LoRAPrune: Structured Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning

Anonymous ACL submission

Abstract

Large Language Models (LLMs), such as LLaMA and T5, have shown exceptional performance across various tasks through fine-tuning. Although low-rank adaption (LoRA) has emerged to cheaply fine-tune these LLMs on downstream tasks, their deployment is still hindered by the vast model scale and computational costs. Post-training model pruning offers a way to compress LLMs. However, the current pruning methods designed for LLMs are not compatible with LoRA. This is due to their utilization of unstructured pruning on LLMs, impeding the merging of LoRA weights, or their dependence on the gradients of pre-trained weights to guide pruning, which can impose significant memory overhead. To this end, we propose LoRAPrune, a new framework that delivers an accurate structured pruned model in a highly memory-efficient manner. Specifically, we first design a LoRA-guided pruning criterion, which uses the weights and gradients of LoRA, rather than the gradients of pre-trained weights for importance estimation. We subsequently integrate this criterion into an iterative pruning process, effectively removing redundant channels and heads. Extensive experimental results demonstrate the superior performance of our LoRAPrune over existing approaches on the LLaMA series models. At a 50% compression rate, LoRAPrune demonstrates superior performance over LLM-Pruner, achieving a reduction in perplexity by 4.81 on WikiText2 and 3.46 on PTB, while also decreasing memory usage by 52.6%. Besides, LoRAPrune also matches semi-structural pruning across multiple LLMs, proving its wide applicability.

1 Introduction

Large Language Models (LLMs) (Touvron et al., 2023; Du et al., 2022; Frantar et al., 2022) have showcased remarkable prowess, exhibiting outstanding performance across numerous tasks. To enable LLMs to perform specific tasks, such as chat-bots (Du et al., 2022; Zeng et al., 2022), they are often efficiently fine-tuned on downstream datasets (Taori et al., 2023; Chenghao Fan and Tian, 2023) by parameter-efficient fine-tuning (PEFT) methods (Jia et al., 2022; Hu et al., 2022; Chen et al., 2022), among which LoRA-based fine-tuning methods (Hu et al., 2022; Luo et al., 2023; He et al., 2023) have gained widespread use. However, the remarkable success of LLMs is accompanied by obstacles from their vast scale and substantial computational costs, making deployment exceedingly arduous (Frantar and Alistarh, 2023).

Table 1: The memory costs for pruning LLaMA-65B. "Iter." indicates whether the method supports iterative pruning and "#GPU" indicates the number of NVIDIA A100 (80G) GPUs required.

Method	Iter.	#GPU	Mem.(G)
PST (Li et al., 2022)	✓	3	234
LLM-Pruner (Ma et al., 2023)	×	2	154
LoRAPrune	✓	1	72

Neural network pruning (Li et al., 2017; Molchanov et al., 2017), a prevailing technique for model compression, can significantly reduce the model size and complexity. Recently, the post-training pruning literature, such as SparseGPT (Frantar and Alistarh, 2023) and WANDA (Sun et al., 2023), have achieved high-performance unstructured sparse LLMs. However, unstructured sparse models face two critical issues: *1) Unstructured sparse models are hard to obtain direct inference speedup*. They often require specialized hardware support to achieve satisfying acceleration benefits, which leads to unstructured pruning not benefiting legacy off-the-shelf platforms, *e.g.*, CPUs, DSPs, and GPUs (Fang et al., 2023; You et al., 2023; Zhou et al., 2022). *2) Unstructured sparse models are not compatible with LoRA*. As shown in Figure 1 (a), since the weights $\mathbf{BA}$ produced by LoRA are dense, it poses challenges when trying to merge $\mathbf{BA}$ into the unstructured sparse weights. For instance, LoRA without merging increases inference time by nearly 54% (see Table 3), diminishing the

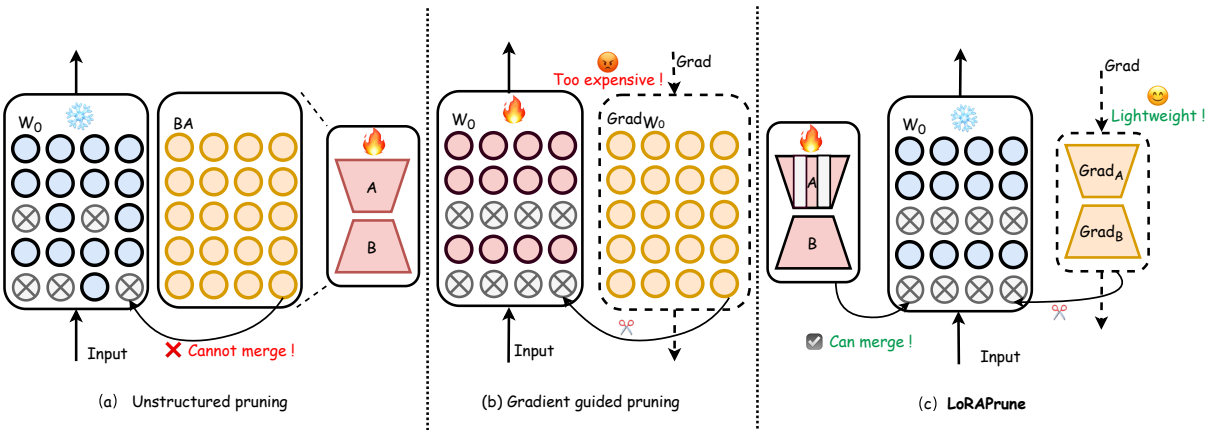

Figure 1: Comparing LoRAPrune with other pruning methods: (a) Unstructured sparse model cannot directly merge LoRA weights, which is computationally inefficient. (b) Gradient-guided pruning requires the gradients of the pre-trained weights, which is memory-intensive. (c) LoRAPrune only needs the gradients of LoRA weights and can seamlessly merge LoRA weights into pre-trained weights, which is efficient in both memory and computation.

benefits of pruning. One potential solution is to perform fine-tuning using LoRA on downstream tasks first and then carry out post-training pruning. However, separating tuning and pruning can lead to sub-optimal results (Molchanov et al., 2019; Sanh et al., 2020). To tackle this challenge, PST (Li et al., 2022) combines unstructured pruning with efficient fine-tuning, which simultaneously prunes LoRA and pre-trained weights. This method ensures a seamless merge of LoRA weights and avoids additional computational overhead that comes from LoRA. However, unstructured pruning of LoRA necessitates computing $\mathbf{BA}$ first and then doing Hadamard product with a binary mask $\mathbf{M}$, which results in significant memory overhead (see Table 1) since $\mathbf{BA}$ and $\mathbf{M}$ share the same shape with pre-trained weights. For instance, when pruning LLaMA-65b, the intermediate variables necessitate the storage capacity of three NVIDIA A100 (80G) GPUs. This poses a significant memory challenge when adapting PST to LLMs. Instead, structured pruning can mitigate this issue since we can directly prune the structured weights of $\mathbf{A}$ in LoRA without storing $\mathbf{BA}$. Therefore, it is significant to combine LoRA with structured pruning to achieve simultaneous PEFT and direct acceleration on general hardware platforms with high performance.

To this end, we propose a unified framework for LoRA and structured pruning, named LoRAPrune. As shown in Figure 1 (c), LoRAPrune not only prunes the structured weights (*e.g.*, heads, channels) from the pre-trained model weights $\mathbf{W}_0$ but also trims the corresponding weights in LoRA weight $\mathbf{A}$ without computing $\mathbf{BA}$ first. Consequently, after pruning and fine-tuning, the weights of LoRA can be *seamlessly* merged with the pre-trained weights, ensuring that no additional computations are needed during inference. To identify weight connections of structural importance, the criterion used in the structured pruning methods (Ma et al., 2023; Molchanov et al., 2019, 2017) is often estimated by gradients or its variants, as shown in Figure 1 (b). However, LoRA typically requires frozen pre-trained weights without computing their gradients, thus pruning approaches that rely on gradients of the pre-trained weights cannot be directly applied. To *efficiently* estimate the importance of pre-trained weights, LoRAPrune introduces a novel criterion that exclusively utilizes the gradients of LoRA. In contrast to the vanilla gradient-guided pruning method, LoRAPrune leverages LoRA's gradients as the approximation for the gradients of the pre-trained weights. Based on the presented criterion, we can *iteratively* perform pruning while simultaneously conducting efficient fine-tuning to restore the performance of the pruned LLMs, requiring only a small calibration set. Specifically, we compute the importance of every batch of data and update the importance using a moving average. Every few iterations, we remove a portion of unimportant structured weights until the desired sparsity is achieved. Through extensive experiments on diverse benchmark datasets and various scales of LLMs, we demonstrate that LoRAPrune consistently outperforms other structured pruning techniques tailored for LLMs. Furthermore, compared to the vanilla gradient-guided pruning, LoRAPrune significantly

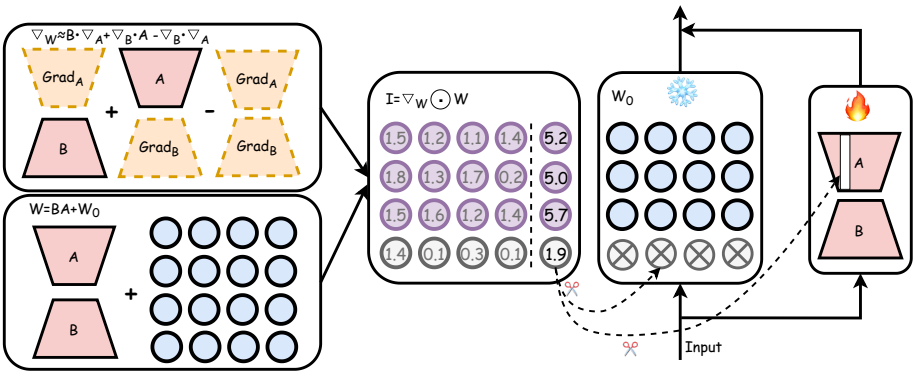

Figure 2: The pruning process for the LoRA-guided criterion involves utilizing the LoRA matrices $\mathbf{A}$, $\mathbf{B}$ and their respective gradients $\nabla_{\mathbf{A}}$, $\nabla_{\mathbf{B}}$ to compute the importance $\mathbf{I}$. Subsequently, weight importance (gray number) with the same group are aggregated to the group importance (black number) and the groups with low scores are removed.

diminishes memory and computational overhead, facilitating efficient pruning and fine-tuning of LLaMA-65b on a single GPU concurrently. This paper has the following key contributions:

- We introduce a novel memory-efficient pruning criterion tailored for LLMs, termed the LoRA-guided criterion, which seamlessly integrates with LoRA. Leveraging the gradients of LoRA, we can efficiently approximate the importance of pre-trained weights without needing to compute their gradients.
- As we can efficiently approximate gradients and update weights using LoRA, LoRAPrune facilitates iterative structured pruning, resulting in precise small models. Our framework ensures both high memory efficiency during pruning and incurs efficient inference.
- Pruning experiments conducted on the LLaMA models demonstrate that LoRAPrune can efficiently perform structured pruning with up to 65 billion weights on a single GPU. Furthermore, the pruning results achieved by LoRAPrune significantly surpass other pruning methods. For example, against LLM-Pruner, LoRAPrune uses only 52.6% of the memory yet scores lower perplexities by 4.81 on WikiText2 and 3.46 on PTB. LoRAPrune also matches semi-structural pruning performance across various LLMs, proving its broad applicability.

2 Related Work

Parameter-efficient fine-tuning. PEFT methods (Jia et al., 2022; Wu et al., 2022; Chen et al., 2022; Hu et al., 2022; Luo et al., 2023; He et al., 2023) have received increasing attention from both academia and industry. Among them, LoRA (Hu et al., 2022) proposes injecting trainable low-rank decomposition matrices into each layer which can be merged into the pre-trained weights, avoiding extra computation in inference. Since inference efficiency, many methods based on LoRA have emerged. For instance, LongLoRA (Chen et al., 2023) improves upon LoRA, enabling efficient fine-tuning of LLMs on long contexts. AnimateDiff (Guo et al., 2023b) obtains a personalized generator by inserting LoRA into the frozen text-to-image model. Quantizing the pre-trained weights into 4-bit, QLoRA (Dettmers et al., 2023) employs LoRA for fine-tuning LLMs in downstream tasks while maintaining efficient memory usage. Therefore, LoRA is indispensable for fine-tuning LLMs. Our method seamlessly integrates LoRA and pruning, making it easily extensible to other PEFT methods based on LoRA.

Neural network pruning. Removing unimportant weights from LLMs to reduce memory and the computational cost of deployment has become a common approach for model compression. Unstructured pruning (Dong et al., 2017; Lee et al., 2019; Wang et al., 2020; Sun et al., 2023; Frantar and Alistarh, 2023; Li et al., 2022) can obtain highly compressed models by directly pruning neurons, which also causes unstructured sparsity and hard deployment. In contrast, structured pruning (Ma et al., 2023; Xia et al., 2023; Guo et al., 2023a) directly discards the whole grouped parameters (*e.g.*heads, channels) and leaves a model with deploy-friendly structures. However, structured pruning models require extensive finetuning to regain their performance levels. For example, Xia et al. (2023) utilized 50B tokens sampled for continued pretraining of their pruned model, a process

that proves to be prohibitively expensive in terms of hardware resources. In contrast, our approach leverages structured pruning alongside a significantly smaller dataset, enabling direct inference acceleration while maintaining training expenses at an acceptable level.

Pruning criterion. Determining the importance of weights in a network is still an open question (Blalock et al., 2020). A common approach to model pruning is to use parameter magnitude (Li et al., 2018; Lee et al., 2020; Elesedy et al., 2020; Han et al., 2015; Li et al., 2017) as a criterion. However, the small weights can still have a significant impact on the model output due to the complex structure of neural networks, while large weights may not be as important. Many methods (Sanh et al., 2020; Yu et al., 2022a; Zhang et al., 2022; Lee et al., 2019; Yu et al., 2022b; Wang et al., 2020; LeCun et al., 1989; Hassibi et al., 1993) employ Taylor expansion to approximate the errors introduced by pruning and use this as the criterion for importance estimation. To avoid computing the Hessian matrix (Hassibi et al., 1993) or Hessian inverse (LeCun et al., 1989) in Taylor expansion, (Molchanov et al., 2017, 2019) only use the first-order term in Taylor expansion. Furthermore, LLM-Pruner (Ma et al., 2023) similarly utilizes the first-order expansion for pruning and extends the pruning technique to LLMs. However, the first-order term in Taylor expansion still requires gradients of the pre-trained weights. As shown in Table 1, computing and storing the gradients of pre-trained weights significantly increases the pruning cost. To avoid computing gradients of pre-trained weights, PST (Li et al., 2022) learns the gradients of pre-trained weights by an extra low-rank matrix, which is motivated by LoRA. Nevertheless, PST conducts unstructured pruning and needs to compute a substantial mask with the equivalent shape of pre-trained weights in each forward pass, which is memory-intensive and hard to be adapted to LLMs. Different from LLM-Pruner (Ma et al., 2023) and PST (Li et al., 2022), our criterion only relies on LoRA's gradients and does not require expensive mask computation, making it memory-efficient.

3 Method

3.1 Preliminary

Initially, we define the notation used in the formula. "**Bold**" letters represent matrices and vectors, while "non-bold" letters indicate scalars. "Subscripts" identify the index of elements within a matrix, and "superscripts" indicate the layer index in a network.

Low-rank adaptation. To efficiently fine-tune LLMs, low-rank adapter LoRA (Hu et al., 2022) constrains the update of model parameters to maintain a low intrinsic rank. During fine-tuning, the pre-trained weights remain frozen, abstaining from gradient computation, while the inserted LoRA is kept trainable. Given two low-rank matrices $\mathbf{A} \in \mathbb{R}^{r\times k}$ and $\mathbf{B} \in \mathbb{R}^{d\times r}$ ($r \ll \min(d,k)$), the update of a linear module can be written as

$$\mathbf{z} = \mathbf{x}\mathbf{W}_0 + \mathbf{x}\mathbf{B}\mathbf{A}, \tag{1}$$

where $\mathbf{W}_0 \in \mathbb{R}^{d\times k}$, $\mathbf{z} \in \mathbb{R}^{n\times k}$ and $\mathbf{x} \in \mathbb{R}^{n\times d}$ denote the pre-trained weights, outputs and inputs, respectively. After adaption, the new weights $\mathbf{W}$ can be re-parameterized as $\mathbf{W} = \mathbf{W}_0 + \mathbf{B}\mathbf{A}$.

Pruning with Taylor expansion. In vanilla pruning approaches (Molchanov et al., 2017, 2019), the importance of a weight $\mathbf{W}_{i,j} \in \mathbf{W}_0$ can be quantified by measuring the impact of its removal on the loss. For an input $\mathbf{x}$ and the ground-truth prediction $\mathbf{y}$, the induced error of $\mathbf{W}_{i,j}$ can be given as:

$$\mathbf{I}_{i,j} = [\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_0) - \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_0 | \mathbf{W}_{i,j} = 0)]^2. \tag{2}$$

Computing $\mathbf{I}_{i,j}$ for each weight is computationally expensive. Following (Molchanov et al., 2019), we can use first-order Taylor expansion to approximate the importance $\hat{\mathbf{I}}_{i,j}$ by:

$$\hat{\mathbf{I}}_{i,j} = (\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{i,j}} \mathbf{W}_{i,j})^2. \tag{3}$$

Dependency-aware structured pruning. In structured pruning, it is crucial to consider that pruned neurons can exhibit dependencies with other neurons due to their interconnected nature. The dependencies of weights are illustrated in Figure 5. We organize the connected weights as a group and estimate the group importance by accumulating the weight importance within the same group. Formally, the importance for the g-th group can be expressed as

$$\hat{\mathcal{G}}_g = \sum_{\mathbf{W}_{i,j} \in \mathbb{G}} \hat{\mathbf{I}}_{i,j}\ , \tag{4}$$

where $\hat{\mathcal{G}} \in \mathbb{R}^{1\times G}$ represents the importance of groups, $\mathbb{G}$ denotes a set of weights within a group and G is the candidate group number in a layer.

Algorithm 1: Progressive pruning with LoRA-guided criterion

Require : Calibration data $\mathcal{D}$; Pre-trained weights $\mathbf{W}_0$; Randomly initialized low-rank matrices $\mathbf{A}$ and $\mathbf{B}$; Loss function $\mathcal{L}$; Target sparsity level S; Fine-tuning iterations T.

Output : Trained low-rank adaption $\mathbf{A}$ and $\mathbf{B}$; Binary mask $\mathbf{M}$.

$\bar{\mathfrak{G}}_g^l \leftarrow 0$, $\mathbf{M}_g^l \leftarrow 1$ for $\forall l, \forall g$;

`// Initialization for masks and group importance`

$s \leftarrow 0$; `// Initialize sparsity level`

for $t \in [1, \ldots, T]$ **do**

  Clear gradient;

  Forward and backward via Eq. (12);

  Update $\mathbf{A}$ and $\mathbf{B}$ via AdamW;

  Calculate $\hat{\mathbf{I}}|_t$ via Eq. (9);

  Calculate $\hat{\mathfrak{G}}|_t$ via Eq. (4);

  Calculate $\bar{\mathfrak{G}}|_t$ via Eq. (10);

  for $l \in [1, \ldots, L]$ **do**

    $p \leftarrow \mathrm{SortDescending}(\bar{\mathfrak{G}})_s$; `// Set threshold`

    $\mathbf{M}_g^l \leftarrow 0$ where $\bar{\mathfrak{G}}_g^l \leq p$, and $g \in \{1, \ldots, G\}$

  end

  `// Remove unimportant groups`

  Progressively increase s until $||\mathbf{M}||_0 > S$;

end

3.2 Pruning with Low-rank Adaption

Motivation. To achieve highly-compressed LLMs, it is essential to accurately evaluate the importance of pre-trained weights. A key approach is to utilize the criteria in Eq. (3) for this evaluation. However, obtaining the gradient of $\mathbf{W}_0$ in a LLM is difficult since it requires a lot of computing power and storage space. Fine-tuning LLMs with LoRA is becoming prevalent (Taori et al., 2023; Chenghao Fan and Tian, 2023). During LoRA fine-tuning, only the gradients of LoRA's weights are computed, yielding remarkable computation and memory efficiency. Therefore, can we rely solely on the weights and gradients of LoRA to accurately estimate the importance of pre-trained weights?

LoRA-guided criterion. In this work, we discuss how to estimate the importance of $\mathbf{W}_0$ by inserting the learnable matrices $\mathbf{A}$ and $\mathbf{B}$ in the downstream task adaption.

The core idea lies in setting the element $(\mathbf{BA})_{ij} = -\mathbf{W}_{ij}$ if the element $\mathbf{W}_{ij} \in \mathbf{W}_0$ is removed. The importance of each parameter in Eq. (2) can be reformulated as follows

$$\mathbf{I}_{i,j} = [\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}) - \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}|(\mathbf{BA})_{i,j} = -\mathbf{W}_{i,j}]^2. \tag{5}$$

Exploiting the first-order Taylor expansion with $(\mathbf{BA})_{i,j} = -\mathbf{W}_{i,j}$ to approximate Eq. (5), the estimated importance $\hat{\mathbf{I}}_{i,j}$ of parameter $\mathbf{W}_{i,j}$ can be represented by

$$\hat{\mathbf{I}}_{i,j} = [\frac{\partial \mathcal{L}}{\partial (\mathbf{BA})_{i,j}}((\mathbf{BA})_{i,j} + \mathbf{W}_{i,j})]^2. \tag{6}$$

However, as shown in Eq. (1), the LoRA computation sequence involves first multiplying by $\mathbf{B}$ and then by $\mathbf{A}$, which means that $\mathbf{BA}$ cannot be obtained during the forward and backward pass. Besides, preserving $\frac{\partial \mathcal{L}}{\partial (\mathbf{BA})_{i,j}}$ still entails the same level of complexity as $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{i,j}}$ since $\mathbf{BA}$ shares the same shape of $\mathbf{W}_0$.

Here, we only save and use the gradients of two low-rank matrices $\mathbf{A}$ and $\mathbf{B}$ to approximate $\frac{\partial \mathcal{L}}{\partial (\mathbf{BA})}$. We can rely on the gradient update that $(\mathbf{BA})_{i,j}|_t = (\mathbf{BA})_{i,j}|_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial (\mathbf{BA})_{i,j}}$ to estimate the gradient, where $(\mathbf{BA})_{i,j}|_t$ and $(\mathbf{BA})_{i,j}|_{t-1}$ represents the $(\mathbf{BA})_{i,j}$ in t-th and $(t-1)$-th step, respectively. For simplicity, we ignore the learning rate η since it has no influence on the importance criterion. Apparently, $\frac{\partial \mathcal{L}}{\partial (\mathbf{BA})_{i,j}}$ is proportional to the change of $\mathbf{BA}$, which can be written as

$$\frac{\partial \mathcal{L}}{\partial (\mathbf{BA})_{i,j}} \propto [(\mathbf{BA})_{i,j}|_{t-1} - (\mathbf{BA})_{i,j}|_t]. \tag{7}$$

Here, $(\mathbf{BA})_{i,j}|_t = \mathbf{B}_{i,:}|_t \mathbf{A}_{:,j}|_t$ is generated by the multiplication of the i-th row of $\mathbf{B}|_t$ with the j-th column of $\mathbf{A}|_t$. Using the above assumption, we can also estimate $\frac{\partial \mathcal{L}}{\partial \mathbf{A}_{:,j}} \propto \mathbf{A}_{:,j}|_{t-1} - \mathbf{A}_{:,j}|_t$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{B}_{i,:}} \propto \mathbf{B}_{i,:}|_{t-1} - \mathbf{B}_{i,:}|_t$, respectively. Subsequently, we substitute $(\mathbf{BA})_{i,j}$ to Eq. (7) and obtain

$$\frac{\partial \mathcal{L}}{\partial (\mathbf{BA})_{i,j}} \propto [\frac{\partial \mathcal{L}}{\partial \mathbf{B}_{i,:}} \mathbf{A}_{:,j} + \mathbf{B}_{i,:} \frac{\partial \mathcal{L}}{\partial \mathbf{A}_{:,j}} - \frac{\partial \mathcal{L}}{\partial \mathbf{B}_{i,:}} \frac{\partial \mathcal{L}}{\partial \mathbf{A}_{:,j}}]. \tag{8}$$

Substitute Eq. (8) to Eq. (6), we can estimate the importance in a gradient-based manner

$$\hat{\mathbf{I}}_{i,j} = [(\frac{\partial \mathcal{L}}{\partial \mathbf{B}_{i,:}} \mathbf{A}_{:,j} + \mathbf{B}_{i,:} \frac{\partial \mathcal{L}}{\partial \mathbf{A}_{:,j}} - \frac{\partial \mathcal{L}}{\partial \mathbf{B}_{i,:}} \frac{\partial \mathcal{L}}{\partial \mathbf{A}_{:,j}}) (\mathbf{W}_{i,j} + (\mathbf{BA})_{i,j})]^2. \tag{9}$$

Table 2: Zero-shot performance of the compressed LLaMA models fine-tuned on the LaMini dataset. We evaluate WikiText2 and PTB on perplexity with 2048-token segments. The average accuracy is calculated among seven classification datasets. **Bold** denotes the best performance at the same compression rate. ⋆ denotes the results obtained by our reproduction.

Pruning Ratio	Method	WikiText2↓	PTB↓	MMLU (5-shot)	OBQA	ARC-e	WinoGrande	ARC-c	PIQA	HellaSwag	Average↑
Ratio = 0%	LLaMA-7B (Touvron et al., 2023)	5.69	8.93	37.10	42.40	67.45	67.01	67.45	78.35	72.99	65.34
Ratio = 20%	Magnitude ⋆	9.06	13.80	27.84	35.80	65.36	61.33	38.74	74.87	63.90	56.67
	WANDA⋆ (Sun et al., 2023)	8.64	12.66	28.35	35.26	68.96	64.01	38.46	74.80	52.63	58.68
	LLM-Pruner⋆ (Ma et al., 2023)	8.14	12.38	33.67	**38.8**	**70.62**	65.82	40.7	77.37	66.6	62.36
	Compresso (Guo et al., 2023a)	-	-	31.90	36.4	68.64	**67.80**	37.97	75.46	53.44	59.82
	LoRAPrune-8bit (Ours)	7.70	11.91	36.45	38.1	70.25	65.93	41.43	77.10	**68.90**	60.29
	LoRAPrune (Ours)	**7.63**	**11.87**	**36.81**	38.6	70.20	66.77	**41.89**	**77.48**	68.64	**62.70**
Ratio = 30%	Magnitude ⋆	11.38	16.90	26.38	33.67	65.58	60.79	37.47	73.15	60.35	55.16
	WANDA⋆ (Sun et al., 2023)	10.10	15.83	27.90	34.90	65.06	61.16	39.44	74.38	60.84	55.96
	LLM-Pruner⋆ (Ma et al., 2023)	9.36	13.82	30.67	34.86	66.2	63.85	40.55	75.60	65.12	57.70
	Compresso (Guo et al., 2023a)	-	-	27.68	29.8	66.23	**64.80**	37.2	75.63	49.16	53.79
	LoRAPrune-8bit (Ours)	8.83	**13.30**	33.36	**36.40**	69.48	62.31	**41.93**	77.40	65.91	58.90
	LoRAPrune (Ours)	**8.79**	13.33	**33.60**	36.20	69.61	62.75	41.21	**77.48**	**66.68**	**58.98**
Ratio = 50%	Magnitude ⋆	18.36	23.88	21.84	30.26	53.61	55.86	36.98	67.10	53.10	49.48
	WANDA⋆ (Sun et al., 2023)	17.38	21.34	24.15	28.78	52.68	55.98	34.20	70.38	54.12	49.35
	LLM-Pruner⋆ (Ma et al., 2023)	16.41	20.85	25.60	33.12	55.36	56.12	34.98	73.25	58.60	51.90
	LoRAPrune-8bit (Ours)	11.65	17.41	27.71	35.30	**60.54**	56.13	**40.58**	74.89	59.86	54.55
	LoRAPrune (Ours)	**11.60**	**17.39**	**27.84**	**35.80**	60.38	**56.97**	40.12	**75.39**	**60.21**	**54.81**

Table 3: Runtime results of the structured pruned LLMs.

Model	Unmerged time (s) ↓	Merged time (s) ↓	Perplexity ↓	Ratio (%)
LLaMA-7B	0.184(+0.0%)	0.105(+0.0%)	5.69	0
	0.120(-34.8%)	0.079(-24.7%)	7.63	20
	0.089(-51.6%)	0.053(-49.5%)	11.60	50

As shown in Figure 2, the LoRA-guided criterion only needs to compute the gradients of $\mathbf{A}$ and $\mathbf{B}$ with the approximation in Eq. (9), which saves memory and computation compared with the gradients of pre-trained weights $\mathbf{W}_0$.

Progressive pruning. To efficiently obtain group importance for structured pruning, we can substitute Eq. (9) into Eq. (4). However, estimating importance and pruning weights with a single batch of data can lead to significant bias and performance loss. To mitigate this, we apply moving average to evaluate group importance $\mathcal{G}$ and incrementally prune less critical groups. Specifically, the group importance at t-th iteration is computed as follows:

$$\bar{\mathcal{G}}|_t = \lambda \bar{\mathcal{G}}|_{t-1} + (1-\lambda)\hat{\mathcal{G}}|_t. \tag{10}$$

Here, $\bar{\mathcal{G}}|_t$ denotes the group importance scores calculated by Eq. (9) and Eq. (4) at the t-th iteration, and $\lambda \in [0, 1]$ balances the importance between historical and current statistics.

To this end, we can efficiently and accurately estimate the importance of each group. We then prune the unimportant groups by setting a binary mask $\mathbf{M} \in \{0,1\}^{1\times G}$ for each pruned layer. The binary mask $\mathbf{M}$ is obtained by

$$\mathbf{M}_g = \begin{cases} 1 & \bar{\mathcal{G}}_g > p \\ 0 & \bar{\mathcal{G}}_g \leq p \end{cases}, \tag{11}$$

where the index $g \in \{1, \ldots, G\}$ denotes the g-th group in the layer, and p represents the threshold of importance. Groups falling below this threshold will be pruned. After setting the mask, the forward process of each pruned layer can be written as

$$\mathbf{z} = (\mathbf{x}\mathbf{W}_0 + \mathbf{x}\mathbf{B}\mathbf{A}) \odot \mathbf{M}, \tag{12}$$

where $\odot$ denotes Hardamard product and can be calculated by broadcast. The complete algorithm of LoRAPrune is given in Algorithm 1.

4 Experiments

4.1 Experimental Setup

Models and metrics. Our method is applied to the LLaMA-1 model family (Touvron et al., 2023), which comprises LLaMA-7B, LLaMA-13B, LLaMA-30B and LLaMA-65B. Following (Frantar and Alistarh, 2023), we evaluate models on the perplexity metric with WikiText (Merity et al., 2016) and PTB (Marcus et al., 1993) dataset. To assess the zero-shot ability of LLMs, we follow LLaMA to perform zero-shot task classification on common sense reasoning datasets: PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy (Clark et al., 2018), ARC-challenge (Clark et al., 2018), OpenbookQA (Mihaylov et al., 2018). We evaluate the in-context learning ability under a 5-shot setting on MMLU (Hendrycks et al., 2020).

Implementation details. We provide results for LoRAPrune as a single-shot method and with post-training recovery fine-tuning. We iteratively prune models on LaMini instruction dataset (Wu et al., 2023) for LLaMA-7b and 20k sampled C4 dataset (Raffel et al., 2020) for LLaMA-13b, LLaMA-30B and LLaMA-65B. Our training configuration includes a batch size of 128, a learning rate set to

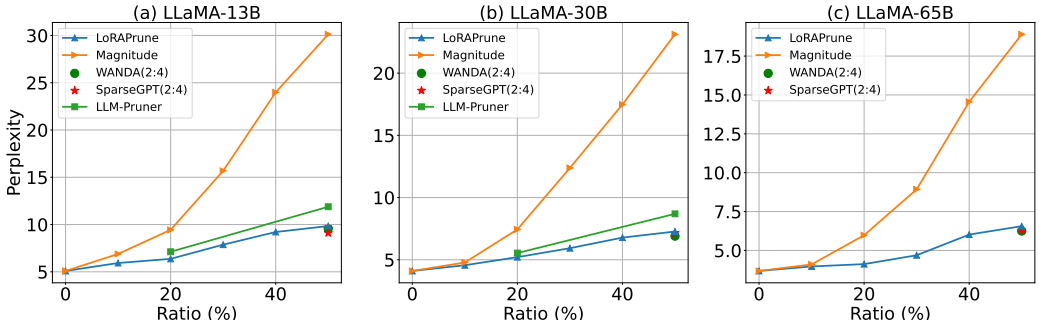

Figure 3: Pruning results on large-scale LLMs: (a) LLaMA-13B, (b) LLaMA-30B, (c) LLaMA-65B.

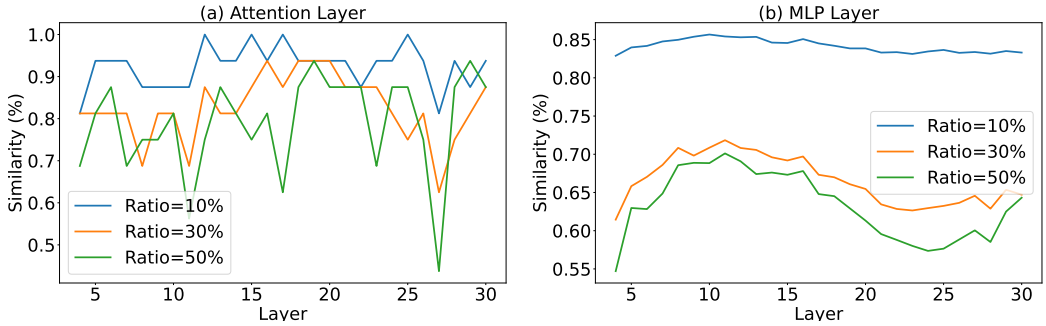

Figure 4: Similarity between LoRA gradient and vanilla criterion on (a) Attention, (b) MLP layers.

1e-4, and a total of 2 training epochs. As the pre-trained weights remain frozen, there is the option to quantize them into 8-bit values to save memory. All models are optimized by AdamW optimizer (He et al., 2020) with a cosine learning rate decay.

Contenders. We compare LoRAPrune with the following pruning methods in both fine-tuning and without fine-tuning settings: **1)** Magnitude Pruning: pruning based on the absolute values of model weights. **2)** LLM-Pruner (Ma et al., 2023): pruning using criterion in Eq. (3). **3)** WANDA (Sun et al., 2023): pruning based on the magnitude of input features and pre-trained weights. **4)** Compresso (Guo et al., 2023a): pruning based on a set of learnable masks.

4.2 Main Results

Zero-shot performance. Table 2 demonstrates the effectiveness of our proposed method. Our LoRAPrune far surpasses other large model pruning methods under structured sparsity. For instance, at a 50% compression rate, LoRAPrune achieves a perplexity of 11.60 on WikiText2, significantly outperforming LLM-Pruner's perplexity of 16.41. We also replicate the experimental results of WANDA under structured pruning scenarios. Our findings reveal that the pruning outcomes achieved by WANDA fell short in comparison to gradient-based pruning methods such as LLM-Pruner and LoRAPrune. This observation underscores the superior performance and effectiveness of gradient-based pruning approaches in our experiments.

It's worth noting that LoRAPrune's efficient approximation for the gradients of the pre-trained weights allows for 8-bit quantization of those weights, greatly reducing the memory requirements for pruning. Moreover, LoRAPrune demonstrates superior pruning results even when models are quantized to 8 bits. These findings underscore the effectiveness and versatility of LoRAPrune in achieving impressive pruning results across various scenarios and compression rates.

Few-shot performance. To verify whether the pruned LLMs retain the in context learning capability, we evaluate on the MMLU with 5-shot. As shown in Table 2, LoRAPrune consistently achieves a higher score than other pruning methods across all sparsity ratios. Notably, LoRAPrune achieves performance on par with the unpruned LLaMA-7B model at a 20% sparsity ratio.

Acceleration for pruned LLMs. Models with structured pruning can be directly sped up in general GPU devices. We conducted tests with 2048 tokens, averaging the results over 100 trials. We specifically examined the inference time with and without merging LoRA weights into the pre-trained weights. As shown in Table 3, we observed that when pruning 20% weights, LLM without merging LoRA has an even slower inference speed than LLM with LoRA merged without pruning. In addition, through structured pruning, the model achieves reductions in inference time of 24.7% and

Table 4: Pruning resource required by different pruning criteria.

Model	Pruning criteria	Fine-tuning Throughput ↓	GPU Memory ↓	Total time ↓	Perplexity ↓
LLaMA-7B (Ratio=50%)	Vanilla	38.87s/iter (+0.0%)	38.6G (+0.0%)	5.3 h (+0.0%)	11.48 (+0.0%)
	Magnitude	13.08s/iter (-66.3%)	16.8G (-56.7%)	1.8 h (-66.04%)	17.38 (+52.9%)
	LoRA-guided	14.13s/iter (-63.6%)	18.3G (-52.6%)	2.0 h (-62.26%)	11.60 (+1.0%)
	LoRA-guided (8-bit)	15.63s/iter (-59.8%)	13.8G (-64.2%)	2.0 h (-62.26%)	12.38 (+9.0%)

49.5% at compression rates of 20% and 50%.

Pruning on large-scale LLMs. Due to the efficient approximation of the pre-trained weights' gradients, LoRAPrune enables iterative pruning on larger-scale LLMs. To ensure that all experiments can be conducted on one GPU, we quantize the pre-trained weights of LLaMA-30b and LLaMA-65b to 8 bits. The experimental results are shown in Figure 3. We observe that, in comparison to the magnitude-based method, LoRAPrune exhibits significant superiority across various scales. Furthermore, in comparison to the 2:4 sparsity model, LoRAPrune achieves comparable pruning results at a 50% sparsity rate. However, it is worth noting that the 2:4 sparsity model also faces challenges in direct weight merging with LoRA, resulting in additional computational overhead during inference. Besides, accelerating 2:4 sparsity models requires specialized hardware support, such as NVIDIA GPUs based on the Ampere architecture, which significantly increases the deployment constraints for 2:4 sparsity models.

4.3 Ablation Study

Efficiency of LoRA-guided criterion vs. vanilla criterion. We conduct a comparative analysis of different pruning criteria with respect to their resource requirements and computational efficiency, including GPU memory and throughput. We adopt the vanilla criterion, as outlined in Eq. (3), as our baseline. For each forward pass, we set the batch size to 1, and we accumulate this process iteratively until we reach a total of 128 accumulations. To ensure robustness and reliability, we compute and subsequently average the results obtained over a span of 100 steps. The comparison results can be found in Table 4. Compared to the vanilla criterion, LoRA-guided and LoRA-guided (8bit) criteria demonstrate a significant reduction in GPU memory usage, saving 52.6% and 64.2% of the memory, respectively. Moreover, as the LoRA-guided criterion does not require the computation of original gradients, it achieves a 64.6% increase in throughput compared to the vanilla criterion with comparable performance, greatly enhancing the speed of the pruning process.

Efficacy of LoRA-guided criterion vs. vanilla criterion. Since the LoRA-guided criterion in Eq. (9) is an efficient approximation of the vanilla criterion in Eq. (3), we evaluate the effectiveness of the proposed LoRA-guided criterion by comparing mask similarity with the vanilla criterion. We randomly sample 128 data and then perform one-shot pruning with both LoRA gradient and vanilla criterion. Figure 4 illustrates that in the case of low compression rates (Ratio=10%), the masks generated by these two criteria exhibit a high degree of consistency. As the compression rates increase, the mask similarity may decrease. However, it is crucial to emphasize that LoRAPrune follows an iterative pruning approach. In each pruning iteration, it only needs to precisely identify the least important weights (about top-5%), thus ensuring the accurate approximation. Hence, the LoRA-guided criterion can attain results that are on par with those of the vanilla criterion while incurring reduced costs.

5 Conclusion

In this paper, we have proposed a method to effectively prune and fine-tune LLMs simultaneously, achieving state-of-the-art efficiency-accuracy trade-offs. Specifically, we have proposed a novel LoRA-guided criterion, for evaluating the parameter importance by only computing the LoRA gradients, which greatly reduces the computational resources required for pruning LLMs. Building upon the proposed criterion, we have presented LoRAPrune, a technique that performs efficient joint pruning and fine-tuning without the need for computing gradients of the pre-trained weights. Finally, comprehensive experiments on various LLMs and benchmarks have demonstrated the superiority of LoRAPrune over other pruning methods. In terms of comparison with the vanilla criterion, the LoRA-guided criterion shows its efficiency and effectiveness. In the future, we aim to further enhance the pruning results of LoRAPrune at higher compression rates.

Limitation. LoRAPrune requires fine-tuning to restore model performance. This limitation can restrict the application of LoRAPrune in scenarios where fine-tuning is unavailable.

References

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proc. AAAI Conf. on Arti. Intel.*, volume 34, pages 7432–7439.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? *Proc. Int. Conf. Mach. Learn. and Syst.*, 2:129–146.

Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. 2022. Adaptformer: Adapting vision transformers for scalable visual recognition. *Proc. Adv. Neural Inf. Process. Syst.*

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*.

Zhenyi Lu Chenghao Fan and Jie Tian. 2023. Chinese-vicuna: A chinese instruction-following llama-based model.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Proc. Adv. Neural Inf. Process. Syst.*, 30.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proc. Annual Associa. Comp. Linguis.*, pages 320–335.

Bryn Elesedy, Varun Kanade, and Yee Whye Teh. 2020. Lottery tickets in linear models: An analysis of iterative magnitude pruning. *arXiv preprint arXiv:2007.08243*.

Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 16091–16101.

Elias Frantar and Dan Alistarh. 2023. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Song Guo, Jiahang Xu, Li Lyna Zhang, and Mao Yang. 2023a. Compresso: Structured pruning with collaborative prompting learns compact large language models. *arXiv preprint arXiv:2310.05015*.

Yuwei Guo, Ceyuan Yang, Anyi Rao, Yaohui Wang, Yu Qiao, Dahua Lin, and Bo Dai. 2023b. Animatediff: Animate your personalized text-to-image diffusion models without specific tuning. *arXiv preprint arXiv:2307.04725*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Proc. Adv. Neural Inf. Process. Syst.*, 28.

Babak Hassibi, David G Stork, and Gregory J Wolff. 1993. Optimal brain surgeon and general network pruning. In *Proc. IEEE Conf. on Neural Networks*, pages 293–299.

Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. 2023. Sensitivity-aware visual parameter-efficient tuning. In *Proc. IEEE Int. Conf. Comp. Vis.*

Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. 2020. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2009–2018.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *Proc. Int. Conf. Learn. Repren.*

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. 2022. Visual prompt tuning. In *Proc. Eur. Conf. Comp. Vis.*

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Proc. Adv. Neural Inf. Process. Syst.*, 2.

Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. 2020. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. 2019. Snip: Single-shot network pruning based on connection sensitivity. In *Proc. Int. Conf. Learn. Repren.*

Guiying Li, Chao Qian, Chunhui Jiang, Xiaofen Lu, and Ke Tang. 2018. Optimization based layer-wise magnitude-based pruning for dnn compression. In *Int. Joi. Conf. on Artificial Intelligence*, pages 2383–2389.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient convnets. In *Proc. Int. Conf. Learn. Repren.*

Yuchao Li, Fuli Luo, Chuanqi Tan, Mengdi Wang, Songfang Huang, Shen Li, and Junjie Bai. 2022. Parameter-efficient sparsity for large language models fine-tuning. *arXiv preprint arXiv:2205.11005*.

Gen Luo, Minglang Huang, Yiyi Zhou, Xiaoshuai Sun, Guannan Jiang, Zhiyu Wang, and Rongrong Ji. 2023. Towards efficient visual adaption via structural reparameterization. *arXiv preprint arXiv:2302.08106*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 11264–11272.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource efficient inference. In *Proc. Int. Conf. Learn. Repren.*

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. 21(1):5485–5551.

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

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Proc. Adv. Neural Inf. Process. Syst.*, 33:20378–20389.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

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

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

Chaoqi Wang, Guodong Zhang, and Roger Grosse. 2020. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*.

Chen Henry Wu, Saman Motamed, Shaunak Srivastava, and Fernando D De la Torre. 2022. Generative visual prompt: Unifying distributional control of pre-trained generative models. *Proc. Adv. Neural Inf. Process. Syst.*, 35:22422–22437.

Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Fikri Aji. 2023. Lamini-lm: A diverse herd of distilled models from large-scale instructions. *CoRR*, abs/2304.14402.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

Haoran You, Zhanyi Sun, Huihong Shi, Zhongzhi Yu, Yang Zhao, Yongan Zhang, Chaojian Li, Baopu Li, and Yingyan Lin. 2023. Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design. In *Proc. IEEE Int. Sym. on High-Perf. Comp. Arch.*, pages 273–286. IEEE.

Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. 2022a. Width & depth pruning for vision transformers. In *Proc. AAAI Conf. on Arti. Intel.*, volume 36, pages 3143–3151.

Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. 2022b. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *Proc. Int. Conf. Mach. Learn.*, pages 25668–25683.

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

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2022. Platon: Pruning large transformer models with upper confidence bound of weight importance. In *Proc. Int. Conf. Mach. Learn.*, pages 26809–26823.

Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *Proc. IEEE Int. Sym. on High-Perf. Comp. Arch.*, pages 1071–1085. IEEE.

Appendix

A Weight Dependency for LLaMA

Here, we use LLaMA architecture as an example to explain the weight dependency. The dependency details are shown in Figure 5. In terms of the Attention module, when we decide to prune a specific head of weights in the Query layer, it is imperative that the corresponding weights with the same index in the Key, Value and Out layers are also pruned. Similarly, for the Feed-Forward Network (FFN) module, when pruning a particular channel of weights in the Up layer, it is essential to prune the weights with matching indices in the Gate and Down layers. This meticulous coordination ensures that pruning maintains the structural integrity and functionality of the model. Following (Ma et al., 2023) and (Fang et al., 2023), we prune heads for Attention and channels for FFN, respectively. The dependency details are shown in Figure 5.

B More Ablation Studies

Pruning on 20k sampled C4 dataset. We also evaluate LoRAPrune on a tiny dataset that randomly samples 20k data from C4 dataset. As presented in Table 5, LoRAPrune outperforms both LLM-Pruner and WANDA across the majority of zero-shot reasoning datasets, thereby securing the highest average score overall. Specifically, LoRAPrune exceeds the performance of LLM-Pruner by margins of 0.82% and 1.02%, respectively.

Effectiveness of the moving average. We verify the rationale behind the moving average through the setting of different values for λ. These experiments were conducted on LLaMA-7b with 20k sampled C4 dataset. The experimental results, as shown in Figure 6 (a), reveal that as λ increases, the pruning results exhibit a significant reduction in perplexity. This effect is especially pronounced when $\lambda = 0$ where pruning is solely determined by the importance of the current batch, confirming the effectiveness of the moving average.

Impact of iterations. To assess the impact of the pruning iterations on pruning results, we conducted experiments on the LLaMA-7b model with different iterations on 20k sampled C4 dataset. The results are shown in Figure 6 (b), which indicates that excessive iterations can lead to a decrease in the model's zero-shot performance, potentially due to overfitting on the calibration dataset. Furthermore, we observe that the model requires more iterations to regain its performance when pruning with high compression (*e.g.*, ratio=50%).

LoRAPrune vs. LLM-Pruner with gradients off-loading. The gradient off-loading strategy can partially mitigate LLM-Pruner's memory demands, such as transferring certain gradients to CPU memory. However, the memory access cost and computational overhead are substantial. Table 6 shows LoRAPrune outperforms LLM-Pruner in efficiency, being 8.19× faster with CPU offloading and 2.75× faster without it. This speed allows iterative pruning to counteract the performance drop due to structured sparsity.

Joint vs. separate. To demonstrate the necessity of integrating pruning and fine-tuning, we conducted experiments that sequentially performed pruning followed by fine-tuning, specifically applying one-shot pruning to the LLaMA-7b model and then employing LoRA fine-tuning to recover the model's performance. The experimental results presented in Table 7 indicate that joint pruning and fine-tuning yields much better performance than the separate counterpart, especially under the high compression ratio.

Pruning frequency. We explore the impact of different pruning frequencies, *i.e.*, how many iterations of fine-tuning before pruning, on the final performance. The experimental results, as shown in Table 8, indicate that our default frequency (frequency=10) obtains the best pruning result. Additionally, we observe that if pruning is too frequent (frequency=1), the model may not have enough iterations to recover through fine-tuning, leading to inaccurate importance estimation. Furthermore, excessive fine-tuning between pruning iterations (frequency=20) leads to overfitting on the calibration data.

C Generative Results

We show the generalization capability of the LoRAPrune by some instructions encompass common sense, translation, and coding tasks in Table 9.

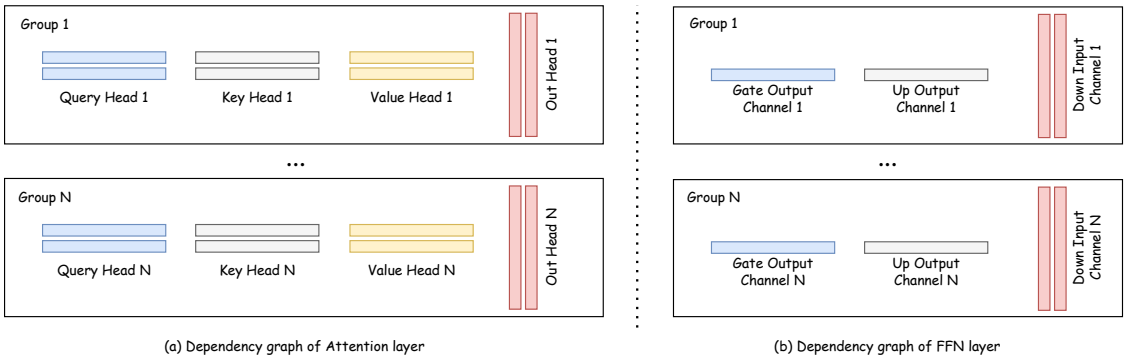

Figure 5: Weight dependency in (a) Attention layer, (b) FFN layer.

Table 5: Zero-shot performance of the compressed LLaMA models fine-tuned on the 20k sampled C4 dataset. The average accuracy is calculated among seven classification datasets. **Bold**/ denotes the best performance at the same compression rate. ⋆ denotes the results obtained by our reproduction.

Pruning Ratio	Method	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average↑
Ratio = 0%	LLaMA-7B (Touvron et al., 2023)	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
Ratio = 20%	Magnitude ⋆	61.89	70.81	58.34	56.87	54.87	34.02	38.40	53.59
	WANDA⋆ (Sun et al., 2023)	65.75	74.70	64.52	59.35	60.65	36.26	39.40	57.23
	LLM-Pruner (Ma et al., 2023)	64.62	77.20	68.80	63.14	64.31	36.77	**39.80**	59.23
	LoRAPrune-8bit (Ours)	65.37	76.65	69.41	**63.78**	65.45	36.12	39.50	59.46
	LoRAPrune (Ours)	**65.62**	**79.31**	**70.00**	62.76	**65.87**	**37.69**	39.14	**60.05**
Ratio = 50%	Magnitude ⋆	47.40	54.36	33.49	53.10	37.88	26.60	30.12	40.42
	WANDA ⋆ (Sun et al., 2023)	50.90	57.38	38.12	55.98	42.68	34.20	38.78	45.43
	LLM-Pruner (Ma et al., 2023)	60.28	69.31	47.06	53.43	45.96	29.18	**35.60**	48.69
	LoRAPrune-8bit (Ours)	61.43	70.88	47.65	**55.12**	**45.78**	30.50	35.62	49.56
	LoRAPrune (Ours)	**61.88**	**71.53**	**47.86**	55.01	45.13	**31.62**	34.98	**49.71**

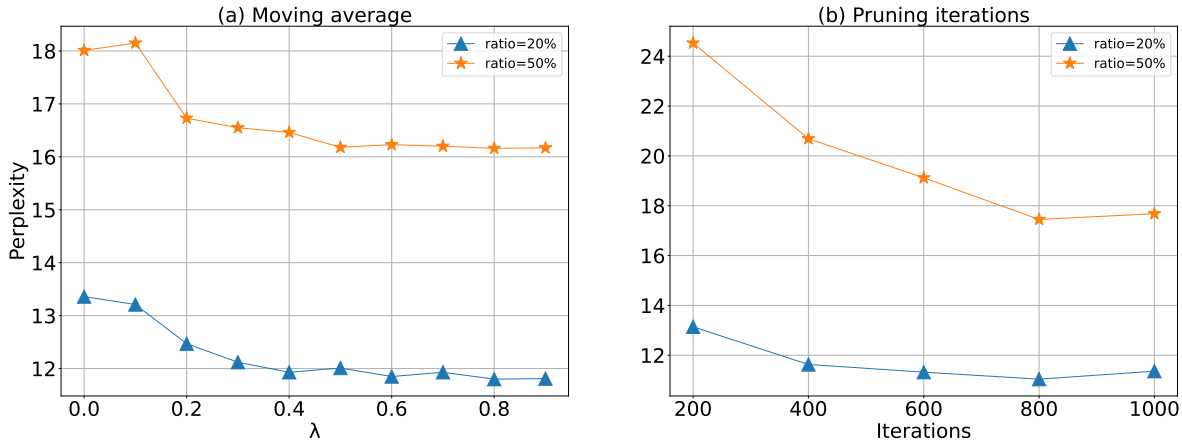

Figure 6: More ablation studies for pruning hyper-parameters: (a) λ value in moving average, (b) fine-tuning iterations.

Table 6: Efficiency comparison between LoRAPrune and LLM-Pruner with CPU off-loading.

Method	Throughput (s/iter)	GPU Memory (GB)	FLOPs (G)	Total time (h)	Pruning time (h)	Fine-tuning time (h)
LLM-Pruner	38.87	38.6	20298	5.3	3.5	1.8
LLM-Pruner + CPU offloading	115.67	19.5	20298	25.8	24	1.8
LoRAPrune (Ours)	**14.13**	**18.3**	**12881**	**2.0**	**0.2**	**1.8**

Table 7: Effect of the joint pruning and fine-tuning. "Average" represents the average performance on seven classification datasets.

	Method	WikiText2↓	PTB↓	Average↑
Ratio=20%	Joint	**12.93**	**22.52**	**60.05**
	Separate	14.51	24.30	57.18
Ratio=50%	Joint	**18.37**	**28.68**	**49.71**
	Separate	21.78	40.39	47.56

Table 8: Results under different pruning frequencies. "Average" denotes the average performance on seven classification datasets.

Frequency	WikiText2↓	PTB↓	Average↑
1	18.57	29.02	49.37
5	18.63	28.73	49.16
10	**18.21**	**28.68**	49.59
20	18.47	28.70	**49.71**

Table 9: Generated examples from the Pruned models

Model	Generated Sentences
LLaMA-7b (ratio=0%)	Instruction: Tell me about alpacas. Response: Alpacas are members of the camelid family and are native to the Andes Mountains of South America. They are known for their soft, luxurious fleece, which is used to make clothing, blankets, and other items. Alpacas are herd animals and live in groups of up to 20 individuals. They are generally docile and easy to care for, making them popular as pets and livestock. Alpacas can live up to 20 years in captivity, and their fleece can be sheared up to three times a year.
	Instruction: Translate the sentence 'I have no mouth but I must scream' into Spanish. Response: I no tengo boca pero tengo que gritar.

	Instruction: Write a Python program that prints the first 10 Fibonacci numbers. Response:
	def fibonacci(n):
	  if n == 0:
	  return 1
	  if n == 1:
	  return 1
	  else:
	  return fibonacci(n - 1) + fibonacci(n - 2)
	for n in range(1, 10):
	  print(fibonacci(n))

LLaMA-7b (ratio=20%)	Instruction: Tell me about alpacas. Alpacas are a species of camelid native to South America. They are known for their soft, luxurious wool, which is used to make clothing, blankets, and other textiles. Alpacas are also used for their meat, which is considered a delicacy in some parts of the world. Alpacas are social animals that live in herds and are known for their gentle demeanor. They are also known for their intelligence and ability to be trained for various tasks, such as pulling carts or performing tricks for entertainment.
	Instruction: Translate the sentence 'I have no mouth but I must scream' into Spanish. Response: Yo no tengo la boca pero tengo que gritar.

	Instruction: Write a Python program that prints the first 10 Fibonacci numbers. Response:
	def fibonacci(n):
	  if n == 0 or n == 1:
	  return 1
	  else:
	  return fibonacci(n - 1) + fibonacci(n - 2)
	for n in range(1, 10):
	  print(fibonacci(n))