLP dimensions for the inner layers of the perfor-

[1] https://huggingface.co/baffo32/decapoda-research-llama-7B-hf

[2] https://huggingface.co/yahma/llama-13b-hf

[3] https://huggingface.co/lmsys/vicuna-7b-v1.5

Pruning Stage		Recover	BoolQ	PIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg
LLM-Pruner	Rate = 20%	AdaLoRA	61.90	76.31	67.82	62.57	64.02	36.70	40.40	58.53
		LoRA	63.30	76.82	68.68	63.38	63.76	37.11	40.60	59.09
		RankAdaptor	**67.34**	**77.31**	**69.07**	**64.17**	**65.36**	**37.80**	**41.60**	**60.38**
	Rate = 25%	AdaLoRA	60.31	75.82	64.57	61.30	61.88	35.10	39.00	56.85
		LoRA	61.93	76.01	**66.08**	61.96	62.21	35.92	39.40	57.64
		RankAdaptor	**67.43**	**76.06**	65.97	**64.40**	**62.63**	**36.77**	**40.40**	**59.09**
	Rate = 30%	AdaLoRA	59.85	73.30	61.53	60.12	59.67	33.21	38.80	55.21
		LoRA	62.45	74.37	63.14	61.96	59.22	33.70	39.60	56.35
		RankAdaptor	**66.21**	**75.19**	**63.61**	**63.14**	**60.10**	**34.64**	**40.20**	**57.58**
	Rate = 50%	AdaLoRA	38.25	67.86	42.80	48.32	42.20	26.31	32.80	42.65
		LoRA	43.76	69.04	45.01	50.99	**45.66**	**28.75**	34.60	45.40
		RankAdaptor	**51.65**	**69.48**	**45.03**	**51.93**	45.20	28.41	**35.00**	**46.67**
Shortened	Rate = 20%	AdaLoRA	70.14	73.85	68.92	65.73	64.93	37.12	39.34	60.00
		LoRA	71.82	**75.31**	70.50	67.36	64.40	38.60	40.80	61.26
		RankAdaptor	**74.53**	75.22	**72.81**	**69.92**	**66.72**	**40.23**	**42.31**	**63.11**
	Rate = 25%	AdaLoRA	67.93	71.54	65.23	63.14	61.83	35.53	36.13	57.33
		LoRA	69.67	73.20	66.85	64.50	63.32	37.02	37.40	58.85
		RankAdaptor	**72.32**	**75.13**	**68.91**	**66.82**	**65.24**	**38.54**	**38.92**	**60.84**
	Rate = 30%	AdaLoRA	61.83	69.72	61.24	62.34	58.13	33.23	35.54	54.58
		LoRA	63.58	71.21	62.75	63.80	**59.42**	34.50	36.80	56.01
		RankAdaptor	**66.23**	**73.42**	**64.82**	**65.93**	59.31	**35.84**	**38.13**	**57.67**
	Rate = 50%	AdaLoRA	44.93	61.24	41.83	52.84	42.93	28.73	32.24	43.53
		LoRA	46.52	**62.76**	43.12	54.37	44.10	**30.07**	33.50	44.92
		RankAdaptor	**48.94**	62.43	**45.24**	**56.53**	**46.14**	29.82	**34.84**	**46.28**

Table 2: Zero-shot performance of pruned LLaMA-7B with AdaLoRA, LoRA, and RankAdaptor recovery. 'Bold' indicates the best performance at each pruning rate. 'Avg' represents the average performance across all benchmarks. Specific rank configurations explored by RankAdaptor are listed in Appendix B. Reported in percentage (%).

mance model are set to (32-32-32-1), meaning each inner MLP consists of three hidden layers with 32 neurons and an output layer with a single neuron. Micro-batch size is configured to 16, which specifies the number of examples processed in each step of model training.

Rank Configuration Candidates and Solution Space. In standard LoRA, setting fixed rank configurations within the range of 2 to 16 achieves favorable model recovery. To ensure that the trainable parameter count of RankAdaptor remains at the same level as standard LoRA, the range of rank configurations in this experiment is set to $\{2, 4, 6, 8, 10, 12\}$. For LLaMA-7B and Vicuna-7B, which have 26 pruned layers, the size of the solution space is 6^{26}. For LLaMA-13B, with 32 pruned layers, the size of the solution space is 6^{32}. Different models follow the same calculation pattern.

4.2 Main Results

Analysis. We present the performance of the recovered LLM finetuned by AdaLoRA, LoRA, and RankAdaptor on each benchmark in Table 2 below, and Tables 5 and 6 in the appendix. The performances of pruned LLM without recovery are listed in Appendix A.2. We have illustrated the specific configuration of the rank configuration explored by RankAdaptor in LLaMA-7B in Appendix B.

RankAdaptor shows strong performance under different pruning strategies. Whether applied with the LLM-Pruner or the Shortened, RankAdaptor generally achieves the highest average scores across all benchmarks. This adaptability to different pruning approaches further highlights its robustness as a pruning recovery method.

At lower pruning rates (20-25%), RankAdaptor shows remarkable effectiveness. For instance, in the LLaMA-13B model with a 20% pruning rate, RankAdaptor achieves the highest scores in 6 out of 7 tasks. This trend continues with Vicuna-7B, where RankAdaptor leads in most tasks at 20% and 25% pruning rates. Even at higher pruning rates (30-50%), where performance typically degrades more significantly, RankAdaptor maintains its edge. In the challenging scenario of 50% pruning, RankAdaptor still manages to outperform other methods in most tasks for both LLaMA-13B and Vicuna-7B.

Furthermore, RankAdaptor's effectiveness is also consistent across different types of tasks. In language-understanding tasks like BoolQ and HellaSwag, as well as in more reasoning-focused tasks like ARC and OBQA, RankAdaptor consistently achieves the best or near-best performance. This broad-spectrum effectiveness suggests that RankAdaptor is adept at preserving various aspects of language model capabilities, from basic comprehension to more complex reasoning.

In summary, the results offer substantial evidence in support of RankAdaptor's efficacy as a pruning recovery approach. Its consistent superiority across diverse scenarios illustrates that RankAdaptor is a highly effective technique, irrespective of model size, architecture, or pruning rate.

Generation Performance. Complementing the evaluation of model performance on classification tasks in the experiments, we further investigate the generative capabilities of the recovered models. Notably, we conduct text generation tasks using LLaMA-7B and Vicuna-7B models recovered by LoRA and RankAdaptor at a 20% pruning rate, as detailed in Appendix C. The results are remarkably promising. For article continuation, the models recovered by RankAdaptor demonstrate superior coherence in their generated sentences. Similarly, when tasked with step listing, RankAdaptor-recovered LLMs produce clearer and more logical step sequences. These compelling comparative results are illustrated in Figure 5 and 6, showcasing the potential of RankAdaptor in preserving and enhancing generative abilities during model compression and recovery.

4.3 Ablation Study

We prune LLaMA-7B with a 20% global pruning rate and the RankAdaptor to recover. More details can be found in Table ??.

Sample Size. We conduct ablation experiments to assess the impact of the estimated sample size during the pruning phase in LLM-Pruner. The larger the estimated sample size in the pruning operation, the better it can evaluate the importance of the model architecture and perform better pruning effects. So we compare performance with $N = 10$ and 50, and results demonstrate that increasing the sample size to $N = 50$ leads to better outcomes. However, while a larger sample size ($N = 50$) tends to improve performance for most tasks, there are instances where the smaller sample size ($N = 10$) yields competitive results, such as in WinoGrande. This underscores the need for careful selection of sample size based on the specific requirements of the task.

Micro-batch Sizes. We finally assess the impact of different micro-batch sizes (4, 8, and 16) in fine-tuning process. The results indicate that larger micro-batch sizes can lead to better performance on certain tasks, though not universally across all benchmarks.

Element-wise Importance. We further conduct tests on the LLM-Pruner's importance estimation techniques. The results compare the first-order (Element^1) and second-order (Element^2) Taylor approximations for evaluating the importance of each parameter, as described in Equation 2. Our findings indicate that Element^1 provides better performance than Element^2 across most benchmarks. While higher-order derivatives may theoretically offer more precise adjustments, their complexity may outweigh the marginal performance gains observed in practice.

Setting of Performance Model. To investigate the impact of different inner MLP dimensions in the performance model, we test three configurations. The first setting consists of three hidden layers with 32 neurons each, followed by an output layer with a single neuron, abbreviated as 32-32-32-1. The other two configurations are 32-64-32-1 and 32-16-32-1, following the same notation. The results illustrate that varying dimensions of inner MLP layers have nuanced impacts on performance across different benchmarks. For inner MLP dimensions, Setting1 provides the highest performance on tasks such as ARC-e and BoolQ, while Setting3 shows competitive performance on PIQA and HellaSwag.

5 Related Work

5.1 Efficient Pruning of LLMs

LLM-Pruner (Ma et al., 2023) employs structured pruning to remove non-essential interconnected

Benchmark	Sample Size		Micro-batch Size			Element-wise Importance		Setting of Performance Model		
	N=10	N=50	Micro-4	Micro-8	Micro-16	Element[1]	Element[2]	Setting1	Setting2	Setting3
ARC-e	63.97	**65.32**	64.52	63.97	**65.24**	**63.97**	62.84	63.97	64.73	64.65
ARC-c	37.29	**37.71**	**38.65**	37.29	37.54	**37.29**	36.77	37.29	36.60	**37.54**
WinoG	**63.61**	63.14	62.04	**63.61**	63.14	**63.61**	63.22	63.61	**63.46**	63.06
OBQA	**39.80**	**41.00**	40.00	39.80	**40.80**	**39.80**	39.80	39.80	**40.80**	**40.80**
BoolQ	**65.81**	64.43	**67.28**	65.81	66.91	65.81	**66.48**	**66.91**	64.43	64.86
PIQA	76.99	**77.15**	76.50	**76.99**	76.93	**76.99**	76.82	76.99	76.99	**77.04**
HellaS	**68.56**	68.52	68.08	68.56	**68.78**	**68.56**	67.88	68.56	68.75	**69.00**

Table 3: Ablation study results comparing performance across seven tasks using LLM-Pruner on LLaMA-7B. 'Bold' indicates better performance. The results are reported in percentage (%).

structures by utilizing gradient information. This approach allows compressed models to restore good performance in multitasks with basic fine-tuning. Xia et al. (2023) introduces "Sheared LLaMA" to compress pre-trained LLMs. It employs dynamic batch loading to improve data efficiency during pruning and retraining. Santacroce et al. (2023) presents Globally Unique Movement (GUM), a novel pruning technique that focuses on the sensitivity and uniqueness of LLMs' network components. GUM selects models' neurons that uniquely contribute to model output and are sensitive to loss changes to prune, thus maintaining high accuracy. SparseGPT (Frantar and Alistarh, 2023) transforms the pruning process into a series of large-scale sparse regression problems, which can be quickly solved through Hessian matrix inversion. It efficiently prunes large models to high sparsity in a single step while maintaining high accuracy. Wanda (Sun et al., 2023) prunes LLMs by selectively removing weights based on their sizes and input activations. It adaptively adjusts sparsity levels to achieve a reduction of more than half without sacrificing accuracy.

5.2 Parameter Efficient Fine-Tuning

Houlsby et al. (2019) introduce a transfer learning method that integrates adapter modules into pre-trained Transformer models. It can efficiently tackle various NLP tasks with few additional parameters and achieve performance similar to full fine-tuning. While the adapter takes a serial approach to integrating trainable components into pre-trained Transformer models, low-rank adaptation (LoRA) (Hu et al., 2021) presents a parallel method of infusing rank decomposition matrices into each layer of the model's architecture. Specifically, LoRA adds trainable matrices to each layer of the model and the pre-trained weights are kept the same. LoRA reduces the number of trainable parameters compared to fine-tuning the entire model, which makes model adaptation faster and less resource-intensive. LoRA-FA (Zhang et al., 2023c) freezes the projection-down weight of the low-rank adaptation (LoRA) layers and only updates the projection-up weight to reduce the memory requirements for fine-tuning. Zhang et al. (2023d) have introduced AdaLoRA, which achieves excellent performance by parameterizing updates in SVD form and employing a novel importance metric to dynamically adjust hierarchical rank configurations during the fine-tuning process.

6 Conclusion

In this work, we present RankAdaptor, an innovative fine-tuning algorithm specifically designed to recover the performance of pruned LLMs. RankAdaptor employs a hierarchical fine-tuning strategy, incorporating a lightweight performance model to optimize rank configuration across different layers. This methodology effectively mitigates the drawbacks of the standard fixed-rank LoRA, which often results in suboptimal performance recovery due to the uneven architectural adjustments caused by structural pruning. Through extensive evaluations of multiple open-source LLMs and benchmark tasks, we demonstrate that RankAdaptor consistently outperforms the standard LoRA approach across various pruning scenarios. The introduction of RankAdaptor marks a significant progression in fine-tuning pruned LLMs. Its adaptive rank scheduling and end-to-end optimization lead to substantial enhancements over traditional techniques, positioning it as a promising tool for boosting the performance of pruned language models in diverse applications.

Limitations

Our methodology delves into the optimization of fine-tuning procedures for pruned models, albeit the offline algorithm necessitates supplementary training data, thereby introducing some level of overhead. Subsequent endeavors will be directed towards refining this phase and integrating a broader range of quantization algorithms, with a focus on effectively fine-tuning the quantized model to achieve final performance and accuracy.

References

Seth Aycock and Rachel Bawden. 2024. Topic-guided example selection for domain adaptation in llm-based machine translation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 175–195.

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.

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

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*.

Xiang Deng, Vasilisa Bashlovkina, Feng Han, Simon Baumgartner, and Michael Bendersky. 2023. What do llms know about financial markets? a case study on reddit market sentiment analysis. In *Companion Proceedings of the ACM Web Conference 2023*, pages 107–110.

Yassir Fathullah, Chunyang Wu, Egor Lakomkin, Junteng Jia, Yuan Shangguan, Ke Li, Jinxi Guo, Wenhan Xiong, Jay Mahadeokar, Ozlem Kalinli, et al. 2024. Prompting large language models with speech recognition abilities. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 13351–13355. IEEE.

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.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

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.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: Asimple depth pruning for large language models.

Janghwan Lee, Minsoo Kim, Seungcheol Baek, Seok Hwang, Wonyong Sung, and Jungwook Choi. 2023. Enhancing computation efficiency in large language models through weight and activation quantization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14726–14739.

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.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

Zeping Min and Jinbo Wang. 2023. Exploring the integration of large language models into automatic speech recognition systems: An empirical study. In *International Conference on Neural Information Processing*, pages 69–84. Springer.

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.

Michael Santacroce, Zixin Wen, Yelong Shen, and Yuanzhi Li. 2023. What matters in the structured pruning of generative language models? *arXiv preprint arXiv:2302.03773*.

Shoetsu Sato, Jin Sakuma, Naoki Yoshinaga, Masashi Toyoda, and Masaru Kitsuregawa. 2020. Vocabulary adaptation for domain adaptation in neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4269–4279.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Shicheng Tan, Weng Lam Tam, Yuanchun Wang, Wenwen Gong, Shu Zhao, Peng Zhang, and Jie Tang. 2023. Gkd: A general knowledge distillation framework for large-scale pre-trained language model. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 134–148.

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*.

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.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*.

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

Biao Zhang, Barry Haddow, and Alexandra Birch. 2023a. Prompting large language model for machine translation: A case study. In *International Conference on Machine Learning*, pages 41092–41110. PMLR.

Boyu Zhang, Hongyang Yang, Tianyu Zhou, Muhammad Ali Babar, and Xiao-Yang Liu. 2023b. Enhancing financial sentiment analysis via retrieval augmented large language models. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pages 349–356.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023c. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023d. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview.

Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for large language models: A survey. *ACM Transactions on Intelligent Systems and Technology*, 15(2):1–38.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

A More Results and Analysis of Experiments

A.1 Performance of Different Model Varients.

We list the performance of the Vicuna-7B in Table 5 and LLaMA-13B in Table 6. These results demonstrate the remarkable versatility and effectiveness of our RankAdaptor method across various dimensions. Firstly, the method exhibits consistent superior performance across different model architectures, as evidenced by its effectiveness on both Vicuna-7B and LLaMA-13B models. This cross-model applicability underscores the scalability and adaptability of our approach. Secondly, RankAdaptor demonstrates robustness to varying pruning rates, consistently outperforming AdaLoRA and LoRA across a wide range of pruning rates from 20% to 50%. This resilience indicates that our method maintains its effectiveness even under aggressive pruning scenarios. Thirdly, the superior performance of RankAdaptor is observed across diverse downstream tasks, including BoolQ, PIQA, HellaSwag, WinoGrande, ARC-easy, ARC-challenge, and OBQA, highlighting its task-agnostic nature. Finally, the method's efficacy is proven under different pruning strategies, namely LLM-Pruner and Shortened, further emphasizing its flexibility. In almost all scenarios, RankAdaptor achieves the highest average performance, often with significant margins, demonstrating its potential as a universal solution for recovering pruned language models across various configurations and applications.

We also use LLM-Pruner to prune LLaMA3-8B by 20%, and recovered it using LoRA and RankAdaptor respectively. The results are shown in Table 4. From the results, we can see that RankAdaptor once again proved its superiority, and it almost completely surpassed LoRA (only less than 2% lower on the ARC-e task).

A.2 Performance of Pruned LLM without Recovery

To demonstrate the critical importance of a recovery phase in achieving optimal performance for pruned models, we conducted extensive benchmark tests on LLaMA-7B models that had undergone only the pruning stage. The results, presented in Table 9, offer compelling evidence when compared with those in Table 2. This comparison reveals a stark contrast in performance between models with and without recovery. Models that have not undergone recovery exhibit significantly diminished performance across all evaluated tasks, highlighting the potential loss of crucial learned representations during the pruning process. In contrast, models that have been subjected to a recovery process, regardless of the specific fine-tuning method employed (be it AdaLoRA, LoRA, or our proposed RankAdaptor), demonstrate substantial performance improvements across nearly all tasks. Furthermore, while all recovery methods show improvements, the degree of enhancement varies, with our proposed RankAdaptor method consistently achieving superior results. These findings emphasize that the recovery phase should not be considered an optional step but rather an integral component of the model compression process.

A.3 Performance on challenging task - GSM8K & MMLU

Our goal is to explore the rank value of each layer that can achieve relative optimal performance in downstream tasks. We have conducted some experiments on GSM8K to confirm the versatility of RankAdaptor across task types. The results using LLM-Pruner and SlimGPT as pruning methods are shown in Table 7. From the results, we can see that although the pruned models with recovery all show extremely poor accuracy on GSM8K, RankAdaptor still performs better than the standard LoRA. We can see that although LLM-Pruner is not as good as SlimGPT in pruning operations, SlimGPT with LoRA is still not as good as LLM-Pruner with RankAdaptor. This is another manifestation of the strength of RankAdaptor.

Finally, to further test the ability of RankAdaptor on the challenge task, we use LLaMA2-7B to test it and LoRA on MMLU. At the same time, we also show the results claimed by the SlimGPT author in Table 8. From the results we can see that we get almost the same pattern as in GSM-8K: RankAdaptor is better than LoRA in any case. And although LLM-Pruner is not as good as SlimGPT in pruning operations, SlimGPT with LoRA is still not as good as LLM-Pruner with RankAdaptor.

B Specific Rank Configuration Allocation

In Table 2, we present the performance achieved by RankAdaptor on each task. Table 10 displays the rank configurations corresponding to the performance results of LLaMA-7B pruned by LLM-Pruner. We make these rank configurations pub-

Method	**BoolQ**	**PIQA**	**HellaS**	**WinoG**	**ARC-e**	**ARC-c**	**OBQA**	**Avg**
LoRA	70.25	78.21	72.37	70.18	75.68	43.49	41.30	64.50
RankAdaptor	73.77	79.98	74.56	72.33	73.82	44.31	42.80	65.93

Table 4: Performance comparison of LoRA and RankAdaptor on LLaMA3-8B.

Pruning Stage		Recover	BoolQ	PIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg
LLM-Pruner	Rate = 20%	AdaLoRA	55.43	75.22	65.31	60.89	64.96	35.61	38.10	56.50
		LoRA	57.77	**77.58**	67.16	63.14	67.30	37.71	40.40	58.72
		RankAdaptor	**61.19**	77.15	**67.32**	**63.85**	**67.68**	**38.05**	**41.20**	**59.49**
	Rate = 25%	AdaLoRA	48.24	72.89	62.25	59.18	61.58	33.57	38.30	53.72
		LoRA	50.34	75.24	64.10	61.33	**63.93**	35.67	40.60	55.89
		RankAdaptor	**58.50**	**76.17**	**64.23**	**61.96**	63.30	**36.01**	**42.00**	**57.45**
	Rate = 30%	AdaLoRA	56.46	71.87	58.85	58.27	56.66	31.69	36.50	52.90
		LoRA	**58.81**	74.37	60.70	**60.62**	59.01	33.79	38.80	55.16
		RankAdaptor	57.58	**75.57**	**61.63**	60.22	**60.94**	**34.81**	**39.00**	**55.68**
	Rate = 50%	AdaLoRA	57.21	64.52	41.83	49.66	46.05	24.35	31.70	45.05
		LoRA	**59.51**	66.87	43.18	**52.01**	48.40	26.45	34.00	47.20
		RankAdaptor	59.91	**67.46**	**43.50**	52.41	**48.70**	**27.65**	**35.80**	**47.92**
Shortened	Rate = 20%	AdaLoRA	69.52	72.95	66.83	63.91	65.82	37.23	38.64	59.27
		LoRA	71.82	74.32	68.45	**67.62**	67.35	38.50	39.80	61.12
		RankAdaptor	**74.31**	**76.92**	**70.73**	65.37	**69.51**	**40.12**	**41.34**	**62.61**
	Rate = 25%	AdaLoRA	67.63	71.84	64.15	62.53	62.41	36.72	37.32	57.51
		LoRA	69.85	73.21	65.72	64.08	63.90	**39.32**	38.50	59.23
		RankAdaptor	**72.41**	**75.62**	**67.93**	**66.21**	**65.84**	37.95	**39.91**	**60.84**
	Rate = 30%	AdaLoRA	60.92	70.54	59.21	61.15	56.73	33.42	35.61	53.94
		LoRA	62.75	71.93	60.68	62.70	**60.32**	34.50	36.80	55.67
		RankAdaptor	**65.13**	**74.21**	**62.84**	**64.92**	58.15	**35.93**	**38.12**	**57.04**
	Rate = 50%	AdaLoRA	58.63	66.42	45.21	52.34	46.92	27.51	33.42	47.21
		LoRA	60.37	**69.73**	46.50	53.76	48.25	28.72	34.60	48.85
		RankAdaptor	**62.54**	67.88	**48.12**	**55.62**	**49.84**	**29.91**	**35.83**	**49.96**

Table 5: Zero-shot performance of pruned Vicuna-7B with AdaLoRA, LoRA, and RankAdaptor recovery. 'Bold' indicates the best performance at each pruning rate. 'Avg' represents the average performance across all benchmarks. The results are reported in percentage (%).

Pruning Stage		Recover	BoolQ	PIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg
LLM-Pruner	Rate = 50%	AdaLoRA	59.63	69.03	51.61	51.24	50.76	28.25	35.70	49.46
		LoRA	61.93	71.38	**53.36**	53.59	53.11	29.95	38.00	51.62
		RankAdaptor	**62.05**	**71.71**	53.33	**54.22**	**53.20**	**30.89**	**39.40**	**52.11**
Shortened	Rate = 50%	AdaLoRA	71.53	70.92	54.32	52.83	50.94	31.62	39.21	53.05
		LoRA	73.75	**73.81**	55.96	54.41	52.35	32.81	40.60	54.81
		RankAdaptor	**75.92**	72.64	**57.83**	**56.23**	**54.12**	**33.96**	**41.92**	**56.09**

Table 6: Zero-shot performance of pruned LLaMA-13B with AdaLoRA, LoRA, and RankAdaptor recovery. 'Bold' indicates the best performance at each pruning rate. 'Avg' represents the average performance across all benchmarks. The results are reported in percentage (%).

licly available to foster reproducibility and enable further research by other scholars.

Pruning Method	Pruning Rate	Recover Method	GSM-8K-Acc
-	-	-	11.00
LLM-Pruner	20%	LoRA	3.75
LLM-Pruner	20%	RankAdaptor	6.18
SlimGPT	20%	-	4.20
SlimGPT	20%	LoRA	6.00
LLM-Pruner	50%	LoRA	1.85
LLM-Pruner	50%	RankAdaptor	3.10

Table 7: Performance comparison of different pruning and recovery methods on GSM-8K accuracy.

Pruning Method	Pruning Rate	Recover Method	MMLU-Acc
-	-	-	45.60
LLM-Pruner	20%	LoRA	33.83
LLM-Pruner	20%	RankAdaptor	38.75
SlimGPT	20%	LoRA	37.80
LLM-Pruner	50%	LoRA	19.25
LLM-Pruner	50%	RankAdaptor	24.31

Table 8: Performance comparison of different pruning and recovery methods on MMLU accuracy.

C Generation Performance Display.

The analysis of Generation Performance has been presented in Section 4.2 of the main section. Therefore, here, we focus solely on presenting the generation comparison results between the pruned LLaMA-7B recovered by LoRA and RankAdaptor, as illustrated in Figures 5 and 6.

D Further Details Supplement

D.1 Detailed Explanation of Equation (4)

Equation (4) is an expression of the goal of our entire work. Thank you for your suggestion that we can benefit from greater mathematical clarity. Let me explain it further:

Given:

1. A pruned model PL
2. A neural network with L layers
3. Rank configurations $r_i \in R_H$ for each layer i, where $i \in \{1, 2, ..., L\}$
4. Solution space $\mathcal{S} = \{\text{all possible } R_H \mid R_H = (r_1, r_2, ..., r_L)\}$

The problem can be formulated as:

$$R_H^* = \arg\max_{R_H \in \mathcal{S}} P(\text{recover}(PL, R_H))$$

where:

- recover(PL, R_H) represents the operation of recovering (fine-tuning) the pruned model PL using rank configuration R_H
- $P(\cdot)$ denotes the performance of the recovered model
- $R_H^* = (r_1^*, r_2^*, ..., r_L^*)$ represents the optimal rank configuration
- r_i^* represents the optimal rank value for layer i

Constraints:

1. $\forall i \in \{1, 2, ..., L\} : r_i \in \{2, 4, 6, 8, 10, 12, 14, 16\}$
2. $R_H \in \mathcal{S}$, where $\mathcal{S}$ is the solution space containing all possible rank configuration combinations

D.2 Approximating Performance for Optimal Rank Search

Due to the huge solution space, we cannot enumerate the rank value of each layer that is the global optimal in actual performance, so we try to get a relatively optimal solution by training a performance model that can directly predict the accuracy of downstream tasks using the rank value of each

	Pruning Stage	BoolQ	PIQA	HellaS	WinoG	ARC-e	ARC-c	OBQA	Avg
LLM-Pruner	Rate = 20%	56.94	75.73	66.83	60.06	60.94	36.43	39.80	56.68
	Rate = 25%	59.94	73.23	62.35	58.80	55.81	34.90	39.40	54.92
	Rate = 30%	58.96	71.22	58.10	58.88	52.19	32.34	38.40	52.87
	Rate = 50%	57.98	60.94	34.35	52.25	31.82	27.30	35.80	42.92
Shortened	Rate = 20%	62.65	77.24	68.83	61.26	62.76	37.52	40.99	58.75
	Rate = 25%	61.74	74.69	64.22	60.56	57.48	35.94	40.58	56.46
	Rate = 30%	60.73	74.64	59.84	59.64	53.75	33.31	39.55	54.49
	Rate = 50%	59.72	62.16	35.38	53.81	32.77	28.12	36.87	44.12

Table 9: Zero-shot performance of pruned LLaMA-7B without recovery. 'Avg' represents the average performance across all benchmarks. The results are reported in percentage (%).

LLaMA-7b with LoRA:

The universe is the entirety of space, time, matter, and energy that exists. There is also a scientific theory called the cosmological principle which asserts that the universe is homogeneous, that is, all the same, with all particles evenly distributed. But, as discussed in Chapter Five, there are many versions of cosmology, including theories that differ from the cosmological principle. These theories, for example, assume that the universe is expanding with a uniform distribution of matter, or that it started from a central source and is being pulled outward by the expansion. In both cases, the cosmological principle would...

LLaMA-7B with RankAdaptor:

The universe is the entirety of space, time, matter, and energy that exists. It is the only thing that exists, and it is all around us. The universe is expanding, and it is expanding at a rate that is faster than the speed of light. This means that the universe is growing and changing. It is also changing all the time, and it is constantly creating new things.The universe is made up of many different things, including galaxies, stars, planets, and other objects. These objects are all made of different things, such as atoms, molecules, and elements.

Figure 5: Article continuation task comparison in LLaMA-7B

Vicuna-7B with LoRA:

10 steps to build an ios app: 1. Create a new project in Xcode. 2. Open the project and create a new folder named "ios" in the project folder. 3. Create a new file named "app.json" in the ios folder. 4. Add the following code to the file:```
{"name": "MyApp","version": "1.0","author": "Me",
"app_name": "...

Vicuna-7B with RankAdaptor:

10 steps to build an ios app: 1. Create a new project in Xcode. 2. Open the project and create a new folder named "Resources" in the project folder. 3. Inside the "Resources" folder, create a folder named "AppIcon" and add an image file named "Icon.png" to it. 4. Open the "AppIcon.png" file in Xcode and drag the image to the "Resources" folder. 5. Open the "AppIcon.png"...

Figure 6: Step listing task comparison in Vicuna-7B

layer. The online incremental learning of the performance model is the process of gradually reducing the potential gap between the predicted performance and the actual performance. When the performance model converges in the iteration phase, we believe that it has the ability to approximate the actual performance of a rank value set. Using this converged performance model, we expect to find a relatively optimal rank value set, rather than the global optimal.

D.3 Why Different Rank Sizes can Meet the Recovery Needs of Different Layers?

Based on our research, we found that this effectiveness mainly stems from the following key factors:

- First, the functional characteristics of different layers determine their differentiated needs for rank size. For example, we observed that the

Pruning Rate	Tasks	Layers' Rank Values (1~32/40)
20%	BoolQ	8, 8, 8, 8, 4, 12, 12, 12, 12, 10, 8, 10, 6, 2, 8, 6, 8, 2, 8, 2, 8, 10, 12, 10, 6, 4, 4, 4, 2, 12, 8, 8
	PIQA	8, 8, 8, 8, 4, 12, 12, 12, 12, 10, 8, 10, 6, 2, 8, 6, 8, 2, 8, 2, 8, 10, 12, 10, 6, 4, 4, 4, 2, 12, 8, 8
	Hella	8, 8, 8, 8, 2, 2, 4, 10, 10, 6, 10, 10, 10, 6, 6, 2, 2, 10, 2, 4, 2, 10, 10, 10, 4, 10, 10, 6, 6, 2, 8, 8
	Wino	8, 8, 8, 8, 8, 10, 4, 10, 4, 6, 6, 2, 10, 8, 12, 12, 10, 12, 12, 10, 6, 6, 8, 8, 10, 6, 6, 12, 2, 8, 8, 8
	ARC-e	8, 8, 8, 8, 4, 12, 12, 12, 12, 10, 8, 10, 6, 2, 8, 6, 8, 2, 8, 2, 8, 10, 12, 10, 6, 4, 4, 4, 2, 12, 8, 8
	ARC-c	8, 8, 8, 8, 4, 12, 12, 12, 12, 10, 8, 10, 6, 2, 8, 6, 8, 2, 8, 2, 8, 10, 12, 10, 6, 4, 4, 4, 2, 12, 8, 8
	OBQA	8, 8, 8, 8, 4, 12, 12, 12, 12, 10, 8, 10, 6, 2, 8, 6, 8, 2, 8, 2, 8, 10, 12, 10, 6, 4, 4, 4, 2, 12, 8, 8
25%	BoolQ	8, 8, 8, 8, 12, 2, 8, 2, 8, 12, 4, 2, 10, 12, 10, 4, 2, 2, 12, 8, 10, 2, 12, 12, 8, 4, 4, 2, 2, 12, 8, 8
	PIQA	8, 8, 8, 8, 4, 2, 2, 10, 10, 2, 10, 10, 10, 2, 2, 2, 4, 10, 4, 6, 10, 2, 2, 6, 10, 2, 2, 10, 10, 2, 8, 8
	Hella	8, 8, 8, 8, 4, 10, 12, 12, 6, 10, 6, 6, 8, 2, 2, 12, 2, 12, 12, 6, 4, 10, 6, 2, 2, 8, 4, 2, 2, 8, 8, 8
	Wino	8, 8, 8, 8, 8, 4, 12, 8, 2, 2, 12, 2, 10, 12, 2, 12, 12, 10, 8, 12, 4, 6, 6, 4, 10, 4, 2, 10, 10, 12, 8, 8
	ARC-e	8, 8, 8, 8, 2, 12, 2, 6, 12, 6, 12, 10, 6, 4, 8, 8, 12, 2, 2, 6, 8, 4, 12, 12, 2, 4, 2, 6, 6, 2, 8, 8
	ARC-c	8, 8, 8, 8, 2, 12, 2, 6, 12, 6, 12, 10, 6, 4, 8, 8, 12, 2, 2, 6, 8, 4, 12, 12, 2, 4, 2, 6, 6, 2, 8, 8
	OBQA	8, 8, 8, 8, 4, 12, 12, 12, 12, 1, 8, 10, 6, 2, 8, 6, 8, 2, 8, 2, 8, 10, 12, 12, 10, 4, 4, 6, 2, 12, 8, 8
30%	BoolQ	8, 8, 8, 8, 12, 2, 8, 2, 8, 12, 4, 2, 10, 12, 10, 4, 2, 2, 12, 8, 10, 2, 12, 12, 8, 4, 4, 2, 2, 12, 8, 8
	PIQA	8, 8, 8, 8, 12, 6, 10, 4, 2, 4, 2, 4, 12, 8, 2, 2, 2, 12, 12, 12, 12, 2, 12, 4, 4, 2, 10, 2, 2, 8, 8, 8
	Hella	8, 8, 8, 8, 12, 6, 8, 4, 2, 12, 10, 4, 4, 2, 6, 4, 6, 10, 4, 2, 8, 6, 12, 10, 4, 6, 6, 6, 8, 2, 8, 8
	Wino	8, 8, 8, 8, 12, 6, 8, 4, 2, 12, 10, 4, 4, 2, 6, 4, 6, 10, 4, 2, 8, 6, 12, 10, 4, 6, 6, 6, 8, 2, 8, 8
	ARC-e	8, 8, 8, 8, 12, 6, 8, 4, 2, 12, 10, 4, 4, 2, 6, 4, 6, 10, 4, 2, 8, 6, 12, 10, 4, 6, 6, 6, 8, 2, 8, 8
	ARC-c	8, 8, 8, 8, 4, 10, 12, 12, 6, 10, 6, 6, 8, 2, 2, 12, 2, 12, 12, 6, 4, 10, 6, 2, 2, 8, 4, 2, 2, 8, 8, 8
	OBQA	8, 8, 8, 8, 12, 6, 8, 4, 2, 12, 10, 4, 4, 2, 6, 4, 6, 10, 4, 2, 8, 6, 12, 10, 4, 6, 6, 6, 8, 2, 8, 8
50%	BoolQ	8, 8, 8, 8, 12, 4, 6, 2, 2, 10, 4, 6, 12, 12, 2, 2, 12, 6, 4, 2, 6, 2, 4, 2, 6, 10, 10, 4, 2, 2, 8, 8
	PIQA	8, 8, 8, 8, 12, 4, 6, 2, 2, 10, 4, 6, 12, 12, 2, 2, 12, 6, 4, 2, 6, 2, 4, 2, 6, 10, 10, 4, 2, 2, 8, 8
	Hella	8, 8, 8, 8, 12, 4, 6, 2, 2, 10, 4, 6, 12, 12, 2, 2, 12, 6, 4, 2, 6, 2, 4, 2, 6, 10, 10, 4, 2, 2, 8, 8
	Wino	8, 8, 8, 8, 12, 4, 6, 2, 2, 10, 4, 6, 12, 12, 2, 2, 12, 6, 4, 2, 6, 2, 4, 2, 6, 10, 10, 4, 2, 2, 8, 8
	ARC-e	8, 8, 8, 8, 12, 4, 6, 2, 2, 10, 4, 6, 12, 12, 2, 2, 12, 6, 4, 2, 6, 2, 4, 2, 6, 10, 10, 4, 2, 2, 8, 8
	ARC-c	8, 8, 8, 8, 12, 4, 6, 2, 2, 10, 4, 6, 12, 12, 2, 2, 12, 6, 4, 2, 6, 2, 4, 2, 6, 10, 10, 4, 2, 2, 8, 8
	OBQA	8, 8, 8, 8, 12, 4, 6, 2, 2, 10, 4, 6, 12, 12, 2, 2, 12, 6, 4, 2, 6, 2, 4, 2, 6, 10, 10, 4, 2, 2, 8, 8

Table 10: Specific value of the rank configuration explored by RankAdaptor in LLaMA-7B pruned by LLM-Pruner

bottom layer mainly processes basic language features (such as grammar, lexical), so it needs a larger rank to maintain its expressiveness; while the high-level layer is mainly responsible for task-related reasoning and is relatively less sensitive to rank size. This functional difference is more obvious after pruning.

- Second, structured pruning based on importance will lead to different degrees of information loss in different layers. A larger rank can introduce more learnable parameters to compensate for the large loss of some layers caused by pruning. At the same time, a relatively complete layer can be retained with a smaller rank to achieve a good recovery effect. This non-uniformity directly affects the selection of the optimal rank of each layer.

- Finally, through comparative experiments, we confirmed the necessity of this differentiated rank allocation. When we use larger ranks for key layers, the overall recovery of the model is significantly improved; on the contrary, assigning too large ranks to non-key layers not only fails to improve performance, but also increases unnecessary computational overhead. These findings directly support our layer-specific rank allocation strategy.