SPP: Sparsity-Preserved Parameter-Efficient Fine-Tuning for Large Language Models

Xudong Lu [* 1] **Aojun Zhou** [* 1] **Yuhui Xu** [* 2] **Renrui Zhang** [1 3] **Peng Gao** [3] **Hongsheng Li** [1 4]

Abstract

Large Language Models (LLMs) have become pivotal in advancing the field of artificial intelligence, yet their immense sizes pose significant challenges for both fine-tuning and deployment. Current post-training pruning methods, while reducing the sizes of LLMs, often fail to maintain their original performance. To address these challenges, this paper introduces SPP, a **S**parsity-**P**reserved **P**arameter-efficient fine-tuning method. Different from existing post-training pruning approaches that struggle with performance retention, SPP proposes to employ lightweight learnable column and row matrices to optimize sparse LLM weights, *keeping the structure and sparsity of pruned pre-trained models intact*. By element-wise multiplication and residual addition, SPP ensures the consistency of model sparsity pattern and ratio during both training and weight-merging processes. We demonstrate the effectiveness of SPP by applying it to the LLaMA and LLaMA-2 model families with recent post-training pruning methods. Our results show that SPP significantly enhances the performance of models with different sparsity patterns (i.e. unstructured and N:M sparsity), especially for those with high sparsity ratios (e.g. 75%), making it a promising solution for the efficient fine-tuning of sparse LLMs. Code will be made available at `https://github.com/Lucky-Lance/SPP`.

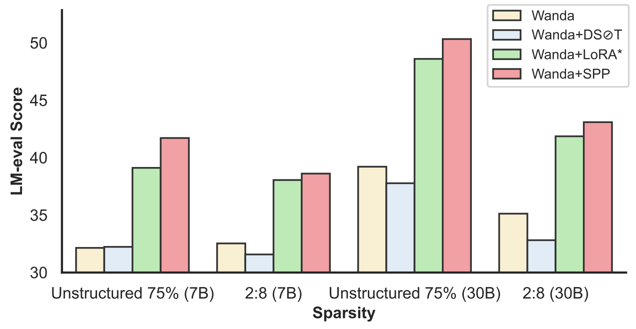

Figure 1. We use LLaMA 7B/30B models at 75% sparsity and test zero-shot accuracies on 7 benchmarks of LM-eval (Gao et al., 2021) to compare different pruning methods. The results of Wanda, Wanda+DS⊘T, Wanda+LoRA*, and Wanda+SPP are visualized. The first two approaches are post-training pruning schemes, LoRA* denotes applying the original Wanda pruning masks to sparse the dense model after LoRA training. Our method achieves overall best results. More experiment details are illustrated in Sec. 4.

1. Introduction

Large language models (LLMs) (Brown et al., 2020; OpenAI, 2023; Anil et al., 2023) have recently shown impressive success in various complex tasks (Wei et al., 2022; Zhou et al., 2024). However, these models are usually characterized by an extensive number of learnable parameters, ranging from several billions to around a hundred billions (Touvron et al., 2023a;b), as exemplified by GPT-4. This enormity makes LLMs cumbersome to be fine-tuned for different scenarios and challenging to deploy on various edge devices.

To reduce the parameters of *pre-trained* large language models without the demanding retraining phase, different post-training pruning (Frantar & Alistarh, 2023) approaches have been presented. SparseGPT, as outlined in (Frantar & Alistarh, 2023), focuses on minimizing the squared errors between pruned and original dense models in a layer-by-layer manner. Wanda (Sun et al., 2023) incorporates both weight magnitude and input activation as metrics for identifying unimportant parameters. Despite their success in language model pruning, these methods often fail to maintain the performance of pre-trained models at even moderate sparsity levels (e.g., 50%) (Jaiswal et al., 2023).

In contrast, pruning methods for smaller-scale deep models – those with fewer than 200 million parameters, such as

[*]Equal contribution [1]Multimedia Laboratory (MMLab), The Chinese University of Hong Kong [2]Salesforce AI Research [3]Shanghai Artificial Intelligence Laboratory [4]CPII under InnoHK. Correspondence to: Hongsheng Li <hsli@ee.cuhk.edu.hk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

ResNet50 (He et al., 2016) and BERT-based models (Devlin et al., 2018) – have realized high sparsity ratios (e.g., >80%) only with negligible performance drop (Evci et al., 2020; Lin et al., 2020; Zhou et al., 2021). The success of these models is largely attributed to the role of **retraining** during their pruning process (Liu et al., 2018). However, compared with training dense neural networks, it will take much longer to train the sparse models to achieve the same performance. For example, 5× more training time is needed in RigL (Evci et al., 2020). These methods heavily rely on back-propagation with full parameters, which is prohibitively expensive for LLMs. This observation raises a crucial question: *Can we introduce an efficient retraining stage for the pruned LLMs?*

In response to these challenges, we propose a novel, plug-and-play method, **SPP**, a **S**parsity-**P**reserved **P**aramater-efficient fine-tuning method designed to effectively retrain or fine-tune sparse LLMs after post-training pruning (e.g. SparseGPT (Frantar & Alistarh, 2023), Wanda (Sun et al., 2023), etc), thereby enhancing their performances. Inspired by the Low-Rank Adaptation (LoRA) method for dense large language models (Hu et al., 2021), **SPP** consists of two phases, training and weight-merging. More specifically, we introduce two sets of lightweight learnable parameters to the sparse matrices of each linear layer. During training and weight-merging phases, these learnable parameters are multiplied with the original frozen post-training pruned weights, achieving the effect of *exactly maintaining the sparse pattern and ratio* throughout all the processes.

SPP is easy to implement and can be applied to a wide range of post-training pruning methods and LLMs with various sparsity ratios and patterns (unstructured and N:M sparsity). We evaluate SPP on the LLaMA (Touvron et al., 2023a) and LLaMA-2 (Touvron et al., 2023b) model families with two recent LLM post-training pruning methods, i.e., SparseGPT (Frantar & Alistarh, 2023) and Wanda (Sun et al., 2023). To illustrate the effectiveness of SPP, zero-shot evaluation results of LLaMA 7B/30B models with 75% sparsity ratio are shown in Fig. 1.

The main contributions of this paper are summarized in three key aspects:

(1) We investigate model pruning methods in the era of LLMs and present a novel parameter-efficient fine-tuning algorithm, SPP, which can maintain model structure and sparsity during both training and weight-merging phases.

(2) Extensive experiments on different post-training pruned LLMs with various sparsity patterns and ratios show the effectiveness of SPP for the efficient training of sparse LLMs.

(3) To the best of our knowledge, this study is the first to systematically explore integrating efficient retraining with advanced post-training pruning methods for LLMs.

2. Related Work

Traditional Model Pruning: Pruning of Deep Neural Networks (DNNs) is a promising direction to compress and accelerate deep learning models (Hoefler et al., 2021). There are mainly two types of techniques to obtain a sparse neural network, iterative pruning-retraining framework, and dynamic sparse training ones. The iterative pruning-retraining framework first finds the unimportant connections (masks), thereby removing the corresponding weights, and then retrains the fixed sparse network to recover its performance. Typical methods include iterative pruning (Han et al., 2015). Later, the lottery ticket hypothesis (Frantar & Alistarh, 2023) shows that the sparse sub-networks (winning tickets) can be trained from scratch with the same dense initialization while the winning tickets are discovered by dense training. Dynamic sparse training methods (Mocanu et al., 2018) start from a randomly sparsified network, then subsequently prune and grow connections during training. These methods can be applied end-to-end within the network training stage and have achieved promising results. Recently proposed state-of-the-art method STR (Kusupati et al., 2020) introduces learnable pruning thresholds to obtain a non-uniform sparse network. RigL (Evci et al., 2020) uses the magnitude-based method to prune and the periodic dense gradients to regrow connections. However, it is imperative to note that these methods heavily rely on back-propagation with full parameters, which is prohibitively expensive for LLMs.

LLM Pruning: In the era of LLMs, methods have been proposed to overcome the challenges mentioned above (Li et al., 2023). Recent research endeavors have evolved towards post-training pruning methods, which start from the pre-trained network and remove redundant parameters *without end-to-end fine-tuning or retraining*. SparseGPT uses second-order information to address a layer-wise reconstruction problem and prunes large models with unstructured and N:M structured sparsity (Zhou et al., 2021) respectively. Wanda (Sun et al., 2023) proposes a new pruning metric that takes both weight magnitude and their corresponding input activations into consideration, achieving comparable perplexity with SparseGPT (Frantar & Alistarh, 2023). However, (Jaiswal et al., 2023) points out that perplexity is not necessarily an accurate metric for evaluating the effectiveness of model compression, with both SparseGPT and Wanda fail to achieve satisfactory performance even with low-level sparsity (25-30%). Based on this issue, we speculate that the lack of *retraining* after removing the unimportant weights will lead to an undesirable decline in performance, and put forward a novel training method with high *training efficiency*.

Parameter-efficient Fine-tuning: In different language and vision tasks, the pre-training and fine-tuning paradigms have been proven to be highly effective. Compared with

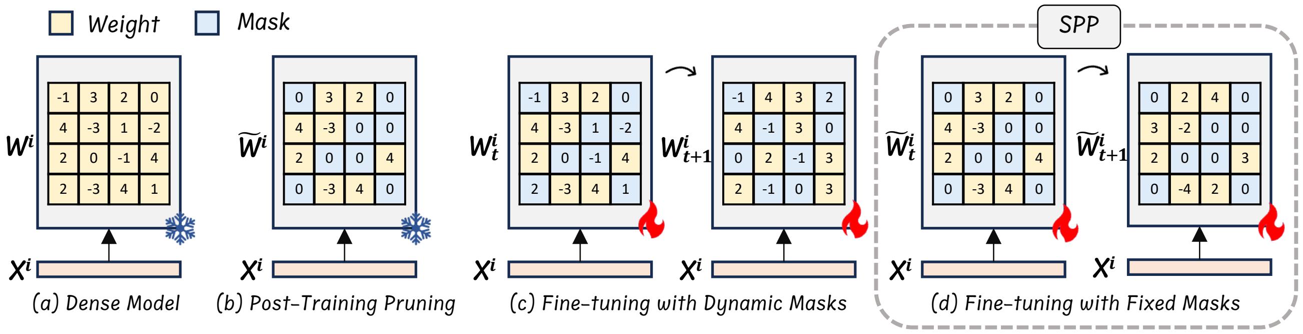

Figure 2. Comparison between different methods for sparsification of neural networks. (a) A dense linear layer with matrix $\mathbf{W}^i$ and activation $\mathbf{X}^i$. (b) Post-training pruning methods leverage weight magnitude and calibration data for weight pruning. (c) Model fine-tuning with dynamic weight masks, typically referred to as full fine-tuning methods in the literature, adaptively changes the weight masks during the fine-tuning process. (d) Model fine-tuning with fixed weight masks. Our proposed SPP distinguishes itself as a parameter-efficient fine-tuning algorithm that can consistently maintain model sparsity during both the training and weight-merging phases.

full parameter fine-tuning, Parameter-Efficient Fine-Tuning (PEFT) (Mangrulkar et al., 2022; Xu et al., 2023) methods freeze most parameters of **dense** pre-trained models and aim to exhibit comparable capabilities on downstream tasks. LoRA (Hu et al., 2021) introduces trainable low-rank decomposition matrices into dense network weights. Adapters (Houlsby et al., 2019) insert lightweight adaption modules into each block of the language models. Different from previous efforts for dense pre-trained language models, we propose the SPP method with few learnable parameters, which is specially designed for sparse LLMs.

3. Method

The objective of neural network pruning is to preserve the performance of the original network as closely as possible by creating a sparse network through the selective removal of certain neural network parameters (Hassibi et al., 1993; Han et al., 2015). In this section, we first introduce the notation used in our paper and categorize some existing algorithms for neural network pruning (especially for LLMs) in Sec. 3.1, and then elaborate on our proposed SPP in Sec. 3.2, which is a parameter-efficient fine-tuning method specifically designed for training sparse LLMs.

3.1. A Revisit of Model Pruning Methods

Post-Training Pruning: Starting from a dense pre-trained model (LLMs in our setting), post-training pruning (Frantar & Alistarh, 2023; Zhang et al., 2023a) algorithms remove redundant parameters in a neural network by calculating a set of weight masks $\mathbf{M} = \{\mathbf{M}^0, \mathbf{M}^1, \dots, \mathbf{M}^{N-1}\}$ leveraging weight magnitude $\mathbf{W} = \{\mathbf{W}^0, \mathbf{W}^1, \dots, \mathbf{W}^{N-1}\}$ together with a set of calibration data. As shown in Fig. 2 (a), given original dense matrix $\mathbf{W}^i$ $(0 \le i \le N-1)$, post-training pruning algorithms compute binary weight masks $\mathbf{M}^i$, and the new set of sparse matrices $\widetilde{\mathbf{W}}$ can be calculated as:

$$\widetilde{\mathbf{W}} = \{\mathbf{W}^0 \odot \mathbf{M}^0, \mathbf{W}^1 \odot \mathbf{M}^1, ..., \mathbf{W}^{N-1} \odot \mathbf{M}^{N-1}\}, \quad (1)$$

where N denotes the number of linear layers, and $\odot$ indicates element-wise multiplication (Hadamard product (Horn, 1990)) between matrices. One example based on 2:4 structured sparsity and using weight magnitude as the pruning metric is illustrated in Fig. 2 (b). Post-training pruning algorithms are known for their relatively low consumption of computational resources. However, as demonstrated in SparseGPT (Frantar & Alistarh, 2023) and Wanda (Sun et al., 2023), even the pruning of LLMs to a moderate level of sparsity, specifically to 50%, inevitably leads to a substantial degradation in performance (Han et al., 2015). This indicates that after post-training pruning, it is essential to retrain the sparse model to reclaim its knowledge capabilities.

Full Fine-tuning with Dynamic Sparse Masks: Another pruning scheme that starts from a dense model is to full-finetune the original dense model with dynamically updated sparse masks, as depicted in Fig. 2 (c). Typically, to improve the performance of sparse models, researchers introduce various training methods to optimize the sparse weights $\widetilde{\mathbf{W}}$ and the corresponding binary masks $\mathbf{M}$ simultaneously (Lin et al., 2020; Bengio et al., 2013):

$$\mathbf{W}^i_{t+1} = \mathbf{W}^i_t - \gamma_t \mathbf{g}(\mathbf{W}^i_t \odot \mathbf{M}^i_t) = \mathbf{W}^i_t - \gamma_t \mathbf{g}(\widetilde{\mathbf{W}}^i_t), \quad (2)$$

where γ_t denotes the learning rate at time step t and function $\mathbf{g}(\cdot)$ calculates the gradients[1]. For small-scale sparse models, such as those based on ResNet and BERT, full fine-tuning methods like STE-based algorithms (Lin et al., 2020; Evci et al., 2020; Zhou et al., 2021) can maintain high performance at even $\ge 75\%$ sparsity. However, these methods usually require substantial computational power

[1]We only show the equation of gradient descent for simplicity.

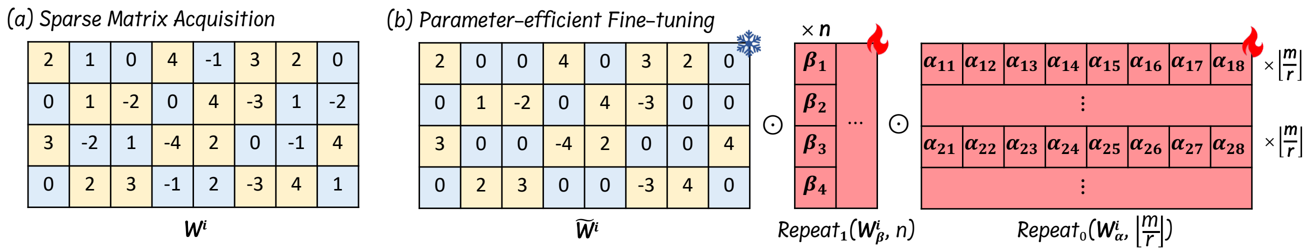
(a) Sparse Matrix Acquisition
(b) Parameter-efficient Fine-tuning
W^i
$\widetilde{W}^i$
$\times n$
β_1 β_2 β_3 β_4
α_{11} α_{12} α_{13} α_{14} α_{15} α_{16} α_{17} α_{18}
α_{21} α_{22} α_{23} α_{24} α_{25} α_{26} α_{27} α_{28}
$\times\lfloor \frac{m}{r} \rfloor$
$Repeat_1(W^i_\beta, n)$
$Repeat_0(W^i_\alpha, \lfloor \frac{m}{r} \rfloor)$

Figure 3. Learnable parameter insertion of our proposed SPP for a $m \times n$ $(m = 4, n = 8)$ weight matrix. (a) We acquire the mask $\mathbf{M}^i$ of the sparse matrix of a linear layer by pruning with post-training algorithms (e.g. SparseGPT (Frantar & Alistarh, 2023), Wanda (Sun et al., 2023)), and obtain sparse linear weights $\widetilde{\mathbf{W}}^i = \mathbf{W}^i \odot \mathbf{M}^i$. (b) SPP adds two sets of parameters to the sparse matrix, one ($\mathbf{W}^i_\beta \in \mathbb{R}^{m\times 1}$) initialized to 0 and multiplied on each row of the matrix, and one ($\mathbf{W}^i_\alpha \in \mathbb{R}^{r\times n}$, $r = 2$ in this example) randomly initialized and multiplied on the columns of the matrix. We then scale $\mathbf{W}^i_\alpha$ and $\mathbf{W}^i_\beta$ to the size of $(m \times n)$ and element-wise multiply then with $\widetilde{\mathbf{W}}^i$.

and memory resources. This poses a significant challenge in the era of training LLMs. For instance, fully fine-tuning a LLaMA-65B model requires at least 780G of GPU memory (Dettmers et al., 2023), not yet including the additional memory needed to update the sparse masks.

Fine-tuning with Fixed Masks: As can be learned from the respective drawbacks of the two approaches described above, there is a need to balance model performance restoration and the training overhead. Actually, Eq. (2) can be decoupled into the iterative optimization of binary masks $\mathbf{M}$ and sparse weights $\widetilde{\mathbf{W}}$. When training with limited computation resources, optimizing sparse weights with fixed masks is considered effective for knowledge restoration (Liu et al., 2018). Generally, if we fix $\mathbf{M}^i_t = \mathbf{M}^i_0$ and apply $\widetilde{\mathbf{W}}^i_t = \mathbf{W}^i_t \odot \mathbf{M}^i_0$ at the t-th iteration, Eq. (2) will be equivalent to the retraining of sparse models with fixed masks:

$$\widetilde{\mathbf{W}}^i_{t+1} = \widetilde{\mathbf{W}}^i_t - \gamma_t \mathbf{g}(\widetilde{\mathbf{W}}^i_t) \odot \mathbf{M}^i_0, \tag{3}$$

but it is still cumbersome to calculate gradients for $\widetilde{\mathbf{W}}^i_t$ associated with fixed mask $\mathbf{M}^i_0$. To this end, we provide the SPP method for efficient fine-tuning post-training pruned LLMs with fixed masks (as illustrated in Fig. 2 (d)).

3.2. Proposed Method

In this subsection, we propose SPP to balance model performance restoration and computational overhead. Current PEFT methods carry out weight updates of huge linear blocks by introducing trainable low-rank decomposition matrices (Hu et al., 2021) and adding them with original frozen linear weights after training. This approach finally results in dense matrices and destroys model sparsity. Reflecting on this issue, we suggest that the direct addition of new parameters to sparse matrices leads to the inevitable change in model sparsity, whereas weight multiplication can avoid the problem. To this end, we add two sets of lightweight *orthogonal* learnable parameters $\mathbf{W}_\alpha = \{\mathbf{W}^0_\alpha, \mathbf{W}^1_\alpha, \ldots, \mathbf{W}^{N-1}_\alpha\}$ and $\mathbf{W}_\beta = \{\mathbf{W}^0_\beta, \mathbf{W}^1_\beta, \ldots, \mathbf{W}^{N-1}_\beta\}$, multiply them separately with column and row weights of each linear matrix in the pruned LLM, and perform efficient fine-tuning for sparse matrices with the original pruned weights fixed.

Learnable Parameter Insertion: The core of SPP is how to introduce learnable parameters such that they do not lead to changes in sparsity during the training and weight-merging process. As shown in Fig. 3, given a sparse linear matrix $\widetilde{\mathbf{W}}^i \in \mathbb{R}^{m\times n}$ $(0 \le i \le N-1)$ after pruning, we insert two sets of learnable parameters: $\mathbf{W}^i_\alpha \in \mathbb{R}^{r\times n}$ and $\mathbf{W}^i_\beta \in \mathbb{R}^{m\times 1}$. $\mathbf{W}^i_\alpha$ adds learnability to the weights of each column, while $\mathbf{W}^i_\beta$ further expands it to each row[2]. This leads to $(m + rn)$ additional parameters for one sparse matrix. We then scale $\mathbf{W}^i_\alpha$ and $\mathbf{W}^i_\beta$ to the same size of $\widetilde{\mathbf{W}}^i$ and multiply them with $\widetilde{\mathbf{W}}^i$ in an element-wise manner:

$$\widetilde{\mathbf{W}}^{i\prime} = \widetilde{\mathbf{W}}^i \odot \text{Repeat}_0(\mathbf{W}^i_\alpha, \left\lfloor \frac{m}{r} \right\rfloor) \odot \text{Repeat}_1(\mathbf{W}^i_\beta, n), \tag{4}$$

where $\text{Repeat}_d(\mathbf{W}, k)$ is a function denoting the repetition of each element of $\mathbf{W}$ by k times along the d-th dimension, which can be easily implemented by the `torch.repeat_interleave()` function in Python. Besides, there is no need to perform an *explicit* "repeat" operation on $\mathbf{W}^i_\beta$, as PyTorch automatically aligns the matrix dimensions when conducting Hadamard product between a matrix and a vector. The floor function $\lfloor\cdot\rfloor$ indicates integer division. In our implementation, r is set to the numbers that can be divided by n (e.g., choosing $r = 4, 8, 16$, etc.). Benefited from element-wise multiplication, the values of 0 in the original sparse matrix will remain unchanged, so $\widetilde{\mathbf{W}}^{i\prime}$ shares the same sparsity pattern and sparsity ratio with $\widetilde{\mathbf{W}}^i$, one example with a 4×8 matrix and $r = 2$ is shown in Fig. 3. In this way, we can train the linear matrices with fixed masks, following the training pattern in Fig. 2 (d). It is worth noting that if we set $r = m$, our method performs a full-parameter gradient update on the original sparse matrix.

[2]We set $r > 1$ to increase capacity. We discuss setting $\mathbf{W}^i_\alpha \in \mathbb{R}^{r\times n}$ instead of $\mathbf{W}^i_\beta \in \mathbb{R}^{m\times r}$ in Sec. A.1 in the Appendix.

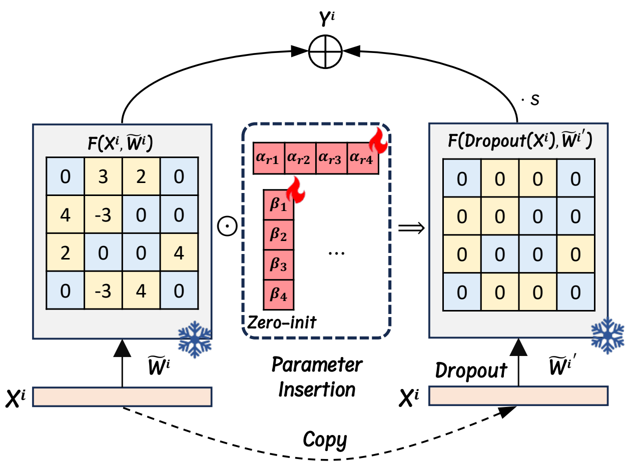

Figure 4. Our proposed SPP framework for sparse weight models. We multiply two sets of learnable parameters to the frozen linear weights with $\mathbf{W}_\alpha$ randomly initialized and $\mathbf{W}_\beta$ zero-initialized (detailed in Fig. 3). Linear operation is conducted on both the original and modified weights, and then the results are added in a residual manner. This framework corresponds to Eq. (5).

Framework of SPP: After explaining how to insert parameters, we introduce our proposed SPP framework, as shown in Fig. 4. During the training process, in the i-th linear layer of the neural network, the output can be calculated by:

$$\mathbf{Y}^i = \mathbf{F}(\mathbf{X}^i, \widetilde{\mathbf{W}}^i) + s \cdot \mathbf{F}(\text{Dropout}(\mathbf{X}^i), \widetilde{\mathbf{W}}^{i'}), \quad (5)$$

where s is a hyper-parameter for scaling, and $\mathbf{X}^i \in \mathbb{R}^{b\times n}$ denotes the input activation at the i-th linear layer. We follow Eq. (4) for the training of $\widetilde{\mathbf{W}}^{i'}$, set initial $\mathbf{W}^i_\beta$ to all zeros, and randomly initialize $\mathbf{W}^i_\alpha$. In this way, the *initial structure and weights of the network are consistent with the post-training pruned model*. More specifically, $\mathbf{Y}^i = \mathbf{F}(\mathbf{X}^i, \widetilde{\mathbf{W}}^i)$ for neural network initialization. This strategy results in more stable training. After training, the linear layer parameters can be merged as: $\widetilde{\mathbf{W}}^i + s \cdot \widetilde{\mathbf{W}}^{i'}$, sharing the same sparsity pattern and ratio with the original pruned model. The structure of SPP is similar to LoRA, but it does not destroy the sparse structure of the model during both the training and weight-merging process. For a more straightforward comparison, recall that the formulation of LoRA with frozen weight $\widetilde{\mathbf{W}}^i$ is:

$$\mathbf{Y}^i = \mathbf{F}(\mathbf{X}^i, \widetilde{\mathbf{W}}^i) + s \cdot \text{Dropout}(\mathbf{X}^i)(\mathbf{A}^i)^T(\mathbf{B}^i)^T, \quad (6)$$

where $\mathbf{A}^i \in \mathbb{R}^{r\times n}$ and $\mathbf{B}^i \in \mathbb{R}^{m\times r}$. During the training and weight-merging process of LoRA, the weights of the linear layer matrix are equivalent to $(\widetilde{\mathbf{W}}^i + s\cdot\mathbf{B}^i\mathbf{A}^i)$, which is actually a dense matrix.

Memory Usage Optimization: We find that for each linear layer with input activation $\mathbf{X}^i$, the insertion of $\mathbf{W}^i_\alpha$ and $\mathbf{W}^i_\beta$ greatly reduces the number of trainable parameters, but we still need to store the intermediate matrix $\text{Repeat}_0(\mathbf{W}^i_\alpha, \lfloor\frac{m}{r}\rfloor)$ when calculating the linear function $\mathbf{Y}^{i'} = \mathbf{X}^i(\widetilde{\mathbf{W}}^{i'})^T$ (this is actually the second addition term in Eq. (5)), which usually takes a huge amount of memory. To overcome this problem, we redesign the procedure of matrix multiplication in SPP following the column parallel linear layer in Megatron-LM (Shoeybi et al., 2019). Specifically, in each linear layer $\mathbf{F}(\mathbf{X}, \mathbf{W})$ with weight $\mathbf{W}^{m\times n}$ and activation $\mathbf{X}^{b\times n}$, the output of the linear layer is $\mathbf{Y} = \mathbf{X}\mathbf{W}^T \in \mathbb{R}^{b\times m}$ (w.l.o.g., we discard the batch of input $\mathbf{X}$ and assume $b = 1$ for simplicity). We split the column of $\mathbf{W}$ into r blocks, denoted as $\left[\mathbf{W}_0^T, \mathbf{W}_1^T, \ldots, \mathbf{W}_{r-1}^T\right]^T$, where each $\mathbf{W}_j \in \mathbb{R}^{\lfloor\frac{m}{r}\rfloor\times n}$. In our setting, we can derive:

$$\begin{aligned}\mathbf{Y}^{i'} &= \mathbf{X}^i(\widetilde{\mathbf{W}}^i \odot \text{Repeat}_0(\mathbf{W}^i_\alpha, \left\lfloor\frac{m}{r}\right\rfloor) \odot \text{Repeat}_1(\mathbf{W}^i_\beta, n))^T \\ &= \left[\cdots(\mathbf{X}^i \odot \mathbf{W}^i_{\alpha j})(\widetilde{\mathbf{W}}^i_j)^T\cdots\right] \odot (\mathbf{W}^i_\beta)^T, \quad (7)\end{aligned}$$

where $\mathbf{W}^i_{\alpha j}$ denotes the j-th row of $\mathbf{W}^i_\alpha$, and $\widetilde{\mathbf{W}}^i_j$ denots the j-th block of $\widetilde{\mathbf{W}}^i$ $(0 \le j \le r-1)$. This will eliminate the need for saving the matrix $\text{Repeat}_0(\mathbf{W}^i_\alpha, \lfloor\frac{m}{r}\rfloor)$, and a more detailed proof of the above Eq. (7) can be found in Sec. A.2 in the Appendix.

Remarks on SPP:

1) Why the residual scheme is needed? Actually, it is simpler to directly initialize $\mathbf{W}_\alpha$ and $\mathbf{W}_\beta$ to 1 and then multiply them to the corresponding positions respectively. But we find that such a training scheme converges more slowly.

2) Why is it important to keep the sparse pattern of the model during training? Previous PEFT schemes tend to turn a sparse model into a dense one after training, and we need to prune the trained model again to obtain the final sparse model. This strategy leads to an unpredictable change in model performance again. In contrast, by keeping the model sparsity during the training process, the performance of the final model is a direct reflection of the effectiveness of the training process (Liu et al., 2018).

4. Experiments

In this section, we report a series of experiments to demonstrate the effectiveness of SPP for the efficient training of sparse pruned models.

Experiment Setup: We choose LLaMA and LLaMA-2 model families: LLaMA 7B/13B/30B/65B (Touvron et al., 2023a) and LLaMA-2 7B/13B/70B (Touvron et al., 2023b) for experiments. For these LLMs, we first prune them using post-training pruning algorithms SparseGPT (Frantar & Alistarh, 2023) and Wanda (Sun et al., 2023), then carry out instruction fine-tuning for the recovery of knowledge. All

	LLaMA				LLaMA-2		
	7B	13B	30B	65B	7B	13B	70B
Trainable Parameters	2.0×10^7	3.1×10^7	6.0×10^7	9.8×10^7	2.0×10^7	3.1×10^7	1.1×10^8
All Parameters	6.8×10^9	1.3×10^{10}	3.3×10^{10}	6.5×10^{10}	6.8×10^9	1.3×10^{10}	6.9×10^{10}
Per mille (‰)	**2.90**	**2.35**	**1.83**	**1.50**	**2.90**	**2.35**	**1.54**

Table 1. Number of trainable parameters, total number of parameters, and per mille of trainable parameters of SPP with $r = 16$. We only need to fine-tune a small fraction of parameters compared with full fine-tuning.

LLaMA	Method	Sparsity	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
7B	None	Dense	75.11	66.43	56.96	69.85	75.25	41.89	34.40	59.98
	SparseGPT	Unstructured 50%	73.36	57.76	51.44	68.03	70.45	36.35	28.40	55.11
	SparseGPT+**SPP**	Unstructured 50%	72.84	65.70	56.40	67.88	72.35	41.04	32.80	**58.43**
	SparseGPT	2:4	70.09	57.76	43.37	63.46	61.62	29.27	22.60	49.74
	SparseGPT+**SPP**	2:4	72.39	59.57	53.33	64.17	68.39	37.54	26.80	**54.60**
	Wanda	Unstructured 50%	71.01	55.23	51.90	66.22	69.36	36.95	28.60	54.18
	Wanda+**SPP**	Unstructured 50%	70.86	66.06	55.92	67.64	72.81	41.64	32.00	**58.13**
	Wanda	2:4	69.27	51.26	42.07	62.67	60.52	27.99	24.60	48.34
	Wanda+**SPP**	2:4	71.19	63.90	52.77	64.88	68.18	37.03	30.00	**55.42**
13B	None	Dense	77.98	70.40	59.92	72.61	77.36	46.50	33.20	62.57
	SparseGPT	Unstructured 50%	76.54	62.09	54.94	71.59	72.35	41.64	32.20	58.76
	SparseGPT+**SPP**	Unstructured 50%	79.20	64.62	59.27	70.32	74.83	46.59	34.60	**61.35**
	SparseGPT	2:4	70.80	56.68	48.09	69.22	66.88	36.26	26.20	53.45
	SparseGPT+**SPP**	2:4	77.65	63.54	56.55	69.69	71.21	40.96	32.60	**58.89**
	Wanda	Unstructured 50%	76.27	62.82	55.78	71.98	73.32	43.77	31.80	59.39
	Wanda+**SPP**	Unstructured 50%	78.29	66.43	58.88	70.32	75.59	46.93	34.40	**61.55**
	Wanda	2:4	70.21	53.79	46.78	68.82	65.74	33.70	26.20	52.18
	Wanda+**SPP**	2:4	75.99	58.12	56.07	68.90	70.37	40.53	32.40	**57.48**
30B	None	Dense	82.63	66.79	63.36	75.85	80.39	52.82	36.00	65.41
	SparseGPT	Unstructured 50%	82.63	58.84	59.20	73.48	78.79	49.15	33.20	62.18
	SparseGPT+**SPP**	Unstructured 50%	84.43	68.23	63.18	73.56	81.57	52.56	37.00	**65.79**
	SparseGPT	2:4	76.57	61.01	53.52	72.30	74.66	42.06	31.60	58.82
	SparseGPT+**SPP**	2:4	81.65	66.43	60.46	72.45	78.75	50.17	36.20	**63.73**
	Wanda	Unstructured 50%	81.93	64.98	60.95	73.64	79.38	50.17	34.80	63.69
	Wanda+**SPP**	Unstructured 50%	84.19	66.79	62.52	71.59	77.10	51.79	34.80	**64.11**
	Wanda	2:4	75.14	63.54	54.53	72.45	74.24	41.89	31.80	59.08
	Wanda+**SPP**	2:4	81.38	69.68	59.99	71.59	76.73	48.63	34.60	**63.23**
65B	None	Dense	84.55	69.68	65.40	77.35	52.82	81.00	38.00	66.97
	SparseGPT	Unstructured 50%	84.90	70.04	63.95	77.27	79.65	50.17	37.40	66.20
	SparseGPT+**SPP**	Unstructured 50%	84.95	70.04	64.25	77.19	79.85	50.94	37.80	**66.43**
	SparseGPT	2:4	84.55	69.31	57.95	76.95	78.00	45.39	31.20	63.34
	SparseGPT+**SPP**	2:4	84.25	68.23	58.40	76.87	78.10	45.99	31.40	63.32
	Wanda	Unstructured 50%	85.05	71.84	64.60	77.35	79.65	50.26	38.40	66.74
	Wanda+**SPP**	Unstructured 50%	85.25	71.84	65.30	77.19	79.95	51.11	38.60	**67.03**
	Wanda	2:4	83.40	61.01	58.55	75.22	76.60	45.56	33.20	61.93
	Wanda+**SPP**	2:4	83.30	61.37	61.85	76.16	78.60	47.70	36.20	**63.60**

Table 2. Zero-shot evaluation results of 7 tasks from EleutherAI LM Harness (Gao et al., 2021) after training LLaMA on Alpaca (Taori et al., 2023) dataset by SPP. SPP improves the performance of sparse models from SparseGPT and Wanda. In 2 cases, the performance of sparsely trained models even exceeds their dense counterparts.

the training and testing processes are conducted on a server with 8 NVIDIA A100-80GB GPUs[3].

[3]This is the largest computing resource we have access to. Fine-tuning schemes outlined in Eq. (2) and (3) result in Out-of-Memory errors when applied to models with more than 30B parameters.

Training and Evaluation Details: We use high quality instruction fine-tuning dataset Stanford-Alpaca (Taori et al., 2023) to train the pruned models.We do not use pre-training datasets (e.g. C4 (Raffel et al., 2020), SlimPajama (Soboleva et al., 2023), etc.) as they are quite large but of low

LLaMA	Method	Sparsity	LM-eval	PPL (↓)
7B	Wanda	Unstructured 75%	32.14	1285.24
	Wanda+DS$\oslash$T	Unstructured 75%	32.23	646.40
	Wanda+**SPP**	Unstructured 75%	**41.71**	**21.80**
	Wanda	2:8	32.53	3284.43
	Wanda+DS$\oslash$T	2:8	31.57	2742.98
	Wanda+**SPP**	2:8	**38.61**	**42.07**
30B	Wanda	Unstructured 75%	39.21	149.63
	Wanda+DS$\oslash$T	Unstructured 75%	37.77	184.51
	Wanda+**SPP**	Unstructured 75%	**50.33**	**10.89**
	Wanda	2:8	35.12	1057.58
	Wanda+DS$\oslash$T	2:8	32.81	903.17
	Wanda+**SPP**	2:8	**43.09**	**19.83**

Table 3. Comparison for pruning LLaMA model family at 75% sparsity. We provide average accuracies of 7 zero-shot tasks together with WikiText perplexity (Merity et al., 2016). SPP achieves far better results than the other two post-training pruning methods.

quality. Unlike previous related works on training sparse models (e.g. Sheared llama (Xia et al., 2023)), our approach requires only several hours of training on a smaller dataset. We add learnable parameters on "q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj, score" linear layers, and r is set to 16 (unless specifically noted). For training convenience, we fine-tune 7B/13B/30B models by 3 epochs, and 65B/70B models by 1 epoch. In terms of evaluation, we mainly report zero-shot performance on seven tasks from EleutherAI LM Harness (Gao et al., 2021) following (Sun et al., 2023). We also test few-shot performances on the MMLU (Hendrycks et al., 2021a;b) benchmark.

Model Sparsity and Baselines: For most experiments, we use the 50% sparsity ratio, including unstructured 50% sparsity and 2:4 sparsity (Mishra et al., 2021; Zhou et al., 2021). We compare the models trained by SPP with post-training pruned models (SparseGPT (Frantar & Alistarh, 2023), Wanda (Sun et al., 2023)) and the original dense models. To extend the experiments to higher sparsity, we test some of the 75% sparsity cases, including unstructured 75% sparsity and 2:8 sparsity, and compare our SPP with LoRA* and recently proposed DS$\oslash$T (Zhang et al., 2023b).

4.1. Number of Trainable Parameters

An important metric for estimating a PEFT method is the number of learnable parameters during training. Tab. 1 summarizes the number and per mille (‰) of trainable parameters in our experiments with $r = 16$. As can be seen, SPP requires training only a very small fraction of parameters.

4.2. Zero-shot Evaluation Results

In Tab. 2 and Tab. 7 (Tab. 7 is in Sec. A.3 in the Appendix due to space limitation), we show the zero-shot accuracies of post-training pruned models and their retrained counterparts by SPP on both LLaMA and LLaMA-2 (task-wise and average results). We follow the evaluation tasks and experiment settings of Wanda (Sun et al., 2023). As seen from

Method	Sparsity	Zero-init	$\mathbf{W}_\beta$	r	LM-eval
Wanda+**SPP**	2:4	✓	✓	4	54.04
		✓	✓	8	54.87
		✓		16	54.52
			✓	16	53.52
		✓	✓	16	**55.42**
	Unstructured 50%	✓	✓	4	57.86
		✓	✓	8	56.39
		✓		16	57.81
			✓	16	57.59
		✓	✓	16	**58.13**
SparseGPT+**SPP**	2:4	✓	✓	4	**54.82**
		✓	✓	8	54.24
		✓		16	54.62
			✓	16	54.01
		✓	✓	16	54.60
	Unstructured 50%	✓	✓	4	57.58
		✓	✓	8	57.32
		✓		16	57.66
			✓	16	57.12
		✓	✓	16	**58.43**

Table 4. Ablation studies on LLaMA-7B. We investigate the utility of zero-initialization of weights, evaluate the influence of parameter $\mathbf{W}_\beta$, as well as test the performance of SPP-trained models for different values of r. Training with zero-init weights, $\mathbf{W}_\beta$, and using $r = 16$ get the overall best results in our experiments.

the tables, after SPP training, the average performances of the vast majority of models are improved. Especially for the models with 2:4 sparsity, we observe around 8% average performance improvement, larger than that for the unstructured 50% sparsity models. By utilizing the cuSPARSELt library, the N:M technique can be efficiently implemented on NVIDIA Ampere Graphics processors, resulting in practical speedups. The enhancement of 2:4 sparsity models is thus of particular importance.

4.3. Extend to Higher Sparsity

To further validate the effectiveness of our SPP method, we extend the existing post-training pruning method Wanda (Sun et al., 2023) to 75% sparsity ratio with unstructured 75% and 2:8 sparsity. The results are shown in Tab. 3. "LM-eval" stands for zero-shot evaluation results of 7 different tasks from EleutherAI LM Harness (Gao et al., 2021), "PPL" stands for Wikitext perplexity (Merity et al., 2016). We find that Wanda (Sun et al., 2023) with unstructured 75% and 2:8 sparsity has a significant performance drop, while the dynamic mask without retraining method DS$\oslash$T (Zhang et al., 2023b) has limited performance improvement. We then apply the SPP method to the pruned models by Wanda and obtain far better results. This further highlights the effectiveness of parameter retraining – just as SPP does – in boosting the performance of sparse LLMs.

4.4. Ablation Study

In this subsection, we carry out an ablation study on the LLaMA-7B model. We train the sparse models obtained

LLaMA	Method	Sparsity	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
7B	Wanda+LoRA*	Unstructured 75%	62.39	53.07	31.81	52.64	38.22	21.08	14.60	39.11
	Wanda+**SPP**	Unstructured 75%	60.67	56.32	35.06	52.64	47.05	22.44	17.80	**41.71**
	Wanda+LoRA*	2:8	61.47	53.07	29.18	53.12	34.68	20.22	14.60	38.05
	Wanda+**SPP**	2:8	54.50	59.21	31.29	52.09	37.46	19.11	16.60	**38.61**
30B	Wanda+LoRA*	Unstructured 75%	65.08	55.60	44.35	62.27	60.69	29.18	23.00	48.60
	Wanda+**SPP**	Unstructured 75%	67.95	54.15	47.28	62.51	63.68	30.72	26.00	**50.33**
	Wanda+LoRA*	2:8	62.17	52.71	35.96	54.30	48.48	23.21	16.20	41.86
	Wanda+**SPP**	2:8	62.05	54.87	38.17	55.09	49.62	23.63	18.20	**43.09**

Table 5. Zero-shot evaluation of 7 different tasks from EleutherAI LM Harness (Gao et al., 2021) with 75% sparsity. We compare SPP with sparse models after LoRA training by applying the dense trained model with the original Wanda pruning masks. SPP achieves overall higher zero-shot performances than LoRA. (*) denotes sparse models obtained from applying Wanda pruning masks after LoRA training.

		LLaMA				LLaMA-2		
Method	**Sparsity**	7B	13B	30B	65B	7B	13B	70B
None	Dense	35.64	47.63	58.58	63.78	46.56	55.30	69.56
SparseGPT	Unstructured 50%	32.19	40.44	52.62	59.37	36.41	47.47	65.57
SparseGPT+**SPP**	Unstructured 50%	30.77	43.91	54.73	59.38	39.78	48.31	65.60
SparseGPT	2:4	28.24	32.31	43.79	49.79	29.16	38.41	57.66
SparseGPT+**SPP**	2:4	27.81	37.55	49.01	49.50	33.28	45.63	57.85
Wanda	Unstructured 50%	31.50	39.43	52.84	58.75	34.20	47.78	64.45
Wanda+**SPP**	Unstructured 50%	31.74	43.34	53.89	59.02	38.08	48.97	64.39
Wanda	2:4	27.14	31.26	41.36	45.68	28.33	35.16	56.86
Wanda+**SPP**	2:4	28.56	35.73	46.19	47.67	30.47	42.79	57.98

Table 6. 5-shot evaluation of MMLU (Hendrycks et al., 2021a;b), with LLaMA and LLaMA-2 model families trained on Alpaca (Taori et al., 2023) dataset. The performances of sparse models are improved after instruction fine-tuning.

from Wanda (Sun et al., 2023) and SparseGPT (Frantar & Alistarh, 2023) at unstructured 50% and 2:4 sparsity, respectively. We test the performance of the models with and without the $\mathbf{W}_\beta$ parameter, as well as the average zero-shot accuracies of the models for different values of r. We also evaluate the effectiveness of the zero-initialization of added weights. The results are shown in Tab. 4. Although different parameter settings may lead to different results, training with $\mathbf{W}_\beta$ and using $r = 16$ can obtain the overall best results in our experiments. In the real application of SPP, appropriate parameters can be selected according to the tasks and available computational resources.

4.5. More Analyses

In this part, more experiments and analyses are provided.

Comparison with LoRA*: We also compare SPP with LoRA*. We adopt $r = 8$ in LoRA and add adapters to the same places as SPP, leading to similar numbers of learnable parameters (around 0.28% for 7B models and 0.18% for 30B models). Since LoRA yields a dense model after training, to maintain the model sparsity, we apply the original Wanda pruning masks to sparse it (the LoRA training-then-pruning method is denoted as LoRA*) and test at 75% sparsity. As shown in Tab. 5, compared to the model pruned after LoRA training, SPP has performance leads in most cases. This highlights the significance of maintaining weight sparsity during the training and weight-merging processes.

Few-shot Results: 5-shot average results of different models on the MMLU (Hendrycks et al., 2021a;b) benchmark are shown in Tab. 6. Compared to the tasks in Sec. 4.2, MMLU is a more difficult benchmark and contains some more difficult questions (e.g., math questions). As can be seen from the table, for the post-training pruned models, SPP also brings large performance gains in the vast majority of cases. However, the sparse models trained by SPP still have considerable gaps in performance compared to the original dense models. It should be noted that we obtain performance improvements by instruction fine-tuning on a small-scale dataset. These models would further improve their performance if we extend the scale of the training dataset (Soboleva et al., 2023), but it is currently outside the scope of our considerations in this paper.

Inference Speedup: The inference speedup for sparse models is only dependent on the sparsity pattern of the pruned model. Since SPP does not change the sparsity pattern and ratio of the pruned model during training and merging of parameters, it will lead to the same inference speedup as the original pruned model. We refer the readers to Section 4.3 of Wanda (Sun et al., 2023) paper for results on the relevant inference speedups, where around 1.6× speed up is noted for linear layers in LLMs and 1.24× end-to-end latency speedup is observed for LLaMA-7B (*from 312ms to 251ms*) with the structured 2:4 sparsity.

Hyper Parameters: For training 7B/13B/30B/65B/70B models, we use learning rates of 4e-3/2e-3/4e-3/5e-4/5e-4 with per-device batch size set to 8/4/16/8/8. Following (Dettmers et al., 2023), we set a 0.03 warm-up ratio, but decay the learning rate after reaching the targeted peak value. We use the AdamW optimizer with default setting in the Transformers package[4] and add a 0.001 weight decay.

After fixing all the hyper-parameters, we train post-training pruned LLMs obtained by SparseGPT (Frantar & Alistarh, 2023) and Wanda (Sun et al., 2023) with different sparsity patterns and ratios. We do not carry out hyper-parameter tuning for specific sparsity patterns or ratios. Experiment results shown in the paper demonstrate that our SPP can bring about stable performance improvements.

5. Conclusion and Discussion

In this paper, we introduce the novel **S**parsity-**P**reserved **P**arameter-efficient fine-tuning (SPP) method, which is a notably efficient approach for retraining or fine-tuning sparse models to tackle the challenge of restoring the performance of LLMs after pruning.

Before the advent of LLMs, the field of model pruning primarily explored methods to identify optimal binary sparse masks while training the remaining parameters, either through gradient-based techniques or heuristic approaches. Our research incorporates the SPP method into existing post-training pruning strategies that utilize fixed masks, and focuses solely on retraining the preserved parameters after pruning. Looking ahead, we aim to further develop our SPP method, and integrate it with iterative mask updating techniques to enhance the performance of sparsity-preserved retraining. We hope this strategy can further boost the development in this field of study.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning, especially the field of large language models (LLMs). Our work can contribute to reducing the computational resources and energy consumption required for operating these huge models, thereby addressing environmental concerns associated with large-scale deep network operations. Additionally, the improved efficiency of LLMs can facilitate more widespread and accessible applications of AI technologies, potentially democratizing the benefits of AI across various sectors. However, as with any advancement in machine learning, there is a need for ongoing consideration of ethical implications, particularly in terms of data privacy, security, and the potential for unintended biases in model outputs. These are questions that all of us in the Machine Learning community need to consider when moving forward.

Acknowledgement

This project is funded in part by National Key R&D Program of China Project 2022ZD0161100, by the Centre for Perceptual and Interactive Intelligence (CPII) Ltd under the Innovation and Technology Commission (ITC)'s InnoHK, by General Research Fund of Hong Kong RGC Project 14204021. Hongsheng Li is a PI of CPII under the InnoHK.

References

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. *ArXiv*, abs/2301.00774, 2023. URL `https:`

[4]https://github.com/huggingface/transformers

//api.semanticscholar.org/CorpusID:255372747.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., et al. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 2021.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural network. In *Neural Information Processing Systems*, 2015. URL https://api.semanticscholar.org/CorpusID:2238772.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hendrycks, D., Burns, C., Basart, S., Critch, A., Li, J., Song, D., and Steinhardt, J. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021a.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021b.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.

Horn, R. A. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pp. 87–169, 1990.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Jaiswal, A., Gan, Z., Du, X., Zhang, B., Wang, Z., and Yang, Y. Compressing llms: The truth is rarely pure and never simple. *arXiv preprint arXiv:2310.01382*, 2023.

Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S., and Farhadi, A. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pp. 5544–5555. PMLR, 2020.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Li, Y., Niu, L., Zhang, X., Liu, K., Zhu, J., and Kang, Z. E-sparse: Boosting the large language model inference through entropy-based n: M sparsity. *arXiv preprint arXiv:2310.15929*, 2023.

Lin, T., Stich, S. U., Barba, L., Dmitriev, D., and Jaggi, M. Dynamic model pruning with feedback. *arXiv preprint arXiv:2006.07253*, 2020.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., and Bossan, B. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mishra, A., Latorre, J. A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., and Micikevicius, P. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9 (1):2383, 2018.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-\deduplicated-version-of-redpajama, 2023.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

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

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=yzkSU5zdwD. Survey Certification.

Xia, M., Gao, T., Zeng, Z., and Chen, D. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Xu, Y., Xie, L., Gu, X., Chen, X., Chang, H., Zhang, H., Chen, Z., Zhang, X., and Tian, Q. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

Zhang, Y., Bai, H., Lin, H., Zhao, J., Hou, L., and Cannistraci, C. V. An efficient plug-and-play post-training pruning strategy in large language models. 2023a.

Zhang, Y., Zhao, L., Lin, M., Sun, Y., Yao, Y., Han, X., Tanner, J., Liu, S., and Ji, R. Dynamic sparse no training: Training-free fine-tuning for sparse llms. *arXiv preprint arXiv:2310.08915*, 2023b.

Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., and Li, H. Learning n: m fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021.

Zhou, A., Wang, K., Lu, Z., Shi, W., Luo, S., Qin, Z., Lu, S., Jia, A., Song, L., Zhan, M., and Li, H. Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=c8McWs4Av0.

A. Appendix

A.1. More explanations

Discussion of setting $\mathbf{W}_{\alpha}^{i} \in \mathbb{R}^{r\times n}$ instead of $\mathbf{W}_{\beta}^{i} \in \mathbb{R}^{m\times r}$.

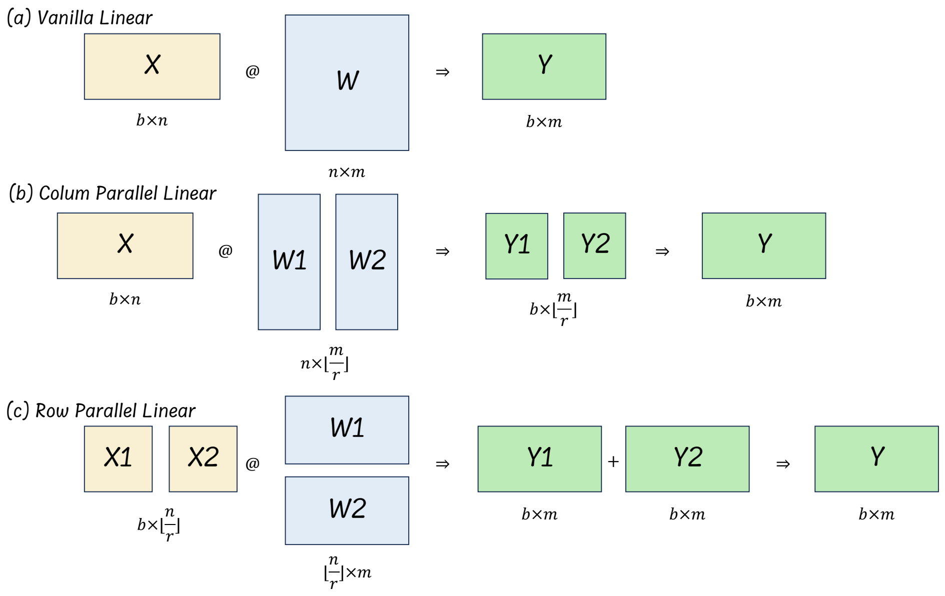

Figure 5. The vanilla linear function and the two parallel methods in Megatron-LM (Shoeybi et al., 2019).

In our paper, we set $\mathbf{W}_{\alpha}^{i} \in \mathbb{R}^{r\times n}$, thus we can utilize the memory optimization technique mentioned in Sec. 3.2 to eliminate the storage of intermediate matrix $\text{Repeat}_0(\mathbf{W}_{\alpha}^{i}, \lfloor \frac{m}{r} \rfloor)$. The optimization approach follows the column parallel linear in Megatron-LM (Shoeybi et al., 2019), corresponding to Fig. 5 (b). As can be seen, if we split the matrices from the dimension of m, we can separately calculate the linear operations $\mathbf{Y}_1 = \mathbf{F}(\mathbf{X}, \mathbf{W}_1), \mathbf{Y}_2 = \mathbf{F}(\mathbf{X}, \mathbf{W}_2), \ldots, \mathbf{Y}_r = \mathbf{F}(\mathbf{X}, \mathbf{W}_r)$ and concat them. Each $\mathbf{Y}_i$ $(1 \le i \le r)$ is a matrix of $\mathbb{R}^{b\times \lfloor \frac{m}{r} \rfloor}$. This does not lead to other additional intermediate weights.

If we use $\mathbf{W}_{\beta}^{i} \in \mathbb{R}^{m\times r}$, the optimization method will correspond to row parallel linear as in Fig. 5 (c). This leads to two disadvantages: **(1)** We need to split both $\mathbf{X}$ and $\mathbf{W}$ (more steps to implement). **(2)** After applying $\mathbf{Y}_1 = \mathbf{F}(\mathbf{X}_1, \mathbf{W}_1), \mathbf{Y}_2 = \mathbf{F}(\mathbf{X}_2, \mathbf{W}_2), \ldots, \mathbf{Y}_r = \mathbf{F}(\mathbf{X}_r, \mathbf{W}_r)$, we get intermediate matrices with size $b \times m$ (we have to store at least two) and need to add them together (less efficient).

A.2. Detailed Proof of Eq. 7

Let us consider a more general case with $b \geq 1$.

Given input activation $\mathbf{X} \in \mathbb{R}^{b\times n}$, $\mathbf{W} \in \mathbb{R}^{m\times n}$, $\mathbf{W}_\alpha \in \mathbb{R}^{r\times n}$ and $\mathbf{W}_\beta \in \mathbb{R}^{m\times 1}$,

$$
\begin{aligned}
&\mathbf{X}(\mathbf{W} \odot \text{Repeat}_0(\mathbf{W}_\alpha, \left\lfloor \frac{m}{r} \right\rfloor) \odot \text{Repeat}_1(\mathbf{W}_\beta, n))^T \\
&= \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{b1} & x_{b2} & \cdots & x_{bn} \end{bmatrix}
\begin{bmatrix} w_{11}\alpha_{11}\beta_1 & w_{12}\alpha_{12}\beta_1 & \cdots & w_{1n}\alpha_{1n}\beta_1 \\ w_{21}\alpha_{11}\beta_2 & w_{22}\alpha_{12}\beta_2 & \cdots & w_{2n}\alpha_{1n}\beta_2 \\ \vdots & \vdots & & \vdots \\ w_{\lfloor \frac{m}{r} \rfloor 1}\alpha_{11}\beta_{\lfloor \frac{m}{r} \rfloor} & w_{\lfloor \frac{m}{r} \rfloor 2}\alpha_{12}\beta_{\lfloor \frac{m}{r} \rfloor} & \cdots & w_{\lfloor \frac{m}{r} \rfloor n}\alpha_{1n}\beta_{\lfloor \frac{m}{r} \rfloor} \\ \vdots & \vdots & & \vdots \\ w_{m1}\alpha_{r1}\beta_m & w_{m2}\alpha_{r2}\beta_m & \cdots & w_{mn}\alpha_{rn}\beta_m \end{bmatrix}^T
\end{aligned}
$$

$$
= \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{b1} & x_{b2} & \cdots & x_{bn} \end{bmatrix}
\begin{bmatrix} w_{11}\alpha_{11}\beta_1 & w_{21}\alpha_{11}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor 1}\alpha_{11}\beta_{\lfloor \frac{m}{r} \rfloor} & \cdots & w_{m1}\alpha_{r1}\beta_m \\ w_{12}\alpha_{12}\beta_1 & w_{22}\alpha_{12}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor 2}\alpha_{12}\beta_{\lfloor \frac{m}{r} \rfloor} & \cdots & w_{m2}\alpha_{r2}\beta_m \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{1n}\alpha_{1n}\beta_1 & w_{2n}\alpha_{1n}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor n}\alpha_{1n}\beta_{\lfloor \frac{m}{r} \rfloor} & \cdots & w_{mn}\alpha_{rn}\beta_m \end{bmatrix} \tag{8}
$$

The matrix on the right can be split horizontally into r blocks, each $\in \mathbb{R}^{n\times \lfloor \frac{m}{r} \rfloor}$. Let's take the first block for simplicity. Some notations should be added for clarity. $\mathbf{W}_{\alpha i}$ denotes the i-th row of $\mathbf{W}_\alpha$, $\mathbf{W}_i$ denotes the i-th block of $\mathbf{W}$ corresponding to the horizontal split, and $\mathbf{W}_i \in \mathbb{R}^{\lfloor \frac{m}{r} \rfloor \times n}$, $\mathbf{W}_{\beta i}$ denotes the i-th block of $\mathbf{W}_\beta$ corresponding to the horizontal split, and $\mathbf{W}_{\beta i} \in \mathbb{R}^{\lfloor \frac{m}{r} \rfloor \times 1}$.

$$
\begin{aligned}
&\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{b1} & x_{b2} & \cdots & x_{bn} \end{bmatrix}
\begin{bmatrix} w_{11}\alpha_{11}\beta_1 & w_{21}\alpha_{11}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor 1}\alpha_{11}\beta_{\lfloor \frac{m}{r} \rfloor} \\ w_{12}\alpha_{12}\beta_1 & w_{22}\alpha_{12}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor 2}\alpha_{12}\beta_{\lfloor \frac{m}{r} \rfloor} \\ \vdots & \vdots & & \vdots \\ w_{1n}\alpha_{1n}\beta_1 & w_{2n}\alpha_{1n}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor n}\alpha_{1n}\beta_{\lfloor \frac{m}{r} \rfloor} \end{bmatrix} \\
&= \begin{bmatrix} x_{11}\alpha_{11} & x_{12}\alpha_{12} & \cdots & x_{1n}\alpha_{1n} \\ x_{21}\alpha_{11} & x_{22}\alpha_{12} & \cdots & x_{2n}\alpha_{1n} \\ \vdots & \vdots & & \vdots \\ x_{b1}\alpha_{11} & x_{b2}\alpha_{12} & \cdots & x_{bn}\alpha_{1n} \end{bmatrix}
\begin{bmatrix} w_{11}\beta_1 & w_{21}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor 1}\beta_{\lfloor \frac{m}{r} \rfloor} \\ w_{12}\beta_1 & w_{22}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor 2}\beta_{\lfloor \frac{m}{r} \rfloor} \\ \vdots & \vdots & & \vdots \\ w_{1n}\beta_1 & w_{2n}\beta_2 & \cdots & w_{\lfloor \frac{m}{r} \rfloor n}\beta_{\lfloor \frac{m}{r} \rfloor} \end{bmatrix} \\
&= \begin{bmatrix} x_{11}\alpha_{11} & x_{12}\alpha_{12} & \cdots & x_{1n}\alpha_{1n} \\ x_{21}\alpha_{11} & x_{22}\alpha_{12} & \cdots & x_{2n}\alpha_{1n} \\ \vdots & \vdots & & \vdots \\ x_{b1}\alpha_{11} & x_{b2}\alpha_{12} & \cdots & x_{bn}\alpha_{1n} \end{bmatrix}
\begin{bmatrix} w_{11} & w_{21} & \cdots & w_{\lfloor \frac{m}{r} \rfloor 1} \\ w_{12} & w_{22} & \cdots & w_{\lfloor \frac{m}{r} \rfloor 2} \\ \vdots & \vdots & & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{\lfloor \frac{m}{r} \rfloor n} \end{bmatrix} \odot
\begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_{\lfloor \frac{m}{r} \rfloor} \\ \beta_1 & \beta_2 & \cdots & \beta_{\lfloor \frac{m}{r} \rfloor} \\ \vdots & \vdots & & \vdots \\ \beta_1 & \beta_2 & \cdots & \beta_{\lfloor \frac{m}{r} \rfloor} \end{bmatrix} \\
&= \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{b1} & x_{b2} & \cdots & x_{bn} \end{bmatrix} \odot
\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \vdots & \vdots & & \vdots \\ \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \end{bmatrix}
\begin{bmatrix} w_{11} & w_{21} & \cdots & w_{\lfloor \frac{m}{r} \rfloor 1} \\ w_{12} & w_{22} & \cdots & w_{\lfloor \frac{m}{r} \rfloor 2} \\ \vdots & \vdots & & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{\lfloor \frac{m}{r} \rfloor n} \end{bmatrix} \odot
\begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_{\lfloor \frac{m}{r} \rfloor} \\ \beta_1 & \beta_2 & \cdots & \beta_{\lfloor \frac{m}{r} \rfloor} \\ \vdots & \vdots & & \vdots \\ \beta_1 & \beta_2 & \cdots & \beta_{\lfloor \frac{m}{r} \rfloor} \end{bmatrix} \\
&= \mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha 0}, b)\mathbf{W}_1^T \odot \text{Repeat}_1(\mathbf{W}_{\beta 0}, b)^T \in \mathbb{R}^{b\times \lfloor \frac{m}{r} \rfloor}
\end{aligned} \tag{9}
$$

Similarly, the result of the i-th block is:

$$\mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha i}, b)\mathbf{W}_i^T \odot \text{Repeat}_1(\mathbf{W}_{\beta i}, b)^T \tag{10}$$

Concatenating the result of all the r blocks, we will get the final result:

$$\begin{aligned}
&\mathbf{X}(\mathbf{W} \odot \text{Repeat}_0(\mathbf{W}_\alpha, \left\lfloor \frac{m}{r} \right\rfloor) \odot \text{Repeat}_1(\mathbf{W}_\beta, n))^T \\
=&[\mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha 0}, b)\mathbf{W}_0^T \odot \text{Repeat}_1(\mathbf{W}_{\beta 0}, b)^T \cdots \mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha(r-1)}, b)\mathbf{W}_{r-1}^T \odot \text{Repeat}_1(\mathbf{W}_{\beta(r-1)}, b)^T] \\
=&[\mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha 0}, b)\mathbf{W}_0^T \cdots \mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha(r-1)}, b)\mathbf{W}_{r-1}^T] \odot [\text{Repeat}_1(\mathbf{W}_{\beta 0}, b)^T \cdots \text{Repeat}_1(\mathbf{W}_{\beta(r-1)}, b)^T] \\
=&[\mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha 0}, b)\mathbf{W}_0^T \cdots \mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha(r-1)}, b)\mathbf{W}_{r-1}^T] \odot \text{Repeat}_1(\mathbf{W}_\beta, b)^T \\
=&[\cdots \mathbf{X} \odot \text{Repeat}_0(\mathbf{W}_{\alpha i}, b)\mathbf{W}_i^T \cdots] \odot \text{Repeat}_1(\mathbf{W}_\beta, b)^T
\end{aligned} \tag{11}$$

If $b = 1$, we get the same result with Eq. 7, as $\text{Repeat}_d(\mathbf{W}, 1) = \mathbf{W}$. Notice that $\mathbf{W}_{\alpha i}$ and $\mathbf{W}_\beta$ are all vectors, and PyTorch automatically aligns the matrix dimensions when doing Hadamard product between a matrix and a vector. Besides, $\mathbf{W}_i$ is part of the frozen weights $\mathbf{W}$. During the training step, we do not need to save other redundant matrices.

A.3. Zero-shot results for LLaMA-2

LLaMA - 2	Method	Sparsity	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
7B	None	Dense	77.74	62.82	57.17	68.98	76.30	43.43	31.40	59.69
	SparseGPT	Unstructured 50%	76.24	55.96	52.87	68.98	71.55	37.80	28.60	56.00
	SparseGPT+**SPP**	Unstructured 50%	75.75	56.32	55.32	69.14	74.20	42.49	30.80	57.72
	SparseGPT	2:4	66.91	57.76	43.55	66.14	64.10	31.40	24.20	50.58
	SparseGPT+**SPP**	2:4	70.89	54.15	51.33	67.40	70.96	38.99	29.00	54.67
	Wanda	Unstructured 50%	76.73	53.43	52.50	68.43	72.35	39.16	30.80	56.20
	Wanda+**SPP**	Unstructured 50%	77.31	54.15	55.19	67.96	74.96	42.24	33.00	57.83
	Wanda	2:4	68.47	53.07	41.33	62.67	62.88	30.55	23.60	48.94
	Wanda+**SPP**	2:4	72.91	54.51	50.61	64.48	70.71	37.88	28.60	54.24
13B	None	Dense	80.55	65.34	60.04	72.14	79.42	48.46	35.20	63.02
	SparseGPT	Unstructured 50%	81.93	62.45	55.83	70.88	75.13	42.49	32.40	60.16
	SparseGPT+**SPP**	Unstructured 50%	80.67	65.70	58.47	70.88	77.02	46.59	34.20	61.93
	SparseGPT	2:4	78.62	56.68	48.32	68.51	68.52	36.01	27.40	54.87
	SparseGPT+**SPP**	2:4	78.10	67.15	55.29	70.24	72.73	43.00	30.80	59.62
	Wanda	Unstructured 50%	81.13	58.84	57.07	70.72	75.67	42.83	32.20	59.78
	Wanda+**SPP**	Unstructured 50%	80.92	69.31	58.45	71.82	78.49	46.84	34.20	62.86
	Wanda	2:4	76.09	55.96	46.19	67.56	68.64	34.13	24.20	53.25
	Wanda+**SPP**	2:4	76.70	66.43	55.02	68.11	72.60	42.24	31.00	58.87
70B	None	Dense	83.45	67.87	66.05	77.98	82.55	54.44	37.20	67.08
	SparseGPT	Unstructured 50%	84.75	71.84	64.10	78.22	81.55	52.73	37.20	67.20
	SparseGPT+**SPP**	Unstructured 50%	84.75	72.20	64.05	78.06	81.65	52.73	37.20	67.23
	SparseGPT	2:4	81.30	69.31	58.60	76.24	79.50	48.21	32.60	63.68
	SparseGPT+**SPP**	2:4	81.30	68.59	58.80	76.24	79.40	48.29	32.60	63.60
	Wanda	Unstructured 50%	82.50	72.56	63.90	78.06	81.60	52.47	37.80	66.98
	Wanda+**SPP**	Unstructured 50%	82.25	72.92	64.40	77.90	81.70	53.24	37.60	67.14
	Wanda	2:4	79.55	67.87	59.05	76.24	79.40	47.70	35.40	63.60
	Wanda+**SPP**	2:4	80.25	69.31	60.65	76.01	80.10	48.81	35.20	64.33

Table 7. Zero-shot evaluation of 7 different tasks from EleutherAI LM Harness (Gao et al., 2021) after training LLaMA-2 on Stanford-Alpaca (Taori et al., 2023) by SPP. As can be seen from the table, SPP can improve the performance of sparse models.

Here we provide in Tab. 7 the zero-shot performance on seven tasks from EleutherAI LM Harness (Gao et al., 2021) after training the LLaMA-2 model family on the Alpaca dataset. We find that after instruction fine-tuning, 11 out of 12 models show performance improvement.

A.4. More Ablations

More ablations concerning zero-initialization of added weight are provided. We carry out experiments with 4 different settings on the LLaMA-7B model (none zero-init, $\mathbf{W}_\alpha$ zero-init, $\mathbf{W}_\beta$ zero-init, and both zero-init). We observe that zero-initializing both $\mathbf{W}_\alpha$ and $\mathbf{W}_\beta$ will lead to gradient vanishment, and the added parameters will remain 0 all the time, so it will not bring any performance enhancement. Average zero-shot accuracies of the other three initialization choices on 7 tasks of LM-eval (Gao et al., 2021) are shown in Tab. 8.

Method	Sparsity	Zero-init W_α	Zero-init W_β	LM-eval
Wanda+**SPP**	2:4			53.52
		✓		**56.16**
			✓	55.42
	Unstructured 50%			57.59
		✓		57.09
			✓	**58.13**
SparseGPT+**SPP**	2:4			54.01
		✓		53.95
			✓	**54.60**
	Unstructured 50%			57.12
		✓		57.59
			✓	**58.43**

Table 8. More ablation studies concerning zero-initialization of added weights. Zero-initializing $\mathbf{W}_\beta$ will get the overall best results in our experiments.

As shown in the table, we find initializing $\mathbf{W}_\beta$ with 0 and randomly initializing $\mathbf{W}_\alpha$ lead to overall better results in our experiment setting.