ADAPT-PRUNER: ADAPTIVE AND STRUCTURED PRUNING FOR EFFICIENT LARGE LANGUAGE MODELS

A PREPRINT

Boyao Wang * **Rui Pan** * **Tong Zhang**
University of Illinois Urbana-Champaign
boyaow2@illinois.edu, rpan2@illinois.edu, tozhang@illinois.edu

ABSTRACT

Large language models have demonstrated exceptional performance in recent years. While wider and deeper network architectures enable these models to learn more comprehensive knowledge, they also pose challenges for deployment on edge devices. Compressing models not only reduces storage requirements, making edge deployment feasible, but also accelerates inference, thereby reducing latency and computational costs. Structured pruning, which removes filters at the architectural level, offers a way to achieve a more compact model while maintaining target accuracy, ensuring compatibility and hardware efficiency. Our method is based on the observation that different decoder layers in LLMs contribute unequally to model performance. We introduce Adapt-Pruner, which leverages this insight by measuring the distance between the input and output tensors of a decoder layer to assign appropriate sparsity. Layers more sensitive to pruning are assigned lower sparsity. We conducted experiments on three cutting-edge open-source models—LLaMA-3.1-8B, Qwen2.5-7B, and Gemma-2-9B. Experiment results show that Adapt-Pruner retains 90.9% of LLaMA-3.1-8B's average benchmark scores at 25% sparsity without post-training. We also compare our approach to other state-of-the-art structured pruning methods to verify its effectiveness.

1 Introduction

Large language models (LLMs) [Kalyan, 2024, OpenAI, 2023] have demonstrated remarkable performance across a wide range of benchmarks. As their size increases, these models exhibit enhanced capabilities in understanding natural language and solving complex tasks through text generation [Zhao et al., 2023]. However, achieving such performance requires models with billions of parameters, which presents significant challenges for practical deployment. The sheer scale of LLMs leads to high computational costs, making inference both resource-intensive and slow, and potentially introducing issues such as increased latency. Consequently, there is a growing demand for methods to compress LLMs [Zhu et al., 2024], aiming to reduce the number of parameters and improve inference speed, all while preserving the original model performance. Effective compression techniques hold the potential to create more efficient and deployable LLMs.

Several techniques have been proposed to compress LLMs, most of which fall into one of four categories: structured and unstructured pruning [Cheng et al., 2024], quantization [Gholami et al., 2022], low-rank factorization [Sainath et al., 2013], and knowledge distillation [Gou et al., 2021]. In this paper, we primarily focus on structured pruning, though we hope the insights from our methods will also inform future work in other categories. Structured pruning removes entire filters from neural networks, enabling both compression and realistic acceleration on standard hardware. Unlike unstructured pruning, it does not require specialized hardware or library support to achieve these benefits [He and Xiao, 2023].

While many works on structured pruning focus on removing a fixed number of filters from weight matrices with minimal performance degradation, these methods often either skip important layers or apply uniform sparsity across all layers. However, as shown in Figure 1, the importance of each decoder layer—and by extension, each weight matrix—varies significantly. To address this, we introduce a novel approach called Adapt-Pruner. Unlike traditional pruning methods

*Equal contribution.

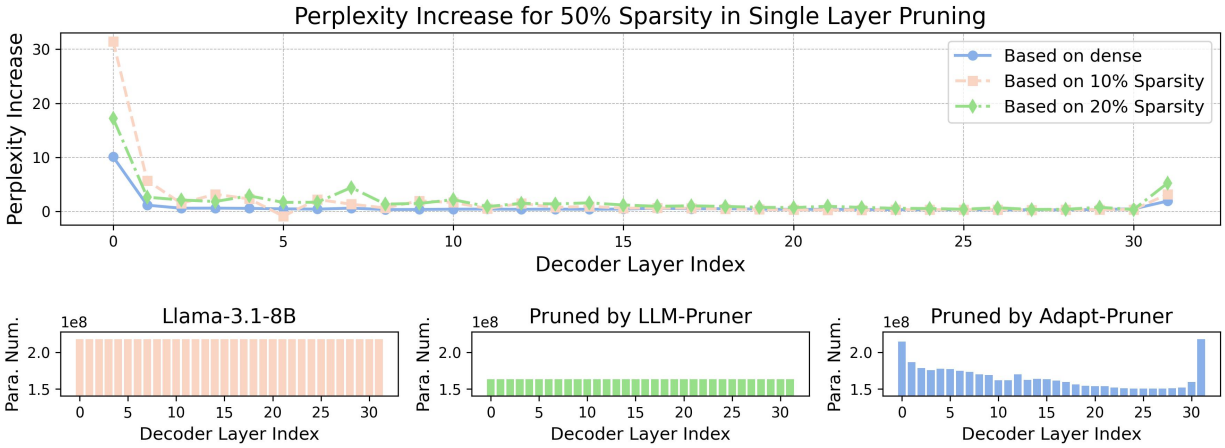

Figure 1: Layer sensitivity and pruned Models. The first row of figures shows the increase in perplexity when a single decoder layer is pruned at 50% sparsity, compared to the dense LLaMA-3.1-8B model, as well as models uniformly pruned across all layers at 10% and 20% sparsity. The second row of figures illustrates the architecture of the pruned models, with each decoder layer represented by its corresponding number of parameters.

that enforce the same sparsity across all decoder layers, Adapt-Pruner operates in multiple steps. At each step, it calculates the relative importance of each decoder layer and applies varying sparsity levels, assigning higher sparsity to less important layers and lower sparsity to more critical ones. After determining the sparsity for each layer, we assess the importance of each weight group using both magnitude and first-order information, selecting the least important groups for pruning. When computational resources allow, post-training stages are periodically applied after several pruning steps to recover any performance loss resulting from the pruning process.

Our main contributions are as follows:

1. We propose a novel structured pruning method, Adapt-Pruner, which quantitatively estimates the importance of each decoder layer and prunes adaptively.
2. This method leverages the insight that the importance of each decoder layer varies, and that their relative importance may shift throughout the pruning process. This adaptiveness leads to higher performance, particularly at high pruning ratios.
3. We conduct extensive experiments on LLaMA-3.1 [AI@Meta, 2024], Qwen2.5 [Team, 2024a], and Gemma [Team, 2024b], demonstrating that Adapt-Pruner can compress these models by up to 25%, while retaining 90.6% of the average benchmark scores of the dense model, without post-training, and outperforming state-of-the-art methods.

2 Related Work

Pruning Pruning removes weights and modifies the model's architecture. Formally, given a neural network $f(x; W)$, pruning produces a new model $f(x; M \odot W)$, where $M \in \{0, 1\}^{|W|}$ is a binary mask that sets certain parameters to zero, and $\odot$ denotes element-wise multiplication. Pruning typically reduces the network's performance, so post-training is often employed to recover this loss [Blalock et al., 2020]. Unstructured pruning removes individual weights, resulting in sparsified weight matrices, but this does not lead to inference speedups unless specialized hardware is available [Dery et al., 2024]. In contrast, structured pruning operates at a larger granularity, removing entire weight groups. This includes width pruning [Ma et al., 2023, Ashkboos et al., 2024], which removes groups of coupled weights, and depth pruning [Kim et al., 2024, Siddiqui et al., 2024], which eliminates entire layers. Some approaches also explore pruning during pretraining, achieving impressive results [Xia et al., 2024]. However, given the immense computational cost—equivalent to pretraining a model—this approach is often impractical. Our focus is on post-training structured pruning, balancing generality and hardware efficiency.

Adaptive Compression Several works have explored adaptive compression. [Dong et al., 2024] selects transformer feedforward experts and removes feedforward neurons during inference based on their high activation magnitudes from input prompts. While this method is effective, we seek an approach that can reduce model size without depending on specific input prompts. [An et al., 2023] computes the sample variance of each input feature and weights it by the

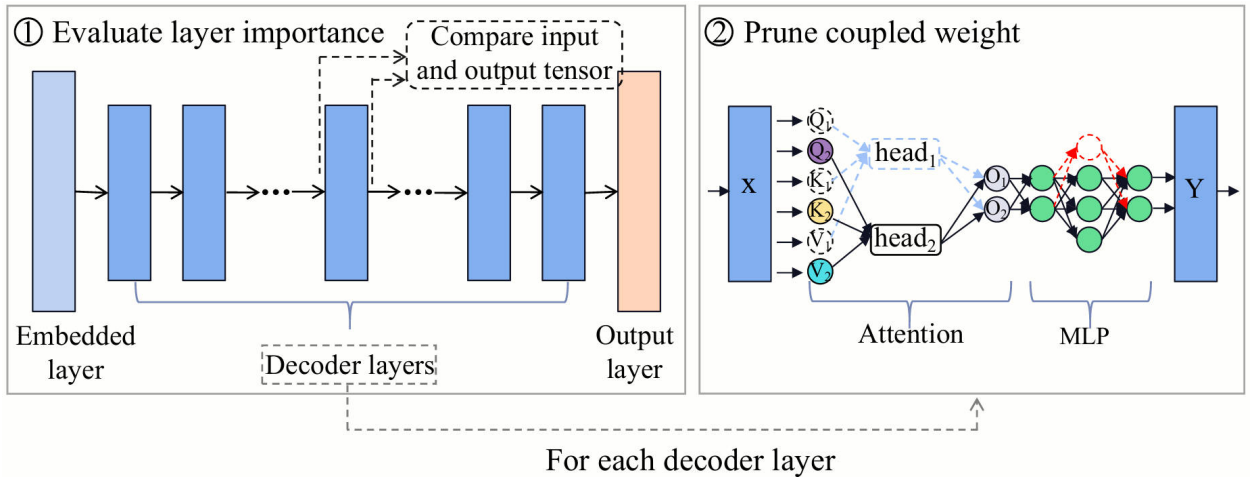

Figure 2: Illustration of Adapt-Pruner. We begin by measuring the distance between each decoder layer's input and output tensors to assess its importance and assign a corresponding sparsity. Based on this assigned sparsity, we then prune the coupled weights in each decoder layer accordingly.

squared norm of the corresponding column in the weight matrix to determine a layer's importance and assign sparsity accordingly. However, compared to our method, which directly measures the distance between a layer's input and output tensors to determine importance, we achieve superior results (see 4).

Knowledge Distillation Knowledge distillation is a technique used to transfer the advanced capabilities of high-performing LLMs to smaller models [Xu et al., 2024]. Combining knowledge distillation with pruning can yield strong performance, where the original model acts as the teacher and the compressed model serves as the student [Sreenivas et al., 2024]. In this paper, we focus on the adaptive pruning algorithm, but it could achieve even better performance when integrated with state-of-the-art knowledge distillation techniques.

3 Method

Given a large language model $\mathcal{M}$, represented as a sequence of embedded layers with $\mathcal{N}$ decoder layers, denoted as $\mathcal{L}^{\mathcal{N}}$, along with a final output layer, our method leverages the insight that each decoder layer contributes differently to the model's overall performance. Adapt-Pruner compresses the model through multiple iterations, with each iteration comprising three steps: (1) quantitatively computing the importance of each decoder layer and assigning a corresponding pruning sparsity; (2) grouping the weights within each decoder layer, evaluating the importance of each coupled structure, and pruning the least important structures based on the assigned sparsity; and (3) periodically applying a post-training stage after a specified number of iterations to recover any performance drop caused by pruning. Figure 2 gives an illustration of our method.

3.1 Assign Sparsity based on Importance

Let $\mathcal{L}^i$ denote the i_{th} decoder layer, $\mathcal{I}^i$ and $\mathcal{S}^i$ represent the importance and sparsity of the i_{th} decoder layer, and $\mathcal{L}^i_{in}$ and $\mathcal{L}^i_{out}$ denote the input and output tensors of the i_{th} decoder layer, respectively. Our goal is to estimate the importance of each decoder layer.

Estimate Decoder Layer's Importance Our pruning method targets only the multi-head attention and multilayer perceptron components within the decoder layers, leaving the hidden size unchanged. Consequently, the input and output tensors for each decoder layer have identical shapes:

$$\forall i = 0, 1, \ldots, N-1, \text{Shape}(\mathcal{L}^i_{in}) = \text{Shape}(\mathcal{L}^i_{out}) = (\mathcal{B}, \mathcal{L}, \mathcal{H}) \tag{1}$$

where $\mathcal{B}, \mathcal{L}, \mathcal{H}$ denote the batch size, sequence length, and hidden size, respectively. Based on this, we use a function that measures the vector similarity or distance between $\mathcal{L}^i_{in}$ and $\mathcal{L}^i_{out}$ to assess the changes in the tensor caused by each decoder layer. The greater the similarity or the smaller the distance between $\mathcal{L}^i_{in}$ and $\mathcal{L}^i_{out}$, the less important that decoder layer is. A practical choice for this distance measurement is cosine similarity. We can compute a decoder layer's importance as follows:

$$\forall i = 0, 1, \ldots, N-1, \mathcal{I}^i = -\text{cosine\_similarity}(\mathcal{L}^i_{in}, \mathcal{L}^i_{out}) \tag{2}$$

Algorithm 1 Adaptive Pruning Algorithm

Require: Number of decoder layer in the LLM N, decoder layer instances in the LLM $\{\mathcal{L}^i\}_{i=1}^N$, overall sparsity after pruning $\mathcal{S}$, iteration times to prune T, iteration period to post-train P, amplitude of sparsity between decoder layers A, similarity function, use cos in default $Sim_{func} \leftarrow$ cosine_similarity()

for $i \leftarrow 1 \dots T$ **do**
  $S_{cur} \leftarrow (S * i/T)$
  $I \leftarrow \{0\}^N$, $S \leftarrow \{0\}^N$
  for $j \leftarrow 1 \dots N$ **do**
    $I_j \leftarrow Sim_{func}(\mathcal{L}_{in}^j, \mathcal{L}_{out}^j)$
  end for
  Normalize I^N to have 0 mean value, limit range to [-1, 1] and times -1 if lower is better
  for $j \leftarrow 1 \dots N$ **do**
    $S_j \leftarrow S_{cur} - A * I_j$
  end for
  Prune the LLM based on the sparsity
  if $i \mod P = 0$ **then**
    Post-train LLM
  end if
end for

This method can easily be extended to alternative similarity or distance functions, such as Euclidean or Manhattan distance. To ensure consistency, we normalize the decoder layer's importance to the range $[-1, 1]$ with a mean of 0, as follows:

$$\mathcal{I}^i = \mathcal{I}^i - \mathcal{I}_{mean} \tag{3}$$

$$\mathcal{I}^i = \frac{\mathcal{I}^i}{\max|\text{abs}(\mathcal{I})|} \tag{4}$$

Assign Sparsity Now that we have computed the importance of each layer, we need a method to link a layer's importance to its sparsity. We adopt an empirical approach to address this. Let A denote the amplitude of sparsity, such that we have:

$$\forall i = 0, 1, \dots, N-1, \mathcal{S}^i = \mathcal{S}_{base} - A * \mathcal{I}^i \tag{5}$$

where $\mathcal{S}_{base}$ is the target overall sparsity of the model. This method ensures that each decoder layer's sparsity is inversely proportional to its importance while maintaining the average sparsity consistent with the intended overall model sparsity.

Based on the observations from 1, the importance of each layer varies throughout the pruning process. This variation necessitates breaking the pruning into multiple iterative steps, allowing each layer's sparsity to be adjusted progressively, leading to improved results.

3.2 Pruning Weight Groups Inside Decoder Layer

We use the method from [Ma et al., 2023, Fang et al., 2023] to build dependency graph for LLMs, which could automatically identify and extract coupled structures in LLMs.

Weight Group Importance With grouped structures and target sparsity defined for each group, the next step is selecting weight matrices to prune to minimize performance degradation. For any grouped weight structure $\mathcal{G} = W^k$, containing k weight matrices, we use a calibration dataset $\mathcal{D}$ to assess the relative importance of each matrix. Following [LeCun et al., 1989, Ma et al., 2023], the importance of the i_{th} weight matrix in layer $\mathcal{L}$ is defined as:

$$I_{W_i} = |\Delta\mathcal{L}(\mathcal{D})| = |\mathcal{L}_{W_i}(\mathcal{D}) - \mathcal{L}_{W_i=0}(\mathcal{D})| = |\frac{\partial\mathcal{L}^\top(\mathcal{D})}{\partial W_i}W_i - \frac{1}{2}{W_i}^\top H W_i + \mathcal{O}\left(\|W_i\|^3\right)| \tag{6}$$

where $H = \frac{\partial\mathcal{L}^2(\mathcal{D})}{\partial W_i^2}$ is the Hessian matrix. Calculating the Hessian requires $\mathcal{O}(N^2)$ computational resources, so we omit it to accelerate pruning, as well as the term $\mathcal{O}\left(|W_i|^3\right)$, which is generally small. This simplifies the estimated weight matrix importance to:

$$\hat{I}_{W_i} = |\frac{\partial\mathcal{L}^\top(\mathcal{D})}{\partial W_i}W_i| \tag{7}$$

Thus, we assess each weight matrix's importance by taking the l_1 norm of the element-wise product between its gradient (derived from the calibration dataset) and its weight value. After computing importance scores, we sort the matrices and prune those with the lowest scores to achieve the desired sparsity level.

3.3 Periodic Recovery Post-Training with Knowledge Distillation

To achieve the target overall sparsity, our model undergoes multiple rounds of pruning, which inevitably leads to performance degradation. To mitigate this, we periodically apply post-training after certain pruning steps to restore lost performance. Balancing performance recovery with computational efficiency, we employ a low-rank approximation method [Hu et al., 2021a] during post-training on the compressed model. Since additional pruning follows each post-training phase, the low-rank matrices generated are merged back into the original weight matrices.

Knowledge distillation [Xu et al., 2024] enables the transfer of knowledge from a larger or more advanced teacher model to a smaller, compressed student model. To preserve performance in the compressed model, we use the original model as the teacher and the corresponding compressed model as the student. Specifically, during post-training, we add a penalty term to the loss function to measure the divergence between the probability distributions of the teacher model, p_t, and the student model, p_s, given an input x. This results in a modified loss function:

$$\mathcal{L} = \alpha * \text{Loss}(p_t(x), p_s(x)) + (1 - \alpha) * \text{Loss}(label(x), p_s(x)) \tag{8}$$

where α is a hyperparameter that controls the influence of the teacher model on the student's learning. By leveraging the knowledge of the original model, the compressed model retains more of its performance. Algorithm 1 gives a detailed description of our method.

4 Experiment

4.1 Settings

Setup We extend the LLM-Pruner framework [Ma et al., 2023, Fang et al., 2023] as our baseline, incorporating modules for computing similarity scores and adaptive pruning using PyTorch [Paszke et al., 2019]. Our experiments utilize an NVIDIA A40 GPU with 48GB of memory for post-training and evaluation, while all pruning processes are computed on an Intel(R) Xeon(R) Gold 6346 CPU, which features 16 cores per socket (32 cores total) running at 3.10 GHz.

In our experiments, we employ cosine similarity between the input and output tensors to assess the importance of decoder layers. The calibration dataset we use is bookcorpus [Zhu et al., 2015], with a default of 10 examples, each truncated to a length of 64 tokens.

Models and Datasets To demonstrate the effectiveness of our methods, we evaluate them on three widely-used open-source models: Llama-3.1-8B [AI@Meta, 2024], Qwen2.5-7B [Team, 2024a, Yang et al., 2024], and Gemma-2-9B [Team, 2024b]. For task-agnostic performance evaluation of the pruned models, we perform zero-shot classification on several common-sense reasoning datasets: ARC-easy and ARC-challenge [Clark et al., 2018], BoolQ Clark et al. [2019], HellaSwag [Zellers et al., 2019], OpenBookQA [Mihaylov et al., 2018], PIQA [Bisk et al., 2020], and WinoGrande [Sakaguchi et al., 2021]. Additionally, we supplement our evaluation with a generation task using WikiText2 [Merity et al., 2016]. Following prior work [Ma et al., 2023, An et al., 2023, Ashkboos et al., 2024], we employ the LM Evaluation Harness [Gao et al., 2024] with default parameters, except that all models use the bfloat16 data type, and the batch size is set to 'auto' during evaluation.

Baseline Setup To validate the effectiveness of our adaptive pruning method based on decoder layer similarity, we compare it to the LLM-Pruner framework. Additionally, we benchmark our approach against other state-of-the-art structured pruning methods, including FLAP [An et al., 2023] and SliceGPT [Ashkboos et al., 2024]. We evaluate all methods at three sparsity ratios—25%, 50%, and 75%—without post-training to isolate the impact of the pruning algorithm and avoid any unintended effects from post-training.

4.2 Results

Zero-shot and Generation Tasks We begin by evaluating our methods on three models: LLaMA-3.1-8B, Qwen2.5-7B, and Gemma-2-9B, across 25%, 50%, and 75% pruning sparsity. We skip the first and last three decoder layers for LLaMA and Qwen, and the first and last four layers for Gemma, as these layers tend to be too important for pruning.

Table 1: Zero-shot and generation task performance of the compressed LLaMA-3.1-8B, Qwen2.5-7B, and Gemma-2-9B models. The average performance is calculated across seven classification datasets. For Gemma-2, the "eager" attention implementation is used, as it is better suited for post-training. The evaluation metric is acc_norm for ARC-e, ARC-c, HellaSwag, OBQA, and PIQA; acc for BoolQ and WinoGrande; and $word\_perplexity$ for WikiText2.

Pruning Sparsity	Model	Para. Num.	ARC-e	ARC-c	BoolQ	HellaSwag	OBQA	PIQA	Winogrande	Average	WikiText2↓
Dense	LLaMA-3.1-8B	8.03B	81.19	53.33	82.08	78.89	44.80	81.12	73.64	70.21	7.33
	Qwen2.5-7B	7.62B	77.36	51.02	84.68	78.93	47.00	79.76	72.93	70.24	8.74
	Gemma-2-9B	9.24B	88.05	66.04	84.34	79.89	46.80	83.19	74.11	74.63	10.60
Ratio = 25% w/o tune	LLaMA-3.1-8B	6.67B	70.16	43.17	73.79	69.44	42.00	77.20	71.19	63.85	14.26
	Qwen2.5-7B	6.39B	67.13	41.30	72.97	68.67	41.40	77.37	65.27	62.02	14.51
	Gemma-2-9B	7.65B	74.62	47.01	75.60	60.81	39.20	75.63	65.04	62.56	20.89
Ratio = 25% w/ tune	LLaMA-3.1-8B	6.67B	75.46	47.18	80.43	73.20	41.00	79.33	72.61	67.03	11.63
	Qwen2.5-7B	6.39B	74.66	47.70	78.75	71.93	43.00	78.51	69.46	66.29	12.44
	Gemma-2-9B	7.65B	80.43	53.75	82.63	71.03	42.20	78.89	68.43	68.19	15.49
Ratio = 50% w/o tune	LLaMA-3.1-8B	5.26B	52.10	31.57	63.27	51.51	33.60	70.51	61.09	51.95	38.52
	Qwen2.5-7B	5.13B	45.45	28.84	55.47	51.36	36.80	69.42	56.27	49.09	44.07
	Gemma-2-9B	5.96B	53.11	29.10	61.59	40.10	28.80	65.18	51.70	47.08	214.08
Ratio = 50% w/ tune	LLaMA-3.1-8B	5.26B	61.74	37.12	71.35	59.62	33.60	72.96	64.72	57.30	19.46
	Qwen2.5-7B	5.13B	52.69	33.62	54.07	57.69	37.20	72.52	59.83	52.52	21.58
	Gemma-2-9B	5.96B	66.58	36.60	71.65	54.43	37.60	70.89	59.19	56.71	28.08
Ratio = 75% w/o tune	LLaMA-3.1-8B	4.20B	32.58	24.74	57.68	33.32	26.40	58.05	52.25	40.72	430.20
	Qwen2.5-7B	3.97B	38.09	22.53	57.25	35.28	27.80	58.98	49.17	41.30	164.23
	Gemma-2-9B	4.32B	30.30	24.74	42.45	27.30	25.60	52.45	50.59	36.20	203176.96
Ratio = 75% w/ tune	LLaMA-3.1-8B	4.20B	46.00	27.30	61.62	42.99	28.40	64.85	55.09	46.61	35.20
	Qwen2.5-7B	3.97B	44.65	26.62	54.83	40.67	30.00	63.28	54.38	44.92	37.90
	Gemma-2-9B	4.32B	49.87	27.13	56.61	37.10	27.60	63.76	51.70	44.82	66.68

Table 1 presents the zero-shot performance and perplexity of pruned models at various sparsity levels. Our results show that the pruned LLaMA models retain 90.9% of the benchmark average scores of the dense model at 25% sparsity and 74.0% at 50% sparsity without post-training. Similarly, the adaptive pruner achieves comparable results for the Qwen and Gemma models, maintaining 88.3% and 83.8% performance at 25% sparsity, respectively, without post-training. While perplexity increases moderately for LLaMA and Qwen at 25% and 50% sparsity, and for Gemma at 25%, it rises dramatically beyond these levels. These findings highlight the effectiveness of the adaptive pruner in compressing models without post-training. However, at 75% sparsity, performance declines significantly, likely due to architectural disruption, suggesting that some post-training may be necessary to sustain the performance of highly compressed models.

We further add experiments here with recovery post training using the dataset OpenHermes 2.5 [Teknium, 2023]. We apply a final post-training using LoRA [Hu et al., 2021b]. For all post-training experiments, we use 0.1 training epochs with each data point truncated to 512 tokens. The LoRA configuration includes $lora_r = 128$ and $lora_alpha = 16$, with default values for all other hyperparameters from the Hugging Face PEFT package [Mangrulkar et al., 2022].

Table 1 also presents the results following post-training. With post-training, the average benchmark score for LLaMA increases to 95.5% at 25% sparsity and 81.6% at 50% sparsity. However, it is important to note that the performance on several benchmarks may drop after post-training. This decline could be attributed to knowledge forgetting, as the model may adapt too closely to the post-training dataset used.

Structured Pruning Methods Comparison We compare our method to the baseline LLM-Pruner, as well as other state-of-the-art structured pruning methods, including FLAP and SliceGPT. To ensure a fair comparison, we do not skip the first or last layers when applying the adaptive pruner. All hyperparameters for these methods are set to their default values. The LM Evaluation Harness [Gao et al., 2024] is used to evaluate the compressed models produced by each method.

Table 2 presents a comparison of structured pruning methods. Adaptive pruning demonstrates improvements of 20.6% and 24.1% in average benchmark scores at 25% and 50% sparsity, respectively, compared to LLM-Pruner, which utilizes uniform pruning. The results also indicate that the adaptive pruner outperforms both FLAP and SliceGPT at these sparsity levels, suggesting that our method, which measures the changes in input and output tensors to evaluate decoder layer importance, is more effective than FLAP's approach, which combines input variance and weight magnitude. However, as pruning sparsity increases, the performance gap between adaptive pruning and other methods narrows. This may be due to higher sparsity leading to significant knowledge loss, resulting in similar performance across all structured pruning methods.

Table 2: Comparison of structured pruning methods, including LLM-Pruner, FLAP, and SliceGPT, on LLaMA-3.1-8B across different sparsity levels. Bold font indicates the optimal method for each criterion within the same sparsity group.

Pruning Sparsity	Method	Para. Num.	ARC-e	ARC-c	BoolQ	HellaSwag	OBQA	PIQA	Winogrande	Average	WikiText2↓
Ratio = 25% w/o tune	LLM-Pruner	6.29B	48.15	30.80	61.41	50.02	29.40	68.28	51.78	48.55	24.85
	FLAP	6.12B	49.20	30.03	57.74	52.18	35.40	68.99	59.98	50.50	**15.35**
	SliceGPT	6.78B	42.93	27.39	37.86	44.99	33.80	58.65	60.69	43.76	30.96
	Adapt-Pruner	6.36B	64.56	38.82	64.83	62.99	39.60	77.48	67.40	**59.38**	18.69
Ratio = 50% w/o tune	LLM-Pruner	4.54B	25.55	24.91	37.86	25.95	27.60	51.03	49.64	34.65	632.58
	FLAP	4.34B	24.20	25.26	40.70	26.37	26.00	51.25	50.36	34.88	270251.54
	SliceGPT	4.58B	30.72	22.01	37.83	29.85	24.60	51.85	48.70	35.08	118.55
	Adapt-Pruner	4.61B	37.29	25.17	60.73	35.60	28.80	60.77	52.09	**42.92**	**95.59**
Ratio = 75% w/o tune	LLM-Pruner	2.80B	25.72	24.83	38.65	25.74	29.60	50.16	50.59	35.04	5478.46
	FLAP	2.77B	25.34	26.71	43.06	26.42	28.60	52.45	49.88	**36.07**	169822.13
	SliceGPT	2.52B	28.20	22.01	37.83	27.77	24.80	51.09	50.12	34.55	**741.92**
	Adapt-Pruner	2.88B	27.74	22.27	39.14	27.15	27.00	52.61	50.83	35.25	734.18

4.3 More Analysis

Efficient Performance Transfer via Model Compression We investigate the efficacy of model compression as an alternative to training from scratch, with particular focus on computational efficiency. Through knowledge distillation and comprehensive parameter fine-tuning on OpenHermes-2.5, we demonstrate our method's versatility by compressing two different architectures: LLaMA-3.2-3B to 1.13B parameters and LLaMA-2-7B to 1.16B parameters. We conduct comparative analyses against similarly-sized pretrained models to evaluate compression effectiveness. The pretrained TinyLlama-1.1 checkpoint serves as our baseline, enabling quantitative assessment of performance preservation through distillation despite pruning-induced degradation.

Table 3: Comparison of Compressed Model with Pretrained Model

Model	Para. Num.	Tokens Needed	ARC-e	ARC-c	BoolQ	HellaSwag	OBQA	PIQA	Winogrande	Average	WikiText2↓
Compressed LLaMA2-7B	1.16B	0.00B	26.52	27.39	38.81	25.79	24.00	49.73	49.80	34.58	69962.30
		0.06B	36.36	22.87	58.84	27.86	26.80	57.45	50.04	40.03	119.74
		0.14B	38.26	24.15	60.40	28.61	25.80	56.91	49.72	40.55	86.58
		0.17B	38.43	24.06	60.12	28.68	27.40	57.02	49.80	40.79	84.76
Compressed LLaMA3.2-3B	1.13B	0.00B	26.26	25.77	41.87	25.86	27.80	51.63	50.91	35.73	1940.24
		0.05B	42.85	24.66	60.49	29.05	28.40	58.87	50.04	42.05	65.94
		0.11B	43.31	24.32	61.56	28.95	31.40	60.55	51.38	43.07	51.91
		0.15B	43.48	24.57	61.87	28.78	29.40	60.34	49.96	42.63	49.59
TinyLlama-1.1	1.10B	10B	38.59	22.53	47.95	33.12	30.20	61.70	50.91	40.71	33.48
		21B	41.08	23.72	58.38	35.79	30.20	63.60	52.49	43.61	27.39

As demonstrated in Table 3, our compressed 1.1B model achieves superior performance to TinyLlama-1.1 while requiring only 0.05B training tokens, compared to TinyLlama's 10B pretraining tokens. This empirical evidence substantiates that structured pruning, combined with knowledge distillation, not only enables model size reduction but also provides a computationally efficient framework for obtaining high-performing compact models. The 200× reduction in required training tokens highlights the significant computational advantages of our compression-based approach.

Sensitivity Analysis of Amplitude Parameter A We empirically investigate the impact of amplitude parameter A in Equation 5 on our algorithm's performance. The amplitude A directly influences the architectural search space of the compressed model: insufficient amplitude constrains the exploration of potential architectural configurations, while excessive amplitude can lead to structural imbalances that degrade model performance. To systematically analyze this relationship, we conduct experiments on LLaMA-3.1-8B using a controlled setup where we maintain consistent parameters (50% overall sparsity ratio and 50 pruning iterations) while varying the amplitude A across pruning steps. This experimental design allows us to isolate and quantify the specific effects of amplitude variation on model compression outcomes.

As demonstrated in Table 4, the baseline case of $A = 0$ represents uniform pruning without our proposed adaptive mechanism. Notably, varying the amplitude A yields different final parameter counts due to its influence on architectural decisions during compression. Our experimental results reveal that $A = 0.02$ achieves optimal performance, maximizing the benchmark average while minimizing model complexity on WikiText2. The performance exhibits a non-monotonic

Table 4: Sensitivity analysis of amplitude parameter A in progressive pruning, with 50% sparsity and 50 pruning steps

Amplitude A	Para. Num.	ARC-e	ARC-c	BoolQ	HellaSwag	OBQA	PIQA	Winogrande	Average	WikiText2↓
0	4.54B	29.04	21.84	45.47	28.22	27.00	53.81	48.30	36.24	621.03
0.005	4.62B	37.25	22.87	55.90	33.89	27.80	59.52	52.64	41.41	122.04
0.01	4.61B	37.29	25.17	60.73	35.60	28.80	60.77	52.09	42.92	95.59
0.02	4.43B	40.61	26.45	62.05	35.18	28.20	62.30	53.67	44.07	90.80
0.04	4.04B	34.51	24.06	56.21	31.17	26.40	57.34	51.78	40.21	314.90

relationship with A: as amplitude increases, model performance initially improves before degrading, consistent with the theoretical trade-off between exploration capacity and architectural stability.

Impact of Distillation Coefficient α We systematically analyze the distillation coefficient α in the objective function (Equation 8), which controls the balance between learning from ground truth labels and the teacher model's behavior during post-training. Using the optimal compressed model configuration identified in Table 4, we conduct controlled experiments with varying α values while maintaining consistent hyperparameters. The experiments utilize LoRA for fine-tuning on the OpenHermes-2.5 dataset, with the original LLaMA-3.1-8B serving as the teacher model.

Table 5: Impact of Distillation Coefficient α using LoRA with LLaMA-3.1-8B as the teacher model

Tokens Needed	Coefficient α	ARC-e	ARC-c	BoolQ	HellaSwag	OBQA	PIQA	Winogrande	Average	WikiText2↓
0.04B	0	49.75	27.56	62.84	46.37	32.20	68.12	56.51	49.05	33.78
	0.25	49.92	27.65	62.20	46.65	30.60	68.50	56.91	48.92	33.67
	0.50	50.42	28.16	62.11	46.95	30.00	68.44	55.96	48.86	34.00
	0.75	50.13	27.99	62.35	47.23	30.40	68.50	56.12	48.96	34.59
	1.00	49.75	28.58	62.42	46.68	29.80	68.61	55.64	48.78	36.13

As demonstrated in Table 5,

Computation Cost LLaMA-3.1-8B, Qwen2.5-7B, and Gemma-2-9B can be pruned adaptively in 2 to 5 minutes on a single NVIDIA A40 GPU, and in 15 to 30 minutes on an Intel(R) Xeon(R) Gold 6346 CPU. After applying iterative post-training, the full compression process takes between 3 and 10 hours. Benchmark evaluation of the compressed models requires an additional 15 to 30 minutes.

5 Conclusion

In this paper, we present Adapt-Pruner, an adaptive and structured pruning method for large language models. Our approach evaluates each decoder layer and assigns corresponding sparsity, prunes coupled weight structures based on weight magnitude and first-order information, and periodically applies post-training to recover any performance drops. We test our method on a variety of popular open-source LLMs. The results show that Adapt-Pruner retains 90.9%, 88.3%, and 83.8% of the benchmark average scores of the dense model at 25% sparsity without post-training for LLaMA-3.1, Qwen2.5, and Gemma-2, respectively. With periodic post-training, these scores increase to 95.5%, 94.4%, and 91.4%, respectively. Our experiments also demonstrate that Adapt-Pruner outperforms other structured pruning methods, including SliceGPT and FLAP, indicating that our method, which measures changes in input and output tensors to evaluate decoder layer importance, is more effective than FLAP's approach, which combines input variance and weight magnitude.

There is room for improvement in our method. Currently, we relate layer importance to sparsity using a straightforward amplitude, which may not be the optimal approach to fully leverage this information for achieving theoretical maximum effectiveness. Additionally, while periodic post-training during pruning helps recover performance losses, it also imposes quality demands on the dataset, potentially causing the LLMs to forget previously learned information. Moreover, post-training can introduce significant computational costs to the pruning process. Future work could focus on enhancing the efficiency of post-training to address these challenges.

We hope our work inspires future research to quantitatively evaluate the importance of decoder layers and leverage this insight when compressing LLMs, ultimately leading to the development of more efficient deep neural networks.

References

Katikapalli Subramanyam Kalyan. A survey of gpt-3 family large language models including chatgpt and gpt-4. *Natural Language Processing Journal*, 6:100048, 2024. ISSN 2949-7191. doi:https://doi.org/10.1016/j.nlp.2023.100048. URL https://www.sciencedirect.com/science/article/pii/S2949719123000456.

OpenAI. Gpt-4 technical report, 2023.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models, 2024. URL https://arxiv.org/abs/2308.07633.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2023.

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Qwen Team. Qwen2.5: A party of foundation models, September 2024a. URL https://qwenlm.github.io/blog/qwen2.5/.

Gemma Team. Gemma. 2024b. doi:10.34740/KAGGLE/M/3301. URL https://www.kaggle.com/m/3301.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 129–146, 2020. URL https://proceedings.mlsys.org/paper_files/paper/2020/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf.

Lucio Dery, Steven Kolawole, Jean-François Kagy, Virginia Smith, Graham Neubig, and Ameet Talwalkar. Everybody prune now: Structured pruning of llms with only forward passes, 2024. URL https://arxiv.org/abs/2402.05406.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns, 2024. URL https://arxiv.org/abs/2401.15024.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. Shortened llama: Depth pruning for large language models with comparison of retraining methods, 2024. URL https://arxiv.org/abs/2402.02834.

Shoaib Ahmed Siddiqui, Xin Dong, Greg Heinrich, Thomas Breuel, Jan Kautz, David Krueger, and Pavlo Molchanov. A deeper look at depth pruning of llms, 2024. URL https://arxiv.org/abs/2407.16286.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning, 2024. URL https://arxiv.org/abs/2310.06694.

Harry Dong, Beidi Chen, and Yuejie Chi. Prompt-prompted adaptive structured pruning for efficient llm generation. In *First Conference on Language Modeling*, 2024.

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models, 2023.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models, 2024. URL https://arxiv.org/abs/2402.13116.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Llm pruning and distillation in practice: The minitron approach, 2024. URL https://arxiv.org/abs/2408.11796.

Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021a.

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

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018. URL https://api.semanticscholar.org/CorpusID:3922816.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi:10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

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, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Teknium. Openhermes 2.5: An open dataset of synthetic data for generalist llm assistants, 2023. URL https://huggingface.co/datasets/teknium/OpenHermes-2.5.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021b. URL https://arxiv.org/abs/2106.09685.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.