# Sliding-Window Merging for Compacting Patch-Redundant Layers in LLMs

**Xuan Ding**[1,2], **Rui Sun**[1,2*], **Yunjian Zhang**[3], **Xiu Yan**[4], **Yueqi Zhou**[5], **Kaihao Huang**[5], **Suzhong Fu**[1,2], **Angelica I Aviles-Rivero**[6], **Chuanlong Xie**[5], **Yao Zhu**[7*]

[1]Shenzhen Future Network of Intelligence Institute and Guangdong Provincial Key Laboratory of Future Networks of Intelligence, The Chinese University of Hong Kong (Shenzhen),
[2]School of Science and Engineering, The Chinese University of Hong Kong (Shenzhen),
[3]University of Chinese Academy of Sciences,
[4]Meituan Group,
[5]Beijing Normal University,
[6]Tsinghua University,
[7]Zhejiang University
202322011119@mail.bnu.edu.cn, ruisun@link.cuhk.edu.cn, ee_zhuy@zju.edu.cn

## Abstract

Depth-wise pruning accelerates LLM inference in resource-constrained scenarios but suffers from performance degradation due to direct removal of entire Transformer layers. This paper reveals "Patch-like" redundancy across layers via correlation analysis of the outputs of different layers in reproducing kernel Hilbert space, demonstrating consecutive layers exhibit high functional similarity. Building on this observation, this paper proposes **S**liding-**W**indow **M**erging (SWM) - a dynamic compression method that selects consecutive layers from top to bottom using a pre-defined similarity threshold, and compacts patch-redundant layers through a parameter consolidation, 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.

**Code** —
https://github.com/920927/Sliding-Window-Merging
**Extended version** — https://arxiv.org/pdf/2502.19159

## 1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across a wide range of applications (Touvron et al. 2023; Chowdhery et al. 2023; Wang et al. 2025), but their ever-increasing scale has also introduced significant deployment challenges. Parameter pruning is considered an effective approach to mitigate redundancy in these over-parameterized systems (Ma, Fang, and Wang 2023; Sun et al. 2024a). Width-wise pruning removes coupled components—such as attention heads and their associated weight connections—while preserving network depth (Ma, Fang, and Wang 2023; An et al. 2024; Sun et al. 2024b), but it struggles to address computational bottlenecks along the temporal dimension. In contrast, depth-wise pruning reduces network depth by entirely removing certain layers (Kim et al. 2024; Men et al. 2024; Song et al. 2024), significantly improving inference efficiency in resource-constrained settings. However, existing depth-wise pruning methods often lack systematic analysis of inter-layer correlations, making it difficult to fully exploit potential redundancy within the model's hierarchical structure. This may lead to structural fragmentation, disrupting the continuity of knowledge flow and ultimately resulting in notable performance degradation.

To resolve this limitation, we focus on the correlations between the layers in LLMs and pose a targeted research question: Do consecutive layers in LLMs exhibit functional redundancy beyond geometric proximity? Inspired by (Raghu et al. 2021), 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. Through systematic evaluation across multiple models and datasets, we reveal a consistent "patch-redundancy" phenomenon: 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 for "patch-redundant" layers, parameter consolidation by layer merging rather than arbitrary removal can preserve knowledge integrity while eliminating structural redundancy.

Building on the "patch-redundancy" discovery, we propose Sliding-Window Merging (SWM) - a novel compressing method that compacts the consecutive redundant layers. 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.

*Corresponding author.

For the layers within the window 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 our Sliding-Window 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, offering useful insights for the design of model compression strategies.
- We propose the Sliding-Windows 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, Levy, and Neubig (2019) introduced pruning and attention head sharing techniques to reduce redundant attention heads, thereby decreasing both computational complexity and parameter requirements. Nova, Dai, and Schuurmans (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, Frantar, and Alistarh 2024). Width pruning reduces FLOPs but fails to address temporal bottlenecks in autoregressive inference, limiting latency improvements.

**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 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. Existing depth pruning relies on heuristic importance scores, which ignore functional redundancy and cause irreversible knowledge loss through layer deletion.

## 3 Method

### 3.1 Motivation

**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.

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

  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.

$$\tilde{K} = HKH, \tilde{L} = HLH, \tag{2}$$

  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.

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

  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.

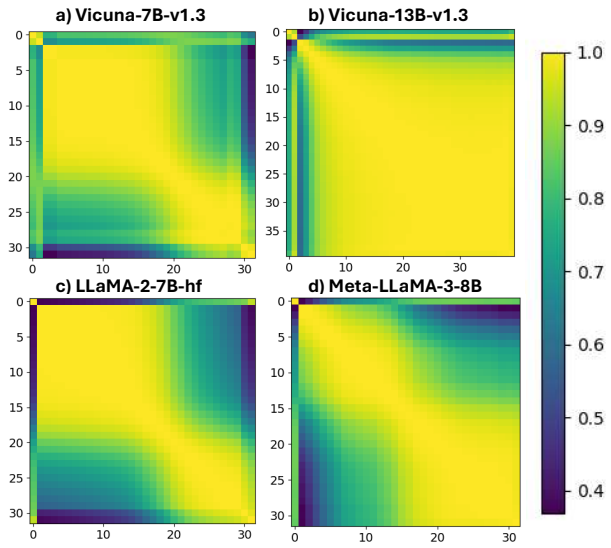


Figure 1: CKA (Center Kernel Alignment) metric between pairs of Transformer layers in LLMs.

**Patch-Redundancy Mechanism in LLMs** 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 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. The results reveal a consistent geometric pattern across all evaluated LLMs: 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.

**Theoretical Basis:** We attribute this phenomenon to two inherent properties of Transformer architectures:

1. **Residual Learning Dynamics**: The residual connection $x_{l+1} = x_l + F(x_l, \theta_l)$ enables gradual feature refinement. When consecutive layers $l$ and $l+1$ satisfy $\|F(x_l, \theta_l)\| \ll \|x_l\|$, their outputs become linearly correlated, manifesting as high CKA similarity.
2. **Optimization-Induced Redundancy**: During pretraining, layers initialized near identity transform accumulate similar gradients ($\frac{\partial \mathcal{L}}{\partial \theta_l} \approx \frac{\partial \mathcal{L}}{\partial \theta_{l+1}}$), converging to functionally equivalent solutions.

## 3.2 Overview of SWM Design

Based on the aforementioned CKA analysis, we propose Sliding-Window Merging (SWM)—a novel compression method that dynamically consolidates Transformer layers from top to bottom while preserving task-critical representations through three core principles:

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**
  $M_{tmp} \leftarrow \text{Merge}(M^*, h\_lay, l\_lay)$
  $s \leftarrow \text{Cal\_Sim}(M, M_{tmp}, D)$
  **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** $M^*$

- **Parameter merging strategy** (Sec.3.3): The observed patch-redundancy indicates functional equivalence and provide space for compression. SWM replaces deletion with parameter consolidation (Eq. (4)) to preserve knowledge flow.
- **Adaptive similarity thresholding** (Sec.3.4): While CKA analysis reveals layer redundancy patterns, SWM employs computationally efficient cosine similarity (O(n) complexity) during runtime compression. Cosine similarity-driven thresholding criteria enable precision compression decisions, preserving CKA's isotropic invariance while accelerating computation.
- **Sliding-window merging algorithm** (Sec.3.5): Variable patch sizes across models necessitate layer-specific compression. SWM achieves this via dynamic window expansion (triggered by the comparison between cosine similarity and a predefined threshold). Moreover, low CKA correlation in boundary layers confirms their task-critical role. SWM excludes these layers from compression entirely (setting protected range [L, H] in Algorithm 1) to prevent irreversible performance damage.

## 3.3 Parameter merging strategy

We compact layers based on inter-layer differences (as shown in Fig.2(b)), which progressively integrates redundant information 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 $\theta^*$ 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 $\theta_i$ maintains fundamental model capabilities, while (2)

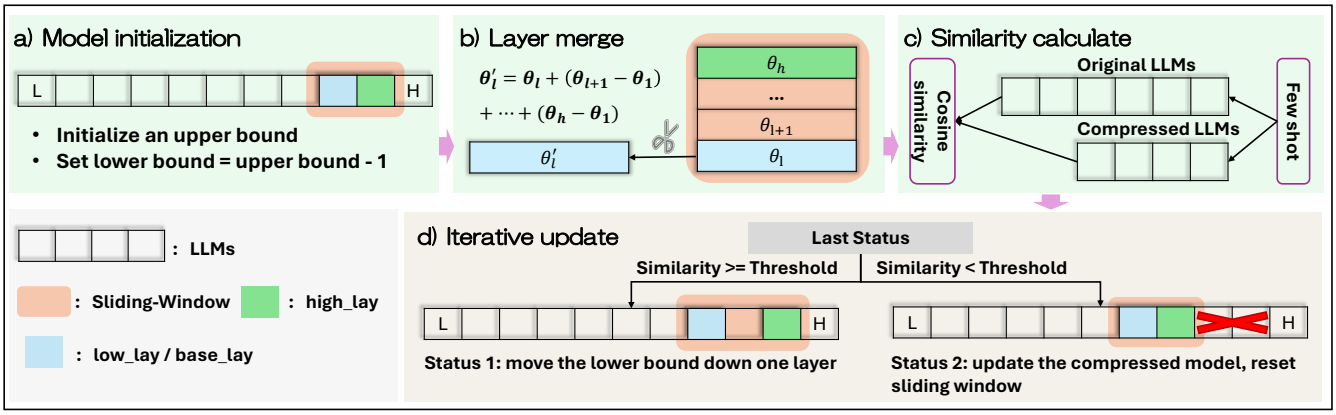


Figure 2: The framework of our sliding-windows merging (SWM) method. (a) Model initialization establishes upper / lower bounds for the sliding window. (b) Layer merging via parameter consolidation: $\theta_l^*$ denotes merged parameters combining adjacent layers within the window. (c) Similarity validation computes cosine similarity between original and compressed models' outputs using few-shot evaluation. (d) Adaptive window adjustment: the window slides downward if similarity meets thresholds (Status 1), otherwise the compressed model updates and window resets (Status 2). Color coding: gray blocks (original LLM layers), red-orange window (active merging region), green / blue markers (current upper / lower bounds).

the difference terms $(\theta_{i+k} - \theta_i)$ incorporate complementary features from other layers. The strategy effectively balances knowledge preservation and redundancy elimination across varying compression requirements.

### 3.4 Adaptive similarity thresholding

The dynamic window updating process relies on comparing cosine similarity between original ($M$) and compressed ($M_{tmp}$) model outputs against predefined thresholds iteratively (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. We optimize thresholds via validation-set grid search under pruning constraints, selecting the highest feasible value maximizing zero-shot capability retention to ensure better model performance.

### 3.5 Sliding window merging algorithm

As shown in Algorithm 1, our method performs iterative layer compression through three steps: (1) model initialization, (2) dynamic window updating, and (3) termination condition evaluation.

**Model initialization**. We initialize the target compressed model $M^*$ as a copy of the original model $M$ while simultaneously configuring the sliding window (upper=$H$, lower=$H-1$) within a predefined compression range $[L, H]$, where $H$ and $L$ denote the highest and lowest layers to be compressed, respectively (see Fig.2 (a)). 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.

**Dynamic window updating.** Our method iteratively merge layers within current sliding 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 (d)). 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.

### 3.6 Performance Recovery by 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, Fang, and Wang 2023; Kim et al. 2024), and we provide a brief introduction to ensure the self-contained aspect of our work in Appendix A.3.

## 4 Experiments

### 4.1 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).

| Pruned Rates | Model | | BoolQ | PIQA | HellaSwag | WinoGrande | ARC-easy | ARC-challenge | OpenbookQA | AVE |
|---|---|---|---|---|---|---|---|---|---|---|
| 0% | LLaMA2-7B | | 77.706 | 78.074 | 76.021 | 69.140 | 76.305 | 46.331 | 44.200 | 66.825 |
| 20% | 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% | 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 |
| 0% | Vicuna-7B-v1.3 | | 78.104 | 77.312 | 73.939 | 69.376 | 74.327 | 44.454 | 43.800 | 65.902 |
| 20% | 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% | 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** |
| 0% | LLaMA3-8B | | 81.101 | 79.489 | 79.167 | 73.402 | 80.093 | 53.242 | 44.800 | 70.185 |
| 20% | 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% | 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: Zero-shot performance comparison of pruning methods at 20% and 35% pruning rates. We compare our method with width pruning (Wanda-sp, FLAP, LLM-Pruner) and depth pruning (SLEB, Shortened-LLM) methods on LLaMA2-7B, Vicuna-7B, and LLaMA3-8B. Note: Wanda-sp does not produce results for LLaMA3-8B due to incompatibility.

**Baselines.** We benchmark SWM against several width pruning (LLM-Pruner (Ma, Fang, and Wang 2023), FLAP & Wanda-sp (An et al. 2024)) and depth pruning (SLEB (Song et al. 2024), Shortened-LLM (Kim et al. 2024)) methods. Following the experimental setup of Shortened-LLM, we assess all methods under two target pruning levels: 20% and 35%. If the product of the total number of Transformer layers and target sparsity is not an integer, we round up to determine the number of layers 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.** Our implementation builds on PyTorch (Paszke et al. 2019) and HuggingFace Transformers (Wolf et al. 2020). Following Ma, Fang, and Wang (2023), we randomly select 10 samples from BookCorpus (Zhu 2015) for similarity computation during pruning. 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 executed on NVIDIA A100 GPUs (80GB memory), with pruning efficiently integrated into forward passes (about 2 minutes for 13B models) and memory footprint comparable to standard inference.

## 4.2 Zero-shot Tasks

Tab.1 demonstrates SWM's consistent superiority over both width and depth pruning baselines across LLaMA2-7B, Vicuna-7B-v1.3, and LLaMA3-8B models. Specifically, under the 20% pruning rate of LLaMA2-7B model, our method achieves a 2.676% higher accuracy than the best-performing

| Pruned Rates | Model | | Latency | Throughout | GPU_Mem | nparam |
|---|---|---|---|---|---|---|
| 0% | LLaMA2-7B | | 2.729 | 46.905 | 13020.25 | 6.7B |
| 20% | 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% | 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 comparison of pruning methods. (Measured with 12 input tokens, 128 output tokens and a batch size of 1.)

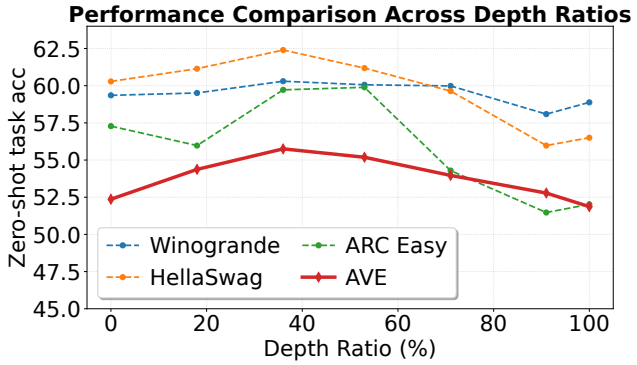


Figure 3: Performance of the integrated method on LLaMA2-7B. The horizontal axis shows the depth pruning ratio, and the vertical axis indicates zero-shot task performance. Dotted lines denote individual task metrics (Winogrande, HellaSwag, ARC-easy), while the solid line shows the average across all seven tasks.

LLM-Pruner method; under the 35% pruning rate of Vicuna-7B model, the average accuracy of our method is 1.654% higher than that of existing methods. Moreover, when handling advanced architectures like LLaMA3-8B, SWM maintains robust performance (66.954 vs original 70.185) while width pruning method FLAP suffers catastrophic failure (34.147 at 35% pruning). These results show that our method can effectively reduce model size and complexity while more fully maintaining performance, attributable to its knowledge-preserving consolidation mechanism.

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.

### 4.3 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 LLaMA2-7B.

Experimental results show that depth pruning generally outperforms width pruning in reasoning efficiency. Specifically, at pruning ratio of 20% and 35%, depth pruning (SLEB, Shortened-LLM and Ours) outperform width pruning methods (Wanda-sp, FLAP, and LLM-Pruner) in both latency and throughout. This suggests that reducing model depth can more effectively enhances inference speed and throughout. Additionally, 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.

### 4.4 Integration of width pruning and depth pruning

We observed in Tab.1 that in the 35% pruning scenario for LLaMA2-7B, the width pruning (LLM-Pruner) slightly outperforms 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 pruning and depth pruning leverage the strengths of both methods to further improve pruning results?

Specifically, using LLaMA2-7B model as an example, we propose a combined compression paradigm integrating SWM's layer consolidation with LLM-Pruner's structured sparsity. In the depth stage, we apply SWM to compress redundant layers of pruning attributed to depth-wise pruning (ranging from 0% to 100%); In the width stage, we employ LLM-Pruner on the model obtained in the depth stage to remove non-essential coupled structures, achieving total 35% sparsity. For the sake of pruning convenience, we selected the following depth pruning rates: 0% (LLM-Pruner), 18%, 36%, 53%, 71%, 91%, and 100% (SWM).

As shown in Fig.3, the combined depth and width 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 those with 0% and 100% depth pruning. Notably, models with depth pruning rates of 36% and 53% rank first and second in performance, respectively. This shows that the integrated methods leverage the advantages of both depth pruning and width pruning methods, achieving better model performance than when using depth pruning or width pruning alone, while mitigating the throughput and inference speed issues associated with width pruning methods. We also performed the same experiments on Vicuna-13B-v1.3 model, with results in Appendix B.2.

### 4.5 The impact of LoRA retraining

We evaluate the impact of LoRA retraining on three baselines requiring fine-tuning (LLM-Pruner, Shortened-LLM, and ours) across LLaMA2-7B and Vicuna-7B at 20% and 35% pruning rates. The results in Fig.4 show that before LoRA fine-tuning, our method significantly outperformed LLM-Pruner and Shortened-LLM, achieving the best per-

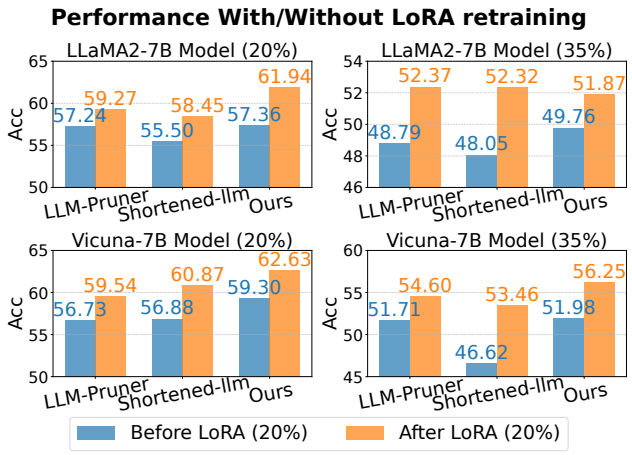


Figure 4: Performance with/without LoRA retraining. The blue column shows performance before LoRA fine-tuning, and the orange column after.

formance at both 20% and 35% pruning ratios. After further employing LoRA fine-tuning, our method's performance significantly improves and still maintains the lead in most cases. Specifically, under the 20% pruning ratio of LLaMA2-7B model, our method improves from 57.360% to 61.944%, a 4.584% increase, far exceeding LLM-Pruner (by 2.031%) and Shortened-LLM (by 2.946%).

### 4.6 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 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 gap gradually expanding as the number of compression layers increases.

### 4.7 The impact of Calibration data

Tab.3 investigates the impact of calibration data selection on LLaMA2-7B model. The results confirm our method's robustness to calibration data selection. Across diverse corpora and sample sizes, performance variations on tasks are minimal. Specifically, HellaSwag accuracy fluctuates within

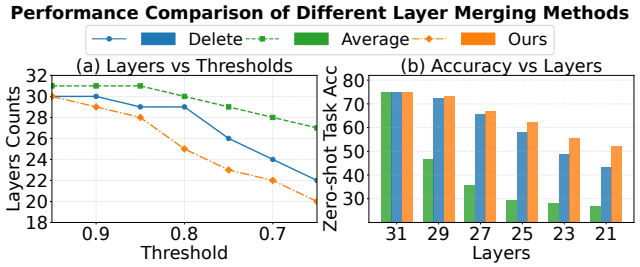


Figure 5: The impact of layer merging methods on LLaMA2-7B model. (a) Layer counts of the pruned model under different similarity thresholds through three layer merging methods: Delete (blue circles), Average (green squares), Ours (orange diamonds). (b) Zero-shot performance of the pruned model on the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

| | 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: Performance impact of calibration dataset selection on LLaMA2-7B model. HellaSwag evaluates contextual understanding while OpenbookQA tests factual knowledge.

1.6% absolute (62.2–63.7%), while OpenbookQA shows less than 1.0% deviation (36.2–37.2%). Crucially, no dataset demonstrates consistent superiority, and larger sample sizes yield diminishing returns. The observed stability suggests that choice of calibration data has minimal practical impact on our method's effectiveness. This relative invariance highlights the operational robustness of our method for real-world compression scenarios.

## 5 Conclusion

By analyzing the correlation between 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 Sliding-Window Merging method to compact the patch-redundant layers. This method 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 pruned 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.

## References


An, Y.; Zhao, X.; Yu, T.; Tang, M.; and Wang, J. 2024. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 10865–10873.

Bisk, Y.; Zellers, R.; Gao, J.; Choi, Y.; et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 7432–7439.

Chiang, W.-L.; Li, Z.; Lin, Z.; Sheng, Y.; Wu, Z.; Zhang, H.; Zheng, L.; Zhuang, S.; Zhuang, Y.; Gonzalez, J. E.; et al. 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.

Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H. W.; Sutton, C.; Gehrmann, S.; et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113.

Clark, C.; Lee, K.; Chang, M.-W.; Kwiatkowski, T.; Collins, M.; and Toutanova, K. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Del Corro, L.; Del Giorno, A.; Agarwal, S.; Yu, B.; Awadallah, A.; and Mukherjee, S. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*.

Gao, L.; Tow, J.; Abbasi, B.; Biderman, S.; Black, S.; DiPofi, A.; Foster, C.; Golding, L.; Hsu, J.; Le Noac'h, A.; Li, H.; McDonell, K.; Muennighoff, N.; Ociepa, C.; Phang, J.; Reynolds, L.; Schoelkopf, H.; Skowron, A.; Sutawika, L.; Tang, E.; Thite, A.; Wang, B.; Wang, K.; and Zou, A. 2024. A framework for few-shot language model evaluation.

Hu, E. J.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W.; et al. 2021. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Kim, B.-K.; Kim, G.; Kim, T.-H.; Castells, T.; Choi, S.; Shin, J.; and Song, H.-K. 2024. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*, 11.

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

Kwon, W.; Kim, S.; Mahoney, M. W.; Hassoun, J.; Keutzer, K.; and Gholami, A. 2022. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems*, 35: 24101–24116.

Lagunas, F.; Charlaix, E.; Sanh, V.; and Rush, A. M. 2021. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*.

Ma, X.; Fang, G.; and Wang, X. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36: 21702–21720.

Men, X.; Xu, M.; Zhang, Q.; Wang, B.; Lin, H.; Lu, Y.; Han, X.; and Chen, W. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Michel, P.; Levy, O.; and Neubig, G. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Mihaylov, T.; Clark, P.; Khot, T.; and Sabharwal, A. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Nova, A.; Dai, H.; and Schuurmans, D. 2023. Gradient-free structured pruning with unlabeled data. In *International Conference on Machine Learning*, 26326–26341. PMLR.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Raghu, M.; Unterthiner, T.; Kornblith, S.; Zhang, C.; and Dosovitskiy, A. 2021. Do vision transformers see like convolutional neural networks? *Advances in neural information processing systems*, 34: 12116–12128.

Raposo, D.; Ritter, S.; Richards, B.; Lillicrap, T.; Humphreys, P. C.; and Santoro, A. 2024. Mixture-of-Depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*.

Sakaguchi, K.; Bras, R. L.; Bhagavatula, C.; and Choi, Y. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9): 99–106.

Santacroce, M.; Wen, Z.; Shen, Y.; and Li, Y. 2023. What matters in the structured pruning of generative language models? *arXiv preprint arXiv:2302.03773*.

Schuster, T.; Fisch, A.; Gupta, J.; Dehghani, M.; Bahri, D.; Tran, V.; Tay, Y.; and Metzler, D. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35: 17456–17472.

Sheng, Y.; Zheng, L.; Yuan, B.; Li, Z.; Ryabinin, M.; Fu, D. Y.; Xie, Z.; Chen, B.; Barrett, C.; Gonzalez, J. E.; Liang, P.; Ré, C.; Stoica, I.; and Zhang, C. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. arXiv:2303.06865.

Song, J.; Oh, K.; Kim, T.; Kim, H.; Kim, Y.; and Kim, J.-J. 2024. SLEB: Streamlining LLMs through Redundancy Verification and Elimination of Transformer Blocks. *arXiv preprint arXiv:2402.09025*.

Sun, P.; Zhu, Y.; Zhang, Y.; Yan, X.; Wang, Z.; and Ji, X. 2024a. Unleashing the Potential of Large Language Models through Spectral Modulation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 3892–3911.

Sun, Q.; Pickett, M.; Nain, A. K.; and Jones, L. 2024b. Transformer layers as painters. *arXiv preprint arXiv:2407.09298*.

Tang, Y.; Liu, F.; Ni, Y.; Tian, Y.; Bai, Z.; Hu, Y.-Q.; Liu, S.; Jui, S.; Han, K.; and Wang, Y. 2024. Rethinking optimization and architecture for tiny language models. *arXiv preprint arXiv:2402.02791*.

Taori, R.; Gulrajani, I.; Zhang, T.; Dubois, Y.; Li, X.; Guestrin, C.; Liang, P.; and Hashimoto, T. B. 2023. Stanford alpaca: An instruction-following llama model.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Voita, E.; Talbot, D.; Moiseev, F.; Sennrich, R.; and Titov, I. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.

Wang, S.; Zhang, Y.; Zhu, Y.; Li, J.; Wang, Z.; Liu, Y.; and Ji, X. 2025. Towards Understanding How Knowledge Evolves in Large Vision-Language Models. *arXiv preprint arXiv:2504.02862*.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 38–45.

Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; and Choi, Y. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Zhu, Y. 2015. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. *arXiv preprint arXiv:1506.06724*.