Pruning Foundation Models for High Accuracy without Retraining

Pu Zhao[1], **Fei Sun**[2], **Xuan Shen**[1], **Pinrui Yu**[1], **Zhenglun Kong**[1], **Yanzhi Wang**[1], **Xue Lin**[1]
[1] Northeastern University
[2] Meta
{p.zhao, shen.xu, yu.pin, kong.zhe, yanz.wang, xue.lin}@northeastern.edu
feisun@meta.com

Abstract

Despite the superior performance, it is challenging to deploy foundation models or large language models (LLMs) due to their massive parameters and computations. While pruning is a promising technique to reduce model size and accelerate the inference, the traditional pruning techniques can hardly be applied for LLMs as they need to finetune the model on the full dataset with multiple epochs consuming massive data and hardware resources. To deal with this problem, post-training pruning methods are proposed to prune LLMs in one-shot without retraining. However, their accuracy after pruning may suffer from certain performance degradation due to the lack of retraining with massive data. To address this issue, in this paper, we first formulate the post-training problem for layer-wise LLM compression to simultaneously prune multiple weights in LLMs. Next, we provide an optimal solution for this problem and design our post-training pruning algorithm for both unstructured and semi-structured sparsity. Our extensive experiments demonstrate the superior performance of the proposed methods in comparison to SOTA baselines across various LLM families including transformer-based LLMs and Mamba-based LLMs.

1 Introduction

Foundation models or large language models (LLMs) have achieved remarkable performance on a variety of tasks. However, it is challenging to deploy LLMs in practical applications due to their massive parameters and computations. To facilitate LLM deployment in practice, various model compression techniques targeting LLMs including pruning (Hubara et al., 2021b; Frantar and Alistarh, 2023) and quantization (Dettmers et al., 2022; Frantar et al., 2022; Yao et al., 2022; Xiao et al., 2023) have been proposed to reduce memory and computation costs.

The traditional pruning techniques, which finetune or retrain models (Li et al., 2020) on full datasets for many epochs (i.e., pruning-aware training), are too expensive for LLMs in terms of data and GPU resources. Thus, post-training pruning based on well-pre-trained models with reduced resource requirements represents a more reasonable approach for LLMs. Notably, SparseGPT (Frantar and Alistarh, 2023) is the representative post-training pruning work with outstanding performance. It reduces memory cost by sequentially loading transformer blocks, one at a time, instead of loading the whole model. Moreover, it reduces the data cost by using only a small amount of calibration data, eliminating the retraining process on massive data. Besides the optimization based SparseGPT, there are some other heuristic post-training pruning methods such as (Sun et al., 2023; Zhang et al., 2024), achieving accuracy close to SparseGPT.

However, the performance of SparseGPT is still sub-optimal as it relies on the solution of *Single Removal Problem* (SRP) (Singh and Alistarh, 2020; Frantar et al., 2021) to address the pruning of multiple weights, which is essentially a *Multiple Removal Problem* (MRP). In particular, the SRP provides the optimal solution to prune one weight at a time and modify all other weights to compensate the pruned single weight and minimize the loss. However, as the optimal solution is unaware of all previous pruned weights and requires to modify all other weights including previous pruned ones (making them unpruned again), it is at odds with the MRP for multiple pruned weights. Thus, the optimal solution in SRP can not be directly applied to solve MRP. To successfully incorporate the SRP solution, a series of approximation methods are adopted in SparseGPT, at the cost of certain performance degradation, as detailed in Sec. 2.3.

Different from the SRP-based SparseGPT, we directly formulate the MRP for layer-wise LLM pruning to simultaneously prune multiple weights in LLMs. Next, we derive the optimal solution for

Findings of the Association for Computational Linguistics: EACL 2024, pages 9681–9694
November 12-16, 2024

the MRP problem with detailed analysis for its advantages. Based on the optimal solution, we design our post-training pruning algorithms for both unstructured and semi-structured sparsity. In our comprehensive evaluation, we demonstrate our superior performance in terms of perplexity and zero-shot accuracy compared with SOTA baselines for both transformer-based and Mamba-based LLMs (such as our 4.278 perplexity on wikitext2 v.s. 5.698 from SparseGPT for LLaMA2-70B under 2:4 sparsity).

Our contributions are summarized as follows:

- We directly formulate the MRP for LLM pruning, to enable simultaneous pruning of multiple weights, covering the SRP as a special case, as detailed in Sec. 3.4.
- We derive the optimal solution for the proposed MRP. Based on that, we design accurate post-training pruning algorithms for both unstructured and semi-structured sparsity.
- Our comprehensive experiments across various LLM families (based on transformers and Mamba), model sizes, and datasets demonstrate our superior performance compared with the optimization-based SparseGPT and other heuristic SOTA baselines.

2 Background

2.1 Post-Training Pruning

The post-training pruning problem can be typically formulated as the following,

$$\min_{\delta \boldsymbol{w}} \ L'(\boldsymbol{w}+\delta\boldsymbol{w}) - L'(\boldsymbol{w}), \quad s.t. \ (\boldsymbol{w}+\delta\boldsymbol{w}) \odot \boldsymbol{M} = \boldsymbol{0}, \tag{1}$$

where $\boldsymbol{w}$ is the original model weights, $\delta\boldsymbol{w}$ is the modifications of model weights and $\boldsymbol{M}$ is the binary pruning mask on model weights with 1 denotes pruning. $L'(\boldsymbol{w})$ is the typical training loss. The problem minimizes the difference of the loss before and after pruning, by optimizing the unpruned weights, with the constraint that the pruned weights following the mask should be zero.

2.2 Single Removal Problem

The single removal problem (SRP) is investigated in many works (Singh and Alistarh, 2020; Frantar and Alistarh, 2022, 2023) due to its simplicity. It optimizes the following (Singh and Alistarh, 2020),

$$\min_{\delta \boldsymbol{w}} \ \mathcal{L}(\delta\boldsymbol{w}) = \frac{1}{2}\delta\boldsymbol{w}\boldsymbol{H}\delta\boldsymbol{w}^T, \quad s.t. \ (\delta\boldsymbol{w}+\boldsymbol{w}) \cdot \boldsymbol{e}_t = 0, \tag{2}$$

where $\boldsymbol{e}_t$ is a one-hot vector denoting the pruned location for the single weight, and $\boldsymbol{H}$ is the Hessian matrix of weights. $\boldsymbol{w}$ and $\delta\boldsymbol{w}$ are formulated as one-dimensional vectors for simplicity. The problem prunes one weight at a time, without any information for other previous pruned weights.

Note that here $L'(\boldsymbol{w}+\delta\boldsymbol{w}) - L'(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L' \delta\boldsymbol{w}^T + \frac{1}{2}\delta\boldsymbol{w}\boldsymbol{H}\delta\boldsymbol{w}^T \approx \frac{1}{2}\delta\boldsymbol{w}\boldsymbol{H}\delta\boldsymbol{w}^T$. It ignores the first-order Jacobian term $\nabla_{\boldsymbol{w}} L' \delta\boldsymbol{w}^T$ and only minimizes the second-order Hessian term $\frac{1}{2}\delta\boldsymbol{w}\boldsymbol{H}\delta\boldsymbol{w}^T$, by assuming that the model is well-trained and thus pruned at local minimum.

2.3 Limitations of SRP

2.3.1 Zero Jacobian Assumption

Following SRP, SparseGPT (Frantar and Alistarh, 2023) applies the SRP solution for each linear layer in LLMs with $\boldsymbol{H} = 2\boldsymbol{x}\boldsymbol{x}^T$. However, the assumption with zero Jacobian does not hold in this layer-wise pruning setting with a local quadratic loss $L'(\boldsymbol{w}) = \|\boldsymbol{w}\boldsymbol{x}\|_2^2$, since we can directly obtain the Jacobian $\nabla_{\boldsymbol{w}} \hat{L} = 2\boldsymbol{w}\boldsymbol{x}\boldsymbol{x}^T$ which is non-zero. The assumption may be unreasonable here.

2.3.2 Sequential Weight Freezing

Another difficulty with the SRP solution is its unawareness of all other pruned weights during pruning. Specifically, when compensating the loss of a single pruned weight, its optimal solution requires to modify all other weights including all previous pruned weights, making them unpruned again and violating the pre-defined sparsity requirement.

Thus, the SRP solution is not able to directly prune multiple weights. To address this problem, SparseGPT applies a series of techniques such as Optimal Partial Updates and Hessian Synchronization. The key idea is to sequentially prune weights in the same row, and freeze/fix all weights (including pruned and unpruned weights) previous to the current pruned weight, so that the previous pruned weights are kept zero. The drawback is that all previous unpruned weights are also frozen without further updating, leading to sub-optimal achievements with potential performance degradation.

3 Multiple Removal Problem

3.1 Notations

For linear layers, the forward computation can be represented as $\boldsymbol{wx}$, where $\boldsymbol{w} \in \mathbb{R}^{n\times m}$ is the weights and $\boldsymbol{x} \in \mathbb{R}^{m\times B}$ (B is the token number in each batch) is the layer input. $[\boldsymbol{A}]_{q,p}$ denotes the weight in the q^{th} row and the p^{th} column of the 2D matrix $\boldsymbol{A}$. $[\boldsymbol{A}]_{q,:}$ denotes the q^{th} row of $\boldsymbol{A}$, and $[\boldsymbol{A}]_{:,p}$ represents the p^{th} column of $\boldsymbol{A}$.

To make the problem tractable, the pruned weights are distributed in k rows ($k \leq n$), and their row indexes are denoted by $q_i, \forall i \in \{1, \ldots, k\}$. In the q_i^{th} row, there are k_i pruned elements, and their column indexes are denoted by $p_{ij}, \forall j \in \{1, \ldots, k_i\}$. Since different rows q_i have different numbers and distributions of pruned locations, we use the representation p_{ij} rather than p_j. Thus, the pruned locations/indexes in the weight matrix $\boldsymbol{w}$ can be expressed as $(q_i, p_{ij}), \forall i \in \{1, \ldots, k\}, \forall j \in \{1, \ldots, k_i\}$, or $(q_i, p_{i1}), (q_i, p_{i2}), ..., (q_i, p_{ik_i}), \forall i \in \{1, \ldots, k\}$.

$\boldsymbol{e}_s$ is a one-hot vector with the s^{th} element as one and all others as zero. So $\boldsymbol{e}_{q_i}^T \boldsymbol{w} \boldsymbol{e}_{p_{ij}}$ means the weight in the q_i^{th} row and the p_{ij}^{th} column of $\boldsymbol{w}$, with $\boldsymbol{e}_{q_i} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{e}_{p_{ij}} \in \mathbb{R}^{m\times 1}$.

$\mathrm{Tr}(\cdot)$ represents the trace function.

3.2 Motivation with MRP

To address the limitations of the SRP, we try to formulate and solve the MRP, which prunes multiple weights simultaneously. Our MRP is specifically formulated for the layer-wise LLM pruning without any assumptions. Furthermore, since our MRP prunes multiple weights at the same time, each pruned weight is aware of all other pruned weights and thus there is no need to freeze any weights, which effectively addresses the limitations of SRP.

3.3 MRP Formulation

In LLMs, the linear layers in transformer (Vaswani et al., 2017) or Mamba (Gu and Dao, 2023) blocks are the main cost of computations and parameters. To reduce the overhead of pruning LLMs, following SparseGPT, we adopt the layer-wise compression strategy to sequentially load and prune one single block instead of the whole model. The significantly reduced memory cost makes it feasible to use only one single GPU for all computations.

For each linear layer, we try to minimize the difference of the linear outputs (measured by ℓ_2 norm) before and after pruning, i.e., $\|(\boldsymbol{w} + \delta\boldsymbol{w})\boldsymbol{x} - \boldsymbol{wx}\|_2^2 = \|\delta\boldsymbol{wx}\|_2^2$. To make the problem tractable, as discussed in Sec. 3.1, the pruned locations/indexes in the weight matrix $\boldsymbol{w}$ can be expressed as $(q_i, p_{i1}), (q_i, p_{i2}), ..., (q_i, p_{ik_i}), \forall i \in \{1, \ldots, k\}$. The pruned weights (q_i, p_{ij}) are set to zero, i.e. $[\boldsymbol{w} + \delta\boldsymbol{w}]_{q_i,p_{ij}} = 0$. To minimize the loss incurred by pruning, the other unpruned weights are updated for the compensation of pruned weights. Our MRP is formulated as the following,

$$
\begin{aligned}
\min_{\delta \boldsymbol{w}} \ & \mathcal{L}(\delta\boldsymbol{w}) = \|\delta\boldsymbol{wx}\|^2, \\
s.t. \ & \boldsymbol{e}_{q_i}^T \delta\boldsymbol{w}\boldsymbol{e}_{p_{i1}} + [w]_{q_i,p_{i1}} = 0, \\
& \boldsymbol{e}_{q_i}^T \delta\boldsymbol{w}\boldsymbol{e}_{p_{i2}} + [w]_{q_i,p_{i2}} = 0, \\
& \ldots\ldots \\
& \boldsymbol{e}_{q_i}^T \delta\boldsymbol{w}\boldsymbol{e}_{p_{ik_i}} + [w]_{q_i,p_{ik_i}} = 0, \\
& \forall i \in \{1, \ldots, k\},
\end{aligned} \tag{3}
$$

where $\boldsymbol{e}_{q_i}^T \delta\boldsymbol{w}\boldsymbol{e}_{p_{ij}}$ denotes the weight in the q_i^{th} row and the p_{ij}^{th} column of $\delta\boldsymbol{w}$.

It can be transformed to vector representation,

$$
\begin{aligned}
\min_{\delta \boldsymbol{w}} \ & \mathcal{L}(\delta\boldsymbol{w}) = \|\delta\boldsymbol{wx}\|^2, \\
s.t. \ & \boldsymbol{e}_{q_1}^T \delta\boldsymbol{w}\boldsymbol{e}_{p\_q_1} + \boldsymbol{w}_{q_1} = \boldsymbol{0}, \\
& \boldsymbol{e}_{q_2}^T \delta\boldsymbol{w}\boldsymbol{e}_{p\_q_2} + \boldsymbol{w}_{q_2} = \boldsymbol{0}, \\
& \ldots\ldots \\
& \boldsymbol{e}_{q_k}^T \delta\boldsymbol{w}\boldsymbol{e}_{p\_q_k} + \boldsymbol{w}_{q_k} = \boldsymbol{0},
\end{aligned} \tag{4}
$$

where $\boldsymbol{e}_{p\_q_i} \in \mathbb{R}^{m\times k_i}$ with $[\boldsymbol{e}_{p\_q_i}]_{:,j} = \boldsymbol{e}_{p_{ij}}$, and $\boldsymbol{w}_{qi} = [[\boldsymbol{w}]_{q_i,p_{i1}}, [\boldsymbol{w}]_{q_i,p_{i2}}, ..., [\boldsymbol{w}]_{q_i,p_{ik_i}}] \in \mathbb{R}^{1\times k_i}$. In MRP, multiple weights are pruned simultaneously. $\boldsymbol{e}_{p\_q_i}$ is a collection of all pruned column indexes in the q_i^{th} row, and $\boldsymbol{w}_{qi}$ is a collection of all pruned weight values in the q_i^{th} row.

3.4 Comparison with SRP

Our problem formulation is different from the SRP (Singh and Alistarh, 2020) in several ways.

Relax the zero Jacobian assumption. Different from SRP with the zero Jacobian assumption which does not hold for the layer-wise LLM pruning, our formulation directly optimize the difference of outputs before and after pruning, without any assumptions or approximations.

Furthermore, we provide an explanation for why SRP can still achieve good performance with the unreasonable zero Jacobian assumption. Specifically, since $\boldsymbol{H} = 2\boldsymbol{xx}^T$ for linear layers with the quadratic loss $L'(\boldsymbol{w}) = \|\boldsymbol{wx}\|_2^2$, we have $\frac{1}{2}\delta\boldsymbol{w}\boldsymbol{H}\delta\boldsymbol{w}^T = \delta\boldsymbol{w}\boldsymbol{xx}^T\delta\boldsymbol{w}^T = \|\delta\boldsymbol{wx}\|_2^2$, which means that the optimization objective of SRP is well aligned with that of our proposed MRP. That is why SRP can still perform well with an unreasonable assumption. We demonstrate that our MRP-based method can achieve better performance than the SRP solutions such as SparseGPT.

Simultaneous multiple removal without any approximations. The pruning removes multiple weights in the model. Compared with SRP, our proposed MRP directly addresses the problem by simultaneously pruning multiple weights, without the need for sequential weight freezing following a series of approximation techniques such as Optimal Partial Updates and Hessian Synchronization (see Sec. 2.3.2). Our straightforward formulation leads to a direct solution to update all unpruned weights, leading to a better accuracy performance than SRP solutions which freeze part of unpruned weights without further updating.

Cover SRP as a special case. The SRP is a special case of our MRP. Our formulation deals with multiple weight removals in 2D weight matrices, which covers the single weight removal within 1D weight vectors in SRP as a special case. Consequently, the SRP solution is also a special case of our MRP solution.

4 Methodology

We first derive our optimal solution for the MRP and then discuss the algorithm design.

4.1 Optimal Solution

The Lagrange function of Problem (4) is

$$\begin{aligned}\mathscr{L}(\delta \boldsymbol{w}, \boldsymbol{\lambda}) =& \|\delta \boldsymbol{w}\boldsymbol{x}\|^2 + (\boldsymbol{e}_{q_1}^T \delta \boldsymbol{w} \boldsymbol{e}_{p\_q_1} + \boldsymbol{w}_{q_1})\boldsymbol{\lambda}_1 \\ &+ (\boldsymbol{e}_{q_2}^T \delta \boldsymbol{w} \boldsymbol{e}_{p\_q_2} + \boldsymbol{w}_{q_2})\boldsymbol{\lambda}_2 + ... \\ &+ (\boldsymbol{e}_{q_k}^T \delta \boldsymbol{w} \boldsymbol{e}_{p\_q_k} + \boldsymbol{w}_{q_k})\boldsymbol{\lambda}_k, \\ =& \operatorname{Tr}(\boldsymbol{x}^T \delta \boldsymbol{w}^T \delta \boldsymbol{w} \boldsymbol{x}) + \sum_i (\boldsymbol{e}_{q_i}^T \delta \boldsymbol{w} \boldsymbol{e}_{p\_q_i} + \boldsymbol{w}_{q_i})\boldsymbol{\lambda}_i,\end{aligned} \tag{5}$$

where $\boldsymbol{\lambda}_i \in R^{k_i \times 1}$ denotes the Lagrange multiplier corresponding to the constraint for the q_i^{th} row in Problem (4). $\boldsymbol{\lambda}_i = [\lambda_{i1}, \lambda_{i2}, ..., \lambda_{ik_i}]$ and each λ_{ij} corresponds to the constraint $\boldsymbol{e}_{q_i}^T \delta \boldsymbol{w} \boldsymbol{e}_{p_{ij}} + [w]_{q_i, p_{ij}} = 0$ in Problem (3). Unlike the SRP with a scalar $\delta \boldsymbol{w}\boldsymbol{x}$, in our problem, $\delta \boldsymbol{w}\boldsymbol{x}$ is a matrix, requiring the trace function $\operatorname{Tr}(\cdot)$.

The gradients with reference to $\delta \boldsymbol{w}$ should be 0.

$$\frac{\delta \mathscr{L}(\delta \boldsymbol{w}, \boldsymbol{\lambda})}{\delta(\delta \boldsymbol{w})} = 2\delta \boldsymbol{w}\boldsymbol{x}\boldsymbol{x}^T + \sum_i \boldsymbol{e}_{q_i} \boldsymbol{\lambda}_i^T \boldsymbol{e}_{p\_q_i}^T = 0. \tag{6}$$

We can obtain $\delta \boldsymbol{w}$ as below,

$$\delta \boldsymbol{w} = -\left(\sum_i \boldsymbol{e}_{q_i} \boldsymbol{\lambda}_i^T \boldsymbol{e}_{p\_q_i}^T\right)(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}. \tag{7}$$

By applying Equation (7) in Equation (5), we have the following,

$$g(\boldsymbol{\lambda}) = -\frac{1}{2}\sum_i \boldsymbol{\lambda}_i^T \boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i} \boldsymbol{\lambda}_i + \sum_i \boldsymbol{w}_{q_i} \boldsymbol{\lambda}_i. \tag{8}$$

Note that $\boldsymbol{e}_{q_i}^T \boldsymbol{e}_{q_i} = 1$ and $\boldsymbol{e}_{q_i}^T \boldsymbol{e}_{q_s} = 0$, for $i \neq s$. Besides, we can switch the position of $\boldsymbol{x}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i} \boldsymbol{\lambda}_i$ and $\boldsymbol{\lambda}_i^T \boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{x}$ in the trace function.

The gradients with reference to $\boldsymbol{\lambda}$ should be 0.

$$\frac{\delta g(\boldsymbol{\lambda})}{\delta \boldsymbol{\lambda}_i} = -\boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i} \boldsymbol{\lambda}_i + \boldsymbol{w}_{q_i}^T = \boldsymbol{0}, \forall i. \tag{9}$$

We can obtain the optimal $\boldsymbol{\lambda}$ as below,

$$\boldsymbol{\lambda}_i^* = [\boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i}]^{-1} \boldsymbol{w}_{q_i}^T, \forall i. \tag{10}$$

The optimal $\delta \boldsymbol{w}$ can be derived as below,

$$\begin{aligned}\delta \boldsymbol{w}^* = -&\left(\sum_i \boldsymbol{e}_{q_i} \boldsymbol{w}_{q_i} [\boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i}]^{-1} \boldsymbol{e}_{p\_q_i}^T\right) \\ &\times (2\boldsymbol{x}\boldsymbol{x}^T)^{-1}.\end{aligned} \tag{11}$$

The minimal loss/error corresponding to the optimal $\delta \boldsymbol{w}$ can be obtained by

$$\begin{aligned}L^* =& \frac{1}{2}\sum_i \boldsymbol{\lambda}_i^T \boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i} \boldsymbol{\lambda}_i \\ =& \frac{1}{2}\sum_i \boldsymbol{w}_{q_i} [\boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i}]^{-1} \boldsymbol{w}_{q_i}^T.\end{aligned} \tag{12}$$

Remark 4.1. **Dampening for the inverse.** If $2\boldsymbol{x}\boldsymbol{x}^T$ is not full rank with difficulties for the inversion $(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}$, the dampening technique is adopted to compute $(2\boldsymbol{x}\boldsymbol{x}^T + \gamma \boldsymbol{I})^{-1}$ instead, with γ as the dampening ratio.

Remark 4.2. **Separate row computation.** For the optimal perturbation in Equation (11), since $\boldsymbol{e}_{q_i}$ is a one-hot vector, $\boldsymbol{e}_{q_i} \times A$ only has non-zero values in the q_i^{th} row with all zeros for all other rows. Thus, in Equation (11), each term with the index i in the sum just computes the q_i^{th} row in the outputs and the computation of the q_i^{th} row does not affect the q_s^{th} row, $\forall s \neq i$. Specifically, we have the following,

$$[\delta \boldsymbol{w}^*]_{q_i,:} = -\boldsymbol{w}_{q_i} [\boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i}]^{-1} \boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \tag{13}$$

Remark 4.3. **Full interactions between pruned weights.** For our optimal perturbation in Equation (11) and optimal loss in Equation (12), our solution is not the simple sum of multiple SRP solutions. Our solution not only depends on the multiple pruned weights, but also takes the interactions of the multiple removals (denoted by $[\boldsymbol{e}_{p\_q_i}^T (2\boldsymbol{x}\boldsymbol{x}^T)^{-1} \boldsymbol{e}_{p\_q_i}]^{-1}$) into considerations, which are unavailable in SRP without the information of other multiple removals.

Algorithm 1 Accurate post-training pruning.

Input: weight matrix $\boldsymbol{w}$, pruning rate α, block-size S, block number N, $2\boldsymbol{x}\boldsymbol{x}^T + \gamma\boldsymbol{I}$.
repeat
  Initialize the pruning mask $\boldsymbol{M} = \boldsymbol{0}$;
  Compute the inversion $[2\boldsymbol{x}\boldsymbol{x}^T + \gamma\boldsymbol{I}]^{-1}$;
  for $i = 1$ **to** N **do**
    Find the pruned weight indexes using either Solution $\mathfrak{S}$ or $\mathfrak{M}$ for *pruning mask*;
    Update $\boldsymbol{M}$ with new pruned locations;
    Compute the modifications on unpruned weights using Solution $\mathfrak{S}$ or $\mathfrak{M}$ for *optimal compensation* with $\boldsymbol{M}$;
    Update unpruned and pruned weights;
  end for
until The final linear layer is pruned.

4.2 Algorithm Design

Our pruning algorithm is shown in Algorithm 1. We need to address two key problems: the pruning mask and optimal compensation. For each problem, we have two choices, including Solution $\mathfrak{M}$ from our MRP and its simplified version, Solution $\mathfrak{S}$.

4.2.1 Pruning Mask

In the algorithm, we need to select the pruned locations and determine the pruning mask.

Solution $\mathfrak{M}$. It is too complex to follow Equation (12) to find out the pruning mask with the minimal pruning loss. Specifically, it needs to select k weights from all weights for each combination, leading to too many combinations. It also needs to compute and sort the losses of all combinations to find out the minimal loss. Thus, for unstructured pruning, we do not implement Solution $\mathfrak{M}$.

For semi-structured pruning with N:M sparsity, we implement our Solution $\mathfrak{M}$ based on our optimal loss in Equation (12). Specifically, in N:M sparsity, we split the weights into groups with M weights in each group, and then select N weights to be pruned in each group. For example, in 2:4 sparsity, there are 2 pruned weights every 4 weights. Thus, in each group with 4 weights, we use Equation (12) to select 2 weights to be pruned with the minimal loss. In particular, there are 6 combinations to select 2 elements from 4. We compute the loss with Equation (12) for each combination and find out the minimal loss with its corresponding 2 elements, which are determined to be pruned. By doing this for each group, we can determine the pruning mask for the whole matrix.

Note that it is still a simplified version of Equation (12), since each group is computed separately without interactions from other groups. Ideally, Equation (12) needs to consider all groups together, which is unaffordable. For example, if there are G groups with 6 combinations in each group for 2:4 sparsity, there are totally 6^G combinations. So we just consider the combinations within each group, without connections between groups.

Solution $\mathfrak{S}$. To reduce the complexity and make the problem tractable, we can assume that $\boldsymbol{e}_{p\_q_i}^T(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}\boldsymbol{e}_{p\_q_i}$ in Equation (12) is a diagonal matrix with all zeros for off-diagonal elements. It means that we ignore the interactions between multiple pruned locations and each pruned weight does not affect other pruned weights. Thus, Equation (12) can be transformed to the following,

$$\hat{L}^* = \frac{[\boldsymbol{w}]_{i,j}^2}{2[(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}]_{j,j}}. \tag{14}$$

We follow Equation (14) to compute the potential pruning loss for each single weight (indexed by (i, j)). Then we sort the losses of all weights and find out the K weights with smaller losses as the pruned weights. It is similar to the mask searching in SparseGPT (Frantar and Alistarh, 2023).

4.2.2 Optimal Compensation

With the pruning mask, we need to update other unpruned weights to compensate the pruning loss.

Solution $\mathfrak{M}$. To achieve the best performance, we directly follow Equation (11) to compute the modifications of other unpruned weights. In Equation (11), we do not need to exactly compute multiple matrix multiplications such as $\boldsymbol{e}_{p\_q_i}^T(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}$ and $\boldsymbol{e}_{p\_q_i}^T(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}\boldsymbol{e}_{p\_q_i}$, since they just select certain rows or columns in a matrix. Besides, the complexity of the inversion $[\boldsymbol{e}_{p\_q_i}^T(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}\boldsymbol{e}_{p\_q_i}]^{-1}$ is smaller than $(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}$ with a reduced dimension.

Solution $\mathfrak{S}$. Similar to the pruning mask, we can reduce the complexity of Equation (11) by assuming that $\boldsymbol{e}_{p\_q_i}^T(2\boldsymbol{x}\boldsymbol{x}^T)^{-1}\boldsymbol{e}_{p\_q_i}$ in Equation (11) is a diagonal matrix with all zeros for off-diagonal elements. It ignores the interactions between multiple pruned weights, and the solution is similar to that in SparseGPT (Frantar and Alistarh, 2023). For simplicity, we directly follow SparseGPT for Solution $\mathfrak{S}$ of optimal compensation.

Model & Setting	Datasets	Origin	Unstructured 50%		2:4 sparsity (50% sparsity)			
			$\mathfrak{S}\mathfrak{S}$ (SparseGPT)	$\mathfrak{S}\mathfrak{M}$ (ours)	$\mathfrak{S}\mathfrak{S}$ (SparseGPT)	$\mathfrak{S}\mathfrak{M}$ (ours)	$\mathfrak{M}\mathfrak{S}$ (ours)	$\mathfrak{M}\mathfrak{M}$ (ours)
LLaMA2-7B S=2048	wikitext2	5.472	7.052	**7.018**	10.85	10.15	10.7	**10.14**
	c4	7.263	9.305	**9.204**	13.65	12.48	13.38	**12.47**
LLaMA2-7B S=all	wikitext2	5.472	7.045	**7.019**	10.92	**10.37**	10.6	10.38
	c4	7.263	9.36	**9.247**	13.62	12.762	13.31	**12.759**
LLaMA2-13B S=2048	wikitext2	4.884	6.028	**6.001**	8.76	**8.219**	8.644	8.224
	c4	6.727	8.275	**8.21**	11.4	10.71	11.28	**10.699**
LLaMA2-13B S=all	wikitext2	4.884	6.082	**6.03**	8.732	8.239	8.65	**8.225**
	c4	6.727	8.374	**8.269**	11.36	10.796	11.23	**10.789**
LLaMA2-70B S=all	wikitext2	3.319	4.509	**4.142**	5.698	4.278	4.353	**4.278**
	c4	5.709	6.932	**6.528**	8.154	6.683	6.84	**6.683**
OPT-2.7B S=512	wikitext2	12.47	13.43	**13.29**	17.13	16.74	16.89	**16.68**
	c4	14.34	15.8	**15.66**	19.34	18.7	19.07	**18.69**
OPT-6.7B S=2048	wikitext2	10.86	11.64	**11.57**	14.16	13.73	14.19	**13.72**
	c4	12.71	13.81	**13.77**	16.42	15.86	16.34	**15.85**
OPT-30B S=all	wikitext2	9.558	9.926	**9.824**	10.9	10.7	10.88	**10.7**
	c4	11.44	12.12	**11.98**	13.16	12.93	13.13	**12.93**
BLOOM-1.7B S=512	wikitext2	15.39	19.1	**18.67**	23.7	22.91	24.01	**22.86**
	c4	19.49	22.53	**22.06**	27.02	25.95	27.42	**25.88**
BLOOM-3B S=2048	wikitext2	13.48	15.99	**15.56**	18.87	18.6	18.8	**18.57**
	c4	17.48	19.76	**19.3**	22.81	22.22	22.77	**22.2**
BLOOM-7.1B S=2048	wikitext2	11.37	13	**12.86**	14.87	14.57	14.82	**14.57**
	c4	15.2	16.71	**16.59**	18.79	18.47	18.74	**18.47**

Table 1: Perplexity comparisons for LLMs with C4 as the calibration dataset. More results are in Appendix A and B.

4.3 Accurate Pruning Algorithms

We design our post-training pruning algorithms for both unstructured and semi-structured sparsity.

4.3.1 Unstructured Post-Training Pruning

To align with SparseGPT for a fair comparison, we adopt the block pruning setting. The weight matrix is split into blocks with a number of S columns (block-size) in each block. All blocks share the same pruning rate α to keep overall pruning rate.

In Algorithm 1, for all blocks, based on how to solve the pruning mask and optimal compensation, we have two combinations, $\mathfrak{S}\mathfrak{S}$ and $\mathfrak{S}\mathfrak{M}$. The first $\mathfrak{S}$ (or $\mathfrak{M}$) denotes using Solution $\mathfrak{S}$ (or $\mathfrak{M}$) for pruning mask, and the second $\mathfrak{S}$ (or $\mathfrak{M}$) represents Solution $\mathfrak{S}$ (or $\mathfrak{M}$) for optimal compensation. We do not implement Solution $\mathfrak{M}$ for pruning mask due to its huge complexity. $\mathfrak{S}\mathfrak{S}$ is just SparseGPT.

For the number of columns S in a block, $S = 1$ leads to too many blocks with high complexity. A typical S value is 128, 512, and 2048. $S = all$ means that all columns are in the same block.

4.3.2 Semi-Structured Post-Training Pruning

Similarly, in semi-structured pruning, we can use Solution $\mathfrak{S}$ or $\mathfrak{M}$ for pruning mask and optimal compensation, leading to 4 combinations: $\mathfrak{S}\mathfrak{S}$, $\mathfrak{S}\mathfrak{M}$, $\mathfrak{M}\mathfrak{S}$, and $\mathfrak{M}\mathfrak{M}$. The first $\mathfrak{S}$ (or $\mathfrak{M}$) denotes using Solution $\mathfrak{S}$ (or $\mathfrak{M}$) for pruning mask, and the second $\mathfrak{S}$ (or $\mathfrak{M}$) is for optimal compensation.

4.4 Comparison with the SRP-based Solution

As discussed in Sec. 2.3, the SRP-based method such as SparseGPT needs to freeze (fix) all weights previous to the current pruned weight, incurring certain performance degradation without further updating the frozen unpruned weights. Different from SparseGPT, our MRP solution updates all unpruned weights, resulting in better performance. Furthermore, ours is not a simple sum of multiple SRP solutions. Instead, it depends on not only the pruned weights, but also the interactions between them, as shown in Equation (11) and (12).

5 Experimental Results

Our implementation for the proposed method is based on PyTorch (Paszke et al., 2019) and HuggingFace (Wolf et al., 2019). We sequentially prune the linear layers of the blocks in LLMs, which only loads one single block each time with significantly less memory cost (Yao et al., 2022; Frantar and

Model	Method	Sparsity: 70%			Sparsity: 80%		
		WT2	PTB	C4	WT2	PTB	C4
BLOOM-7.1B	Original	11.37	20.82	15.2	11.37	20.82	15.2
	SparseGPT	26.79	62.24	30.3	150.77	266.9	121.6
	Ours-$\mathfrak{S}\mathfrak{M}$	**22.69**	**49.35**	**25.47**	**93.48**	**168.2**	**70.75**
LLaMA2-13B	Original	4.884	50.94	6.727	4.884	50.94	6.727
	Wanda[1]	-	-	-	2e3	-	-
	SparseGPT	26.47	568	27.81	339.4	1872	262.9
	Ours-$\mathfrak{S}\mathfrak{M}$	**19.05**	**451.2**	**22.12**	**93.43**	**861.8**	**88.36**
OPT-66B	Original	9.339	13.36	10.99	9.339	13.36	10.99
	SparseGPT	16.62	28.14	16.87	1.5e4	1e4	6e3
	Ours-$\mathfrak{S}\mathfrak{M}$	**14.41**	**23.78**	**14.92**	**58.39**	**147.7**	**42.75**
LLaMA2-70B	Original	3.319	24.25	5.709	3.319	24.25	5.709
	Wanda[1]	-	-	-	1e2	-	-
	SparseGPT	9.042	56.36	11.69	30.12	285.3	33.12
	Ours-$\mathfrak{S}\mathfrak{M}$	**8.31**	**51.69**	**11.12**	**26.35**	**219.5**	**28.2**

[1] Wanda is not able to run on a single GPU for large LLMs such as LLAMA2-13B and OPT-30B. Its results are from the Wanda paper.

Table 2: Perplexity comparison with baselines. WT2 denotes WikiText2. The calibration dataset is C4. More results are shown in Appendix C.

Alistarh, 2022, 2023). We conduct experiments on one NVIDIA A100 GPU. Similar to (Dettmers et al., 2022; Frantar and Alistarh, 2023), we do not incorporate any finetuning.

Our experiments can be finished on one single GPU within a few hours. For example, for a LLaMA2-7B model, our pruning method can be finished on one single GPU within 4 hours, which is still not very long. Our method can achieve a better perplexity and accuracy especially for large sparsity, which can hardly or never be achieved by the baselines. Compared with other typical pruning-aware training methods which require to fine-tune/re-train the pruned model with massive data, our method is very efficient with a few GPU hours cost on a single GPU, since we do not require finetuning. For example, the LLaMA Pro method (Wu et al., 2024) needs over 2800 GPU hours to finetune a small part of the LLaMA2-7B model on multiple GPUs.

Models. We test our method for transformer-based LLM families (including LLaMA2 (Touvron et al., 2023), OPT (Zhang et al., 2022a), and BLOOM (Scao et al., 2022)) and Mamba-based LLMs (Gu and Dao, 2023). For each LLM family, we experiment with multiple models of different sizes to demonstrate the general performance.

Datasets. For the calibration data, for a fair comparison, we adopt the same setting as SparseGPT and Wanda (Sun et al., 2023) to randomly choose 128 segments each with 2048 tokens from the first shard of the C4 dataset (Raffel et al., 2020) or the LAMBADA dataset (Paperno et al., 2016).

For performance evaluation, we test the models on commonly used datasets including raw-WikiText2 (Merity et al., 2016), PTB (Marcus et al., 1994) and C4. We also test on ZeroShot datasets including LAMBADA (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), PIQA (Tata and Patel, 2003), ARC-Easy and ARC-Challenge (Boratko et al., 2018), and WinoGrade (ai2, 2019).

Baselines. We compare with SparseGPT, Wanda (Sun et al., 2023), and Magnitude (Zhu and Gupta, 2017) methods. We do not compare with AdaPrune (Hubara et al., 2021b) as it performs worse than SparseGPT.

Configurations. We adopt the *perplexity* to evaluate the accuracy of sparse models. To make a fair comparison, we adopt the same hyperparameters as SparseGPT, including the dampening ratio, the calibration data number, token length, and so on. We also test the accuracy on zero-shot datasets.

5.1 Results on Transformer-based LLMs

The results on transformer-based LLMs under various block-size settings are presented in Table 1. More results for OPT and BLOOM models are shown in Appendix A and B. We compare the perplexity for unstructured pruning (50% sparsity) and semi-structured pruning (2:4 sparsity). As discussed in Section 4.2, in unstructured pruning, we only have $\mathfrak{S}\mathfrak{S}$ and $\mathfrak{S}\mathfrak{M}$ since the complexity to address pruning mask with Solution $\mathfrak{M}$ is too high. In semi-structured pruning, we have 4 combinations, $\mathfrak{S}\mathfrak{S}$, $\mathfrak{S}\mathfrak{M}$, $\mathfrak{M}\mathfrak{S}$, and $\mathfrak{M}\mathfrak{M}$. $\mathfrak{S}\mathfrak{S}$ corresponds to the original SparseGPT, and other methods are based on our proposed optimal solution.

As demonstrated in Table 1, our method can achieve a lower perplexity for various models on different datasets under the same setting. Specifically, for unstructured sparsity, our perplexity is lower than SparseGPT. For 2:4 sparsity, all of our methods achieve lower perplexity than SparseGPT. $\mathfrak{M}\mathfrak{M}$ typically performs the best since it adopts more advanced techniques following our optimal solution. However, it has the highest complexity. We notice that the performance of $\mathfrak{S}\mathfrak{M}$ is very close to (or even better than) that of $\mathfrak{M}\mathfrak{M}$, with lower complexity. Thus, we suggest to use $\mathfrak{S}\mathfrak{M}$ with a limited computation budget. In the following, we mainly show the results of $\mathfrak{S}\mathfrak{M}$.

More results for other models are demonstrated in Table A1, A2 and A3 with similar observations. We also demonstrate the comparison with other baselines under different sparsity in Table 2 and Appendix C. Ours can achieve better perplexity.

Model	Method	Sparsity	Perplexity ↓	Accuracy ↑						
			LAMBADA	LAMBADA	HellaSwag	PIQA	Arc-E	Arc-C	WinoGrade	Average
Mamba-130M	Magnitude	50%	1e20	0.19	27.21	53.92	30.18	25.94	50.59	31.338
	Sparsegpt	50%	29.8	**35.65**	**32.37**	**60.88**	**41.88**	23.38	51.62	40.963
	Wanda	50%	44.99	29.38	32.1	60.28	42.47	23.89	51.14	39.877
	Ours-$\mathfrak{SM}$	50%	**28.97**	35.2	32.21	60.83	41.58	**24.23**	**51.93**	**40.997**
Mamba-370M	Magnitude	50%	9e9	2.37	29.57	55.93	29.92	24.06	50.28	32.022
	Sparsegpt	50%	13.39	45.8	39.71	**65.13**	**48.48**	26.28	53.59	46.498
	Wanda	50%	17.69	41.16	38.96	65.02	47.64	25	53.28	45.177
	Ours-$\mathfrak{SM}$	50%	**12.33**	**47.88**	**40.21**	64.69	47.64	**26.54**	**54.3**	**46.877**
Mamba-790M	Magnitude	50%	2e58	0.04	28.98	56.8	27.36	24.74	51.14	31.510
	Sparsegpt	50%	8.31	54.43	47.71	68.28	**51.94**	25.17	55.8	50.555
	Wanda	50%	11.89	48.19	46.81	68.61	52.74	26.19	53.75	49.382
	Ours-$\mathfrak{SM}$	50%	**7.865**	**56.01**	**47.96**	**68.88**	51.56	**26.28**	**55.88**	**51.095**
Mamba-1.4B	Magnitude	70%	5e6	0.29	27.29	53.24	31.27	21.25	50.2	30.590
	Sparsegpt	70%	31.66	34.66	34.66	61.1	40.91	22.7	**55.01**	41.507
	Wanda	70%	1936	4.68	28.35	56.91	35.02	21.84	51.54	33.057
	Ours-$\mathfrak{SM}$	70%	**19.65**	**41.96**	**35.74**	**61.1**	**41.16**	**22.87**	54.38	**42.868**
Mamba-2.8B	Sparsegpt	70%	9.964	53.58	42.19	63.82	**46.97**	24.83	56.27	47.943
	Wanda	70%	160.7	17.62	32.91	59.36	39.14	21.42	52.8	37.208
	Ours-$\mathfrak{SM}$	70%	**7.511**	**58.82**	**43.25**	**64.64**	46.63	**25.17**	**58.25**	**49.460**

Table 3: Results for Mamba models. The calibration dataset is LAMBADA. Magnitude, Wanda and SparseGPT are not implemented for Mamba models in original papers. We implement and adapt these baselines for Mamba.

5.2 Results for Mamba-based LLMs

The results for Mamba models (Gu and Dao, 2023) are demonstrated in Table 3. Similarly, our method can achieve a better perplexity than other baselines for various Mamba models under the same setting.

5.3 Zero-Shot Evaluation

The zero-shot results for Mamba model are demonstrated in Table 3. Our method with a better perplexity can achieve a higher average accuracy on zero-shot datasets than other baselines.

In Table 3, we can observe that with 50% sparsity, the accuracy for LAMBADA under Magnitude pruning is very low, such as 2.37% for Mamba-370M, while it is a bit higher on other datasets such as HellaSwag with 29.57%. This observation highlights that the LAMBADA dataset is quite sensitive to the model sparsity. The reason is that LAMBADA is a token prediction dataset, while other datasets are based on selection from candidate answers, such as Hellaswag to choose one from 4 candidates with 25% accuracy for random guessing. Thus, with a relatively large sparsity such as 50%, the magnitude pruned model does not perform well, just similar to or a bit better than random guessing. Then for the token prediction of LAMBADA, it achieves extremely low accuracy such as 2.37% since random guessing can hardly guess the correct token with too many choices. But its accuracy on HellaSwag can still be 30%, which is just a bit better than random guessing (25% with 4 choices). We can see that achieving good performance on the LAMBADA dataset demonstrates the superior performance of our method.

5.4 Ablation Study

We ablate different values of the dampening ratios and the number of calibration data. We test our method on the LLaMA2-7B model. As shown in Appendix D and Figure A1, by using a smaller dampening ratio or more calibration data, our performance can be better. But to make a fair comparison, we set $\gamma = 0.01$ and use 128 samples.

6 Related Work

Post-training pruning. It is challenging to apply the traditional pruning methods for LLMs, since it requires to retrain or finetune the model on the full dataset for many epochs with massive data and computation costs (Yang et al., 2023; Zhang et al., 2022b; Li et al., 2022; Zhan et al., 2021, 2024). To address this problem, post-training pruning for LLMs are explored to prune the model with a small amount of calibration data, requiring much less resources compared with retraining. The post-training idea is originally proposed in quantization (Nagel et al., 2020; Hubara et al., 2021b; Shen et al., 2024a; Frantar et al., 2022) for transformers and LLMs, and then successfully applied for pruning (Hubara et al., 2021a; Kwon et al., 2022; Shen et al., 2024b; Frantar and Alistarh, 2023; Shen et al., 2024c). Post-training compression usually investigates the compression for a single layer of the LLM instead of the whole model for simplicity. The memory cost is reduced significantly as the memory only loads one block at a time (Hubara et al., 2021a; Frantar et al., 2022).

Post-training solvers. AdaPrune (Hubara et al., 2021a) uses weight magnitudes to determine the pruning mask, and then uses an optimizer such as SGD to update unpruned weights and improve the performance based on a small amount of calibration data. Its performance is sub-optimal due to the limited number of finetuning data. To further improve the performance, the optimization-based post-training methods are proposed such as OBC (Frantar and Alistarh, 2022) and SparseGPT (Frantar and Alistarh, 2023). Based on the SRP and its solution (Singh and Alistarh, 2020), OBC introduces a greedy solver to remove one single weight at a time, and then reconstructs the remaining weights following closed-form solutions in each iteration. SparseGPT further improves the solution of OBC and applies to the largest available open-source LLM models, achieving 60% unstructured sparsity with SOTA performance in perplexity. Besides optimization-based methods, the heuristic Wanda (Sun et al., 2023) suffers from significant accuracy loss when the sparsity is large.

7 Conclusion

We first formulate the MRP in LLMs to prune multiple weights simultaneously. Then we derive the optimal solution. Based on the optimal solution, we propose accurate post-training pruning algorithms for unstructured and semi-structured sparsity. Our comprehensive experiments demonstrate that our method is more accurate than SOTA baselines under the same configurations.

Limitations

The complexity of our method may be higher than SparseGPT, as we need to invert the matrix multiple times. As discussed in Sec. 4.2.2, we can avoid certain matrix multiplications with row or column selection, and the complexity of matrix inversion is reduced due to a smaller matrix dimension. Our method can still be finished with one single GPU within a few hours.

References

2019. Winogrande: An adversarial winograd schema challenge at scale.

Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, Pritish Yuvraj, Rajarshi Das, Andrew McCallum, Maria Chang, Achille Fokoue-Nkoutche, Pavan Kapanipathi, Nicholas Mattei, et al. 2018. A systematic classification of knowledge, reasoning, and context within the arc dataset. *arXiv preprint arXiv:1806.00358*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

Elias Frantar and Dan Alistarh. 2022. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

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

Elias Frantar, Eldar Kurtic, and Dan Alistarh. 2021. M-fac: Efficient matrix-free approximations of second-order information. *Advances in Neural Information Processing Systems*, 34:14873–14886.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. 2021a. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in neural information processing systems*, 34:21099–21111.

Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. 2021b. Accurate post training quantization with small calibration sets. In *International Conference on Machine Learning*, pages 4466–4475. PMLR.

Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems*, 35:24101–24116.

Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. 2020. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3187–3199, Online. Association for Computational Linguistics.

Yanyu Li, Pu Zhao, Geng Yuan, Xue Lin, Yanzhi Wang, and Xin Chen. 2022. Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization. *arXiv preprint arXiv:2206.01198*.

Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

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

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Xuan Shen, Peiyan Dong, Lei Lu, Zhenglun Kong, Zhengang Li, Ming Lin, Chao Wu, and Yanzhi Wang. 2024a. Agile-quant: Activation-guided quantization for faster inference of llms on the edge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18944–18951.

Xuan Shen, Zhenglun Kong, Changdi Yang, Zhaoyang Han, Lei Lu, Peiyan Dong, et al. 2024b. Edgeqat: Entropy and distribution guided quantization-aware training for the acceleration of lightweight llms on the edge. *arXiv preprint arXiv:2402.10787*.

Xuan Shen, Pu Zhao, Yifan Gong, Zhenglun Kong, Zheng Zhan, Yushu Wu, Ming Lin, Chao Wu, Xue Lin, and Yanzhi Wang. 2024c. Search for efficient large language models. *arXiv preprint arXiv:2402.10787*.

Sidak Pal Singh and Dan Alistarh. 2020. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109.

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*.

Sandeep Tata and Jignesh M Patel. 2003. Piqa: An algebra for querying protein data sets. In *15th International Conference on Scientific and Statistical Database Management, 2003.*, pages 141–150. IEEE.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ping Luo, and Ying Shan. 2024. Llama pro: Progressive llama with block expansion. *arXiv preprint arXiv:2401.02415*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Changdi Yang, Pu Zhao, Yanyu Li, Wei Niu, Jiexiong Guan, Hao Tang, Minghai Qin, Bin Ren, Xue Lin, and Yanzhi Wang. 2023. Pruning parameterization with bi-level optimization for efficient semantic segmentation on the edge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15402–15412.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

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*.

Zheng Zhan, Yifan Gong, Pu Zhao, Geng Yuan, Wei Niu, Yushu Wu, Tianyun Zhang, Malith Jayaweera, David Kaeli, Bin Ren, Xue Lin, and Yanzhi Wang. 2021. Achieving on-mobile real-time super-resolution with neural architecture and pruning search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4821–4831.

Zheng Zhan, Zhenglun Kong, Yifan Gong, Yushu Wu, Zichong Meng, Hangyu Zheng, Xuan Shen, Stratis Ioannidis, Wei Niu, Pu Zhao, and Yanzhi Wang. 2024. Exploring token pruning in vision state space models. *arXiv preprint arXiv:2409.18962*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022a. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Yihua Zhang, Yuguang Yao, Parikshit Ram, Pu Zhao, Tianlong Chen, Mingyi Hong, Yanzhi Wang, and Sijia Liu. 2022b. Advancing model pruning via bi-level optimization. *Advances in Neural Information Processing Systems*, 35:18309–18326.

Yingtao Zhang, Haoli Bai, Jialin Zhao, Haokun Lin, Lu Hou, and Carlo Vittorio Cannistraci. 2024. Plug-and-play: An efficient post-training pruning method for large language models. In *The Twelfth International Conference on Learning Representations*.

Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

Appendix

A Results on OPT Models

The results for OPT models are demonstrated in Table A1. Similarly, our methods can achieve a better perplexity than SparseGPT for various models on different datasets under the same setting. For 2:4 sparsity, $\mathfrak{MM}$ typically can achieve the best performance and the performance of $\mathfrak{SM}$ is very close to (or even better than) that of $\mathfrak{MM}$ with lower complexity.

Model & Setting	Datasets	Original perplexity	Unstructured 50%		2:4 sparsity			
			$\mathfrak{SS}$ (SparseGPT)	$\mathfrak{SM}$ (ours)	$\mathfrak{SS}$ (SparseGPT)	$\mathfrak{SM}$ (ours)	$\mathfrak{MS}$ (ours)	$\mathfrak{MM}$ (ours)
OPT-125M S=128	wikitext2	27.65	37.02	**35.75**	58.78	52.4	59.13	**52.31**
	ptb	38.99	55.4	**55.37**	92.42	**83.22**	91.68	83.23
	c4	26.56	33.49	**32.72**	51.49	45.76	49.64	**45.59**
OPT-350M S=128	wikitext2	22	31.21	**30.12**	50.57	50.06	51.94	**48.46**
	ptb	31.07	43.44	**42.71**	72.45	**70.8**	72.23	70.96
	c4	22.59	29.17	**28.36**	46.48	42.79	46.04	**42.16**
OPT-2.7B S=512	wikitext2	12.47	13.43	**13.29**	17.13	16.74	16.89	**16.68**
	ptb	17.97	20.45	**20.28**	26.97	25.99	26.63	**25.91**
	c4	14.34	15.8	**15.66**	19.34	18.7	19.07	**18.69**
OPT-6.7B S=2048	wikitext2	10.86	11.64	**11.57**	14.16	13.73	14.19	**13.72**
	ptb	15.77	17.45	**17.33**	21.53	**20.38**	21.11	20.4
	c4	12.71	13.81	**13.77**	16.42	15.86	16.34	**15.85**
OPT-30B S=all	wikitext2	9.558	9.926	**9.824**	10.9	10.7	10.88	**10.7**
	ptb	14.04	15.3	**15.05**	16.58	**16.19**	16.53	16.2
	c4	11.44	12.12	**11.98**	13.16	12.93	13.13	**12.93**

Table A1: Perplexity comparisons for OPT models under various block-size settings.

B Results on BLOOM Models

The results for BLOOM models are demonstrated in Table A2. Similarly, our methods can achieve a better perplexity than SparseGPT for various models on different datasets under the same setting. For 2:4 sparsity, the performance of $\mathfrak{SM}$ is very close to (or even better than) that of $\mathfrak{MM}$ with lower complexity.

Model & Setting	Datasets	Original perplexity	Unstructured 50%		2:4 sparsity			
			$\mathfrak{SS}$ (SparseGPT)	$\mathfrak{SM}$ (ours)	$\mathfrak{SS}$ (SparseGPT)	$\mathfrak{SM}$ (ours)	$\mathfrak{MS}$ (ours)	$\mathfrak{MM}$ (ours)
BLOOM-560M S=128	wikitext2	22.41	29.12	**28.77**	37.58	36.28	38.78	**36.02**
	ptb	43.66	60.52	**60.36**	77.53	73.93	79.4	**73.01**
	c4	26.59	32.83	**32.12**	40.72	39.6	41.98	**39.46**
BLOOM-1.7B S=512	wikitext2	15.39	19.1	**18.67**	23.7	22.91	24.01	**22.86**
	ptb	29.99	39.23	**39.21**	48.65	45.79	48.2	**45.41**
	c4	19.49	22.53	**22.06**	27.02	25.95	27.42	**25.88**
BLOOM-3B S=2048	wikitext2	13.48	15.99	**15.56**	18.87	18.6	18.8	**18.57**
	ptb	25.34	30.47	**29.56**	38.44	**37.34**	38.91	37.48
	c4	17.48	19.76	**19.3**	22.81	22.22	22.77	**22.2**
BLOOM7.1B S=2048	wikitext2	11.37	13	**12.86**	14.87	14.57	14.82	**14.57**
	ptb	20.82	24.26	**23.97**	28.28	27.86	28.5	**27.8**
	c4	15.2	16.71	**16.59**	18.79	18.47	18.74	**18.47**

Table A2: Perplexity comparisons for BLOOM models under various block-size settings.

C Results of Other Sparsity and Baselines

We demonstrate the comparison with other baselines under different sparsity in Table A3. Our method can achieve better perplexity.

Table A3: Perplexity comparison of our method and baselines. WT2 denotes WikiText2. Wanda is not able to run on a single GPU for large LLMs such as LLAMA2-13B and OPT-30B. Its results are from the Wanda paper.

Model	Method	block-size	Sparsity: 0.7			Sparsity: 0.8		
			WT2	PTB	C4	WT2	PTB	C4
OPT-6.7B	Original	-	10.86	15.77	12.71	10.86	15.77	12.71
	SparseGPT	512	20.7	31.3	21.68	84.43	103.6	71.8
	Ours-$\mathfrak{SM}$	512	**19.84**	**31.06**	**21.18**	**80.52**	**101.34**	**69.2**
LLaMA2-7B	Original	-	5.472	37.91	7.263	5.472	37.91	7.263
	Wanda	-	-	-	-	1e5	-	-
	SparseGPT	512	26.25	2203	28.49	104	4358	104.6
	Ours-$\mathfrak{SM}$	512	**24.53**	**1812**	**26.75**	**91.92**	**3422**	**90.12**
BLOOM-7.1B	Original	-	11.37	20.82	15.2	11.37	20.82	15.2
	SparseGPT	2048	26.79	62.24	30.3	150.77	266.9	121.6
	Ours-$\mathfrak{SM}$	2048	**22.69**	**49.35**	**25.47**	**93.48**	**168.2**	**70.75**
LLaMA2-13B	Original	-	4.884	50.94	6.727	4.884	50.94	6.727
	Wanda	-	-	-	-	2e3	-	-
	SparseGPT	all	26.47	568	27.81	339.4	1872	262.9
	Ours-$\mathfrak{SM}$	all	**19.05**	**451.2**	**22.12**	**93.43**	**861.8**	**88.36**
OPT-30B	Original	-	9.558	14.04	11.44	9.558	14.04	11.44
	SparseGPT	all	15.42	25.63	17.09	604.4	303.7	349
	Ours-$\mathfrak{SM}$	all	**13.54**	**22.42**	**15.56**	**50.61**	**71.98**	**39.15**
OPT-66B	Original	-	9.339	13.36	10.99	9.339	13.36	10.99
	SparseGPT	all	16.62	28.14	16.87	1.5e4	1e4	6e3
	Ours-$\mathfrak{SM}$	all	**14.41**	**23.78**	**14.92**	**58.39**	**147.7**	**42.75**
LLaMA2-70B	Original	-	3.319	24.25	5.709	3.319	24.25	5.709
	Wanda	-	-	-	-	1e2	-	-
	SparseGPT	all	9.042	56.36	11.69	30.12	285.3	33.12
	Ours-$\mathfrak{SM}$	all	**8.31**	**51.69**	**11.12**	**26.35**	**219.5**	**28.2**

D Ablation Study

We ablate different values of the dampening ratios and the number of calibration data. We test our $\mathfrak{SM}$ method on the LLAMA2-7B model. As shown in Figure A1, by using a smaller dampening ratio or more calibration data, our performance can be better. But to make a fair comparison, we set $\gamma = 0.01$ and use 128 samples.

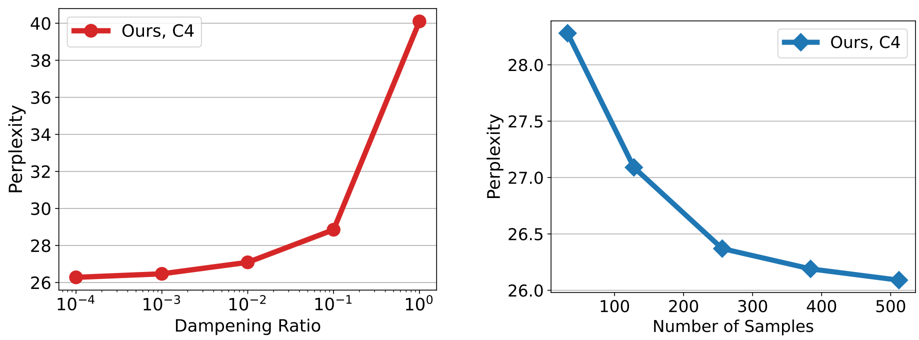

Figure A1: Ablation study for the dampening ratio and the number of samples.