UDC: 004.8 https://doi.org/10.15407/jai2025.04.088

V. Shvets[1], N. Shapoval[2]
[1,2]National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine
37, Beresteysky Avenue, Kyiv, 03056
[1]shvets.vitaliy@lll.kpi.ua
[2]shovgun@gmail.com
[1]https://orcid.org/0009-0009-2998-158X
[2]https://orcid.org/0000-0002-8509-6886

STRUCTURED PRUNING METHOD FOR LARGE LANGUAGE MODELS WITH ADAPTIVE COMPRESSION RATIOS

Abstract. The article addresses the important challenge of deploying large language models (LLMs) on resource-constrained devices. We analyze the evolution of neural network pruning methods from classical approaches (Optimal Brain Damage, Optimal Brain Surgeon) to modern one-shot techniques for LLMs (SparseGPT, Wanda, SliceGPT, 2SSP). The research demonstrates that while unstructured pruning achieves high compression ratios with small quality loss, it fails to provide real size reduction and inference acceleration on standard hardware due to chaotic sparse matrix structures. In contrast, structured pruning methods ensure hardware efficiency by removing entire structural blocks. We propose the Adaptive 2SSP method (modification of the 2SSP method), which combines adaptive compression ratio selection based on block redundancy with two-stage structured pruning: attention block removal (depth pruning) followed by FFN layer neuron removal (width pruning). Experimental validation on Llama-3.2-3B, Llama-2-7B, and Qwen2.5-3B models demonstrates the method's superiority over existing alternatives (GLU Aware Pruning, Dynamic Slicing, original 2SSP). When removing 40% of Llama-3.2-3B parameters, the method maintains perplexity at 26.35 and average benchmark accuracy at 39.57%, representing the best results among compared methods. Hardware efficiency evaluation for Llama-3.2-3B achieved 35.12% reduction in VRAM consumption and 34.78% acceleration in token generation. For Llama-2-7B, a 3.7-fold speedup was obtained at 20% pruning by overcoming VRAM limitations. The results demonstrate that the proposed method provides an optimal balance between compression degree, execution speed, and model quality preservation, making it an effective tool for adapting modern LLMs to deployment on devices with limited computational resources.

Keywords: language models, pruning, model compression, structured pruning, adaptive pruning, LLM optimization, transformers, neural network compression.

Introduction

The rapid advancement of natural language processing in recent years has been driven by large language models (LLMs) such as GPT, Claude, Gemini, LLaMA, Mistral, etc. These models demonstrate unprecedented capabilities in text generation, translation, decision support, and creative tasks. The effectiveness of such models directly correlates with their scale: increasing the number of parameters often leads to quality improvements and the emergence of new "emergent" properties [1]. However, this massive scale creates serious challenges – models with hundreds of billions of parameters require enormous computational resources, memory, and energy for both training and inference.

The primary challenges of large-scale language models include high computational complexity of inference, substantial memory requirements (up to hundreds of gigabytes for 70B-level models), increased response latency, environmental costs, and impossibility of deployment on resource-constrained devices. For example, training GPT-3 (175 billion parameters) cost approximately $4.6 million and resulted in CO_2 emissions equivalent to over 500 tons [2]. A model with 70 billion parameters in FP16 format requires approximately 140 GB of VRAM for inference alone, making its use impossible on most edge devices.

These factors stimulate research into optimization and compression methods for large language models. Among the main approaches are quantization (reducing the bit size of weight representation), knowledge distillation (training a smaller model to mimic the behavior of a large one), and pruning – removing redundant parameters with minimal quality loss. While quantization is the most popular method for reducing VRAM usage, it has compression limits (typically up to 4 bits without significant

quality loss). Knowledge distillation is the most computationally expensive of all LLM optimization methods. Pruning remains a method that has been actively developing in the field of LLM applications in recent years.

Research Problem Statement

Academic research in neural network compression faces a fundamental challenge: how to make powerful large language models accessible for deployment on devices with limited computational resources without significant quality degradation. The problem is multifaceted and has several dimensions.

First, there is a computational accessibility gap. Current state-of-the-art LLMs require industrial-scale computing infrastructure for deployment, limiting their use to organizations with substantial resources. This creates barriers to democratization of AI technology and restricts innovation potential, particularly in resource-constrained environments such as mobile devices, edge computing, etc.

Second, the environmental sustainability concern is increasingly urgent. Energy consumption associated with training and deploying large models contributes significantly to carbon emissions. For instance, the training of GPT-3 alone resulted in emissions equivalent to over 500 tons of CO_2 [2]. As AI adoption accelerates globally, the cumulative environmental impact becomes a pressing issue that demands more efficient model architecture.

Third, there exists a methodological challenge in compression techniques themselves. While several approaches to model compression have been developed, including quantization, knowledge distillation, and pruning, each presents distinct trade-offs between compression ratio, quality preservation, and hardware efficiency. Specifically, unstructured pruning methods, though achieving high compression ratios, fail to deliver real-world speedups due to sparse matrix patterns that cannot be efficiently processed by standard GPU architectures. And existing structured pruning methods often apply uniform compression across all model components, failing to account for varying redundancy levels across different layers and blocks.

Fourth, the theoretical understanding of model redundancy remains incomplete. Recent research [3] has revealed that deep layers in transformer architectures exhibit significant redundancy, with the last 25-50% of layers making minimal contributions to final outputs. However, the extent of this redundancy varies across different architectural components (attention mechanisms versus feed-forward networks) and across different layers. Existing methods lack fine-grained adaptivity to exploit these varying redundancy patterns optimally.

Finally, there is a practical implementation gap. Many proposed compression methods are either too computationally expensive for one-shot application (requiring iterative fine-tuning), too architecture-specific (failing to generalize across model families), or too coarse-grained (applying uniform compression ratios without considering block-level redundancy variations). This gap between theoretical compression potential and practical deployment feasibility motivates the need for a new approach that combines structural efficiency with adaptive, block-level compression strategies.

The present research addresses these interconnected challenges by developing a structured pruning method that provides: (1) real hardware acceleration through physical removal of model components, (2) adaptive compression ratios based on redundancy assessment at the block level, (3) one-shot pruning capability without expensive iterative fine-tuning, and (4) generalization across different model sizes and families.

Related work

Neural network pruning began in the late 1980s with theoretical methods like “Optimal Brain Damage” (OBD) [4] and “Optimal Brain Surgeon” (OBS) [5], which used the Hessian matrix to assess and remove the least important weights for improved generalization. Although foundational, these Hessian-based techniques are computationally impractical for modern, billion-parameter models due to the quadratic complexity of Hessian calculation.

Two theoretical frameworks provide justification for why aggressive pruning is possible. The "Lottery Ticket Hypothesis" [7] states that within a large, randomly initialized network exists a "winning lottery ticket" – a small subnetwork that, when trained with the same initialization, achieves comparable performance to the full network. This suggests most parameters are redundant for the result. Formally, for a dense network with random initialization θ_0, there exists a subnetwork with mask m such that after training with initialization θ_0, it achieves accuracy comparable to the full network.

Complementarily, research on layer redundancy [3] in transformer architectures reveals that in very deep LLMs, the final 25-50% of layers make minimal contributions to output. Experiments using cosine similarity between consecutive layer hidden states show that deeper layers perform primarily "refinement" function, with main semantic processing occurring in early and middle layers. This architectural characteristic makes depth pruning particularly promising for transformer-based LLMs.

Modern pruning methods can be classified along two dimensions: granularity (what is removed) and mechanism (how the process occurs).

By granularity: unstructured pruning removes individual weights regardless of position. This approach offers maximum flexibility and can achieve high compression ratios (50%+ sparsity) with minimal accuracy loss [8]. However, it suffers from a limitation: the resulting sparse matrices have chaotic zero patterns that standard GPU hardware cannot efficiently exploit. Without specialized sparse matrix multiplication libraries or/and custom hardware, unstructured pruning provides no real speedup despite reducing weights count [9].

Structured pruning addresses hardware efficiency by removing entire structural units, it can operate along two axes: depth pruning (removing entire layers or blocks) and width pruning (removing neurons, channels, or attention heads). The resulting models are standard dense architectures, just smaller, requiring no specialized software for efficient execution. In Transformer architectures [10], the primary targets are Attention blocks and especially FFN blocks, since FFN blocks contain approximately 60% of total model parameters.

By mechanism: iterative pruning gradually reduces model size through cycles of pruning and fine-tuning [5]. While theoretically good for quality preservation, this approach is impractical for LLMs due to prohibitive computational costs – even a single fine-tuning epoch for a 175B parameter model requires enormous resources.

One-shot pruning removes all target weights in a single pass without subsequent fine-tuning, making it the only viable approach for very large models [8, 11]. The challenge lies in accurately identifying which parameters to remove without iterative feedback from the loss function.

Several recent methods have pioneered one-shot pruning for large language models:

SparseGPT [8] formulates layer-wise pruning as a reconstruction problem: for each layer l, it finds a sparse weight matrix W_l' that minimizes output difference from the original W_l on a calibration dataset. The method approximates the inverse Hessian using efficient block-wise computation. SparseGPT achieves 50% sparsity with minimal quality loss but remains unstructured, limiting practical speedups.

Wanda [11] introduces a simpler criterion that combines weight magnitude with input activation norms:

$$S_{ij} = |w_{ij}| \cdot \left||X_j|\right|_2 \quad (1)$$

where $|w_{ij}|$ – single weight magnitude, $\left||X_j|\right|_2$ – L_2norm of input activations.

This captures the intuition that a small weight may be critical if it multiplies large activations. Despite its simplicity, Wanda achieves competitive results with SparseGPT while being computationally much lighter. But the problem of unstructured pruning is still present.

SliceGPT [12] pioneered structured pruning for LLMs by leveraging computational

invariance in transformers that use RMSNorm. The method uses Principal Component Analysis (PCA) to compute orthogonal transformation matrices Q, then exploits the following property to later remove dimensions corresponding to smallest principal components:

$$RMSNorm(X_l Q)Q^T = RMSNorm(X_l) \quad (2)$$

where X_l – RMSNorm input, Q – orthogonal transformation matrix for this RMSNorm.

This physically reduces matrix sizes, ensuring real speedups. However, SliceGPT has limitations: it requires large calibration sets (1024+ samples), shows quality degradation for smaller models, and increases memory usage at low pruning ratios due to added Q matrices.

Dynamic LLM Slicing [13] extends SliceGPT with adaptive layer-wise compression. It introduces Layer Redundancy (LR) metric (cosine similarity):

$$LR(L_i) = \frac{(X_{in} \cdot X_{out})}{||X_{in}||||X_{out}||} \quad (3)$$

where L_i – i-th layer, X_{in} – layer input, X_{out} – layer output.

Layers with higher LR values (closer to 1) exhibit greater redundancy and can be compressed more aggressively. While this represents progress toward adaptivity, the method operates at layer granularity, missing redundancy variations between attention and FFN blocks within layers.

2SSP [14] introduced a two-stage structured approach combining width and depth pruning. Stage 1 prunes FFN neurons based on L_2-norm of activations. Stage 2 iteratively removes attention modules based on perplexity change causing minimal perplexity increase. The number of attention modules to remove follows:

$$N^{Attn} = round\left(B \cdot s^{\frac{|W_{FFN}|}{\alpha|W_{Attn}|}}\right) \quad (4)$$

where B – total transformer blocks, s – target global sparsity, $|W_{FFN}|$ and $|W_{Attn}|$ – parameter counts in FFN and attention respectively, α – 1.5 (regulation hyperparameter).

The method works efficiently with small calibration sets (32 samples) but lacks adaptivity across layers.

The Purpose of Research

The purpose of this research is to enhance deploying large language models on resource-constrained devices by developing an adaptive structured pruning method.

To achieve this, we need to achieve the following objectives:

1. Develop a pruning method that combines two-stage structured pruning (attention block removal followed by FFN neuron pruning) with adaptive, block-level compression ratios based on redundancy assessment.

2. Evaluate the proposed method across multiple model sizes and families using standardized benchmarks for both quality (perplexity, downstream task accuracy) and efficiency (memory consumption, inference speed).

Proposed Method: Adaptive 2SSP

The method consists of three sequential stages implementing reverse-order two-stage structured pruning with block-level adaptivity.

Stage 1: Attention Block Pruning (Depth Pruning). Attention blocks are analyzed first due to research showing [15] that it is possible to remove entire blocks with minimal impact.

Step 1.1 – Redundancy Assessment: For each attention block in layer L_i, collect input x_j and output y_j (after residual connection) on the calibration set. Redundancy is computed as:

$$R(L_i) = \frac{1}{N}\sum_{j=1}^{N} \mathrm{cossim}(\mathrm{x_j}, \mathrm{y_j}) = \frac{1}{N}\sum_{j=1}^{N} \frac{\mathrm{x_j} \cdot \mathrm{y_j}}{\left||\mathrm{x_j}|\right|\left||\mathrm{y_j}|\right|} \quad (5)$$

Higher $R(L_i)$ values (approaching 1) indicate the block makes minimal transformations, marking it as redundant.

Step 1.2 – Pruning Target Calculation: The number of attention blocks to remove follows the empirical formula from 2SSP [14].

Step 1.3 – Block Removal: Select N^{Attn} blocks with highest redundancy scores and physically remove them from the model architecture.

Stage 2: FFN Layer Redundancy Assessment and Adaptive Compression Ratio Calculation. After attention pruning, the model's structure has changed, necessitating fresh redundancy computation for FFN blocks and assigning layer-specific pruning ratios based on measured redundancy.

Step 2.1 – Data Pass: Run calibration data through the pruned model (post–attention removal).

Step 2.2 – FFN Block Redundancy: For each FFN block, compute redundancy using the same cosine similarity metric.

Step 2.3 – Normalization: Normalize redundancy scores to [0,1] range for subsequent ratio calculation.

Step 2.4 – Keep Ratio Function: For each FFN layer, calculate the fraction of neurons to preserve [13]:

$$Keep_{ratio(FFN_i)} = 1 - \left[R(FFN_i) \cdot \left(\frac{target_{rate}}{mean(R)}\right)\right] \quad (6)$$

where $target_{rate}$ – desired overall FFN pruning rate, $mean(R)$ – average redundancy across all FFN blocks.

This formula ensures: (1) high-redundancy layers receive aggressive compression, (2) low-redundancy layers preserve more capacity, (3) overall compression approximately matches the target rate.

Stage 3: FFN Neuron Pruning (Width Pruning).

Step 3.1 – Neuron Importance: For each neuron j in each FFN layer, compute importance as average activation magnitude on calibration data:

$$s_j = \frac{1}{N}\sum_{c=1}^{N}\left|\left|z_c^j\right|\right|_2 \quad (7)$$

where z_c^j – activation of neuron j for sample c.

Higher magnitude indicates greater importance.

Step 3.2 – Selective Removal: For each FFN layer, rank neurons by importance and remove the bottom fraction. Removal is implemented by deleting corresponding rows in W_{up} and W_{gate} matrices and corresponding columns in W_{down} matrix.

This physically reduces matrix dimensions, ensuring real memory savings and speedup.

Evaluation Metrics

To assess the quality of the pruned models, we evaluate perplexity (PPL) on WikiText-2[17], which measures the model's probabilistic accuracy (how confident the model is in the next token), where lower is better. We also evaluate the model's accuracy on five downstream benchmarks (MMLU, Hellaswag, ARC-E, BoolQ, MathQA) [18-22], which test preservation of reasoning, knowledge, and language understanding capabilities. An average across the five tasks is also reported.

For efficiency evaluation, we use memory consumption, which tracks VRAM usage in GB, and inference speed, which measures tokens per second (TPS) during generation. The proposed method is compared against Wanda [11], a state-of-the-art unstructured pruning approach; GLU Aware Pruning [16], which uses simple magnitude-based structured pruning; Dynamic Slicing [13], an adaptive structured pruning method at the layer level; and 2SSP [14], the original two-stage structured pruning without adaptivity.

Experiments were conducted on the CPU – AMD Ryzen 5 7500F, 3.7 GHz; GPU – NVIDIA GeForce RTX 4070 SUPER, 12GB VRAM and RAM 32 GB DDR5.

Experimental Results

Before full comparative evaluation, we validated the two key modifications introduced in Adaptive 2SSP: reverse pruning order and block-level redundancy assessment.

Reverse Pruning Order Validation: Table 1 compares reverse order (Attention→FFN) versus standard 2SSP order (FFN→Attention)

on Llama-3.2-3B (orders were tested on our adaptive modification not on original 2SSP).

Table 1. Pruning Order Comparison on Llama-3.2-3B

Pruning % and method	AVG Benchmark Acc	Wikitext2 (PPL)
20% reverse	48,66%	12,81
20% standard	47,77%	12,5
30% reverse	44,50%	17,99
30% standard	43,71%	17,51
40% reverse	39,57%	26,34
40% standard	38,45%	25,69

The reverse order consistently achieves higher benchmark accuracy with slightly worse perplexity. This validates the hypothesis that removing coarse-grained structures (attention blocks) first allows subsequent fine-grained adjustments (FFN neurons) to better compensate.

Table 2 compares block-level redundancy assessment versus layer-level (redundancy assessments were tested on our adaptive modification not on original 2SSP).

Table 2. Redundancy Granularity Comparison on Llama-3.2-3B

Pruning % and method	AVG Benchmark Acc	Wikitext2 (PPL)
10% blocks	52,95%	9,58
10% layers	52,45%	9,58
20% blocks	48,66%	12,81
20% layers	47,95%	12,88
30% blocks	44,50%	17.99
30% layers	41,45%	17,68
40% blocks	39,57%	26,34
40% layers	38,73%	26,27

Block-level redundancy assessment consistently achieves higher benchmark accuracy with slightly worse perplexity in some cases. This confirms that attention and FFN blocks within the same layer indeed have different redundancy patterns, justifying the added granularity. Figures 1 and 2 present heatmaps of FFN blocks and Attention blocks of Llama-3.2-3B model.

Analysis of Llama-3.2-3B blocks redundancy heatmaps revealed that deeper layers exhibit higher redundancy and redundancy could differ significantly between attention and FFN blocks within the same layer. These patterns support the adaptive, block–level approach.

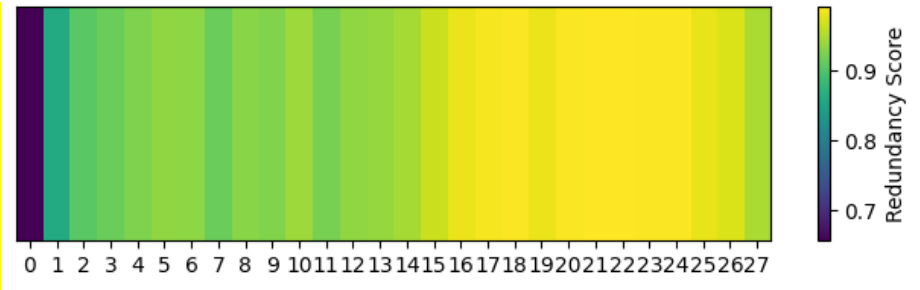

Figure 1. Heatmap of Llama-3.2-3B FFN blocks redundancy

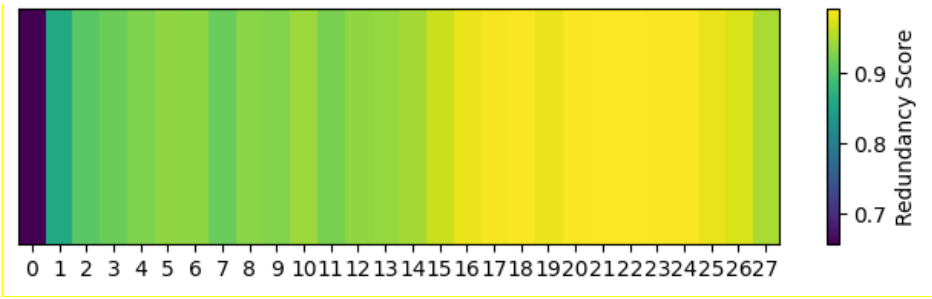

Figure 2. Heatmap of Llama-3.2-3B FFN blocks redundancy

Tables 3-5 present quality results on different benchmarks for Llama-3.2-3B, Llama-2-7B and Qwen2.5-3B across all chosen for comparison methods. And Tables 6-7 present efficiency gains by using proposed method on Llama-3.2-3 and Llama-2-7B models to compare different model sizes.

Table 3. Quality Metrics for Llama-3.2-3B

Pruning %	Model/Method	WikiText2 (PPL)	ARC–Easy (Acc, %)	BoolQ (Acc, %)	HellaSwag (Acc, %)	MMLU (Acc, %)	MathQA (Acc, %)	AVG Acc, %
0%	Llama-3.2-3B (Baseline)	7,81	74,00%	74,00%	48,50%	57,75%	35,80%	58,01%
20%	Wanda	8,28	71,10%	74,20%	48,50%	55,75%	35,10%	56,93%
10%	GLU Aware Pruning	11,12	66,50%	48,90%	44,80%	40,50%	33,00%	46,74%
	Dynamic slicing	15,88	63,00%	68,50%	42,70%	41,75%	25,40%	48,27%
	2SSP	9,64	68,30%	65,60%	47,00%	50,00%	30,40%	52,26%
	Adaptive 2SSP	9,58	67,80%	68,60%	47,00%	51,25%	30,10%	52,95%
20%	GLU Aware Pruning	17,14	52,80%	42,00%	41,80%	30,50%	28,60%	39,14%
	Dynamic slicing	29,93	55,30%	66,50%	39,60%	29,50%	23,50%	42,88%
	2SSP	13,46	60,60%	59,50%	42,50%	37,25%	27,00%	45,37%
	Adaptive 2SSP	12,81	62,10%	61,30%	45,10%	46,00%	28,80%	48,66%
30%	GLU Aware Pruning	34,9	47,40%	40,90%	35,90%	28,00%	23,60%	35,16%
	Dynamic slicing	58,33	46,40%	60,70%	36,50%	24,75%	23,80%	38,43%
	2SSP	19,35	52,40%	56,50%	38,10%	28,75%	26,90%	40,53%
	Adaptive 2SSP	18	54,40%	60,70%	42,50%	38,00%	26,90%	44,50%
40%	GLU Aware Pruning	73,67	36,30%	52,30%	33,10%	25,50%	24,30%	34,30%
	Dynamic slicing	99,94	37,70%	60,10%	33,80%	23,50%	21,90%	35,40%
	2SSP	28,38	43,30%	61,90%	35,90%	23,25%	24,20%	37,71%
	Adaptive 2SSP	26,35	45,20%	63,50%	37,50%	26,25%	25,40%	39,57%

Table 4. Quality Metrics for Llama-2-7B

Pruning %	Model/Method	WikiText2 (PPL)	ARC–Easy (Acc, %)	BoolQ (Acc, %)	HellaSwag (Acc, %)	MMLU (Acc, %)	MathQA (Acc, %)	AVG Acc, %
0%	Llama-2-7B (Baseline)	5,47	76,10%	76,40%	49,80%	40,25%	28,00%	54,11%
10%	GLU Aware Pruning	6,6	71,10%	70,60%	46,30%	27,75%	29,20%	48,99%
	Dynamic slicing	7,35	71,70%	73,40%	49,20%	29,50%	27,20%	50,20%
	2SSP	6,36	73,50%	73,30%	48,90%	31,50%	26,50%	50,74%
	Adaptive 2SSP	6,52	73,60%	73,60%	49,30%	35,50%	28,10%	52,02%
20%	GLU Aware Pruning	9,56	62,00%	64,00%	42,80%	23,50%	26,00%	43,66%
	Dynamic slicing	11,21	63,10%	66,10%	46,10%	24,75%	27,00%	45,41%
	2SSP	7,9	67,70%	67,50%	46,40%	26,00%	24,00%	46,32%
	Adaptive 2SSP	8,01	70,90%	74,20%	48,00%	29,50%	25,30%	49,58%
30%	GLU Aware Pruning	23,99	46,40%	49,70%	35,30%	25,25%	23,50%	36,03%
	Dynamic slicing	19,56	53,70%	65,40%	42,00%	23,75%	23,90%	41,75%
	2SSP	11,12	61,80%	63,80%	45,10%	24,50%	25,10%	44,06%
	Adaptive 2SSP	12	63,30%	69,40%	46,00%	24,00%	25,80%	45,70%
40%	GLU Aware Pruning	89,94	34,30%	45,80%	32,50%	26,25%	24,00%	32,57%
	Dynamic slicing	35,78	41,90%	63,40%	39,00%	23,75%	22,50%	38,11%
	2SSP	17,38	49,70%	63,80%	41,50%	24,75%	24,70%	40,89%
	Adaptive 2SSP	20,65	50,60%	65,30%	41,70%	24,50%	25,40%	41,50%

Table 5. Quality Metrics for Qwen2.5-3B

Pruning %	Model/Method	WikiText2 (PPL)	ARC–Easy (Acc, %)	BoolQ (Acc, %)	HellaSwag (Acc, %)	MMLU (Acc, %)	MathQA (Acc, %)	AVG Acc, %
0%	Qwen2.5–3B (Baseline)	8,03	76,70%	76,20%	47,50%	68,50%	37,90%	61,36%
10%	GLU Aware Pruning	13,64	75,00%	59,70%	47,50%	62,25%	36,90%	56,27%
	Dynamic slicing	9,92	74,80%	77,10%	46,20%	49,25%	30,00%	55,47%
	2SSP	9,51	72,80%	72,30%	46,90%	61,75%	33,10%	57,37%
	Adaptive 2SSP	9,21	70,60%	70,50%	47,30%	61,50%	32,80%	56,54%
20%	GLU Aware Pruning	20,02	66,40%	69,80%	42,60%	53,75%	35,10%	53,53%
	Dynamic slicing	14,52	65,70%	72,60%	42,10%	24,25%	25,00%	45,93%
	2SSP	11,93	65,00%	71,50%	44,20%	51,00%	30,40%	52,42%
	Adaptive 2SSP	11,49	65,60%	68,90%	44,00%	48,50%	29,60%	51,32%
30%	GLU Aware Pruning	37,11	56,00%	51,40%	39,80%	42,75%	30,10%	44,01%
	Dynamic slicing	24,89	52,80%	64,70%	37,10%	23,50%	23,70%	40,36%
	2SSP	16,17	56,50%	64,70%	41,80%	26,50%	24,80%	42,86%
	Adaptive 2SSP	14,73	58,30%	64,10%	41,70%	28,25%	26,70%	43,81%
40%	GLU Aware Pruning	76,68	46,80%	51,70%	33,60%	27,25%	24,60%	36,79%
	Dynamic slicing	39,23	42,30%	63,50%	34,40%	23,50%	23,20%	37,38%
	2SSP	22,1	48,80%	62,50%	38,60%	23,00%	26,10%	39,80%
	Adaptive 2SSP	19,56	50,80%	63,50%	38,60%	23,50%	25,40%	40,36%

Table 6. Llama-3.2-3B Hardware Efficiency

Pruning %	VRAM Usage (GB)	Memory usage reduction	Throughput (TPS)	Throughput increase
0%	5,98	–	92	–
10%	5,45	8,86%	99	7,61%
20%	4,93	17,56%	108	17,39%
30%	4,4	26,42%	118	28,26%
40%	3,88	35,12%	124	34,78%

Table 7. Llama-2-7B Hardware Efficiency

Pruning %	VRAM Usage (GB)	Memory usage reduction	Throughput (TPS)	Throughput increase
0%	12,5	–	19	–
10%	11,3	9,60%	27	42,11%
20%	10,1	19,20%	71	273,68%
30%	8,93	28,56%	79	315,79%
40%	7,72	38,24%	89	368,42%

The results obtained for the Llama-3.2-3B model indicate that the Adaptive 2SSP method provides the optimal balance between preserving text generation quality and maintaining logical reasoning capabilities, particularly at high degrees of compression. To establish a baseline for quality preservation, the unstructured Wanda method was evaluated at 20% pruning. While Wanda maintained performance nearly on par with the original model, achieving an average accuracy of 56.93% versus the baseline's 58.01% with only a small rise in perplexity from 7.81 to 8.28, but since it's structured pruning, it failed to deliver any improvements in inference speed or memory consumption. Among structured pruning methods at moderate compression (20%), Adaptive 2SSP demonstrated superior performance with an average accuracy of 48.66%. This significantly outperformed the standard 2SSP (45.37%) and Dynamic Slicing (42.88%), as well as GLU Aware Pruning, which recorded the lowest accuracy at 39.14%.

The divergence between methods becomes most visible in the high-compression range of 30% to 40%. Both Dynamic Slicing and GLU from catastrophic degradation, at 40% pruning, their perplexity values explode to 99.94 and 73.67 respectively, rendering the generated text unpredictable. In contrast, the proposed Adaptive 2SSP method along with original 2SSP manages to maintain perplexity. Even at 40% compression, it has perplexity of 26.35, while its average accuracy of 39.57% remains the highest among all competing structured methods.

These trends persist when method applied to the larger Llama-2-7B model. While the original 2SSP method retains a slight advantage in raw perplexity values for this specific architecture, Adaptive 2SSP secures superior accuracy on downstream benchmarks. This performance consistency across different model sizes suggests that the proposed method maintains algorithmic stability regardless of architectural scale.

The Qwen2.5-3B model, however, exhibits behavior distinct from the other architectures tested. Notably, the GLU Aware method presented an anomaly at 20% pruning: it achieved a relatively high accuracy of 53.53%, surpassing Adaptive 2SSP's 51.32%, but this was accompanied by a severe degradation in perplexity (20.02 compared to 11.49 for the proposed method). This discrepancy implies that while the GLU Aware model may successfully "guess" answers for the given tasks, it loses the capacity to generate coherent, connected text. At the same time, Adaptive 2SSP demonstrates the smoothest degradation curve, avoiding sharp spikes in perplexity and achieving the lowest comparative value of 19.56 at 40% pruning. Consequently, for the Qwen2.5 architecture, Adaptive 2SSP represents the most balanced choice, as it prioritizes the text coherence

essential for real-world scenarios, such as chatbots, unlike the behavior observed with GLU Aware.

So, Adaptive 2SSP demonstrates the most balanced preservation of model quality among the compared alternatives. The method successfully prevents model collapse even when removing 40% of parameters, a threshold where Dynamic Slicing and GLU Aware fail, while consistently outperforming the original 2SSP approach.

Conclusion

This research developed and validated modified version of 2SSP pruning method – Adaptive 2SSP, a structured pruning method for large language models that combines two-stage architectural pruning with block-level adaptive compression ratios to achieve state-of-the-art compression-quality trade-offs and real hardware acceleration. Methodologically, the approach modifies existing pruning method by introducing block-level redundancy assessment using cosine similarity metrics, which enables differentiated compression between attention and FFN components. This is further enhanced by a reverse pruning order that prunes attention blocks before FFN neurons to leverage the structural complementarity between coarse and fine-grained pruning.

Comprehensive experiments across Llama-3.2-3B, Llama-2-7B, and Qwen2.5-3B demonstrate the method's consistent superiority over existing techniques. For example, at 40% compression on Llama-3.2-3B, Adaptive 2SSP achieved a perplexity of 26.35 and an average accuracy of 39.57%, outperforming all other structured pruning approaches. These algorithmic gains translate directly into practical deployment value, with hardware efficiency measurements confirming a 35.12% memory reduction and 34.78% speedup for Llama-3.2-3B at 40% pruning. Also, for Llama-2-7B on 12 GB VRAM, a 20% pruning rate enabled full on-device loading, providing a massive 273.68% speedup by eliminating RAM offloading bottlenecks.

Despite promising results, several limitations suggest directions for future research. There is room for refinement in hyperparameter optimization; since the Attention-FFN ratio parameter α was adopted directly from 2SSP, researching other ways for ratio calculation could potentially enhance results. Also, study's scope regarding architectural generalization also presents opportunities for expansion. As testing was limited to decoder-only transformer variants, extending validation to encoder-decoder or encoder models, as well as mixture-of-experts architectures is necessary. Furthermore, because experiments were restricted to models with up to 7 billion parameters due to hardware constraints, validation on very large-scale models exceeding 70 billion parameters would significantly strengthen generalization claims.

Finally, combining structured pruning with quantization could yield compound compression benefits, though the interaction effects between these techniques require careful study.

References

1. Wei, J., et al. (2022). Emergent abilities of large language models. arXiv preprint arXiv:2206.07682.
2. Patterson, D., et al. (2021). Carbon emissions and large neural network training. arXiv preprint arXiv:2104.10350.
3. Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., & Roberts, D.A. (2024). The unreasonable ineffectiveness of the deeper layers. arXiv preprint arXiv:2403.17887.
4. LeCun, Y., Denker, J.S., & Solla, S.A. (1989). Optimal brain damage. Neural Information Processing Systems, 2, 598–605.
5. Hassibi, B., Stork, D.G., & Wolff, G.J. (1993). Optimal Brain Surgeon and general network pruning. IEEE International Conference on Neural Networks, 293–299.
6. Han, S., Pool, J., Tran, J., & Dally, W.J. (2015). Learning both Weights and Connections for Efficient Neural Networks. Advances in neural information processing systems, 28.
7. Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635.
8. Frantar, E., & Alistarh, D. (2023). SparseGPT: Massive language models can be accurately pruned in One–Shot. arXiv preprint arXiv:2301.00774.
9. Siddiqui, M. H. (2024, March 22). Difference Between Structured and Unstructured Pruning in Neural.

Medium.https://medium.com/@mhammadsiddiqui/difference-between-structured-and-unstructured-pruning-in-neural-cca5603581fb.

10. Vaswani, A., et al. (2017). Attention is all you need. Advances in neural information processing systems, 30.
11. Sun, M., Liu, Z., Bair, A., & Kolter, J.Z. (2023). A simple and effective pruning approach for large language models. arXiv preprint arXiv:2306.11695.
12. Ashkboos, S., et al. (2024). SliceGPT: Compress large language models by deleting rows and columns. arXiv preprint arXiv:2401.15024.
13. Dumitru, R.–G., Clotan, P.–I., Yadav, V., Peteleaza, D., & Surdeanu, M. (2024). Change Is the Only Constant: Dynamic LLM Slicing based on Layer Redundancy. arXiv preprint arXiv:2411.03513.
14. Sandri, F., Cunegatti, E., & Iacca, G. (2025). 2SSP: A Two–Stage Framework for Structured Pruning of LLMs. arXiv preprint arXiv:2501.17771.
15. He, S., Sun, G., Shen, Z., & Li, A. (2024). What Matters in Transformers? Not All Attention is Needed. arXiv preprint arXiv:2406.15786.
16. Making LLMs Smaller Without Breaking Them: A GLU–Aware Pruning Approach. (2024). Huggingface.co. https://huggingface.co/blog/oopere/making–llms–smaller–without–breaking–them
17. Jelinek, F., Mercer, R.L., Bahl, L.R., & Baker, J.K. (1977). Perplexity –a measure of the difficulty of speech recognition tasks. The Journal of the Acoustical Society of America, 62(S1), S63.
18. Hendrycks, D., et al. (2020). Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300.
19. Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). HellaSwag: Can a machine really finish your sentence? arXiv preprint arXiv:1905.07830.
20. Clark, P., et al. (2018). Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. arXiv preprint arXiv:1803.05457.
21. Clark, C., Lee, K., Chang, M.–W., Kwiatkowski, T., Collins, M., & Toutanova, K. (2019). BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. arXiv preprint arXiv:1905.10044.
22. Amini, A., Gabriel, S., Lin, P., Koncel–Kedziorski, R., Choi, Y., & Hajishirzi, H. (2019). MathQA: Towards Interpretable Math Word Problem Solving with Operation–Based Formalisms. arXiv preprint arXiv:1905.13319.

The article has been sent to the editors 15.12.25.
After processing 20.12.25.
Submitted for printing 30.12.25.