A Sliding Layer Merging Method for Efficient Depth-Wise Pruning in LLMs

Anonymous ACL submission

Abstract

Compared to width-wise pruning, depth-wise pruning can significantly accelerate inference in resource-constrained scenarios. However, treating the entire Transformer layer as the minimum pruning unit may degrade model performance by indiscriminately discarding the entire information of the layer. This paper reveals the "Patch-like" feature relationship between layers in large language models by analyzing the correlation of the outputs of different layers in the reproducing kernel Hilbert space. Building on this observation, we propose a sliding layer merging method that dynamically selects and fuses consecutive layers from top to bottom according to a predefined similarity threshold, thereby simplifying the model structure while maintaining its performance. Extensive experiments on LLMs with various architectures and different parameter scales show that our method outperforms existing pruning techniques in both zero-shot inference performance and retraining recovery quality after pruning. In particular, in the experiment with 35% pruning on the Vicuna-7B model, our method achieved a 1.654% improvement in average performance on zero-shot tasks compared to the existing method. Moreover, we further reveal the potential of combining depth pruning with width pruning to enhance the pruning effect.

1 Introduction

Large language models (LLMs) have attracted widespread attention in deep learning, owing to their exceptional performance and broad application potential (Touvron et al., 2023; Chowdhery et al., 2023; Wang et al., 2025). However, in pursuit of better performance, the size of LLMs has grown increasingly larger, posing significant technical and resource challenges for practical deployment and application. Given that not all parameters in the large parameter space of the model contribute equally to the output, pruning methods are effective for reducing redundancy and addressing model size challenges (Ma et al., 2023; Sun et al., 2024a; Kim et al., 2024; Men et al., 2024; Song et al., 2024). The width-wise approach reduces the network width by pruning coupled structures, such as attention heads and their associated weight connections, while preserving the number of layers (Ma et al., 2023; An et al., 2024; Sun et al., 2024b). In contrast, the depth-wise approach reduces the network depth by completely removing certain layers (Kim et al., 2024; Men et al., 2024; Song et al., 2024). Although the depth-wise pruning method can significantly accelerate inference in resource-constrained scenarios that require running LLMs with limited batch sizes, it remains underexplored in terms of analyzing the correlations between Transformer layers at different depths. Moreover, arbitrary removing specific layers may degrade the performance of the pruned model.

We first focus on the question: what is the correlation between the features extracted by different layers in LLMs? To capture the subtle distinctions between features in high-dimensional space, we assess the correlations between the outputs of different layers of the model within a reproducing kernel Hilbert space and normalize the evaluation metric to ensure isotropic scaling invariance, an approach inspired by (Raghu et al., 2021). We conduct observational experiments across multiple models and datasets, and the results reveal a high degree of similarity in the representations of certain consecutive Transformer layers in large language models, exhibiting a clear "patch-like" structure. This observation provides new insights for model compression, suggesting that when feature representations across layers are

highly similar, parameter sharing or layer merging can be considered to reduce both computational load and memory usage.

Building on the aforementioned observation, we propose a novel compression method - Sliding Layer Merging (SLM). This method dynamically selects the base layer and its adjacent layers with similar representations for merging, starting from the deepest layers and moving progressively towards the shallower layers, utilizing a sliding window mechanism. For the layers to be merged, we calculate the parameter differences between them and the base layer, incorporating these differences into the base layer's parameters, thereby merging multiple layers into one. The sliding window mechanism selects adjacent layers of the base layer for merging by comparing the similarity between the outputs of the pruned model and the original model. When the similarity exceeds a predefined threshold, the window expands towards the shallower layers; when the similarity falls below the threshold, the layers to be merged are combined, and the window slides to update the base layer's index. Once the iterative merging is completed, a fast recovery phase is performed, utilizing limited data to post-train the pruned model.

Extensive experiments, encompassing zero-shot performance comparisons with baseline methods as well as evaluations of inference speed and throughput, demonstrate that the proposed Sliding Layer Merging method consistently outperforms existing approaches in both model accuracy and computational efficiency. Moreover, we introduce an innovative fusion of width-wise and depth-wise pruning techniques, which further enhances the model compress performance.

The contributions of this study are summarized as:

- We analyze the inter-layer correlations in LLMs within a reproducing kernel Hilbert space, observing an interesting "Patch-Like" correlation distribution, which provides valuable insights for the design of model compression strategies.

- We propose the Sliding Layer Merging method, which dynamically merges layers with strong representational similarity in LLMs. This method can be seamlessly applied to various LLM architectures.

- We conduct extensive experiments across multiple LLM architectures of varying scales, demonstrating that our method outperforms existing depth-wise pruning methods in zero-shot performance, both in retraining-free scenarios and in scenarios where pruning is followed by retraining to restore quality. Specifically, when pruning the Vicuna-7B model by 35%, our method achieved superior average performance across multiple datasets, outperforming method LLM-Pruner by 1.654%.

2 Related Work

Large language models' multi-layer Transformer architecture often contains substantial redundancy, motivating research on width-wise and depth-wise pruning to reduce this redundancy and improve model efficiency.

Width-wise pruning reduces the network width by pruning coupled structures. For example, Voita et al. (2019) and Michel et al. (2019) introduced pruning and attention head sharing techniques to reduce redundant attention heads, thereby decreasing both computational complexity and parameter requirements. Nova et al. (2023) and Santacroce et al. (2023) optimized the feedforward network by reducing the dimension of the FFN hidden layer, thereby reducing the memory footprint and computational complexity. More complex hybrid optimization methods have also been explored (Lagunas et al., 2021; Kwon et al., 2022; Kurtić et al., 2024).

Depth-wise pruning directly removes the entire least important layer and can significantly accelerate inference. Shortened-LLM (Kim et al., 2024) selected Taylor+ and PPL indicators as the importance measure of the Transformer layer, and deleted the unimportant Transformer layer to reduce the consumption of computing resources and improve the inference speed. The layer-skipping strategy (Schuster et al., 2022; Del Corro et al., 2023; Raposo et al., 2024) further reduces computational burden and boosts inference efficiency by dynamically selecting which layers to skip during execution. Additionally, Song et al. (2024) and Tang et al. (2024) investigated

depth pruning methods, which reduce model depth by eliminating redundant layers, optimizing both computational overhead and model performance while retaining essential layers.

3 Motivation

3.1 CKA vector similarity

Center Kernel Alignment (CKA) is a metric used to compare the internal representations of neural networks. Its main advantages are its invariance to orthogonal transformations (e.g. changes in neuron arrangement) and its robustness to isotropic scaling achieved through a normalization term (Raghu et al., 2021). These properties make CKA particularly suitable for studying the underlying relationships between different Transformer layers within large language models. Our calculation procedure for CKA is outlined as follows:

- Step1: Calculate the Gram matrix of two representation matrices to measure the similarity of representations.

$$K = XX^T, L = YY^T, K, L \in R^{n \times n}, \quad (1)$$

where $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^{n \times q}$ denote the outputs of the two Transformer layers for which CKA is to be computed, n is the number of samples, and p and q represent the dimensionalities of X and Y, respectively.

- Step2: Centralize the Gram matrix to eliminate the potential impact of sample distribution deviation.

$$\tilde{K} = HKH, \tilde{L} = HLH, \quad (2)$$

where $H = I_n - 1/n 1_n 1_n^T$ is the centralization matrix, I_n is the $n \times n$ identity matrix, and 1_n is an all-ones vector of length n.

- Step3: Calculate the normalized alignment between the central Gram matrices K and L to get CKA.

$$\text{CKA}(K, L) = \frac{\langle \tilde{K}, \tilde{L} \rangle_F}{\|\tilde{K}\|_F \|\tilde{L}\|_F}, \quad (3)$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product and $\|\cdot\|_F$ represents the Frobenius norm.

The final CKA value is between 0 and 1. The closer the value is to 1, the more similar the two representation matrices are.

3.2 Representation Structure between LLM Transformer Layers

We begin our investigation by leveraging the CKA metric to examine the internal representation structures of various models, with a particular focus on two key questions: What are the internal relationships between different Transformer layers in large language models (LLMs)? And is there redundancy among these layers? To explore these questions, we present inter-layer CKA similarity heatmaps for several LLMs, including LLaMA2-7B, Vicuna-7B-v1.3, Vicuna-13B-v1.3, and Meta-LLaMA3-8B, as shown in Fig.1. Analysis of these heatmaps revealed several key findings:

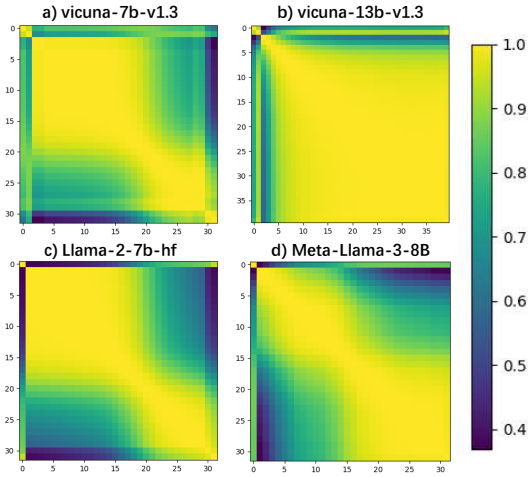

Figure 1: CKA (Center Kernel Alignment) metric between pairs of Transformer layers in LLMs.

- **Redundancy of intermediate layers:** When performing CKA measurements across different model architectures, we observed strong inter-layer correlations between adjacent intermediate layers, which appear as bright patches on the heatmaps, forming "patch-like" structures. This result implies that these layers have high functional redundancy and provide space for compression.

- **Inter-layer correlation differences:** We found that the first two and final layers of the model exhibited lower CKA correlations than other layers. This suggests that the representations of some layers may be relatively independent, weakly functionally related to other layers, and may not be suitable for large-scale compression. In

subsequent studies, we "protect" these layers by not compressing them to avoid unnecessary damage to model performance.

4 Method

4.1 Overview

Based on the aformentioned CKA analysis, we propose a Sliding Layer Merging method to prune large language models (LLMs), which dynamically compresses Transformer layers from top to bottom while preserving task-critical representations. In the following, we first describe the algorithm workflow (Sec.4.2), followed by detailed explanations of its two core components: parameter merging (Sec.4.3) and adaptive similarity thresholding (Sec.4.4).

4.2 Sliding layer merging algorithm

As shown in Algorithm 1, our method performs iterative merging of adjacent layers within a sliding window via three steps: (1) model initialization, (2) dynamic window updating, and (3) termination condition evaluation.

Algorithm 1 Iterative Layer Compression Algorithm

Input: Original model M
Parameters: Layer range $[L, H]$; Similarity threshold T; Few-shot calibration samples D
Output: Pruned model M^*

$M^* \leftarrow M$
$h\_lay \leftarrow H$
$l\_lay \leftarrow h\_lay - 1$
while $l\_lay \geq L$ **do**
  $M_{tmp} \leftarrow \text{Merge}(M^*, h\_lay, l\_lay)$
  $s \leftarrow \text{Cal\_Sim}(M, M_{tmp})$
  if $s > T$ **then**
    $l\_lay \leftarrow l\_lay - 1$
  else
    $M^* \leftarrow M_{tmp}$
    $h\_lay \leftarrow l\_lay$
  end if
end while
return M^*

Model initialization. We initialize the target compressed model M^* as a copy of the original model M, with a predefined compression range $[L, H]$, where H and L denote the highest and lowest layers to be compressed, respectively. To preserve critical functionality, we exclude the top layers from compression based on CKA analysis, which reveals their distinct role due to lower inter-layer correlation. This protection mechanism prevents excessive degradation while enabling aggressive compression of redundant lower layers.

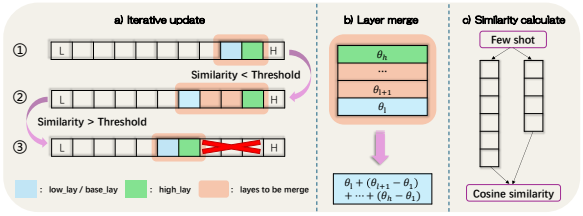

Figure 2: The framework of our sliding layer merging method.

Dynamic window updating. Our method employs a dynamic sliding window (initialized with layer H as upper bound and $H-1$ as lower bound) to determine merging ranges. Each iteration proceeds as follows: First, we merge layers within the window to create M_{tmp} and compute its cosine similarity with the original model M using last hidden states from few-shot calibration data. When the similarity exceeds threshold T (indicating minimal performance impact), we expand the merging range by moving the lower bound down one layer. Conversely, when the similarity falls below threshold T (suggesting significant performance impact), we stop window expanding, update the compressed model $M^* = M_{tmp}$, and reset the upper bound to the current lower bound before proceeding to the next merging round (see Fig.2 (a)). This adaptive process systematically balances compression efficiency with performance preservation through representation-aware thresholding, terminating when further merging would violate the similarity constraint to maintain model integrity.

Termination condition evaluation. The dynamic window updating process continues until the lowest level L is processed. Ultimately, the pruned model M^* output by the algorithm reduces redundant computing and storage requirements by retaining the merged representation of key layers.

4.3 Parameter merging strategy

We adopt a layer merging strategy based on inter-layer differences (as shown in Fig.2 (b)), which progressively integrates redundant in-

formation while preserving core functionality. Specifically, given layers $\{L_i, L_{i+1}, ..., L_j\}$ with parameters $\{\theta_i, \theta_{i+1}, ..., \theta_j\}$ within the sliding window, our Merge function computes the merged parameters θ^* as:

$$\begin{aligned}\theta_i^* &= \theta_i + (\theta_{i+1} - \theta_i) + \cdots + (\theta_j - \theta_i) \\ &= \theta_i + \sum_{k=1}^{j-i} (\theta_{i+k} - \theta_i), \end{aligned} \tag{4}$$

This formulation offers two key advantages: (1) the base layer θ_i maintains fundamental model capabilities, while (2) the difference terms $(\theta_{i+k} - \theta_i)$ incorporate complementary features from other layers. The strategy effectively captures inter-layer correlations while remaining adaptable to various model compression requirements.

4.4 Adaptive similarity thresholding

The dynamic window updating process of our method relies on the iteratively evaluation by comparing the cosine similarity of final states between the original model M and compressed model M_{tmp} against predetermined thresholds (see Fig.2 (c)). Through systematic ablation (Section 5.6), we observe that lower thresholds enable more aggressive compression but degrade performance, while higher thresholds better preserve model behavior at the cost of reduced compression efficiency. To ensure optimal performance at each target compression ratio, we select the highest achievable threshold for each target ratio, ensuring maximal performance preservation under the given compression constraints.

4.5 Performance Recovery with Low-rank Approximation

We use the low-rank approximation technique, LoRA(Hu et al., 2021), to fine-tune the pruned model and recover its performance. This is a common practice in many pruning methods (Ma et al., 2023; Kim et al., 2024), and we provide a brief introduction to ensure the self-contained aspect of our work in Appendix A.3.

5 Experiments

5.1 Experimental setup

Foundation LLMs. We conducte experiments on existing popular open-source language models, including LLaMA2-{7B, 13B} (Touvron et al., 2023), LLaMA3-{8B} and Vicuna-{7B, 13B}-v1.3 (Chiang et al., 2023).

Baselines. The proposed method is compared with several previous works, categorized by their pruning strategy. For width pruning, we compare with LLM-Pruner (Ma et al., 2023), FLAP (An et al., 2024), and Wanda-sp (An et al., 2024), a structured variant of Wanda (Sun et al., 2024b). For depth pruning, we examine SLEB (Song et al., 2024) and Shortened-LLM (Kim et al., 2024). Following the experimental setup of the existing baseline method, Shortened-LLM, we assess all methods under two target pruning levels: 20% and 35%. If the product of the total number of transformer blocks and target sparsity is not an integer, we round up to determine the number of blocks to remove.

Benchmarks. Following Touvron et al. (2023), we measure model performance on seven commonsense reasoning datasets (i.e., BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARCeasy (Clark et al., 2018), ARC-challenge (Clark et al., 2018), and OpenbookQA (Mihaylov et al., 2018)) using the lm-evaluation-harness package (Gao et al., 2024).

Implementation Details. We implement our method in PyTorch (Paszke et al., 2019) using the HuggingFace Transformers library (Wolf et al., 2020). Following Ma et al. (2023), we randomly select 10 samples from BookCorpus (Zhu, 2015) to calculate the model similarity in the iterative pruning process. We also use this calibration dataset for baselines to ensure a fair comparison. In LoRA retraining, we use 50K samples of refined Alpaca (Taori et al., 2023) for instruction tuning. All experiments covered in this article were performed on an NVIDIA A100 GPU with 80GB memory.

5.2 Zero-shot Tasks

Tab.1 shows the zero-shot performance of different pruning methods on the LLaMA2-7B model, Vicuna-7B-v1.3 model and LLaMA3-8B model. Our method consistently outperforms existing techniques in both width and depth pruning. Specifically, under the 20% pruning rate of the LLaMA2-7B model, our method achieves a 2.676% higher accuracy than the best-performing LLM-Pruner method; under

			BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLaMA2-7B(Original)			77.706	78.074	76.021	69.140	76.305	46.331	44.200	66.825
20%Pruned	width	Wanda-sp	62.600	76.440	70.660	63.770	69.610	42.150	40.000	60.747
		FLAP	72.050	73.390	64.690	64.720	62.250	32.510	36.800	58.059
		LLM-Pruner	63.731	77.476	67.128	61.878	65.783	38.481	40.400	59.268
	depth	SLEB	62.875	73.939	63.951	59.747	63.468	35.154	38.000	56.733
		Shortened-LLM	61.560	76.061	67.994	58.800	68.813	37.884	38.000	58.445
		Ours	69.450	73.667	70.484	67.088	69.108	41.212	42.600	**61.944**
35%Pruned	width	Wanda-sp	59.790	68.820	53.140	54.060	52.270	31.570	32.800	50.350
		FLAP	66.970	67.850	52.100	61.480	49.490	28.070	32.400	51.194
		LLM-Pruner	45.260	74.760	60.290	59.350	57.280	32.420	37.200	**52.366**
	depth	SLEB	54.801	67.410	46.545	53.197	48.527	29.010	33.000	47.499
		Shortened-LLM	62.171	71.926	54.800	51.776	59.259	30.887	35.400	52.317
		Ours	63.119	65.452	56.503	58.879	52.020	31.911	35.200	51.869
Vicuna-7B-v1.3(Original)			78.104	77.312	73.939	69.376	74.327	44.454	43.800	65.902
20%Pruned	width	Wanda-sp	63.270	73.780	68.620	63.930	67.210	38.820	37.200	58.976
		FLAP	73.520	74.810	68.760	66.460	69.110	38.990	40.000	61.664
		LLM-Pruner	67.645	76.115	66.660	63.931	65.446	36.604	40.400	59.543
	depth	SLEB	46.758	51.850	26.170	51.223	25.505	28.840	24.800	36.449
		Shortened-LLM	72.355	74.701	67.576	64.562	70.034	38.225	38.600	60.865
		Ours	77.431	74.755	69.040	68.745	69.318	38.908	40.200	**62.628**
35%Pruned	width	Wanda-sp	50.760	60.450	43.060	55.960	43.520	26.190	28.000	43.991
		FLAP	57.860	69.640	59.120	63.300	57.830	35.670	36.000	54.203
		LLM-Pruner	63.976	73.069	59.560	58.564	56.524	32.679	37.800	54.596
	depth	SLEB	37.829	53.264	25.921	49.961	25.926	29.096	25.800	35.399
		Shortened-LLM	64.281	70.783	56.722	57.380	59.596	31.485	34.000	53.464
		Ours	69.235	70.294	60.705	62.273	60.227	33.618	37.400	**56.250**
LLaMA3-8B(Original)			81.101	79.489	79.167	73.402	80.093	53.242	44.800	70.185
20%Pruned	width	FLAP	37.830	52.180	26.360	49.960	26.810	24.830	26.000	34.853
		LLM-Pruner	74.037	79.489	74.268	70.324	74.285	46.587	42.600	65.941
	depth	SLEB	62.171	73.286	64.748	63.062	64.562	37.713	37.000	57.506
		Shortened-LLM	66.208	78.074	72.695	62.747	75.295	44.795	43.400	63.316
		Ours	76.789	77.639	73.770	71.744	76.599	50.939	41.200	**66.954**
35%Pruned	width	FLAP	37.830	52.340	26.050	47.990	24.790	24.830	25.200	34.147
		LLM-Pruner	64.465	74.048	61.800	59.353	64.646	34.386	37.200	56.557
	depth	SLEB	59.755	64.635	45.061	51.539	47.306	27.133	27.600	46.147
		Shortened-LLM	63.180	72.851	62.985	58.090	66.877	37.116	37.000	56.871
		Ours	67.554	73.830	61.472	62.747	64.352	36.007	37.600	**57.652**

Table 1: Performance comparison of pruning methods across multiple baselines at 20% and 35% pruning rates. We compare our method with width pruning approaches (Wanda-sp, FLAP, LLM-Pruner) and depth pruning approaches (SLEB, Shortened-LLM) on LLaMA2-7B, Vicuna-7B, and LLaMA3-8B models. Note that Wanda-sp does not produce results for the LLaMA3-8B model due to incompatibility.

the 35% pruning rate of the Vicuna-7B model, the average accuracy of our method is 1.654% higher than that of existing methods. These results show that our proposed method can effectively reduce model size and complexity while more fully maintaining model performance.

To verify the broad applicability of our method, we also provide relevant experimental results on larger models (such as LLaMA2-13B and Vicuna-13B-v1.3) in Appendix B.4.

5.3 Latency and Throughput

We follow Sheng et al. (2023) to evaluate the LLM inference speedup achieved by our pruning methods. Given a batch size M and an output sequence length L, the latency T represents the time required to handle the given prompts and produce ML output tokens. The throughput is computed as ML/T . We report the average results from 20 runs after the initial 10 warm-up batches. Tab.2 present throughput and latency results for LLaMA-2-7B.

			Latency(sec)	Throughout(tokens/s)	GPU_Memory	nparam
LLaMA-2-7B(Original)			2.729	46.905	13020.25	6.7B
20%Pruned	width	Wanda-sp	4.628	27.663	10676	5.5B
		FLAP	4.045	31.656	10707.25	5.4B
		LLM-Pruner	5.655	22.635	10951.5	5.5B
	depth	SLEB	2.529	50.622	10682.45	5.5B
		Shortened-LLM	2.585	49.542	10682.45	5.5B
		Ours	2.339	54.758	10682.45	5.5B
35%Pruned	width	Wanda-sp	4.619	27.726	8901	4.5B
		FLAP	4.127	31.051	8855.95	4.5B
		LLM-Pruner	5.630	22.736	9043.9	4.5B
	depth	SLEB	1.938	66.048	8725.9	4.5B
		Shortened-LLM	2.084	61.433	8725.85	4.5B
		Ours	1.889	67.770	8725.9	4.5B

Table 2: Inference efficiency of pruned models. (Measured with 12 input tokens, 128 output tokens, and a batch size of 1.)

Experimental results show that depth prun-

ing generally outperforms width pruning in reasoning efficiency. Specifically, at pruning ratio of 20% and 35%, depth pruning methods (SLEB, Shortened-LLM and Ours) outperform width pruning methods (Wanda-sp, FLAP, and LLM-Pruner) in both latency and throughout metrics. This suggests that reducing the depth of the model can more effectively enhance inference speed and throughout. At the same time, depth pruning methods maintain relatively stable GPU memory usage while ensuring efficient inference. Therefore, from the perspective of inference efficiency, depth pruning is a more effective pruning strategy.

5.4 Integration of width-wise pruning and depth-wise pruning

We observed in Tab.1 that in the 35% pruning scenario of the LLaMA2-7B model, the width-wise LLM-Pruner method performs slightly better than our depth-wise method. This may be due to the fact that under specific pruning ratios and model structures, width pruning can also exhibit advantages, and depth pruning does not necessarily always outperform it. This naturally raises the question: Can the intergration of width-wise pruning and depth-wise pruning leverage the strengths of both methods to further improve pruning results?

Specifically, using LLaMA2-7B model as an example, we divide the pruning process into depth-wise pruning and width-wise pruning. In the first stage, we compress the model based on Transformer layers by our proposed method. In the second stage, we use the LLM-Pruner method to remove non-essential coupled structures from the model obtained in the first stage. As a result, the pruned model achieves a pruning rate of 35% relative to the original model. For the sake of pruning convenience, we selected the following depth pruning rates: 0% (LLM-Pruner), 18%, 36%, 53%, 71%, 91%, and 100% (ours). The performance evaluation results are shown in Fig.3.

The results show that the combined depth-wise and width-wise pruning strategy achieves better performance at the same pruning rate compared to using either method alone. Specifically, models with depth pruning rates of 18%, 36%, 53%, 71% and 91% consistently surpass the models with depth pruning rates of 0% and 100%. Notably, the models with depth pruning

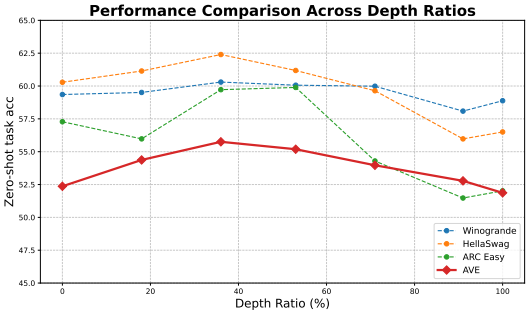

Figure 3: Performance of the integrated method on LLaMA2-7B. The horizontal axis shows the depth pruning ratio, and the vertical axis indicates zero-shot task performance. Results for Winogrande, HellaSwag, and ARC Easy are plotted as dotted lines, with the solid line showing the average performance across seven tasks.

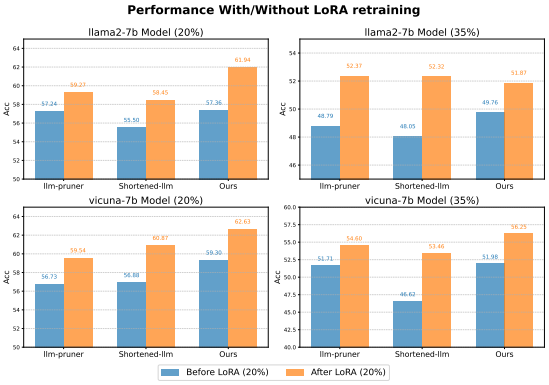

Figure 4: Performance with/without LoRA retraining. The blue column represents the model performance before lora fine-tuning, and the orange column represents performance after lora fine-tuning.

rates of 36% and 53% rank first and second in terms of performance, respectively. This shows that the integrated methods can combine the advantages of depth-wise pruning and width-wise pruning methods to achieve better model performance than when using depth-wise pruning or width-wise pruning alone, while mitigating the throughput and inference speed issues associated with width-wise pruning methods. We also performed the same experiments on the Vicuna-13B-v1.3 model, and the results can be found in Appendix B.2.

5.5 The impact of LoRA retraining

We evaluate the impact of LoRA retraining on three pruning methods requiring fine-tuning (LLM-Pruner, Shortened-LLM, and ours). Fig.4 shows the performance of the

LLaMA2-7B model and Vicuna-7B model at 20% and 35% pruning rate.

The results show that before LoRA fine-tuning, the performance of our method was significantly better than that of LLM-Pruner and Shortened-LLM, with performance indicators at the best level whether under both 20% and 35% pruning ratio. After further employing LoRA fine-tuning, the performance of our method is significantly improved and still maintains the lead in most cases. Specifically, under the 20% pruning ratio of the LLaMA2-7B model, the performance improvement of the our method is particularly significant, from 57.360% to 61.944%, an increase of 4.584%, far exceeding LLM-Pruner (by 2.031%) and Shortened-LLM (by 2.946%).

5.6 The impact of Layer merging methods

We evaluate three layer merging methods — direct layer deletion ("Delete"), replacing multiple layers' parameters with their average ("Average"), and our method ("Ours") — on the LLaMA2-7B model, analyzing their impact on pruned model performance using the HellaSwag dataset (without LoRA fine-tuning).

We first analyzed the layer count after different merging methods at varying similarity thresholds in Fig.5(a). As the threshold decreases, the number of layers reduces, indicating fewer redundant parameters. At the same threshold, our layer merging method achieves higher compression. For example, at a threshold of 0.75, our method retains only 23 layers, significantly fewer than "Delete" method (26 layers) and "Average" method (29 layers), resulting in a more efficient model compression.

In Fig.5(b), we compared the zero-shot accuracy at various compression levels after different merging methods. The results show that, while "Average" degrades significantly under aggressive pruning, our method consistently outperforms both baselines, with the performance difference gradually expands as the number of compression layers increases.

5.7 The impact of Calibration data

Tab.3 investigates the impact of calibration data selection on LLaMA2-7B model. The results show that varying the choice and size of calibration data has minimal impact on the performance of compressed models. This demonstrates our method's robustness to calibration data selection, with negligible differences across configurations.

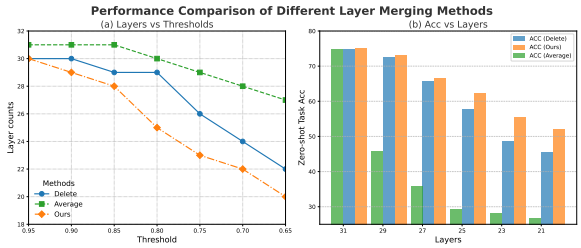

Figure 5: The impact of layer merging methods on LLaMA2-7B model. (a) Layer counts of the pruned model under different similarity thresholds through three layer merging methods (Delete, Average, Ours). (b) Zero-shot performance of the pruned model on the HellaSwag dataset through three layer merging methods.

	HellaSwag	OpenbookQA
C4 (num=10)	63.374	36.600
WikiText2 (num=10)	63.730	37.200
BookCorpus (num=10)	62.179	36.600
BookCorpus (num=20)	62.420	36.200

Table 3: The impact of calibration dataset on LLaMA2-7B model.

6 Conclusion

By analyzing the correlation between the outputs of different layers in the reproducing kernel Hilbert space, this paper reveals the "patch-like" relational patterns between layers in LLMs. Based on this insight, we propose a depth-wise pruning method that dynamically merges consecutive layers using a similarity threshold, enabling rapid model compression while effectively preserving performance. Experimental results show that our method significantly outperforms existing pruning methods on both inference efficiency and zero-shot tasks. Moreover, our method can be seamlessly integrated with width pruning methods, leading to pruning models that achieve enhanced performance. We hope this study will inspire further research into depth-wise pruning methods and foster the development of a unified framework that combines both depth-wise and width-wise pruning strategies, ultimately contributing to the efficient deployment of LLMs in resource-constrained environments.

Limitations

While our sliding layer merging method demonstrates strong performance across various models, datasets, and compression scales, several limitations remain.

First, due to computational constraints, our experiments were limited to models up to 13B parameters. The scalability of our method to larger-scale models (e.g., 70B or beyond) requires further validation. Second, the similarity threshold for layer merging was selected empirically based on optimal compression-peformance trade-offs under fixed compression ratios. While effective, the heuristic may not generalize optimally across all scenarios. Future work could explore more sophisticated threshold selection methods, to further improve merging efficiency.

Nevertheless, despite these limitations, our method consistently outperforms existing width-wise and depth-wise pruning methods across diverse evaluation settings, demonstrating its robustness and practical effectiveness.

References

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2024. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10865–10873.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, and 1 others. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

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. *arXiv preprint arXiv:1905.10044*.

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*.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. A framework for few-shot language model evaluation.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 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: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*, 11.

Eldar Kurtić, Elias Frantar, and Dan Alistarh. 2024. Ziplm: Inference-aware structured pruning of language models. *Advances in Neural Information Processing Systems*, 36.

Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems*, 35:24101–24116.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*.

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*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

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. *arXiv preprint arXiv:1809.02789*.

Azade Nova, Hanjun Dai, and Dale Schuurmans. 2023. Gradient-free structured pruning with unlabeled data. In *International Conference on Machine Learning*, pages 26326–26341. PMLR.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, and 1 others. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. 2021. Do vision transformers see like convolutional neural networks? *Advances in neural information processing systems*, 34:12116–12128.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*.

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*.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. *Preprint*, arXiv:2303.06865.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. 2024. Sleb: Streamlining llms through redundancy verification and elimination of transformer blocks. *arXiv preprint arXiv:2402.09025*.

Peng Sun, Yao Zhu, Yunjian Zhang, Xiu Yan, Zizhe Wang, and Xiangyang Ji. 2024a. Unleashing the potential of large language models through spectral modulation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 3892–3911.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2024b. Transformer layers as painters. *arXiv preprint arXiv:2407.09298*.

Yehui Tang, Fangcheng Liu, Yunsheng Ni, Yuchuan Tian, Zheyuan Bai, Yi-Qi Hu, Sichao Liu, Shangling Jui, Kai Han, and Yunhe Wang. 2024. Rethinking optimization and architecture for tiny language models. *arXiv preprint arXiv:2402.02791*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.

Sudong Wang, Yunjian Zhang, Yao Zhu, Jianing Li, Zizhe Wang, Yanwei Liu, and Xiangyang Ji. 2025. Towards understanding how knowledge evolves in large vision-language models. *arXiv preprint arXiv:2504.02862*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and 1 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

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*.

Yukun Zhu. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *arXiv preprint arXiv:1506.06724*.

A Experimental Setup

A.1 Baseline Methods

We primarily consider the width-wise pruning methods and depth-wise pruning methods as our baseline methods in our analysis. The

specific information of baseline methods are described below, where we use their official code for implementation. To ensure a fair comparison, we employ the same calibration dataset across all methods.

Width-wise method. The width-wise pruning methods include Wanda-sp, FLAP and LLM-Pruner.

Wanda-sp is a variant of Wanda (Sun et al., 2024b). The original metric of Wanda was based on the product of weight magnitudes and input activation norms, while Wanda-sp presented in (An et al., 2024) extends this in a struced way while using common dimensions among different modules.

Model	Pruned Ratio	Metrics	Structure	Params
LLaMA2-7B / Vicuna-7B	0.2	WIFV	AL-AM	5470556160
	0.35	WIFV	AL-AM	4485812224
LLaMA2-13B / Vicuna-13B	0.2	WIFV	AL-AM	10479211520
	0.35	WIFV	AL-AM	8575800320

Table 4: Hyperparameter settings for Wanda-sp.

FLAP (An et al., 2024) is a new LLM retraining-free structured pruning framework that determines the recoverability of the feature map after removing weight columns based on the fluctuation pruning index. It adaptively determines the compressed model structure using normalized importance scores and adds additional bias terms to the pruned feature maps to restore performance.

Model	Pruned Ratio	Metrics	Structure	Params
LLaMA2-7B / Vicuna-7B	0.2	WIFV	AL-AM	5444964352
	0.35	WIFV	AL-AM	4473626624
LLaMA2-13B / Vicuna-13B	0.2	WIFV	AL-AM	10481551360
	0.35	WIFV	AL-AM	8578908160
Meta-LLaMA3	0.2	WIFV	AL-AM	6721589248
	0.35	WIFV	AL-AM	5631029248

Table 5: Hyperparameter settings for FLAP.

LLM-Pruner (Ma et al., 2023) utilizes a Taylor-based importance metric to identify and remove attention heads from MHA and intermediate neurons from FFN. The pruning process is conducted locally, selecting removable groups within each module while ensuring that the dimensions across the blocks remain consistent. Following the original approach, we preserve the first and last few blocks without pruning. Both their pruned models and ours are retrained using Low-Rank Adaptation (LoRA).

Model	Pruned Ratio	Block_mlp_layer_start / BloCK_attention_layer_start	Block_mlp_layer_end / Block_attention_layer_start	Params
LLaMA2-7B/Vicuna-7B	0.24	4	30	5512646656
	0.43	4	30	4517122048
LLaMA2-13B/Vicuna-13B	0.24	4	38	10480911360
	0.42	4	38	8557153280
LLaMA3-8B	0.24	4	30	6794588160
	0.43	4	30	5651673088

Table 6: Hyperparameter settings for LLM-Pruner.

Depth-wise method. The depth-wise pruning methods use the Transformer module in LLM as the pruning unit. SLEB (Song et al., 2024) uses a logit-based approach to find unnecessary blocks and updates the importance score after removing each block. SLEB pursues a no-retraining setting, but it cannot maintain sufficient performance as the pruning rate increases. Shortened-LLM (Kim et al., 2024) uses the PPL standard to determine the importance of the transformer layer, and deletes unimportant transformer layers after sorting. Shortened-LLM method uses lora to retrain to restore the performance of the model.

Model	Pruned Ratio	Block	Head	FFN-D	Params
LLaMA2-7B/Vicuna-7B	0.2	26	32	11008	5524115456
	0.35	21	32	11008	4512198656
LLaMA2-13B/Vicuna-13B	0.2	32	40	13824	10478228480
	0.35	26	40	13824	8575001600
LLaMA3-8B	0.2	26	32	14336	6721589248
	0.35	21	32	14336	5631029248

Table 7: Hyperparameter settings for depth-wise methods.

A.2 Selected Transformer Layers

We summarize the number and indices of transformer blocks selected for removal using our method in Tab.8. The specified sets of Transformer layers are fused into a single layer, represented by the index of the first layer in the set (e.g., [25, 26, 27, 28, 29, 30] are fused into layer 25).

Model	Threshold	Num of layers	Merged Layers
LLaMA2-7B	0.81	26	[[25, 26, 27, 28, 29, 30], [10, 11]]
	0.68	21	[[22, 23, 24, 25, 26, 27, 28, 29, 30], [14, 15], [10, 11], [6, 7]]
Vicuna-7B	0.78	26	[[25, 26, 27, 28, 29, 30], [13, 14]]
	0.57	21	[[21, 22, 23, 24, 25, 26, 27, 28, 29, 30], [13, 14], [10, 11]]
LLaMA3-8B	0.72	26	[[28, 29, 30], [25, 26, 27], [22, 23], [14, 15]]
	0.51	21	[[25, 26, 27, 28, 29, 30], [21, 22], [19, 20], [14, 15, 16], [12, 13], [10, 11]]
LLaMA2-13B	0.8	32	[[30, 31, 32, 33, 34, 35, 36, 37, 38]]
	0.72	26	[[26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38], [16, 17], [12, 13]]
Vicuna-13B	0.76	32	[[33, 34, 35, 36, 37, 38], [31, 32], [29, 30], [15, 16]]
	0.65	26	[[31, 32, 33, 34, 35, 36, 37, 38], [29, 30], [27, 28], [9, 10, 11, 12, 13, 14]]

Table 8: Corresponding layer indices of the pruned models under different threshold setting of our depth-wise pruning method used in the main results

A.3 LoRA Retraining

We apply a LoRA adapter (Hu et al., 2021) to every projection weight matrix by following (Ma et al., 2023). We employ a LoRA rank of 8, a learning rate of 0.0001, and a batch size of 64 over 2 epochs. The retraining costs are notably low, with the entire process being executed on a single NVIDIA A100 (80GB VRAM) GPU. For example, retraining a 20%-pruned model from 7B parameters takes about 2 hours and utilizes 22GB GPU memory. Here we provide a brief introducion of LoRA retraining to ensure the self-contained aspect of our work.

Each learnable weight matrix (including both pruned and unpruned linear projections in LLMs) is represented by W. The update to W, denoted as $\triangle W$, is factorized as $\triangle W = PQ \in R^{d^- \times d^+}$, where $P \in R^{d^- \times d}$ and $Q \in R^{d \times d^+}$. Here, d^-, d, and d^+ correspond to the dimensions of the input, hidden, and output layers, respectively. The forward computation is then expressed as follows:

$$f(x) = (W + \triangle W)x + b = (WX + b) + (PQ)X, \quad (5)$$

where b represents the bias in the dense layer. By training only the low-rank matrices P and Q, we considerably reduce both the computational complexity and the dependence on large-scale training data. Furthermore, the additional parameters P and Q can be reparameterized as $\triangle W$, thereby introducing no extra parameters in the final pruned model.

B Supplementary Experiment Results

B.1 Additional Results of Inference Efficiency

Tab.9 shows the inference of our depth-wise pruning method in different LLMs (LLaMA2-7B, LLaMA2-13B, Vicuna-13B-v1.3, and LLaMA3-8B). As the pruning ratio increases, the latency of the model decreases, the throughput increases, and the GPU memory usage and number of parameters also decrease accordingly.

B.2 Integration of Depth-wise Pruning and Width-wise Pruning

We integrate our method with LLM-Pruner to further improve the pruning effect. In the first stage, we apply our proposed pruning method, based on the layer compression model. In the second stage, we use the LLM-Pruner method to remove unnecessary coupling structures from the model obtained in the first stage. As a result, the pruned model achieved a pruning rate of 35% relative to the original model. Tab.10 shows the pruning proportions and corresponding parameter settings of depth-wise pruning and width-wise pruning on the LLaMA2-7B model using the integrated method.

We also tested our integrated method on the Vicuna-13B model, and the experimental results are shown in Fig6. Tab.11 shows the pruning proportions and corresponding parameter settings of depth-wise pruning and width-wise pruning.

	Latency(sec)	Throughout(tokens/s)	GPU_Memory	nparam
LLaMA2-7B	2.729	46.905	13020.25	6738415616
20% Pruned	2.339	54.758	10682.45	5524115456
35% Pruned	1.889	67.770	8725.9	4512198656
LLaMA2-13B	3.635	35.210	25188.85	13015864320
20% Pruned	2.908	44.016	20274.25	10478228480
35% Pruned	2.380	53.793	16593.1	8575001600
Vicuna-7B-v1.3	2.865	44.681	13021.8	6738415616
20% Pruned	2.370	54.065	10682.45	5524115456
35% Pruned	2.000	64.087	8725.9	4512198656
Vicuna-13B-v1.3	3.593	35.623	25186.95	13015864320
20% Pruned	2.885	44.386	20274.25	10478228480
35% Pruned	2.395	53.461	16593.1	8575001600
LLaMA3-8B	3.150	40.640	15364	8030261248
20% Pruned	2.635	48.616	12862	6721589248
35% Pruned	2.218	57.795	10776	5631029248

Table 9: Inference efficiency of our pruned models. (Measured with 12 input tokens, 128 output tokens, and a batch size of 1 on a NVIDIA H100 GPU.)

Depth-wise(ours)			Width-wise(LLM-Pruner)		Nparam	AVE
Proportion	Threshold	Remove layers	Proportion	Pruner_ratio		
0%		0	100%	0.43	4532772064	52.366
18%	0.96	2	82%	0.35	4502876160	54.372
36%	0.86	4	64%	0.3	4501803008	55.753
53%	0.81	6	47%	0.24	4486926336	55.188
71%	0.77	8	29%	0.16	4476604416	53.967
91%	0.7	10	9%	0.06	4530630656	52.780
100%	0.68	11	0%		4512198656	51.869

Table 10: Pruning proportions and corresponding parameter settings of integrated method on the LLaMA2-7B model.

Depth-wise(ours)			Width-wise(LLM-Pruner)		Nparam	AVE
Proportion	Threshold	Remove layers	Proportion	Pruner_ratio		
0%		0	100%	0.42	8557153280	59.863
14%	0.94	2	86%	0.35	8606530560	61.236
28%	0.86	4	72%	0.3	8565928960	60.321
43%	0.8	6	57%	0.28	8606039040	60.455
57%	0.76	8	43%	0.24	8582722560	60.257
71%	0.7	10	29%	0.16	8551490560	61.203
100%	0.57	14	0%		8575001600	59.617

Table 11: Pruning proportions and corresponding parameter settings of integrated method on the Vicuna-13B model.

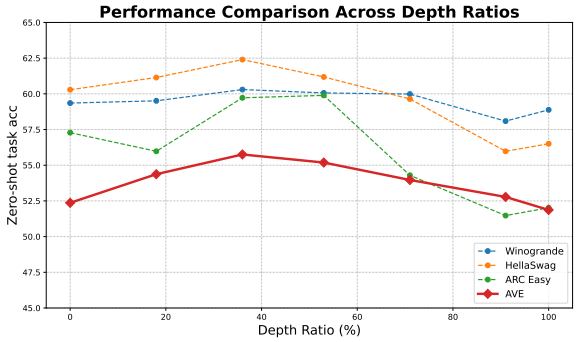

Figure 6: Performance of the integrated method on the Vicuna-13B model.

B.3 Three different layer merging methods

For the l-th layer of an LLM, we denote all its parameters as θ_l. We considered three different layer merging methods to merge the parameters of the subsequent m consecutive layers $\theta_{l+1}, \theta_{l+2}, ..., \theta_{l+m}$ into θ_l to form θ_l^*.

- **Remove:** We directly ignore the parameters of the $l+1$ to $l+m$ layers, and only retain the parameters of the l-th layer (although in actual operation, we may still label the combined parameter set as θ_l^*, but here θ_l^* actually only contains the parameters of the original l-th layer).

$$\theta_l^* = \theta_l \tag{6}$$

- **Average:** We calculate the average value of the parameters from layer l to $l+m$, and use this average value to form a new parameter set θ_l^*.

$$\theta_l^* = \frac{\sum_{i=l}^{l+m} \theta_i}{m} \tag{7}$$

- **Our method:** We adopt a parameter merging strategy based on inter-layer differences, adding the differences between adjacent layer and base layer parameters to gradually integrate redundant information.

$$\theta_l^* = \theta_l + \sum_{i=l+1}^{l+m} (\theta_i - \theta_l) \tag{8}$$

B.4 Zero-shot performance in larger scale

Tab.14 presents the zero-shot performance of various downstream tasks with the proposed method applied to the LLaMA2-13B model and Vicuna-13B model. Our method shows superior pruning capabilities.

Threshold	Num of layers	Merged Layers
0.95	30	[[27, 28], [13, 14]]
0.9	30	[[27, 28, 29]]
0.85	29	[[29, 30], [27, 28], [24, 25]]
0.8	29	[[27, 28, 29, 30]]
0.75	26	[[26, 27, 28, 29, 30], [24, 25], [9, 10]]
0.7	24	[[24, 25, 26, 27, 28, 29, 30], [22, 23], [9, 10]]
0.65	22	[[22, 23, 24, 25, 26, 27, 28, 29, 30], [17, 18], [9, 10]]

Table 12: The layer index corresponding to the pruned model obtained by the "delete" layer merging method under different threshold settings.

Threshold	Num of layers	Merged Layers
0.95	31	[[21, 22]]
0.9	31	[[28, 29]]
0.85	31	[[29, 30]]
0.8	30	[[29, 30], [23, 24]]
0.75	29	[[29, 30], [26, 27], [22, 23]]
0.7	28	[[29, 30], [27, 28], [23, 24], [10, 11]]
0.65	27	[[29, 30], [27, 28], [25, 26], [22, 23], [10, 11]]

Table 13: The layer index corresponding to the pruned model obtained by the "average" layer merging method under different threshold settings.

B.5 Additional Results of Moderate Pruning and LoRA Retraining

Tab.15-19 show the zero-shot results of several pruning strategies that require retraining, including LLM-Pruner, Shortened-LLM, and our proposed method on different model.

#Param & Method			BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLaMA-2-13B(Original)			80.550	79.053	79.367	72.139	79.377	49.147	45.200	69.262
20%Pruned	width	Wanda-sp	69.630	77.480	74.750	67.010	73.480	44.110	44.000	64.351
		FLAP	72.780	74.650	69.070	68.350	70.830	40.610	40.000	62.327
		LLM-Pruner	71.315	79.162	74.836	67.324	73.485	43.771	41.600	64.499
	depth	SLEB	63.211	76.061	70.116	65.430	70.749	39.932	39.200	60.671
		Shortened-LLM	68.318	76.279	75.204	71.113	74.790	46.672	42.400	64.968
		Ours	64.006	77.421	76.369	71.665	76.557	48.549	43.800	**65.481**
35%Pruned	width	Wanda-sp	59.020	55.110	33.580	52.800	29.840	24.910	28.600	40.551
		FLAP	71.250	69.590	61.680	64.400	59.050	34.130	36.000	56.586
		LLM-Pruner	66.086	76.061	67.785	59.195	67.382	39.334	41.800	59.663
	depth	SLEB	62.385	70.620	58.415	55.643	62.500	36.689	33.800	54.293
		Shortened-LLM	63.333	72.307	67.128	63.694	66.667	39.761	37.000	58.556
		Ours	59.939	73.286	68.751	65.983	68.981	41.724	39.000	**59.666**
Vicuna-13B-v1.3(Original)			82.813	78.346	77.017	71.113	75.547	47.611	45.400	68.264
20%Pruned	width	Wanda-sp	77.090	77.090	74.420	67.960	67.800	42.320	42.800	64.211
		FLAP	81.100	76.770	73.720	68.350	71.510	42.490	41.000	64.991
		LLM-Pruner	74.526	78.346	72.426	69.219	69.739	40.529	43.200	63.998
	depth	SLEB	62.385	70.620	58.415	55.643	62.500	36.689	33.800	54.293
		Shortened-LLM	75.535	77.476	73.571	68.272	72.180	44.198	43.200	64.919
		Ours	81.040	76.442	74.846	70.324	71.801	44.625	41.800	**65.840**
35%Pruned	width	Wanda-sp	61.650	71.220	63.960	61.400	57.490	35.320	37.000	55.434
		FLAP	75.170	73.990	65.540	67.560	61.320	36.770	37.400	59.679
		LLM-Pruner	70.581	76.659	67.317	65.272	63.258	35.154	40.800	59.863
	depth	SLEB	37.829	51.034	25.503	50.908	26.221	27.218	27.400	35.159
		Shortened-LLM	68.532	74.102	66.421	64.009	67.593	41.126	38.600	**60.055**
		Ours	69.450	74.918	67.447	63.378	66.414	38.311	37.400	59.617

Table 14: Performance comparison of pruning methods across multiple baselines on LLaMA2-13B and Vicuna-13B models.

20%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	53.761	76.659	66.132	61.168	64.857	37.884	40.200	57.237
	w_lora	63.731	77.476	67.128	61.878	65.783	38.481	40.400	59.268
Shortened-LLM	wo_lora	60.489	73.776	63.364	57.459	64.015	33.191	36.200	55.499
	w_lora	61.560	76.061	67.994	58.800	68.813	37.884	38.000	58.445
Ours	wo_lora	62.324	70.239	65.097	66.298	61.448	38.311	37.800	57.360
	w_lora	69.450	73.667	70.484	67.088	69.108	41.212	42.600	61.944
35%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	50.703	69.695	51.275	52.960	49.495	31.399	36.000	48.790
	w_lora	45.260	74.755	60.287	59.353	57.281	32.423	37.200	52.366
Shortened-LLM	wo_lora	61.101	67.791	45.987	52.960	48.485	27.218	32.800	48.049
	w_lora	62.171	71.926	54.800	51.776	59.259	30.887	35.400	52.317
Ours	wo_lora	62.171	63.874	52.131	59.511	46.170	31.826	32.600	49.755
	w_lora	63.119	65.452	56.503	58.879	52.020	31.911	35.200	51.869

Table 15: Performance with/without LoRA retraining on LLaMA2-7B.

20%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	57.554	74.810	65.216	59.037	64.815	36.263	39.400	56.728
	w_lora	67.645	76.115	66.660	63.931	65.446	36.604	40.400	59.543
Shortened-LLM	wo_lora	63.670	72.470	63.105	62.194	64.352	36.775	35.600	56.881
	w_lora	72.355	74.701	67.576	64.562	70.034	38.225	38.600	60.865
Ours	wo_lora	63.394	72.742	66.152	66.219	66.919	39.078	40.600	59.301
	w_lora	77.431	74.755	69.040	68.745	69.318	38.908	40.200	62.628
35%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	60.642	70.294	54.860	52.881	53.662	33.618	36.000	51.708
	w_lora	63.976	73.069	59.560	58.564	56.524	32.679	37.800	54.596
Shortened-LLM	wo_lora	60.245	64.527	43.856	53.986	47.348	26.792	29.600	46.622
	w_lora	64.281	70.783	56.722	57.380	59.596	31.485	34.000	53.464
Ours	wo_lora	62.202	66.431	53.834	61.484	52.946	33.532	33.400	51.976
	w_lora	69.235	70.294	60.705	62.273	60.227	33.618	37.400	56.250

Table 16: Performance with/without LoRA retraining on Vicuna-7B-v1.3.

20%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	56.942	77.040	67.785	68.666	68.603	39.078	40.400	59.788
	w_lora	74.037	79.489	74.268	70.324	74.285	46.587	42.600	65.941
Shortened-LLM	wo_lora	45.443	73.232	60.994	57.853	65.404	34.044	36.000	53.282
	w_lora	66.208	78.074	72.695	62.747	75.295	44.795	43.400	63.316
Ours	wo_lora	38.073	71.980	61.800	69.613	66.035	41.809	38.400	55.387
	w_lora	76.789	77.639	73.770	71.744	76.599	50.939	41.200	66.954
35%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	47.829	69.369	45.150	53.118	48.485	27.816	33.200	46.424
	w_lora	64.465	74.048	61.800	59.353	64.646	34.386	37.200	56.557
Shortened-LLM	wo_lora	61.651	66.431	49.801	51.697	51.431	29.352	30.400	48.680
	w_lora	63.180	72.851	62.985	58.090	66.877	37.116	37.000	56.871
Ours	wo_lora	40.428	62.350	40.291	55.485	39.394	28.242	28.200	42.056
	w_lora	67.554	73.830	61.472	62.747	64.352	36.007	37.600	57.652

Table 17: Performance with/without LoRA retraining on LLaMA3-8B.

20%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	65.566	78.509	72.018	64.167	69.992	43.601	40.600	62.065
	w_lora	71.315	79.162	74.836	67.324	73.485	43.771	41.600	64.499
Shortened-LLM	wo_lora	63.180	75.027	71.191	70.481	69.529	43.089	40.800	61.900
	w_lora	68.318	76.279	75.204	71.113	74.790	46.672	42.400	64.968
Ours	wo_lora	38.318	72.361	67.726	70.797	64.815	39.676	42.800	56.642
	w_lora	64.006	77.421	76.369	71.665	76.557	48.549	43.800	65.481
35%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	52.049	74.048	61.522	55.801	60.816	36.689	39.800	54.389
	w_lora	66.086	76.061	67.785	59.195	67.382	39.334	41.800	59.663
Shortened-LLM	wo_lora	62.141	69.042	59.361	60.221	53.830	31.741	34.400	52.962
	w_lora	63.333	72.307	67.128	63.694	66.667	39.761	37.000	58.556
Ours	wo_lora	40.489	67.900	56.234	63.694	51.347	33.532	36.400	49.942
	w_lora	59.939	73.286	68.751	65.983	68.981	41.724	39.000	59.666

Table 18: Performance with/without LoRA retraining on LLaMA2-13B.

20%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	74.006	77.040	71.679	64.799	66.498	39.078	41.800	62.129
	w_lora	74.526	78.346	72.426	69.219	69.739	40.529	43.200	63.998
Shortened-LLM	wo_lora	66.239	74.918	70.056	66.456	68.476	43.857	39.600	61.372
	w_lora	75.535	77.476	73.571	68.272	72.180	44.198	43.200	64.919
Ours	wo_lora	75.199	75.898	71.679	69.692	70.370	43.942	43.200	64.283
	w_lora	81.040	76.442	74.846	70.324	71.801	44.625	41.800	65.840
35%Pruned		BoolQ	PIQA	HellaSwag	WinoGrande	ARC-easy	ARC-challenge	OpenbookQA	AVE
LLM_Pruner	wo_lora	63.609	73.449	63.683	58.800	52.778	34.471	38.200	54.999
	w_lora	70.581	76.659	67.317	65.272	63.258	35.154	40.800	59.863
Shortened-LLM	wo_lora	42.385	69.532	57.897	60.063	60.017	37.372	34.800	51.724
	w_lora	68.532	74.102	66.421	64.009	67.593	41.126	38.600	60.055
Ours	wo_lora	63.609	73.123	58.703	61.089	62.837	36.604	34.600	55.795
	w_lora	69.450	74.918	67.447	63.378	66.414	38.311	37.400	59.617

Table 19: Performance with/without LoRA retraining on Vicuna-13B-v1.3.