Structured Optimal Brain Pruning for Large Language Models

Jiateng Wei[1] Quan Lu[2] Ning Jiang[2]
Siqi Li[1] Jingyang Xiang[1] Jun Chen[3]* Yong Liu[1]*
[1]APRIL Lab, Zhejiang University
[2]Mashang Consumer Finance Co, Ltd [3]Zhejiang Normal University
{jiatengwei, lsq4747, jingyangxiang}@zju.edu.cn
{quan.lu02, ning.jiang02}@msxf.com
junc.change@gmail.com, yongliu@iipc.zju.edu.cn

Abstract

The massive parameters and computational demands hinder the widespread application of Large Language Models (LLMs). Network pruning provides a practical solution to this problem. However, existing pruning works for LLMs mainly focus on unstructured pruning or necessitate post-pruning fine-tuning. The former relies on special hardware to accelerate computation, while the latter may need substantial computational resources. In this paper, we introduce a retraining-free structured pruning method called SoBP (**S**tructured **O**ptimal **B**rain **P**runing). It leverages global first-order information to select pruning structures, then refines them with a local greedy approach, and finally adopts module-wise reconstruction to mitigate information loss. We assess the effectiveness of SoBP across 14 models from 3 LLM families on 8 distinct datasets. Experimental results demonstrate that SoBP outperforms current state-of-the-art methods.

1 Introduction

Extensive research has shown that Large Language Models (LLMs) (Zhang et al., 2022a; Touvron et al., 2023; Workshop et al., 2022) possess outstanding language comprehension and text generation capabilities, which provide them with significant potential and value across diverse industries (Hadi et al., 2023; Zhao et al., 2023). However, such high performance often comes with enormous model size, high computational overhead, and significant memory consumption (Brown et al., 2020; Xiao et al., 2023), making it impractical for many scenarios. Many studies in model compression (Wei et al., 2023; Frantar et al., 2023; Frantar and Alistarh, 2023; Sun et al., 2023) have been investigated to tackle this problem. Among them, network pruning is an effective technique to compress the model size while maintaining the performance.

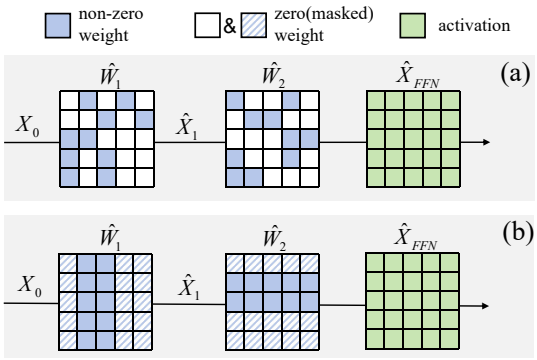

Figure 1: General overview of computation in Feed Forward Network. (a) Unstructured pruned weights. (b) Structured pruned weights.

For LLMs' pruning, most methods adopt unstructured pruning. They analyze the importance of individual weight, zeroing out unimportant weights to obtain sparse weight matrices, as shown in Fig.1(a). However, unstructured sparse weights require specialized data structure to store and specific hardware to accelerate computation (Zhang et al., 2020; Xie et al., 2021). For general computing devices, unstructured pruning does not bring any reduction in storage or computation.

Another branch of network pruning focuses on structured pruning. Structured pruning eliminates several rows or columns of the weight matrix or deletes entire layers from the network, which makes it more general for various computing devices (Anwar et al., 2017). However, these coarser pruning granularities suffer from non-negligible performance degradation. Therefore, most structured pruning methods require fine-tuning to restore network performance (Han et al., 2015). Given the large number of parameters in LLMs, it might be incredibly resource-intensive and time-consuming.

To address the above issues, we propose SoBP (**S**tructured **O**ptimal **B**rain **P**runing), a structured pruning method for LLMs that does not require fine-tuning. It takes into account both the hardware-

*Corresponding authors.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 13991–14007
November 12-16, 2024

friendliness of the pruned network and the efficiency of the pruning process. When considering this problem of retraining-free structured pruning, two principal aspects need to be critically addressed: ❶ Finding the appropriate pruning structures. ❷ Applying a suitable strategy to recover the performance of the pruned network.

Regarding the first aspect, considering the setting of structured pruning, we prune the heads in Multi-Head Attention (MHA) and the intermediate dimension of Feed Forward Network (FFN), similar to previous studies (Kwon et al., 2022; An et al., 2024). Since the importance level varies among different layers in LLMs (Dalvi et al., 2020; Kim et al., 2024), it is necessary to select pruning structures according to their importance within the network. We use global importance scores, which are based on first-order information, to select the pruning structures of each MHA and FFN module. However, first-order information ignores the correlation between different structures, which means that the structures selected by it might fail to minimize the change in local output. So we further refine the pruning structures of each module by a local greedy approach.

For the second aspect, since fine-tuning is omitted, we reconstruct the output weights of each module after pruning. Our method inherits the legacy of the previous Optimal Brain series (LeCun et al., 1989; Hassibi et al., 1993; Frantar and Alistarh, 2022), extending them into the field of structured pruning and implementing some transformations to address numerical stability issues. Meanwhile, we also introduce a method to compensate for the cumulative errors at a high compression rate.

The overview of SoBP is shown in Fig.2. To summarize the contribution of our work:

• We adopt a global-local manner to find out appropriate pruning structures. We select the pruning structures of each module based on global importance scores and further refine them by a local greedy approach.

• Based on the explored pruning structures, we adopt module-wise reconstruction to align the outputs between the pruned network and the original network, avoiding post-pruning fine-tuning.

• We evaluate the effectiveness of SoBP through extensive experiments. The results demonstrate that SoBP exhibits superior performance across various models at different pruning rates, outperforming current state-of-the-art methods.

2 Related Work

2.1 Network pruning for LLMs

Extensive studies have suggested that by removing certain weights from the network, network pruning helps in reducing the model's parameters and accelerating inference (LeCun et al., 1989; Hassibi et al., 1993; He et al., 2018; Behnke and Heafield, 2020). This method also receives considerable attention with respect to LLMs.

Unstructured pruning. Unstructured pruning sets individual weights to zero according to their importance. Most LLMs' pruning work follows this paradigm. SparseGPT (Frantar and Alistarh, 2023) zeros out unimportant weights and adjusts the remaining weights based on the Hessian matrix. Wanda (Sun et al., 2023) uses the product of the weight's magnitude and the norm of the activation to remove unimportant weights. E-Sparse (Li et al., 2023) further introduces information entropy to measure the importance of weights. Plug-and-Play (Zhang et al., 2024) ensures unstructured sparsity of the pruned weights by relative importance.

Structured pruning. Structured pruning prunes entire rows or columns of weight matrices or even deletes entire network layers. LLM-pruner (Ma et al., 2023) removes non-critical coupled structures and restores network performance via LoRA (Hu et al., 2022). Compresso (Guo et al., 2023) combines LoRA with L_0 regularization, simultaneously performing network pruning and parameter fine-tuning. Shortened LLaMA (Kim et al., 2024) deletes entire layers with low importance and retrains the pruned network to avoid performance degradation. To further improve the efficiency of the pruning process, recent researches pay attention to retraining-free structured pruning. PTP (Kwon et al., 2022) utilizes second-order information to determine pruning structures and a least squares method to minimize information loss. FLAP (An et al., 2024) discovers the stability of certain channels in LLMs' activation, the weights corresponding to these channels can be directly removed and compensated with additional bias. SliceGPT (Ashkboos et al., 2024) makes use of the network's rotational invariance and applies PCA to reduce the embedding dimension of LLMs.

2.2 Low-rank decomposition for LLMs

Parallel to pruning, low-rank decomposition is also an effective network compression technique (Sainath et al., 2013; Idelbayev and Carreira-

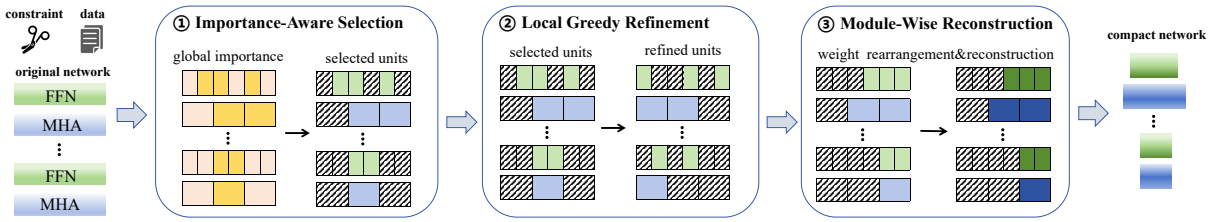

Figure 2: Framework of SoBP. ① Select pruning units of each module based on global importance scores. ② Refine the selected units by a greedy approach. ③ Reconstruct the weight matrix to maintain the output.

Perpinán, 2020). Recent research suggests that decomposing LLMs' weights is effective in reducing parameters. LoRD (Kaushal et al., 2023) first applies Singular Value Decomposition (SVD) to LLMs, demonstrating the high potential of low-rank decomposition. ASVD (Yuan et al., 2024) considers the activation outliers during the decomposition of weights. SVD-LLM (Wang et al., 2024) discovers the misalignment between singular values and compression loss and proposes a data whitening strategy to address this problem.

3 Preliminary

SoBP prunes heads of MHA and the intermediate dimension of FFN. We refer to them as pruning structures or pruning units and use neurons to denote the intermediate dimension of FFN in the following text.

3.1 Hessian Matrix Based Pruning

Given an input feature $X \in \mathbb{R}^{T\times C_{in}}$ and weight $W \in \mathbb{R}^{C_{in}\times C_{out}}$, where T represents the number of tokens of the sequence, C_{in},C_{out} are the number of input and output channels of a linear layer, respectively. A pruning algorithm finds $\widehat{W}$ to minimize the output error, formulated as:

$$\arg\min_{\widehat{W}} ||XW - X\widehat{W}||_2^2 \quad \text{s.t} \quad r(\widehat{W}) \geq r_t \quad (1)$$

where $r(\cdot)$ represents the current pruning rate and r_t is the target pruning rate.

Recent study OBC (Frantar and Alistarh, 2022) observes that squared term in Eq.(1) can be reformulated as $\sum_{j=1}^{C_{out}} ||XW_{:,j} - X\widehat{W}_{:,j}||_2^2$. So we can consider each output channel (column of weight matrix) independently. For weight $W_{i,j}$, the minimal error incurred by pruning it is $e_{i,j}$ and the update to the weights in the j^{th} column is $\delta W_{:,j}$, they can be calculated as:

$$e_{i,j} = \frac{W_{i,j}^2}{H_{i,i}^{-1}} \quad \delta W_{:,j} = -\frac{W_{i,j}}{H_{i,i}^{-1}} \cdot H_{:,i}^{-1} \quad (2)$$

where $H = X^T X + \alpha I$ represents the local Hessian matrix (Hassibi et al., 1993; Frantar and Alistarh, 2022), α is a small positive number. After pruning $W_{i,j}$, the inverse of the Hessian matrix corresponding to the remaining weights needs to be recalculated. OBC proposes an efficient algorithm, which calculates the inverse through Gaussian elimination:

$$[H^{-1}]_{-i} = (H^{-1} - \frac{1}{H_{i,i}^{-1}} H_{:,i}^{-1} H_{i,:}^{-1})_{-i} \quad (3)$$

3.2 MHA and FFN Under Masks

As the basic block of LLMs, Transformer (Vaswani et al., 2017) contains two modules, MHA and FFN. In the l^{th} layer of a Transformer network, MHA usually contains four linear layers, with the weight matrices W_q^l,W_k^l,W_v^l,$W_o^l \in \mathbb{R}^{D\times D}$, where D is the embedding dimension. FFN usually contains two linear layers, with weight matrices $W_1^l \in \mathbb{R}^{D\times N}$,$W_2^l \in \mathbb{R}^{N\times D}$, N is the number of neurons. Our work focuses on deleting entire rows or columns of these matrices. We define binary masks $M_H^l \in \{0,1\}^{H_h}$, $M_N^l \in \{0,1\}^N$ to remove (0) or retain (1) rows and columns, H_h is the number of heads. For simplicity, the following equations ignore the skip connection and normalization layer. The operations of the MHA module under the mask are denoted by:

$$X_{head}^{l,i} = \text{Attn}\left(X^{l-1}, W_q^{l,i}, W_k^{l,i}, W_v^{l,i}\right) \circ M_H^{l,i} \quad (4)$$

$$\widehat{X}_{MHA}^l = \text{Concat}\left(X_{head}^{l,1}, \ldots, X_{head}^{l,H_h}\right) W_o^l \quad (5)$$

where Attn represents the attention mechanism (Vaswani et al., 2017), X^{l-1} is the input of the l^{th} layer, $X_{head}^{l,i}$ is the output of the i^{th} head, Concat is the concatenation operation, $\widehat{X}_{MHA}^l$ is the output of the masked MHA module. The symbol $\circ$ represents element-wise multiplication. Right multiplication means it acts on the columns. For the FFN module, its operation under the mask is

denoted by:

$$\widehat{X}^l_{FFN} = \sigma\left(\widehat{X}^l_{MHA} W^l_1 \circ M^l_N\right) W^l_2 \quad (6)$$

where σ is a non-linear activation function. The overview of Eq.(6) is shown in Fig.1(b).

4 Method

The framework of SoBP has three key components: global importance-aware selection, local greedy refinement, and module-wise reconstruction.

4.1 Global Importance-Aware Selection

We first select pruning units based on their global importance scores. When conducting network pruning, we aim to identify and remove structures that have minimal impact on the network's final prediction. Taylor expansion is a commonly used method for analyzing the importance of structures. (Michel et al., 2019; Molchanov et al., 2019). The global importance of a pruning unit can be assessed by its impact on loss. Removing more important units will result in greater loss. Similar to previous studies (Zhang et al., 2022b; Kwon et al., 2022), we perform a Taylor expansion on the loss with the variable being the mask $M = \{M^l_H, M^l_N\}^L_{l=1}$. Given calibration data $\mathcal{D}$, the expansion at $M = \mathbb{1}$ can be formulated as:

$$\mathcal{L}(M, \mathcal{D}) \approx \mathcal{L}(\mathbb{1}, \mathcal{D}) + g(M - \mathbb{1}) \quad (7)$$

where $\mathcal{L}$ represents the perplexity loss, g is the gradient of the masks, with each element $g_i = \frac{\partial \mathcal{L}}{\partial m_i}$. Most previous works ignore the first-order term since they assume that the model has converged on the training data. However, if the calibration data does not come from the original training data, this assumption does not hold (Ma et al., 2023), then we can utilize first-order information.

Our global importance score is built on the basis of first-order information. For the i^{th} unit, U_i, its initial importance score is $\widehat{I}_i = \left(\mathcal{L}(M_i, \mathcal{D}) - \mathcal{L}(\mathbb{1}, \mathcal{D})\right)^2 \approx (g_i)^2$. Since heads and neurons have different parameters and levels of importance, we introduce a transformation to balance the scores between them. The final global importance score can be expressed as:

$$I_i = \begin{cases} \widehat{I}_i & U_i \text{ is a head} \\ \frac{\lambda P_N}{P_H} \widehat{I}_i & U_i \text{ is a neuron} \end{cases} \quad (8)$$

where λ is a hyperparameter, P_H and P_N denote parameters per head and parameters per neuron, respectively. After obtaining the importance score of each unit, we have the following optimization problem:

$$\arg\min_{\mathcal{S}} \sum_{i \in \mathcal{S}} I_i \quad \text{s.t.} \sum_{i \in \mathcal{S}} P(U_i) \geq r \cdot \Theta \quad (9)$$

where $\mathcal{S}$ is a set of pruning units, $P(\cdot)$ represents the number of parameters of a unit, r is the given pruning rate, Θ is the total parameters of the model. Eq.(9) is a knapsack problem (Kwon et al., 2022). Since each unit has its own parameter size (volume) and importance score (value), the sum constraint can be seen as the knapsack capacity.

By solving the knapsack problem, we can obtain a set of pruning units with the lowest importance. However, it's important to note that we neglect higher-order terms in Eq.(7). Because given massive parameters in LLMs, they are computationally infeasible. For example, LLaMA1-13B has about 1600 heads and 5×10^5 neurons, which makes the second-order term approximately a 5×10^5 by 5×10^5 square matrix. Neglecting the higher-order terms means that we cannot consider the correlations between different units. To mitigate the negative effect of it, we introduce a greedy approach to refine the selected units within each module.

4.2 Local Greedy Refinement

Consider a problem of pruning a linear layer (with weight matrix W). Assuming the selected set of pruning units from Section 4.1 is S, we prune several rows of W based on S. The minimum error incurred by pruning this set can be calculated as:

$$E = \sum\nolimits_{j=1}^{C_{out}} [W_{S,j}]^T \left[H^{-1}_{S,S}\right]^{-1} [W_{S,j}] \quad (10)$$

Eq.(10) has an important implication, the total error incurred by pruning is related to the pruning units selected. In Section 4.1, we only consider the global importance of each unit while ignoring the correlation between them, so S may be suboptimal for minimizing E. A straightforward approach to finding the optimal set is to brute-force check all sets that have the same parameters as S. From these sets, the one with the lowest E is optimal. The number of computations required by this method is $C(row, |S|)$, C represents the calculation of combinations, row denotes the number of rows of W, $|S|$ denotes the number of selected units (rows). For the weight matrices of LLMs, the number of rows is huge, which makes the brute-force method

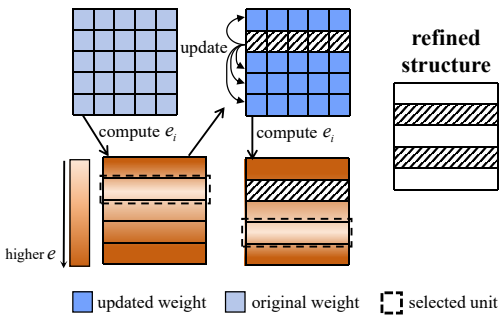

Figure 3: Greedily select pruning units. Each time select the row with the minimum error then update the remaining weights.

infeasible. For example, LLaMA1-13B, its W_2 contains 1.3×10^4 rows, the brute-force method requires about 10^{2820} computations at a pruning rate of 20%. Inspired by recent studies (Frantar and Alistarh, 2022; Kurtić et al., 2023), this problem of finding the optimal combination set can be approximated in a one-by-one greedy manner. A key insight is that the total error caused by pruning all units in a certain set, Eq.(10), is equal to pruning these units one by one while considering weight updates through Eq.(2). This one-by-one manner allows us to adopt a greedy algorithm. For the i^{th} unit, the error incurred by pruning it is e_i:

$$e_i = \sum_{j=1}^{C_{out}} \frac{W_{i,j}^2}{H_{i,i}^{-1}} \tag{11}$$

At each step, we select and prune the row with the lowest error among all rows and update the remaining weights as $W_{:,j} = W_{:,j} + \delta W_{:,j}$. $\delta W_{:,j}$ is calculated by Eq.(2), then H^{-1} is updated by Eq.(3). The process iterates until the parameter constraint is satisfied.

This greedy approach selects the local optimum at each step. Although it can not guarantee a global optimum, the result can still be considered a good approximation. The overview of this method is shown in Fig.3. We validate the effectiveness of it in the ablation study. The proof of Eq.(10) and the equivalence between pruning the whole set and the one-by-one pruning and updating manner can be found in Appendix A. Through greedy selection, we can further refine the pruning structures of each module, so that to minimize output error E between the pruned network and the original network.

4.3 Module-Wise Reconstruction

SoBP considers pruning each MHA and FFN module individually. Module-wise reconstruction aims at aligning the pruned modules' output with their counterparts in the original network. This is achieved through the following two key strategies.

Output weights reconstruction. When conducting output weights reconstruction, we refer to W_o of MHA and W_2 of FFN. Theoretically, in Section 4.2, when greedily selecting pruning units, we have updated the remaining weights. The resulting weight matrix is reconstructed. However, in practice, for LLMs with billions of parameters, running Eq.(3) tens of thousands of times can cause numerical stability problems. This prevents weights from correct updates. So after greedy refinement, we need to perform a numerically stable pruning and reconstruction. GPTQ (Frantar et al., 2023) points out that the Gaussian elimination in Eq.(3) corresponds to a Cholesky decomposition. Performing a pre-decomposition can solve the numerical stability problem. However, the decomposed Hessian inverse can only handle weights sequentially. For sparsely distributed pruning units, as shown in the right part of Fig.3, the decomposition can not be directly utilized.

This problem can be addressed by rearranging the input and weight matrices. Note that for the matrix multiplication XW, if the rows of W are permuted, the corresponding columns of X must be permuted accordingly to preserve the output. Following the rearrangement of X, the Hessian matrix is also permuted. By rearranging input and weight matrices, the decomposed Hessian inverse can correctly update the remaining weights.

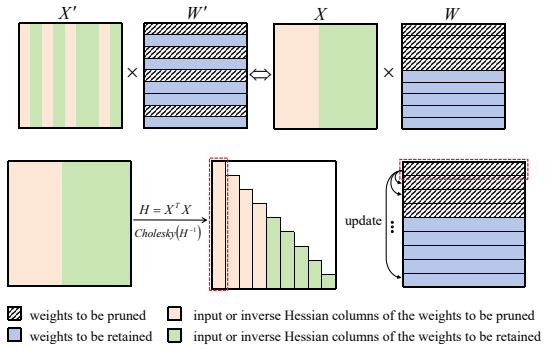

Figure 4: Rearrange input and weight matrices to ensure correct reconstruction.

When conducting reconstruction, we only consider output weights. Once the rows of these weights are pruned, the corresponding columns of the preceding weights in the same module become irrelevant and can consequently be pruned.

Input weights error compensation. The input

weights refer to W_q,W_k,W_v of MHA and W_1 of FFN. In output weights reconstruction, the pruned network aligns with the original network by minimizing $||XW - X\widehat{W}||_2^2$. A potential issue is that after pruning the previous layers, the input feature $\widehat{X}$ deviates from X of the original network. It results in the optimization problem changing to $\arg\min_{\widehat{W}}||\widehat{X}W - \widehat{X}\widehat{W}||_2^2$. This change causes a deviation between the reconstructed output and the original output. Therefore, we need to update the input weights before reconstruction to compensate for the error incurred by pruning previous layers, which ensures that $\widehat{X}$ is sufficiently close to X. It leads to the following problem:

$$\arg\min_{\delta W_p}||X_pW_p - \widehat{X}_p(W_p + \delta W_p)||_2^2 \quad (12)$$

where W_p represents the preceding weights of output weights, X_p denotes the input of W_p in the original network, $\widehat{X}_p$ denotes the counterpart of X_p in the pruned network. The closed-form solution of Eq.(12) is as follow:

$$\delta W_p = (\widehat{X}_p^T\widehat{X}_p)^{-1}\widehat{X}_p^T(X_p - \widehat{X}_p)W_p \quad (13)$$

When the input weights are updated, we also need to transform the skip connection to ensure information alignment. The skip connection can be seen as a special linear layer, its W_p is an identity matrix. This transformation will add extra parameters to the skip connection, so we only use it under certain conditions[1].

Algorithm 1 provides a comprehensive overview of SoBP.

Algorithm 1 Prune LLM with SoBP

Require: model, pruning rate r, calibration data $\mathcal{D}$

```
# global importance-aware selection
loss ← PPL(label, model(mask = 𝟙, D))
inital_scores ← loss.backward()
scores ← transform_score(inital_scores, P_H, P_N, λ)
                                                    ▷ Eq.(8)
S = {S_1, ..., S_2L} ← knapsack(scores, P_H, P_N, r, Θ)
                                                    ▷ Eq.(9)
for i ← 1 to 2L do
    W_p ← input weights in the i^th module
    if need to update W_p then
        W_p = W_p + δW_p                            ▷ Eq.(13)
    end if
    W ← output weights in the i^th module
    X ← input feature of W
    # local greedy refinement
    S_i ← greedy_refinement(W, X, S_i)              ▷ Eq.(2-3)
    # module-wise reconstruction
    W_p ← prune_input_weight(W_p, S_i)
    W, X ← matrix_rearrange(W, X, S_i)
    W ← prune_with_cholesky(W, X, S_i)
    W ← rearrange_back(W, S_i)
end for
return pruned compact model
```

5 Experiments

We compare SoBP with different methods on generative and zero-shot tasks. Meanwhile, we perform ablation studies to evaluate the contribution of each component in SoBP, as well as the impact of calibration data and the hyperparameter on the pruned models' performance. Finally, we measure the inference time and throughput after pruning.

5.1 Experimental Settings

Baselines. We compare SoBP against 4 SOTA methods, two pruning methods: FLAP (An et al., 2024), SliceGPT (Ashkboos et al., 2024) and two decomposition methods: ASVD (Yuan et al., 2024), SVD-LLM (Wang et al., 2024) since both pruning and decomposition aims to reduce parameters.

[1]More details and discussions in Section 5.4.

Setup. SoBP is implemented with PyTorch(Paszke et al., 2019) and HuggingFace Transformers(Wolf et al., 2019). Unless otherwise stated, the calibration data is from the training dataset of WikiText2. The calibration data size and sequence length are 128 and 2048, respectively, using the same settings as SliceGPT. All experiments are conducted on Nvidia 24GB 3090 GPUs and 80GB A800 GPUs.
Models, Datasets. We conduct experiments on 14 models from LLaMA1, LLaMA2 (Touvron et al., 2023) and OPT (Zhang et al., 2022a) families, evaluating the compressed models via lm-eval-harness (Gao et al., 2021) on WikiText2 (Merity et al., 2017) and seven zero-shot tasks: ARC-c, ARC-e (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2020), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2019).

5.2 Performance after Compression

SliceGPT introduces additional parameters in each residual connection. There is a discrepancy in the total parameters. SliceGPT-eq is an implementation in which we control its slicing rate to achieve the same number of parameters as other methods. As shown in Table 1, SoBP outperforms other methods in 53 out of 56 comparative experiments. On generative tasks, SoBP consistently outperforms other methods, achieving lower perplexity across all models. On seven zero-shot tasks, under a low compression rate (15%), ASVD is comparable to

Model		LLaMA1-7B			LLaMA1-13B			LLaMA1-30B			LLaMA1-65B			LLaMA2-7B			LLaMA2-13B			LLaMA2-70B		
Rate	Method	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑
0%	Dense	6.7B	5.68	66.05	13.0B	5.09	68.21	32.5B	4.10	71.92	65.3B	3.56	73.01	6.7B	5.47	66.69	13.0B	4.88	69.24	69.0B	3.32	73.61
15%	FLAP	5.8B	6.44	62.69	11.1B	5.71	64.87	27.7B	4.75	69.28	55.6B	4.12	71.91	5.8B	6.50	59.97	11.1B	5.84	63.21	59.1B	3.81	72.07
	SliceGPT	6.5B	6.49	57.46	12.6B	5.74	62.81	31.5B	4.90	68.10	63.3B	4.25	70.52	6.5B	6.33	55.77	12.6B	5.61	60.96	66.5B	4.09	71.89
	SliceGPT-eq	5.8B	7.55	52.67	11.1B	6.56	57.84	27.7B	5.65	62.41	55.6B	4.95	66.97	5.8B	7.43	52.01	11.1B	6.52	54.56	59.1B	4.76	67.12
	SVD-LLM	5.8B	7.51	57.02	11.1B	F	F	27.7B	5.39	64.64	55.6B	4.67	69.84	5.8B	7.86	51.96	11.1B	6.35	61.11	59.1B	4.39	69.26
	ASVD	5.8B	6.66	**62.94**	11.1B	5.76	66.94	27.7B	4.63	**71.47**	55.6B	4.08	71.39	5.8B	6.64	63.21	11.1B	5.53	68.39	59.1B	5.88	62.65
	SoBP	5.8B	**6.39**	62.29	11.1B	**5.59**	**67.66**	27.7B	**4.56**	71.42	55.6B	**3.86**	**72.98**	5.8B	**6.13**	**63.74**	11.1B	**5.38**	**68.55**	59.1B	**3.70**	**72.37**
20%	FLAP	5.4B	6.89	61.19	10.4B	6.05	63.15	26.1B	5.13	67.77	52.3B	4.45	71.59	5.4B	7.16	56.62	10.4B	6.31	61.55	55.7B	4.12	**71.60**
	SliceGPT	6.1B	6.99	56.16	11.8B	6.13	60.66	29.5B	5.27	65.45	59.2B	4.58	68.66	6.1B	6.84	54.25	11.8B	6.06	56.78	62.2B	4.46	69.60
	SliceGPT-eq	5.4B	8.27	48.11	10.4B	7.08	56.08	26.1B	6.09	59.39	52.3B	5.33	64.43	5.4B	8.18	48.27	10.4B	7.12	51.86	55.7B	5.15	63.65
	SVD-LLM	5.4B	7.89	56.25	10.4B	F	F	26.1B	5.68	63.29	52.3B	4.91	68.48	5.4B	8.38	48.70	10.4B	6.66	58.98	55.7B	4.66	68.14
	ASVD	5.4B	8.87	61.55	10.4B	6.71	65.29	26.1B	5.21	70.22	52.3B	4.52	70.48	5.4B	8.65	59.49	10.4B	6.51	66.18	55.7B	18.34	44.05
	SoBP	5.4B	**6.78**	**62.19**	10.4B	**5.82**	**66.96**	26.1B	**4.84**	**70.87**	52.3B	**4.08**	**72.74**	5.4B	**6.53**	**63.27**	10.4B	**5.62**	**67.73**	55.7B	**3.88**	71.24
30%	FLAP	4.8B	8.23	57.30	9.2B	6.97	60.14	22.9B	5.87	64.58	45.8B	5.06	69.85	4.8B	8.85	50.91	9.2B	7.57	57.27	48.9B	4.82	69.68
	SliceGPT	5.3B	8.69	46.90	10.2B	7.35	54.26	25.5B	6.33	58.05	51.1B	5.49	63.46	5.3B	8.64	46.70	10.2B	7.44	50.10	53.7B	5.41	61.61
	SliceGPT-eq	4.8B	11.19	42.52	9.2B	8.69	48.69	22.9B	7.40	51.50	45.8B	6.33	57.01	4.8B	10.80	43.46	9.2B	9.17	46.25	48.9B	6.23	56.19
	SVD-LLM	4.8B	9.52	51.39	9.2B	F	F	22.9B	6.49	59.61	45.8B	5.58	65.35	4.8B	10.66	44.77	9.2B	8.00	53.16	48.9B	5.44	64.65
	ASVD	4.8B	84.72	45.55	9.2B	13.30	57.47	22.9B	8.16	61.88	45.8B	6.86	64.78	4.8B	1.6e2	42.82	9.2B	15.92	53.32	48.9B	1.1e3	34.68
	SoBP	4.8B	**7.57**	**59.61**	9.2B	**6.52**	**64.50**	22.9B	**5.40**	**69.62**	45.8B	**4.56**	**72.46**	4.8B	**7.58**	**59.15**	9.2B	**6.27**	**66.82**	48.9B	**4.36**	**70.30**
40%	FLAP	4.1B	10.16	52.45	7.9B	8.02	56.80	19.7B	6.95	59.55	39.4B	5.76	68.00	4.1B	11.49	48.70	7.9B	9.07	53.18	42.1B	6.24	67.96
	SliceGPT	4.5B	15.94	39.64	8.6B	9.79	47.00	21.5B	8.22	48.90	43.2B	6.92	54.05	4.5B	12.80	41.47	8.6B	10.60	44.31	45.5B	7.08	52.00
	SliceGPT-eq	4.1B	46.08	36.78	7.9B	11.89	44.76	19.7B	9.89	46.10	39.4B	8.10	50.07	4.1B	16.02	39.63	7.9B	13.38	41.32	42.1B	8.93	48.63
	SVD-LLM	4.1B	13.85	42.95	7.9B	F	F	19.7B	7.96	54.21	39.4B	6.69	58.89	4.1B	16.14	40.47	7.9B	10.79	45.40	42.1B	6.83	58.05
	ASVD	4.1B	1.7e3	36.79	7.9B	149.94	40.13	19.7B	17.78	49.79	39.4B	15.23	50.62	4.1B	2.1e3	36.29	7.9B	71.21	42.88	42.1B	4.8e3	34.32
	SoBP	4.1B	**9.09**	**56.10**	7.9B	**7.61**	**60.34**	19.7B	**6.06**	**67.20**	39.4B	**5.10**	**71.24**	4.1B	**9.28**	**56.06**	7.9B	**7.39**	**60.86**	42.1B	**4.96**	**68.58**

Table 1: Performance of models compressed by different methods. **Bold** for the best performance. PPL is the perplexity on the WikiText2 test dataset. Avg is the average accuracy on 7 zero-shot tasks. The detailed results are shown in Table 7. F indicates that we failed to conduct comparisons due to errors in the open-source code.

SoBP. However, as the compression rate increases, its performance significantly declines, while SoBP still maintains relatively good performance. When compressing 20% parameters of LLaMA2-70B, FLAP outperforms SoBP by 0.36% on average accuracy. However, in other experiments, SoBP consistently outperforms FLAP. Specifically, at a 40% compression rate of LLaMA1-30B, SoBP outperforms FLAP by 7.65%. Moreover, we find that the advantage of SoBP over FLAP is more pronounced in OPT family[2]. Additionally, we find that SoBP is more favorable for larger models. After pruning 30% of the parameters from LLaMA1-65B and OPT-66B, the models retain 99.2% (72.46/73.01) and 99.9% (63.05/63.08) of their performance on seven zero-shot tasks, respectively. Experimental results of the OPT family are shown in Table 8.

5.3 Compression Time

Most LLaMA and OPT models can be compressed by SoBP on a single 80G A800 GPU (models larger than 30B need 2 A800 to obtain first-order information, then the rest pruning work can be done on a single A800). Table 2 presents the time consumption of different methods on different models at a compression rate of 20%. As shown in the table, the compression time of SoBP is longer than that of FLAP and SliceGPT, comparable to SVD-LLM, and shorter than that of ASVD.

5.4 Ablation Study

Evaluation of each component. We evaluate the contribution of each component in SoBP by incrementally incorporating them, denoted as GI, Rec, and LGR. GI: select pruning units based on global importance scores and directly remove them. Rec: apply module-wise reconstruction after pruning. GI+LGR+Rec: select pruning units for each module, then refine them via local greedy refinement, and finally, apply module-wise reconstruction. After applying GI, different modules of the network have varying pruning rates. As shown in Fig.5, GI prefers to prune the latter part of the network. This phenomenon may be explained by recent work (Fan et al., 2024), which suggests that LLMs can produce reliable results without using all transformer layers for inference, indicating that deeper layers may exhibit a higher degree of redundancy.

Method	LLaMA2-7B	LLaMA1-30B	OPT-6.7B	OPT-30B
FLAP	1min	3min	1min	1.5min
SliceGPT	12min	40min	14min	36min
SVD-LLM	24min	2h09min	33min	2h41min
ASVD	17h10min	109h50min	20h54min	103h05min
SoBP	18min	1h05min	35min	3h47min

Table 2: Compression time of different methods.

To demonstrate the effectiveness of GI, we show the results of weight magnitude pruning, denoted as Mag. Mag: select pruning units based on weight magnitude $\sum_{j=1}^{C_{out}} |W_{i,j}|$ and directly remove them. We present the pruned models' perplexity of WikiText2 in Table 3. After incorporating each component, there is a decrease in perplexity, indicating that each component has a positive effect.

Impact of input weights error compensation. Since the input weight error compensation mechanism requires adding an additional mapping matrix to the skip connection, we apply it only to the mid-

[2]Detailed results can be found in the Appendix B.3.

Model	Mag	GI	GI+Rec	GI+LGR+Rec
LLaMA1-7B(-30%)	55243	8.53	8.04	**7.57**
LLaMA1-7B(-40%)	102250	12.30	10.85	**9.09**
LLaMA2-7B(-30%)	39589	8.38	7.80	**7.58**
LLaMA2-7B(-40%)	50374	13.66	11.46	**9.28**
LLaMA1-13B(-30%)	61222	6.76	**6.51**	6.52
LLaMA1-13B(-40%)	73490	8.66	7.99	**7.61**
LLaMA2-13B(-30%)	18688	6.58	6.32	**6.27**
LLaMA2-13B(-40%)	57253	8.84	7.74	**7.39**
OPT-6.7B(-30%)	162.73	17.10	12.69	**12.17**
OPT-13B(-30%)	1.2e6	22.09	11.43	**10.83**

Table 3: Ablation study of each component. -30% denotes pruning 30% of the parameters.

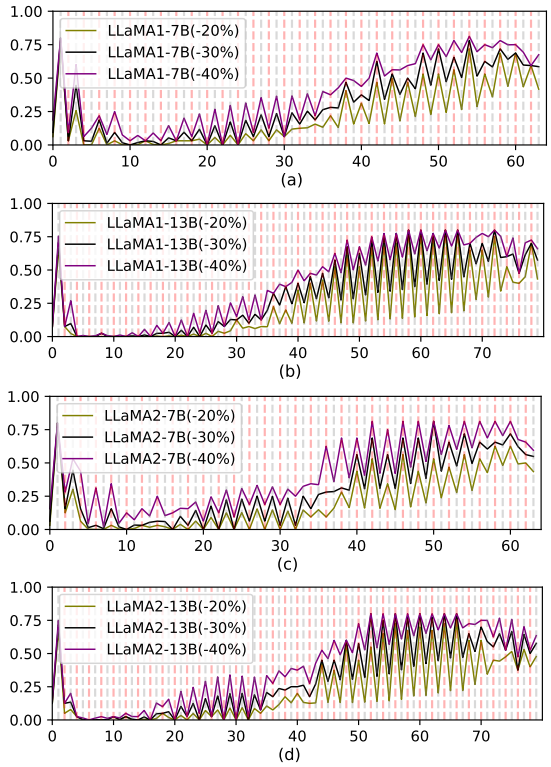

Figure 5: Pruning rates of different modules. The X-axis represents the module index, and the Y-axis represents the pruning rate. The pink and grey dashed lines correspond to the MHA and FFN modules, respectively.

dle and the fourth-to-last FFN modules when the global pruning rate exceeds 30%. We validate its effectiveness on the OPT, BLOOM(Workshop et al., 2022), and LLaMA models. Table 4 presents the perplexity of the pruned models on the WikiText2. As shown in the table, when the model size is less than 7B, the incorporation of this mechanism results in a decrease in perplexity across all cases. However, when the model size reaches 7B, this mechanism only has a positive impact under a 60% pruning rate. We think this phenomenon is due to the growth in the condition number. Eq.(13) corresponds to a system of linear equations: $AX = B$. If the condition number of A is large, X becomes numerically sensitive, and a small perturbation in B can lead to significant changes in the solution. As the model size increases, $A = (\widehat{X}_p^T \widehat{X}_p)$ tends to have a large condition number. Under lower pruning rates, $B = \widehat{X}_p^T (X_p - \widehat{X}_p) W_p$ is closer to 0, making the perturbation more impactful on the solution. Therefore, we prefer to use this mechanism when the model size is less than 7B.

Model	40%		50%		60%	
	w/o	w	w/o	w	w/o	w
OPT-125M	37.07	36.10	47.58	44.59	73.59	59.98
OPT-1.3B	20.57	20.14	31.26	27.38	48.94	39.42
OPT-2.7B	16.78	16.64	26.30	24.73	43.01	38.36
OPT-6.7B	15.24	14.93	19.99	19.02	27.38	24.39
BLOOM-560M	45.08	38.52	69.42	50.48	103.30	68.09
BLOOM-1.1B	29.59	26.41	44.75	36.89	85.97	57.74
BLOOM-1.7B	21.97	20.89	28.27	25.80	44.75	40.25
BLOOM-3B	17.86	17.65	24.96	22.69	41.06	33.35
LLaMA1-7B	9.09	9.39	12.07	12.05	18.75	17.70
LLaMA2-7B	9.28	9.66	11.69	12.17	17.43	17.04
LLaMA1-13B	7.61	7.87	9.26	9.78	12.53	13.12

Table 4: Impact of input weight error compensation. **w** denotes with compensation, and **w/o** denotes without.

Impact of calibration data. We investigate the impact of calibration data from two aspects: different data sizes and different datasets. Fig.6 presents the results at a pruning rate of 30%. The performance of SoBP improves as calibration data increases. It stabilizes when the data size reaches around 128 samples. Compared to LLaMA families, the models of the OPT family are more sensitive to the calibration dataset. When using C4 (Raffel et al., 2020) for calibration, the pruned OPT models' performance on zero-shot tasks improves. Therefore, for zero-shot tasks of OPT models, we choose 128 C4 samples for calibration.

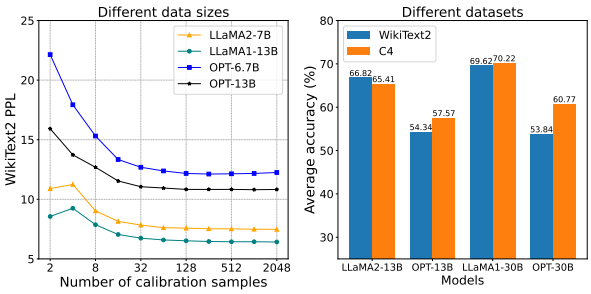

Figure 6: Impact of calibration data.

Impact of hyperparameter. We introduce a hyperparameter λ in Section 4.1 to balance the importance of head and neuron. So we evaluate its impact on the pruned models. Fig.7 presents the results at a pruning rate of 30%, which exhibits bowl-shaped curves. Hyperparameters used in our experiments can be found in the Appendix B.1.

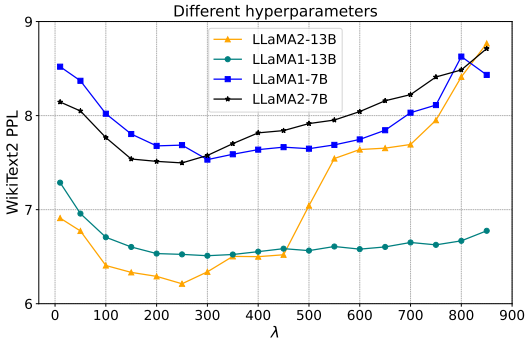

Figure 7: Impact of hyperparameter

5.5 Inference time and Throughput

The inference of LLMs can be divided into the prefill phase and the decode phase (Sheng et al., 2023). The former is compute-bound, while the latter is memory-bound. We conduct experiments on these two phases. Specifically, in the prefill phase, we set the sequence length to 2048 and batch size to 1, then measure the time consumption of inference. In the decode phase, in order to measure the maximum throughput, we set the sequence length to 128 and progressively increase the batch size until the GPUs run out of memory. For OPT-66B and LLaMA2-70B, we use two A800 GPUs. In other cases, an A800 GPU is required. As shown in Fig.8, compared to the original model (dense), pruning via SoBP achieves efficiency improvements. Unlike SliceGPT, it does not introduce additional parameters to each residual connection. In contrast to decomposition methods, it does not need to recover the original weight matrix through low-rank matrix multiplication. At a pruning rate of 30%, SoBP achieves 16.5% $(1 - 1.62/1.94)$ speed up on OPT-66B during the prefill phase. At a pruning rate of 40%, LLaMA2-70B achieves 16.3% $(1-1.6/1.91)$ speed up. The compressed models also show improvements during the decode phase. OPT-66B (-30%) and LLaMA2-70B (-40%) achieve 2.05× and 1.42× throughput, respectively. It is worth noting that the model pruned by SliceGPT-eq has a shorter inference time than SoBP during the prefill stage. That is because SliceGPT-eq controls all dimensions of the weight matrix are multiples of 8, which accelerates tensor computations in PyTorch. If we similarly ensure that all weight dimensions in the SoBP pruned model are multiples of 8, denoted as SoBP (/8), the inference performance can be further improved. At the prefill stage, the inference time with SoBP (/8) pruning is shorter on OPT-13B (-30%) and OPT-66B (-30%) compared to SliceGPT-eq pruning , and is slightly longer on

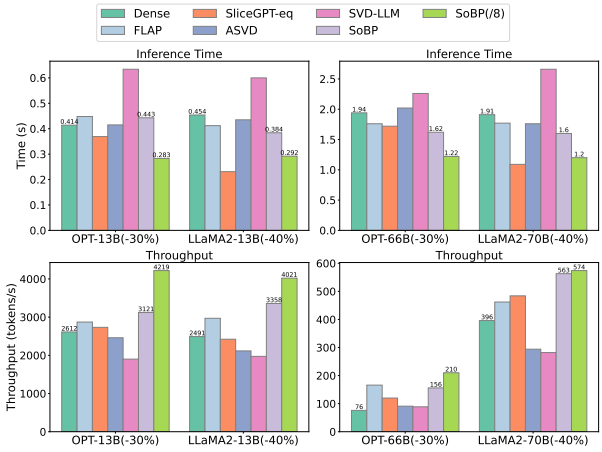

Figure 8: Inference time and throughput of different models. Dense denotes the original model. -30% denotes compressing 30% of the parameters.

LLaMA2-13B (-40%) and LLaMA2-70B (-40%). With SoBP (/8), OPT-13b (30%) and LLaMA2-13B (-40%) achieve higher throughput, 1.62× and 1.61×, respectively.

6 Conclusion

In this paper, we investigate retraining-free structured pruning for large language models. We propose a three-stage pruning algorithm named SoBP. It determines the appropriate pruning structures in a global-local manner. By reconstructing the output feature of each module, we preserve the pruned model's performance while avoiding retraining. The effectiveness of this approach is confirmed through extensive experiments across various models.

Limitations

Our method assesses the importance of each pruning structure based on first-order information while ignoring higher-order terms. It cannot consider the correlations between different pruning structures. Although the local greedy refinement approach optimizes pruning structures within each module, it still neglects the inter-module correlations. In future research, we aim to explore methods for thoroughly analyzing the correlations between pruning structures. Additionally, the global importance metric we proposed is still hand-crafted, introducing an extra hyperparameter that requires manual tuning. Thus, our method is not fully automated and necessitates domain expertise to balance model size, speed, and performance. Future research could

consider integrating AutoML to achieve a human-intervention-free pruning process.

Ethics Statement

Our SoBP aims to reduce the model size of LLMs while maintaining their performance. It can help extend the application of LLMs to resource-constrained scenarios and does not introduce additional ethical concerns.

Acknowledgements

We sincerely thank the anonymous reviewers for their thorough evaluations and insightful suggestions.

References

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2024. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10865–10873.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18.

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*.

Maximiliana Behnke and Kenneth Heafield. 2020. Losing heads in the lottery: Pruning transformer. In *The 2020 Conference on Empirical Methods in Natural Language Processing*, pages 2664–2674. Association for Computational Linguistics (ACL).

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Fahim Dalvi, Hassan Sajjad, Nadir Durrani, and Yonatan Belinkov. 2020. Analyzing redundancy in pretrained transformer models. *arXiv preprint arXiv:2004.04010*.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*.

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 *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation.

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

Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. 2023. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Babak Hassibi, David G Stork, and Gregory J Wolff. 1993. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yerlan Idelbayev and Miguel A Carreira-Perpinán. 2020. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059.

Ayush Kaushal, Tejas Vaidhya, and Irina Rish. 2023. Lord: Low rank decomposition of monolingual code llms for one-shot compression. *arXiv preprint arXiv:2309.14021*.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*.

Eldar Kurtić, Elias Frantar, and Dan Alistarh. 2023. Ziplm: Inference-aware structured pruning of language models. *Advances in Neural Information Processing Systems*, 36.

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.

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

Yun Li, Lin Niu, Xipeng Zhang, Kai Liu, Jianchen Zhu, and Zhanhui Kang. 2023. E-sparse: Boosting the large language model inference through entropy-based n: M sparsity. *arXiv preprint arXiv:2310.15929*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, EdwardZ. 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. *arXiv: Learning,arXiv: Learning*.

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. *Journal of machine learning research*, 21(140):1–67.

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, page 8732–8740.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

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*.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. 2024. Svd-llm: Truncation-aware singular value decomposition for large language model compression. *arXiv preprint arXiv:2403.07378*.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. 2023. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1648–1665.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Frank Morgan, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *Cornell University - arXiv,Cornell University - arXiv*.

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

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.

Xiaoru Xie, Jun Lin, Zhongfeng Wang, and Jinghe Wei. 2021. An efficient and flexible accelerator design for sparse convolutional neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(7):2936–2949.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. 2024. Asvd: Activation-aware singular value decomposition for compressing large language models.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Jie-Fang Zhang, Ching-En Lee, Chester Liu, Yakun Sophia Shao, Stephen W Keckler, and Zhengya Zhang. 2020. Snap: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference. *IEEE Journal of Solid-State Circuits*, 56(2):636–647.

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*.

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, 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*.

Yuxin Zhang, Mingbao Lin, Mengzhao Chen, Fei Chao, and Rongrong Ji. 2022b. Optg: Optimizing gradient-driven criteria in network sparsity. *arXiv preprint arXiv:2201.12826*.

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

A Appendix-A

A.1 Proof of Equation 10

Assuming we need to prune n rows from the weight matrix W according to the selected set S. For simplicity, consider a simple case, the weights to be pruned are the first n rows of W, $S = \{1, ..., n\}$. We have the following optimization problem:

$$\begin{aligned} &\arg\min_{\delta W} \sum_{j=1}^{C_{out}} \|XW_{:,j} - X(W_{:,j} + \delta W_{:,j})\|_2^2 \\ &\text{s.t. } v_1^T \delta W_{:,j} + W_{1,j} = 0, ..., v_n^T \delta W_{:,j} + W_{n,j} = 0, \forall j \end{aligned} \tag{14}$$

where v_i is a one-hot column vector with the i^{th} position set to 1 and all other positions set to 0. Following the approach of OBC (Frantar and Alistarh, 2022), we decompose it into an optimization problem for each column:

$$\begin{aligned} &\arg\min_{\delta w} \|Xw - X(w + \delta w)\|_2^2 \\ &\text{s.t.} \quad v_1^T \delta w + w_1 = 0, ..., v_n^T \delta w + w_n = 0 \end{aligned} \tag{15}$$

where w represents a column of W, and δw denotes the corresponding update for this column. To solve Eq.(15), we form the following Lagrangian equation:

$$\frac{1}{2}\delta w^T X^T X \delta w + \sum_{i=1}^{n} \lambda_i (v_i^T \delta w + w_i) \tag{16}$$

where λ_i is the Lagrange multiplier. Taking the derivative of Eq.(16) with respect to δw and setting the result to zero yields:

$$\delta w = -H^{-1} V \boldsymbol{\lambda} \tag{17}$$

where $H = X^T X$, $V = [v_1, ..., v_n]$ and $\boldsymbol{\lambda} = [\lambda_1, ..., \lambda_n]^T$. Noting that $\delta w_i = -w_i$, we have following linear equations:

$$H_{S,S}^{-1} \boldsymbol{\lambda} = w_S \tag{18}$$

where $H_{S,S}^{-1}$ is the submatrix of H^{-1} according to S, and $w_S = [w_1, ..., w_n]^T$ is the subvector of w. Solving Eq.(18) yields:

$$\delta w = -H^{-1} V \left[H_{S,S}^{-1}\right]^{-1} w_S \tag{19}$$

Substituting Eq.(19) into $e = \delta w^T X^T X \delta w$ and simplify it , we have:

$$e = w_S^T \left[H_{S,S}^{-1}\right]^{-1} w_S \tag{20}$$

The Eq.(20) represents the incurred error in a single column. Considering all columns yields Eq.(10).

A.2 Analysis of Equation 10

Following A.1, the selected set is $S = \{1, ..., n\}$, w is a column of weight matrix W, w_S is a subvector of w. Assuming $H_{S,S}^{-1}$ is as follows:

$$H_{S,S}^{-1} = \begin{bmatrix} A & B \\ B^T & c \end{bmatrix} \tag{21}$$

where A is a $(|S|-1)$ by $(|S|-1)$ matrix, B is a column vector and c is a scalar. The inverse of it is:

$$\left[H_{S,S}^{-1}\right]^{-1} = \begin{bmatrix} c\Lambda & -\Lambda B \\ -B^T\Lambda & \frac{1}{c} + \frac{1}{c}B^T\Lambda B \end{bmatrix} \tag{22}$$

where $\Lambda = (cA - BB^T)^{-1}$. According to Eq.(20), the pruning-induced error in this column is:

$$\begin{aligned} e &= \begin{bmatrix} w_{\widehat{S}}^T & w_n \end{bmatrix} \left[H_{S,S}^{-1}\right]^{-1} \begin{bmatrix} w_{\widehat{S}} \\ w_n \end{bmatrix} \\ &= \begin{bmatrix} w_{\widehat{S}}^T & w_n \end{bmatrix} \begin{bmatrix} c\Lambda & -\Lambda B \\ -B^T\Lambda & \frac{1}{c} + \frac{1}{c}B^T\Lambda B \end{bmatrix} \begin{bmatrix} w_{\widehat{S}} \\ w_n \end{bmatrix} \\ &= \frac{w_n^2}{c} + c w_{\widehat{S}}^T \Lambda w_{\widehat{S}} - w_n B^T \Lambda w_{\widehat{S}} - w_n w_{\widehat{S}}^T \Lambda B + \frac{w_n^2}{c} B^T \Lambda B \\ &= \frac{w_n^2}{c} + \left(w_{\widehat{S}}^T - \frac{w_n}{c} B^T\right) c\Lambda \left(w_{\widehat{S}} - \frac{w_n}{c} B\right) \\ &= \underbrace{\frac{w_n^2}{c}}_{\text{first term}} + \underbrace{\underbrace{\left[w_{\widehat{S}} - \frac{w_n}{c} B\right]^T}_{\text{updated weights}} \underbrace{\left[A - \frac{1}{c} BB^T\right]^{-1}}_{\text{updated Hessian inverse}} \underbrace{\left[w_{\widehat{S}} - \frac{w_n}{c} B\right]}_{\text{updated weights}}}_{\text{second term}} \end{aligned} \tag{23}$$

where $\widehat{S} = \{1, ..., n-1\}$. Eq.(23) has two terms. The first term is the error incurred by pruning w_n, according to Eq.(2). The second term consists of the updated weights and the updated inverse Hessian, which are calculated by Eq.(2) and Eq.(3), respectively. This indicates that the error incurred by pruning the entire set is equivalence to the error incurred by pruning a single weight (w_n) plus the error incurred by pruning the subset (without w_n) while considering weight updating and inverse Hessian updating. Repeating this method on the updated $w_{\widehat{S}}$, we can break down the problem of pruning the entire set into a one-by-one weight pruning, updating, then pruning iteration.

Thinking in reverse, to find the optimal set S that minimizes $w_S^T \left[H_{S,S}^{-1}\right]^{-1} w_S$, we can iteratively select and prune the weight with the smallest error while updating the weights and inverse Hessian until the parameter constraints are satisfied. The pruning error of the selected set equals the sum of the error at each step. By iteratively reaching the local optimum at each, we approximate the global optimum.

The above only considers the case of a single column of W, as the processing of each column is independent. Simultaneously considering all columns corresponds to the structured pruning of the entire weight matrix.

A.3 Proof of Equation 13

Consider Eq.(12) from the perspective of each column, we have:

$$\begin{aligned}&\left\|X_pW_p-\widehat{X}_p(W_p+\delta W_p)\right\|_2^2\\&=\sum_{j=1}^{C_{out}}\left\|\left(X_p-\widehat{X}_p\right)(W_p)_{:,j}-\widehat{X}_p(\delta W_p)_{:,j}\right\|_2^2\end{aligned}\tag{24}$$

where C_{out} is the number of columns in W_p. For the simplicity of notation, denote $w=(W_p)_{:,j}$, $\delta w=\delta(W_p)_{:,j}$, $X=X_p$, $\widehat{X}=\widehat{X}_p$, and $\triangle X=X_p-\widehat{X}_p$. Then we have:

$$\begin{aligned}&\left\|\triangle Xw-\widehat{X}\delta w\right\|_2^2\\&=\left(\Delta Xw-\widehat{X}\delta w\right)^T\left(\Delta Xw-\widehat{X}\delta w\right)\\&=w^T\Delta X^T\Delta Xw-w^T\Delta X^T\widehat{X}\delta w\\&\quad-\delta w^T\widehat{X}^T\Delta Xw+\delta w^T\widehat{X}^T\widehat{X}\delta w\end{aligned}\tag{25}$$

Taking the derivative with respect to δw and setting the result to zero, we obtain:

$$\delta w=(\widehat{X}^T\widehat{X})^{-1}\widehat{X}^T\Delta Xw\tag{26}$$

Eq.(26) represents the solution for each column. By considering all columns simultaneously, we obtain Eq.(13)

B Appendix-B

B.1 Details for SoBP

Hyperparameter λ for different models. In Section 4.1 we introduce a hyperparameter λ to balance the importance between head and neuron. The hyperparameters used in our experiments are shown in Table 5 and Table 6.

Model	15%	20%	30%	40%
LLaMA1-7B	250	300	300	300
LLaMA1-13B	300	350	350	300
LLaMA1-30B	250	250	250	250
LLaMA1-65B	250	200	250	250
LLaMA2-7B	250	250	300	450
LLaMA2-13B	250	250	250	250
LLaMA2-70B	250	200	150	250

Table 5: Hyperpatameters for LLaMA models

Model	10%	15%	20%	30%
OPT-125M	100	250	250	300
OPT-1.3B	250	250	550	550
OPT-2.7B	550	550	550	550
OPT-6.7B	900	900	1000	2600
OPT-13B	650	850	1000	3000
OPT-30B	650	650	650	2750
OPT-66B	450	450	450	3000

Table 6: Hyperpatameters for OPT models

Constrained knapsack algorithm. When solving Eq.(9), we do not use the naive knapsack algorithm. Instead, we add some constraints to prevent the module from being excessively pruned. Specifically, when the pruning rate of a certain module exceeds the threshold, we set the "volume" of all unpruned units of that module to infinity. This ensures that they will not be selected into the knapsack in subsequent iterations. In our experiments, we set the pruning rate threshold for MHA and FFN to 80%.

B.2 Detailed Results of LLaMA Family

Table 7 presents the accuracy of the compressed LLaMA models on seven zero-shot tasks with a compression rate r=30%. As shown in the table, SoBP outperforms other methods on most tasks.

B.3 Detailed Results of OPT Family

In the compression of the LLAMA models, FLAP is comparable to SoBP. However, when compressing OPT models, SoBP significantly outperforms FLAP. As shown in Table 8, when compressing OPT-13B, OPT-30B, and OPT-66B models by 30% of their parameters, SoBP achieves average accuracies on zero-shot tasks that exceed FLAP by 6.76%, 8.16%, and 9.34%, respectively. We suppose the reason is that FLAP relies on the stability of feature channels, which the OPT models lack. This further emphasizes the greater applicability of SoBP to different models.

When compressing OPT models, SliceGPT is comparable to SoBP. As shown in Table 9, SliceGPT outperforms SoBP in certain tasks. However, SoBP consistently achieves higher average accuracy. Meanwhile, as shown in Table 8, the models compressed via SliceGPT have a larger parameter size. When compared at an equivalent parameter size, SoBP shows a more significant advantage over SliceGPT-eq.

Model	Methods	ARC-c	ARC-e	BoolQ	HellaSwag	OpenBookQA	PIQA	WinoGrande	Avg
LLaMA1-7B	Dense	44.62	72.90	75.02	76.22	44.40	79.16	70.01	66.05
	FLAP	33.87	58.42	66.88	61.70	**40.40**	73.23	66.61	57.30
	SliceGPT	31.40	53.37	37.83	45.68	33.60	64.31	62.12	46.90
	SliceGPT-eq	29.18	44.82	37.83	37.85	29.80	60.07	58.09	42.52
	SVD-LLM	31.14	50.93	61.68	50.73	37.00	65.61	62.67	51.39
	ASVD	28.16	40.28	64.01	42.71	29.20	60.72	53.75	45.55
	SoBP	**37.97**	**61.20**	**68.41**	**67.62**	40.20	**73.56**	**68.35**	**59.61**
LLaMA1-13B	Dense	47.78	74.83	77.98	79.10	44.80	80.14	72.85	68.21
	FLAP	38.91	62.46	66.70	67.58	42.40	74.27	68.67	60.14
	SliceGPT	36.69	60.40	55.20	54.06	38.00	67.30	68.19	54.26
	SliceGPT-eq	33.36	55.18	38.35	48.56	37.00	63.93	64.48	48.69
	SVD-LLM	-	-	-	-	-	-	-	-
	ASVD	35.84	58.50	70.58	63.04	37.60	73.34	63.38	57.47
	SoBP	**43.86**	**70.41**	**71.50**	**74.92**	**42.40**	**77.09**	**71.35**	**64.50**
LLaMA1-30B	Dense	52.90	78.91	82.69	82.61	48.20	82.26	75.85	71.92
	FLAP	39.93	66.46	74.37	76.32	43.60	78.67	72.69	64.58
	SliceGPT	42.15	69.23	55.44	59.29	41.60	69.75	68.90	58.05
	SliceGPT-eq	36.86	60.19	38.81	52.20	40.20	64.91	67.32	51.50
	SVD-LLM	40.02	64.35	62.57	64.06	42.20	71.76	72.30	59.61
	ASVD	41.98	67.89	73.52	67.45	39.40	75.68	67.25	61.88
	SoBP	**50.00**	**75.34**	**80.28**	**80.12**	**47.40**	**80.20**	**74.03**	**69.62**
LLaMA1-65B	Dense	55.63	79.71	84.89	84.15	47.00	82.32	77.35	73.01
	FLAP	49.06	76.68	81.74	80.32	46.40	80.36	74.35	69.84
	SliceGPT	47.95	73.65	67.95	64.43	43.20	73.56	73.48	63.46
	SliceGPT-eq	41.47	67.51	51.35	57.57	41.60	68.99	70.56	57.01
	SVD-LLM	46.84	74.41	71.59	70.42	43.00	76.17	74.98	65.34
	ASVD	41.98	69.91	79.91	71.41	44.00	76.93	69.30	64.78
	SoBP	**53.41**	**79.71**	**84.68**	**83.13**	**46.60**	**81.83**	**77.82**	**72.46**
LLaMA2-7B	Dense	46.25	74.58	77.74	76.02	44.20	79.05	68.98	66.69
	FLAP	31.23	53.83	44.71	56.58	37.00	71.27	61.72	50.91
	SliceGPT	31.74	49.58	38.32	49.09	32.60	64.91	60.69	46.70
	SliceGPT-eq	28.07	45.45	37.83	43.06	31.40	60.01	58.41	43.46
	SVD-LLM	26.79	42.47	44.50	45.40	33.40	62.13	58.72	44.77
	ASVD	26.96	36.95	61.56	36.82	27.20	56.69	53.59	42.82
	SoBP	**37.63**	**59.81**	**71.19**	**67.27**	**38.40**	**73.50**	**66.22**	**59.15**
LLaMA2-13B	Dense	49.23	77.48	80.58	79.37	45.20	80.52	72.30	69.24
	FLAP	36.26	59.43	65.14	61.86	40.20	72.42	65.59	57.27
	SliceGPT	36.86	53.28	38.81	52.37	38.60	65.13	65.67	50.10
	SliceGPT-eq	31.57	47.31	37.80	45.93	37.40	61.37	62.35	46.25
	SVD-LLM	31.14	52.86	66.85	50.96	39.40	66.76	64.17	53.16
	ASVD	29.95	53.11	69.24	53.30	36.60	70.18	60.85	53.32
	SoBP	**47.78**	**74.45**	**79.45**	**74.55**	**43.20**	**76.50**	**71.82**	**66.82**
LLaMA2-70B	Dense	57.25	80.98	83.70	83.81	48.80	82.75	77.98	73.61
	FLAP	49.32	73.86	**82.78**	80.41	45.80	**80.90**	74.66	69.68
	SliceGPT	50.43	74.03	56.18	62.62	44.00	71.38	72.61	61.61
	SliceGPT-eq	43.94	68.06	43.27	56.11	42.00	68.12	71.82	56.19
	SVD-LLM	50.51	75.63	64.10	69.41	43.20	74.37	75.30	64.65
	ASVD	22.95	29.00	40.64	27.45	22.80	50.33	49.57	34.68
	SoBP	**56.40**	**79.21**	68.62	**81.80**	**48.20**	80.58	**77.27**	**70.30**

Table 7: Detailed zero-shot tasks results of LLaMA1 and LLaMA2 models. Compression rate $r = 30\%$.

B.4 Results of Inference time and Throughput

The inference time and throughput are tested on Pytorch (version 2.2.2) and Accelerate (version 0.29.2). Table 10-11 provide detailed results on the inference time and throughput of various compressed models across different batch sizes. During the prefill stage, the inference time increases almost linearly with the batch size, indicating that this stage is compute-bound. In the decode stage, the throughput of the model increases as the batch size increases. Compared to the original model, SoBP achieves improvements in both inference speed and throughput. The model compressed by SliceGPT-eq has a faster inference speed during the prefill stage compared to SoBP, in other cases, SoBP achieve better performance.

Model		OPT-125M			OPT-1.3B			OPT-2.7B			OPT-6.7B			OPT-13B			OPT-30B			OPT-66B		
Rate	Method	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑	Para	PPL↓	Avg↑
0%	Dense	125M	27.64	41.39	1.3B	14.61	51.09	2.7B	12.46	53.91	6.7B	10.85	58.16	12.9B	10.12	59.15	30.0B	9.56	61.85	65.7B	9.33	63.08
10%	FLAP	117M	29.27	41.14	1.2B	15.58	48.40	2.4B	13.70	52.57	6.0B	11.35	57.31	11.6B	11.31	58.10	27.0B	10.45	59.26	59.2B	10.51	60.81
	SliceGPT	163M	29.50	41.16	1.5B	15.16	49.92	2.8B	12.82	52.86	7.1B	11.07	57.07	13.5B	10.30	59.18	31.3B	9.65	61.61	68.4B	9.43	**63.14**
	SliceGPT-eq	117M	77.48	36.45	1.2B	18.01	47.59	2.4B	14.58	50.29	6.0B	11.86	55.66	11.6B	10.86	57.34	27.0B	9.99	60.50	59.2B	9.64	62.85
	ASVD	117M	31.56	40.69	1.2B	19.48	47.56	2.4B	16.99	51.69	6.0B	12.22	55.18	11.6B	14.61	56.32	27.0B	10.99	59.11	59.2B	10.53	60.33
	SoBP	117M	**27.74**	**41.97**	1.2B	**14.80**	**50.65**	2.4B	**12.46**	**53.51**	6.0B	**10.88**	**58.41**	11.6B	**10.15**	**59.29**	27.0B	**9.56**	**61.96**	59.2B	**9.34**	63.01
15%	FLAP	112M	32.01	41.23	1.13B	16.44	47.94	2.3B	14.80	52.50	5.7B	11.88	55.35	10.9B	12.19	56.98	25.5B	11.74	58.09	55.9B	11.32	59.56
	SliceGPT	156M	31.17	41.35	1.4B	15.70	49.20	2.7B	13.28	51.93	6.7B	11.26	56.58	12.7B	10.49	58.70	29.4B	9.78	61.08	64.2B	9.51	63.07
	SliceGPT-eq	112M	93.21	36.96	1.13B	19.44	46.92	2.3B	15.54	49.18	5.7B	12.32	54.76	10.9B	11.15	56.68	25.5B	10.16	59.75	55.9B	9.76	62.22
	ASVD	112M	37.13	40.59	1.13B	74.56	42.19	2.3B	43.27	43.51	5.7B	23.33	47.92	10.9B	2.7e2	44.05	25.5B	12.74	56.66	55.9B	1.4e2	52.29
	SoBP	112M	**28.21**	**41.66**	1.13B	**15.16**	**50.46**	2.3B	**12.55**	**53.56**	5.7B	**11.02**	**58.36**	10.9B	**10.21**	**59.15**	25.5B	**9.57**	**61.86**	55.9B	**9.35**	**63.26**
20%	FLAP	108M	34.45	40.99	1.07B	17.37	47.50	2.1B	15.38	50.91	5.4B	12.80	54.72	10.3B	13.18	55.36	24.0B	12.81	56.52	52.3B	12.25	57.95
	SliceGPT	148M	34.10	40.02	1.3B	16.51	48.46	2.5B	13.89	51.51	6.2B	11.60	55.50	11.9B	10.71	57.84	27.5B	9.92	60.86	60.1B	9.61	62.96
	SliceGPT-eq	108M	112.17	36.25	1.07B	21.45	45.71	2.1B	16.95	48.45	5.4B	12.89	54.08	10.3B	11.54	56.02	24.0B	10.38	59.53	52.3B	9.91	62.41
	ASVD	108M	45.52	40.14	1.07B	5.4e2	38.43	2.1B	3.9e2	38.64	5.4B	51.92	45.11	10.3B	1.2e3	39.2	24.0B	25.97	49.48	52.3B	3.7e2	44.12
	SoBP	108M	**29.01**	**41.26**	1.07B	**15.57**	**50.18**	2.1B	**12.72**	**52.65**	5.4B	**11.32**	**57.71**	10.3B	**10.40**	**59.19**	24.0B	**9.63**	**61.42**	52.3B	**9.38**	**63.29**
30%	FLAP	99.7M	40.06	**40.66**	0.95B	20.78	45.62	1.9B	18.32	47.18	4.7B	15.42	52.77	9.1B	16.01	50.81	21.1B	16.50	52.61	46.1B	16.61	53.71
	SliceGPT	134M	44.23	38.30	1.1B	19.59	46.97	2.2B	16.31	48.96	5.4B	12.80	54.16	10.3B	11.49	55.92	23.8B	10.39	59.49	51.9B	9.95	62.24
	SliceGPT-eq	99.7M	1.8e2	35.41	0.95B	28.87	43.66	1.9B	22.31	45.46	4.7B	14.78	51.79	9.1B	12.66	53.98	21.1B	11.03	58.35	46.1B	10.35	61.70
	ASVD	99.7M	1.0e2	38.51	0.95B	3.4e3	38.07	1.9B	1.7e3	38.37	4.7B	1.1e3	37.86	9.1B	7.6e3	36.85	21.1B	1.2e2	41.12	46.1B	6.6e3	35.90
	SoBP	99.7M	**32.14**	40.50	0.95B	**17.18**	**49.12**	1.9B	**13.89**	**51.45**	4.7B	**12.17**	**56.02**	9.1B	**10.83**	**57.57**	21.1B	**9.89**	**60.77**	46.1B	**9.56**	**63.05**

Table 8: Performance of compressed OPT models.

Model	Methods	ARC-c	ARC-e	BoolQ	HellaSwag	OpenBookQA	PIQA	WinoGrande	Avg
OPT-125M	Dense	22.78	39.94	55.47	31.36	27.80	61.97	50.43	41.39
	FLAP	20.65	35.52	**61.74**	**30.13**	27.80	59.03	49.72	**40.66**
	SliceGPT	**22.44**	34.89	43.43	29.22	**27.80**	58.32	**52.01**	38.30
	SliceGPT-eq	21.5	29.97	39.51	27.73	25.60	53.54	50.04	35.41
	ASVD	21.76	35.56	49.63	29.60	26.80	56.91	49.33	38.51
	SoBP	21.93	**36.66**	55.41	30.35	26.60	**60.99**	51.54	40.50
OPT-1.3B	Dense	29.69	50.97	57.74	53.69	33.20	72.42	59.91	51.09
	FLAP	25.00	42.42	61.41	41.06	28.00	66.65	54.78	45.62
	SliceGPT	25.43	45.54	**62.08**	43.38	**32.20**	67.30	52.88	46.97
	SliceGPT-eq	23.21	41.08	60.80	36.96	30.00	62.35	51.22	43.66
	ASVD	24.74	28.28	58.01	26.70	25.40	51.09	52.25	38.07
	SoBP	**27.22**	**47.56**	61.41	**48.85**	32.00	**69.59**	**57.22**	**49.12**
OPT-2.7B	Dense	31.23	54.38	60.37	60.63	35.20	74.76	60.77	53.91
	FLAP	25.17	45.88	62.23	45.79	27.60	68.66	54.93	47.18
	SliceGPT	26.37	47.35	**62.26**	49.23	33.20	68.28	56.04	48.96
	SliceGPT-eq	24.23	42.68	61.74	42.16	28.80	64.80	53.83	45.46
	ASVD	26.02	26.77	61.87	26.19	27.60	49.73	50.43	38.37
	SoBP	**30.80**	**50.80**	56.09	**56.51**	**34.40**	**72.69**	**58.88**	**51.45**
OPT-6.7B	Dense	34.64	60.14	66.06	67.19	37.40	76.50	65.19	58.16
	FLAP	30.29	51.47	62.14	54.94	**37.40**	73.18	59.98	52.77
	SliceGPT	30.12	55.85	64.43	58.32	36.20	73.45	60.77	54.16
	SliceGPT-eq	28.33	51.47	63.61	53.81	34.00	71.06	60.22	51.79
	ASVD	25.26	28.07	55.84	26.75	25.00	52.72	51.38	37.86
	SoBP	**32.85**	**57.24**	**65.29**	**63.22**	36.60	**75.24**	**61.72**	**56.02**
OPT-13B	Dense	35.67	61.87	65.72	69.86	39.00	76.82	65.11	59.15
	FLAP	30.63	47.69	63.98	53.60	32.20	71.11	56.43	50.81
	SliceGPT	31.31	57.87	66.30	61.93	**36.60**	74.59	**62.83**	55.92
	SliceGPT-eq	29.69	54.12	64.37	58.81	35.80	73.07	61.96	53.97
	ASVD	24.83	25.80	53.85	25.99	26.60	50.71	50.20	36.85
	SoBP	**34.98**	**60.23**	**68.01**	**67.01**	36.20	**75.24**	61.33	**57.57**
OPT-30B	Dense	38.05	65.36	70.46	72.30	40.20	78.18	68.43	61.85
	FLAP	28.75	47.69	62.17	59.30	38.40	73.07	58.88	52.61
	SliceGPT	33.36	62.08	**71.87**	67.65	38.40	76.88	66.22	59.49
	SliceGPT-eq	33.28	60.77	70.86	65.51	36.80	76.28	64.96	58.35
	ASVD	24.23	32.74	60.64	36.68	25.20	57.45	50.91	41.12
	SoBP	**37.54**	**65.36**	68.44	**71.62**	**38.40**	**77.75**	**66.30**	**60.77**
OPT-66B	Dense	40.02	67.26	69.72	74.89	41.00	79.87	68.82	63.08
	FLAP	31.48	53.75	60.24	60.46	36.00	74.86	59.19	53.71
	SliceGPT	37.54	65.11	**72.75**	72.35	**40.80**	78.56	**68.59**	62.24
	SliceGPT-eq	37.63	65.40	72.72	70.92	39.60	77.69	67.96	61.70
	ASVD	25.09	29.46	40.00	26.94	27.40	52.12	50.28	35.90
	SoBP	**41.13**	**67.55**	69.97	**74.72**	40.60	**79.05**	68.35	**63.05**

Table 9: Detailed zero-shot tasks results of OPT models. Compression rate $r = 30\%$.

Methods	Model	GPUs	Batchsize	Time (s)
Dense	LLaMA2-13B	1	1	0.454
		1	2	0.896
		1	4	1.832
	LLaMA2-70B	2	1	1.913
		2	2	4.062
		2	3	6.072
	OPT-13B	1	1	0.414
		1	2	0.877
		1	4	1.754
	OPT-66B	2	1	1.942
		2	2	OOM
		2	3	OOM
SliceGPT-eq	LLaMA2-13B (-40%)	1	1	0.231
		1	2	0.524
		1	4	1.023
	LLaMA2-70B (-40%)	2	1	1.093
		2	2	2.009
		2	3	3.094
	OPT-13B (-30%)	1	1	0.369
		1	2	0.870
		1	4	1.480
	OPT-66B (-30%)	2	1	1.722
		2	2	3.490
		2	3	5.593
ASVD	LLaMA2-13B (-40%)	1	1	0.435
		1	2	0.890
		1	4	1.791
	LLaMA2-70B (-40%)	2	1	1.761
		2	2	3.473
		2	3	5.152
	OPT-13B (-30%)	1	1	0.425
		1	2	0.830
		1	4	1.644
	OPT-66B (-30%)	2	1	2.023
		2	2	OOM
		2	3	OOM
SVD-LLM	LLaMA2-13B (-40%)	1	1	0.605
		1	2	1.277
		1	4	2.499
	LLaMA2-70B (-40%)	2	1	2.661
		2	2	5.131
		2	3	7.974
	OPT-13B (-30%)	1	1	0.634
		1	2	1.270
		1	4	2.522
	OPT-66B (-30%)	2	1	2.262
		2	2	4.530
		2	3	6.993
SoBP	LLaMA2-13B (-40%)	1	1	0.384 (0.292)
		1	2	0.793 (0.586)
		1	4	1.552 (1.164)
	LLaMA2-70B (-40%)	2	1	1.603 (1.203)
		2	2	3.226 (2.391)
		2	3	4.718 (3.609)
	OPT-13B (-30%)	1	1	0.443 (0.283)
		1	2	0.922 (0.566)
		1	4	1.786 (1.158)
	OPT-66B (-30%)	2	1	1.620 (1.224)
		2	2	3.077 (2.528)
		2	3	4.551 (3.763)

Table 10: Inference time of different models. OOM denotes out of memory. The results in () of SoBP correspond to the results of SoBP(/8).

Methods	Model	GPUs	Batchsize	Token/s
Dense	LLaMA2-13B	1	256	2456
		1	512	2491
		1	768	OOM
	LLaMA2-70B	2	64	306
		2	128	396
		2	256	OOM
	OPT-13B	1	256	2176
		1	512	2612
		1	768	OOM
	OPT-66B	2	8	51
		2	16	76
		2	32	OOM
SliceGPT-eq	LLaMA2-13B (-40%)	1	256	2282
		1	512	2422
		1	768	OOM
	LLaMA2-70B (-40%)	2	128	423
		2	256	484
		2	512	OOM
	OPT-13B (-30%)	1	256	2664
		1	512	2733
		1	768	OOM
	OPT-66B (-30%)	2	32	109
		2	48	120
		2	64	OOM
ASVD	LLaMA2-13B (-40%)	1	256	2004
		1	512	2116
		1	768	OOM
	LLaMA2-70B (-40%)	2	32	206
		2	64	294
		2	128	OOM
	OPT-13B (-30%)	1	256	2458
		1	512	2460
		1	768	OOM
	OPT-66B (-30%)	2	32	84
		2	64	91
		2	96	OOM
SVD-LLM	LLaMA2-13B (-40%)	1	256	1740
		1	512	1973
		1	768	OOM
	LLaMA2-70B (-40%)	2	128	249
		2	256	282
		2	512	OOM
	OPT-13B (-30%)	1	256	1743
		1	512	1901
		1	768	OOM
	OPT-66B (-30%)	2	16	66
		2	32	89
		2	64	OOM
SoBP	LLaMA2-13B (-40%)	1	512	3167 (3782)
		1	768	3358 (4021)
		1	1024	OOM
	LLaMA2-70B (-40%)	2	256	530 (546)
		2	512	563 (574)
		2	1024	OOM
	OPT-13B (-30%)	1	512	3074 (4162)
		1	768	3121 (4219)
		1	1024	OOM
	OPT-66B (-30%)	2	32	97 (168)
		2	64	156 (210)
		2	128	OOM

Table 11: Throughput of different models. OOM denotes out of memory. The results in () of SoBP correspond to the results of SoBP(/8).