WRP: Weight Recover Prune for Structured Sparsity

Zhendong Tan, Xingjun Zhang, Zheng Wei
School of Computer Science and Technology, Xi'an Jiaotong University
`{772316639, frank.wei}@stu.xjtu.edu.cn, xjzhang@xjtu.edu.cn`

Abstract

As the scale of Large Language Models (LLMs) increases, it is necessary to compress the models to reduce the substantial demand on computational resources. Network pruning significantly reduces the model size by converting the weight matrix from dense to sparse data format. Current methodologies advocate for one-shot pruning to avoid the expense of retraining, ensuring the maintenance of model performance under conditions of 50%-60% unstructured pruning. Nevertheless, matrices characterized by this level of sparsity could not be treated as sparse matrices, because the indices would incur significant costs. To mitigate this problem, NVIDIA introduced the 2:4 structured sparsity. However, we observe a notable decline in model performance when adopting 2:4 structured sparsity due to group constraints. In this paper, we introduce the Weight Recover Prune (WRP) approach. By recovering a minimal set of critical weights, WRP aims to enhance model performance while maintaining the efficiency of the compression. Our evaluation of the WRP method on the LLAMA2 and OPT models shows that it outperforms other 2:4 pattern one-shot pruning methods. Meanwhile, WRP can guarantee that the size of the pruned model is about 60% of the dense model. Our code is available at: `https://github.com/TanZhendong/WRP`.

1 Introduction

Nowadays, many Large Language Models (LLMs) have been developed, based on the transformer architecture (Zhang et al., 2022; Touvron et al., 2023a; Achiam et al., 2023). These models have demonstrated astonishing capabilities across a variety of tasks. However, the deployment of LLMs, characterized by their billions of parameters, demands substantial hardware resources. For instance, the LLAMA2-70B model, with a size of 129GB, necessitates at least two A100-80GB GPUs for inference. To mitigate the extensive resource requirements for model deployment, pruning and quantization algorithms emerge as two prevalent strategies. Existing quantization algorithms could compress LLMs to 4 bits without retraining (Frantar et al., 2022; Lin et al., 2023; Dettmers et al., 2023), which could significantly reduce the size of the models.

Network pruning is a model compression approach orthogonal to quantization algorithms. Based on the granularity of the pruning algorithm, it is principally categorized into unstructured pruning and structured pruning. Unstructured pruning offers higher flexibility and typically results in less precision loss. It converts dense matrices into sparse matrices by setting certain values in the weight matrix to zero, thereby achieving model compression and acceleration. Considering the significant training overhead of LLMs, some pruning algorithms pursue one-shot pruning— that is, they avoid retraining to recover accuracy (Frantar and Alistarh, 2023; Sun et al., 2023). Such pruning methods have demonstrated minimal accuracy loss but generally could not achieve high levels of sparsity, with an optimal sparsity between 50%-60%. As for the compression of sparse matrices, taking the Compressed Sparse Row (CSR) data format as an example, as shown in Figure 1, a sparsity level of over 70% is usually required to realize compression benefits. In this context, matrices couldn't be treated as sparse for compression and computation if they do not meet this level of sparsity.

NVIDIA has introduced a solution known as structured sparsity, or 2:4 sparsity (Mishra et al., 2021). This pattern mandates that within every group of 4 values, at most 2 values can be retained. Firstly, this leads to a 50% degree of sparsity, which is beneficial for the performance of models following one-shot pruning. Moreover, by compressing the indices, it ensures efficient compression under 50% sparsity. Additionally, this approach could achieve a 2x math throughput increase on the NVIDIA Ampere GPU architecture. As a result, this pattern and compressed format are more hardware-friendly than unstructured sparsity at 50%. However, due to its grouping constraints, there is a notable decline in accuracy compared to unstructured pruning at 50%. For instance, when applying Wanda pruning (Sun et al., 2023) to the LLAMA-7B model, the resulting perplexity on the WikiText dataset under unstructured 50% and 2:4 pattern conditions are 7.26 and 11.53, respectively. This indicates that there is still room for improvement in performance of 2:4 pattern.

In this work, we observe a performance gap between the 2:4 pattern and unstructured 50% pruning. Consequently, our primary objective is to enhance the model performance of the one-shot 2:4 pattern pruning, while ensuring the model compression is achieved. In section 3.1, we observed that some crucial weights might be incorrectly pruned in the 2:4 pattern pruning. Based on this phenomenon, we propose the Weight Recover Prune (WRP) approach, which aims to improve model accuracy by restoring a minor portion of critical weights, while for the majority of the matrix is still adopting a 2:4 pattern. To safeguard the compression efficacy, we partition the weight matrix into two separately stored entities: one is the 2:4 pattern and the other is high sparsity matrix, as shown in Figure 1. This approach allows us to achieve a balance between model size and performance, addressing the challenge inherent in the structured 2:4 sparsity pattern.

The main contributions of this work are:

- We explore the differences in mask selection between 2:4 pruning and unstructured 50% pruning. The results indicate that 2:4 pruning might incorrectly prune a small number of values with larger metrics.

- We propose the Weight Recover Prune (WRP) approach, which enhances the model performance after 2:4 pruning by recovering the crucial weights.

- We evaluate our approach on the LLAMA2 and OPT models. The results indicate that our approach can recover the model performance while ensuring the model compression effect.

2 Related Work

Network Pruning. Network pruning is a commonly used method for model compression. It typically results in a loss of model accuracy, necessitating the adoption of various techniques for its restoration. Training is a common method for recovering accuracy. Based on the relationship between training and pruning, this process could be categorized into three distinct approaches: pruning before (re)training, pruning during training, and pruning without retraining.

Han et al. (2015) introduced Deep Compression, which designed a three-stage pipeline: pruning, trained quantization, and Huffman coding. This approach is considered a milestone in the field of model compression. Additionally, the Lottery Ticket Hypothesis shows that pruning could be conducted at the network initialization phase (Frankle and Carbin, 2018; Wang et al., 2020). Pruning during training typically needs to design a weight importance estimation to accurately remove non-essential weights during the training process (Molchanov et al., 2019; Evci et al., 2020). Finally, the second-order information is commonly used for restoring accuracy without retraining (LeCun et al., 1989; Hassibi et al., 1993; Frantar and Alistarh, 2022).

Structured 2:4 Sparsity. NVIDIA proposed the 2:4 sparsity pattern, which typically requires retraining to recover model accuracy (Mishra et al., 2021). Channel permutations could be utilized to enhance model accuracy and alleviate the limitations of group constraints (Pool and Yu, 2021). For the 2:4 sparse matrix format, cuSparseLt provides the corresponding operators (NVIDIA, 2020). PyTorch has also implemented support for 2:4 Sparsity (Cai, 2023).

LLMs Pruning. Due to the substantial resources required for training LLMs, LLMs pruning aims to restore accuracy with minimal overhead, primarily through fine-tuning or one-shot pruning.

LLM-Pruner (Ma et al., 2023) implements structured pruning by identifying dependencies and removing some of them. LoRAPrune (Zhang et al., 2023) designs a LoRA-guided pruning criterion, integrating LLM pruning with LoRA (Hu et al., 2021). These approaches typically rely on fine-tuning to achieve improved accuracy, which might necessitate high-quality fine-tuning datasets and additional computational resources.

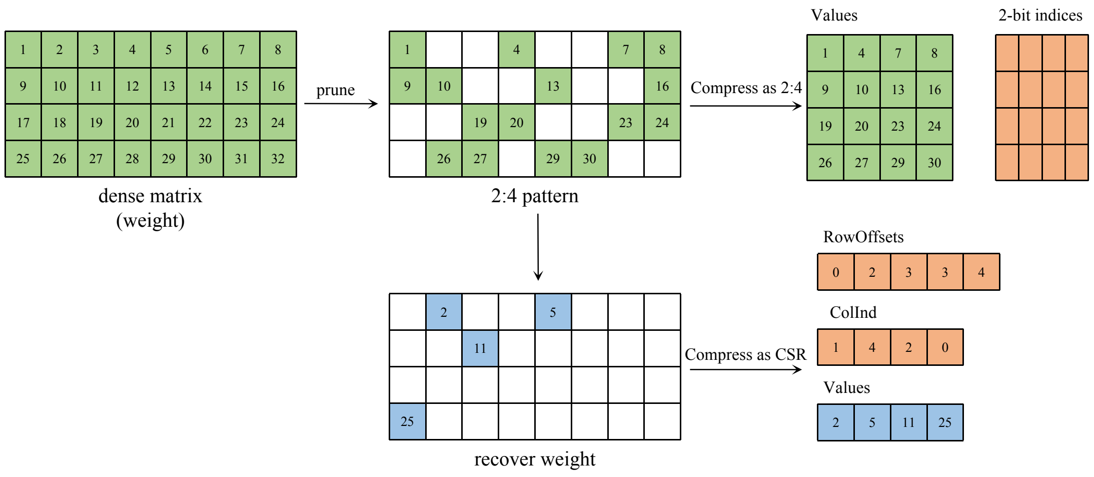

Figure 1: Illustration of the Weight Recover Prune (WRP) approach: we respectively store the sparse matrix in a 2:4 pattern and the recover weights. Indices storage is the primary additional overhead.

SparseGPT (Frantar and Alistarh, 2023) is an approach based on second-order information, enhancing accuracy through weight reconstruction. Dettmers et al. (2022, 2023) demonstrates the presence of outliers in LLMs during model quantization. Building on this observation, Sun et al. (2023) introduced the Wanda metric, which not only performs superiorly on LLMs but also achieves faster pruning speeds. These methods typically exhibit great performance in unstructured pruning; however, they fall short of achieving higher levels of sparsity, such as 70%. When adopting a 2:4 pattern, their accuracy suffers due to group constraints. Inspired by these challenges, we focus on improving the accuracy of the 2:4 pattern with minimal overhead.

3 Weight Recover Prune

3.1 2:4 Pattern vs. Unstructured 50%

In Section 1, we have discussed the benefits and drawbacks of the 2:4 pattern compared to an unstructured 50% approach. It is obvious that the 2:4 pattern is more practical than the unstructured 50% pruning. In this part, we will focus on: what distinguishes the choice of pruning masks between the 2:4 pattern and the unstructured 50% pruning when using the same metric?

First of all, we must clarify that in implementing unstructured 50% pruning, we typically do not prune the 50% of weights with the lower metrics across the *entire* weight matrix. Instead, the approach targets *each row* individually. This means sorting the weights within each row of the weight matrix and pruning the 50% with lower metrics.

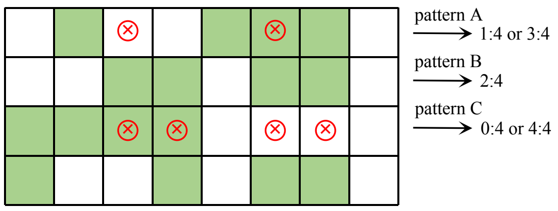

Figure 2: Three patterns of X:4. ⊗ represents different weight choices of 50% unstructured and a 2:4 pattern.

To understand different mask choice between the 2:4 pattern and unstructured 50% pruning, we divide unstructured pruning matrix into three patterns, as shown in Figure 2. Figure 2 depicts a weight matrix that has been unstructured 50% pruned. We consider every four elements as a group, resulting in five possible scenarios: X:4, where X denotes the quantity of remaining elements ranging from 0 to 4. Regarding pattern A (1:4 or 3:4), when applying 2:4 pattern pruning, exactly one weight is *incorrectly* removed. Taking the 3:4 case of pattern A as an example, let's assume these four values are a_1, a_2, a_3, and a_4. If during the unstructured 50% pruning, a_2 is pruned, then the following is observed:

$$a_2 \in S_{m \leq 50\%} \leq a_1, a_3, a_4 \in S_{m \geq 50\%}$$

where m represents the metric used for pruning. Consequently, when applying the 2:4 pattern, a_2 would definitely be pruned again, and additionally, the smallest among a_1, a_3, and a_4 would also be incorrectly pruned. Similarly, for pattern B, there would not be any weights incorrectly pruned. For pattern C, there would be two weights incorrectly pruned.

layer	0:4	1:4	2:4	3:4	4:4
0.q	6.16	25.05	37.66	24.92	6.22
0.k	6.16	25.09	37.59	24.94	6.23
0.v	6.18	25.04	37.59	24.97	6.22
16.q	6.25	25.00	37.52	24.98	6.26
16.k	6.23	25.01	37.53	25.01	6.23
16.v	6.24	24.99	37.53	25.00	6.24
30.q	6.25	24.99	37.52	24.99	6.25
30.k	6.23	25.01	37.53	24.98	6.24
30.v	6.24	24.99	37.54	25.00	6.24

Table 1: LLAMA2-7B proportions of X:4 with Wanda pruning (%)

To analyze the proportions of three patterns within unstructured pruning, we use the Wanda metric (Sun et al., 2023) to prune the LLAMA2-7B model. The results are summarized in Table 1. In total, approximately 40% of the groups conform to a 2:4 pruning pattern, around 25% of the groups probably would prune one crucial weight element, and roughly 6.25% of groups might prune two crucial weight elements. This results in suboptimal accuracy. Consequently, a natural thought arises: *we could recover those values that were potentially incorrectly pruned in 2:4 pattern to enhance the model performance.*

3.2 Determining the Weights for Recover

In this section, we focus on the process of recovering those values that were potentially incorrectly pruned in a 2:4 pattern. To address this issue, we need to determine the following two aspects:

1. How can we identify the weights that need to be recovered?

2. How many weights need to be recovered to enhance the model performance?

The primary pruning metrics contain three main types: magnitude, second-order information (Hassibi et al., 1993), and Wanda. Within the framework of a one-shot pruning, we typically calculate metrics for each element first. Then, prune masks are selected based on the magnitude of these metrics. Generally, the larger the metric, the more important that weight element is considered. Similarly, we believe that these metrics could effectively reflect the impact of the elements on model accuracy. Therefore, we introduce a ratio factor α, indicating that the elements with the highest α metrics are referred to as *crucial weights*, which should not be pruned in 2:4 pattern.

In this case, the elements that need to be recovered are those crucial weights that were pruned by the 2:4 pattern. We formalize the recover elements as follows:

$$W_{recover} = W_\alpha \cap \overline{W_{2:4}}$$

Furthermore, the linear layer in the model could be modified as:

$$xW^T + b = xW_{2:4}^T + xW_{recover}^T + b$$

By adjusting the value of α, we could control the proportion of elements to recover. Typically, an increase in the value of α would make more elements recovered. This would reduce the sparsity of $W_{recover}$, but improve more in model accuracy. We provide the pseudo code of our approach in algorithm 1.

Algorithm 1 Weight Recover Prune

```
Ensure: $W_{2:4}$, $W_{recover}$
Require: $W$, $M(metrics), \alpha$
1: $mask_{2:4}$=prune(W, M, "2:4")
2: $mask_\alpha$ = topk(W, M, $\alpha$)
3: $mask_{recover} = mask_\alpha \cap \overline{mask_{2:4}}$
4: $W_{2:4}$ = to_sparse_semi_structured($W[mask_{2:4}]$)
5: $W_{recover}$ = to_sparse_csr($W[mask_{recover}]$)
```

3.3 Recover with Block

The sparse format of unstructured matrix commonly poses challenges in exploiting Tensor Cores, resulting in suboptimal performance during large-scale Sparse Matrix Multiplication (SpMM), even with a high degree of sparsity. A potential solution to this issue is the adoption of the Blocked-ELL sparse matrix format (NVIDIA, 2022; Yamaguchi and Busato, 2021). As a result, we extend the WRP to accommodate block sparse formats, thereby mitigating the impact on performance.

Unlike Section 3.2, we couldn't directly compute the recovered weights for each element. Instead, the weight matrix must be processed in blocks. Accordingly, we introduce two additional hyperparameters: b and k. Here, b represents the block size for matrix, and k denotes the number of blocks to be recovered in each row. Following the application of a 2:4 pattern pruning, we calculate the sum of the metrics for the pruned weights within each block, serving as the metric for the entire block. This is expressed as follows:

$$M_{block} = \sum_{i=i_0}^{i_0+b} \sum_{j=j_0}^{j_0+b} m_{ij}, \ (mask_{ij} \neq 1)$$

In computation, the metric of the retained weights within the 2:4 pattern should be set to zero. Then, for each block, calculate the sum of the metrics. We recover the k blocks with the highest metrics in each row after tiled. Parameters b and k jointly control the sparsity of the recovery matrix, with its sparsity increasing as values of b increase and k decrease. To achieve better performance, b is typically chosen as a power of 2. Overall, the method of block recovery could achieve better computational efficiency, while the recovery of weights in an unstructured manner showcases the best trade-off between compression effects and improvement in model performance. We provide the pseudocode for block recovery in Algorithm 2.

Algorithm 2 Weight Recover Prune for Block

```
Ensure: $W_{2:4}$, $W_{recover}$
Require: $W$, $M(metrics), b(blocksize), k$
1: $mask_{2:4}$=prune(W, M, "2:4")
2: $M[mask_{2:4}] = 0$
3: $M\_block$ = block_sum($M, b$)
4: $mask_{recover}$ = topn($M\_block, k$)
5: $W_{2:4}$ = to_sparse_semi_structured($W[mask_{2:4}]$)
6: $W_{recover}$ = to_blocked_ELL($W[mask_{recover}]$)
```

4 Experiment

4.1 Setup

Models. We evaluate our approach using the LLAMA2 (Touvron et al., 2023b) and OPT (Zhang et al., 2022) model families. LLAMA2 is a suite of pre-trained and fine-tuned generative text models, comprising models of 7 B, 13 B, and 70 B parameters, respectively. We apply the WRP to each of these configurations. In contrast, OPT is GPT-3 style, with a more classical Transformer decoder-only architecture. Compared to LLAMA2, it offers more selection of model sizes, thereby facilitating our exploration into the scaling trends of LLMs.

Evaluation. We evaluate the performance of the model on language capabilities: perplexity, a metric also utilized by prior works (Frantar et al., 2022; Frantar and Alistarh, 2023; Sun et al., 2023). We conducted tests on the perplexity metric using WikiText (Merity et al., 2016) and the c4 dataset (Raffel et al., 2019). To enhance the evaluation of LLMs, we use the Language Model Evaluation Harness (Gao et al., 2023) to assess performance on 5 zero-shot tasks of models: HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2019), OpenBookQA (Mihaylov et al., 2018), RTE (Wang et al., 2018).

Baselines. We compare our method WRP against two prior pruning methods which could readily adopt a 2:4 pattern:

- Wanda (Sun et al., 2023) is a pruning metric that is simple and effective on LLMs. Furthermore, we use Wanda as the 2:4 pattern pruning for WRP, which implies that Wanda 2:4 could be considered as a scenario of WRP without recovering.

- SparseGPT (Frantar and Alistarh, 2023) is a pruning method based on second-order information and uses weight reconstruction to restore model performance. Through comparison, we aim to verify the effectiveness of crucial weights in recovering model accuracy.

Both approaches adopt a 2:4 pattern. Regarding calibration data, as recommended by SpQR (Dettmers et al., 2023), we utilize the RedPajama dataset (Computer, 2023) for LLAMA2 and the c4 dataset for OPT. The length of the samples is uniformly set at 128.

Pruning what? Following the approaches of SparseGPT and Wanda, we skip pruning the embedding layer and the final classification head layer. For the remaining layers in the Transformers architecture, which are all Linear layers, we uniformly apply the 2:4 pattern pruning to all weight matrices.

4.2 Model Perplexity

Recover effect. We use SparseGPT, Wanda and WRP to prune the LLAMA2 model family separately. We select the recover ratio α of 0.25. And the average density of recover matrix is approximately 1.5%. The perplexity results are summarized in Table 2.

Compared to weight reconstruction, WRP demonstrate a more effective capability in recovering the perplexity of the LLAMA2 model family during 2:4 pattern. Due to the group constraints,

Size	Method	PPL(wikitext2)	PPL(c4)
7B	Dense	5.47	6.97
	SparseGPT	10.58	13.32
	Wanda	11.97	14.22
	WRP	**8.69**	**10.58**
13B	Dense	4.88	6.47
	SparseGPT	8.54	11.29
	Wanda	8.87	11.28
	WRP	**7.01**	**9.02**
70B	Dense	3.32	5.52
	SparseGPT	5.63	8.15
	Wanda	5.47	7.50
	WRP	**4.78**	**6.74**

Table 2: WRP ($\alpha = 0.25$) can recover the model perplexity of 2:4 pattern pruning on LLAMA2-family.

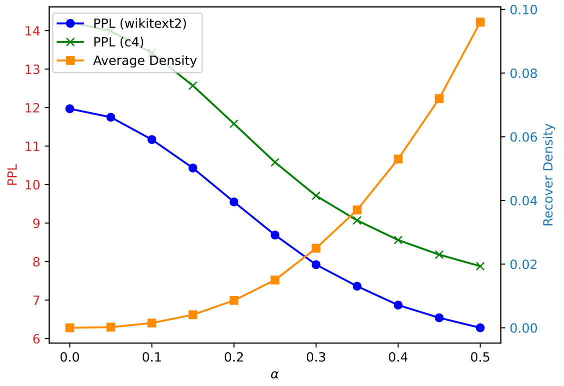

Figure 3: Exploring the impact of factor α on model perplexity and the average density of recover weight in LLAMA2-7B.

Wanda's performance demonstrate a significant decline in the 2:4 pattern compared to the unstructured 50% pruning. Considering that WRP utilize the Wanda metric, this result confirm that Wanda could identify the crucial weights efficiently. Furthermore, it is observed that with only a minimal set of these crucial weights (approximately 1.5%), a significant reduction in model perplexity could be achieved.

The impact of α. To explore the influence of the recover ratio α, we execute WRP on LLAMA2-7B model with varying α values. The results are shown in Figure 3. As α increases, a notable decrease in the model perplexity is observed, alongside a gradual increase in the average sparsity of the recover matrix. Overall, with an additional weight of less than 2%, the model's perplexity on the Wikitext2 dataset decreased from 11.97 to 8.69. We recommend selecting an α value between 0.2 and 0.3 to achieve an optimal trade-off between model perplexity and size. Furthermore, we provide results of block recovery, with details presented in Appendix B.

4.3 Zero-shot Tasks

We evalute pruned LLAMA2 and OPT models on 5 zero-shot tasks. Resuls are shown in Table 3 and Appendix A respectively. For HellaSwag, PIQA, and OpenBookQA tasks, we present the normalized accuracy.

For the OPT models, we choose more model sizes, from 2.7B to 30B parameters. As the scale increases, we observe that the accuracy on some datasets does not improve and even declines. For instance, the accuracy of OPT-30B on the RTE dataset is 57.8%, whereas for OPT-13B, it is 58.1%. We speculate that this instability in performance across different model scales might be attributable to factors inherent to the models or the evaluation tasks themselves. WRP is capable of outperforming Wanda for most tasks, while SparseGPT achieves better results in certain cases such as OPT-13B. However, for the LLAMA2 model, an increase in scale consistently leads to improvements in accuracy across all tasks. WRP significantly enhances the 2:4 pattern accuracy of Wanda and also surpasses SparseGPT in most tasks.

4.4 Model Size

To explore the efficacy of WRP on model compression, we prune the OPT model and compress the 2:4 pattern and recover weight matrix with the data format illustrated in Figure 1. Additionally, we directly use Wanda to perform a 2:4 pattern pruning to verify the additional overhead associated with the recover matrix. The results are presented in Table 4.

We select different recover ratios α, because the value of α influences the size of the model. We use 32 bits for storing CSR indices, which is the main additional overhead of a sparse matrix. Given that the actual proportion of recover crucial weights varies across different layers, the compression ratio of the model exhibits some degree of fluctuation. Overall, the model size after WRP compression is approximately 62% of the dense model, while for the 2:4 pattern compression is about 58%. Consequently, WRP demonstrates little addtional overhead compared to 2:4 pattern. Furthermore, we

Size	Method	HellaSwag	PIQA	WinoGrande	OpenBookQA	RTE	Average
7B	Dense	75.9	79.0	69.1	44.0	63.2	66.2
	SparseGPT(50%)	70.0	77.3	69.7	42.4	53.4	62.6
	Wanda(50%)	70.7	76.7	67.0	42.0	54.2	62.1
	SparseGPT(2:4)	56.7	70.8	64.5	35.4	**55.2**	56.5
	Wanda(2:4)	54.5	70.9	61.9	37.4	53.4	55.6
	WRP(α=0.25)	**63.1**	**74.2**	**66.1**	**38.8**	54.2	**59.3**
13B	Dense	79.4	80.6	72.2	45.4	65.0	68.5
	SparseGPT(50%)	74.4	78.1	72.5	43.2	62.8	66.2
	Wanda(50%)	76.1	78.7	70.9	44.4	60.6	66.1
	SparseGPT(2:4)	62.7	73.8	**69.6**	36.6	58.8	60.3
	Wanda(2:4)	62.1	74.0	65.7	35.6	57.0	58.9
	WRP(α=0.25)	**69.9**	**77.3**	69.1	**41.2**	**61.0**	**63.7**

Table 3: Accuracies (%) for 5 zero-shot tasks with 2:4 pattern on LLAMA2-family. For HellaSwag, PIQA, and OpenBookQA tasks, we present the normalized accuracy (acc_norm). (50%) means unstructured 50% pruned model.

Size	Dense	2:4	WRP(α=0.20)	WRP(α=0.25)	WRP(α=0.30)	WRP(α=0.35)
1.3B	2.5GB	1.5GB	1.6GB	1.6GB	1.7GB	1.8GB
2.7B	5.0GB	2.9GB	3.1GB	3.2GB	3.3GB	3.5GB
6.7B	13GB	7.2GB	7.5GB	7.7GB	8.1GB	8.5GB
13B	24GB	14GB	15GB	15GB	16GB	17GB
30B	56GB	32GB	34GB	35GB	36GB	38GB

Table 4: The compressed model size for the OPT-family with different α.

provide the model size results of Blocked-ELL data format in Appendix C.

4.5 Inference Kernel

PyTorch has supported the 2:4 pattern SpMM using either CUTLASS or cuSparseLt libraries. Consequently, we directly use PyTorch to evaluate the 2:4 pattern latency. Considering that PyTorch does not support the Blocked-ELL data format, we implement a SpMM kernel for the Blocked-ELL format as a PyTorch Extension using cuSparse. The performance of sparse matrix computations is generally influenced by the matrix sparsity, computing hardware, and the scale of the problem. We assume a density level of 6.25% for Blocked-ELL, which is enough to offer a balance between recovering model performance and achieving acceleration. Furthermore, we set the batch size to 1 and the sequence length to 2K, controlling problem scales through hidden states. These tests were conducted on both A100 and RTX 4090 GPUs, with the results detailed in Table 5.

For A100 GPUs, we observe that acceleration is not guaranteed and depend on the scale of the problem. Specifically, for the acceleration using a 2:4 pattern on A100 GPUs, we were able to achieve a speedup of approximately 1.3×, which aligns with (Cai, 2023). Compared to the theoretical maximum of a 2× increase in mathematical throughput, there is still room for improvement. Typically, significant acceleration is observed when dealing with larger matrices. For the RTX 4090 GPUs, a more pronounced acceleration effect could be achieved, with speedup ratios generally exceeding 1.1×. We speculate that different GPU architectures might result in different capabilities to process sparse and dense matrices. As a result, the actual acceleration achieved is dependent on the specific application context.

5 Conclusions

In this work, we propose the Weight Recover Prune (WRP) methodology for achieving structured sparsity in LLMs. Observing the notable performance gap between the 2:4 pruning pattern and the unstructured 50% pruning, our WRP technique enhances the model performance associated with the 2:4 pattern by recovering a minimal set of crucial weights, thereby ensuring the efficiency of model

Device	Hidden State	2:4(ms)	Blocked-ELL(ms)	Dense(ms)	Speedup
A100	4096	0.274	0.064	0.324	0.95×
A100	7168	0.733	0.147	0.971	1.1×
RTX 4090	4096	0.276	0.081	0.412	1.15×
RTX 4090	7168	0.718	0.230	1.377	1.45×

Table 5: Kernel test of WRP, including 2:4 and blocked-ELL matmul. The density of blocked-ELL is 6.25%.

compression. With the recovery of approximately 1.5% of these crucial weights, the WRP approach can significantly improve the perplexity of models, while the compressed models are approximately 60% of their original size. We hope that our work could contribute to the semi-structured pruning for LLMs.

6 Limitations

WRP achieves a good trade-off between model performance and compression effectiveness. However, due to the fact that sparse matrices in CSR format typically couldn't utilize NVIDIA Tensor Cores for acceleration, our recover matrix is unable to achieve enhanced inference speed even if very high sparisty. If, in the future, it becomes feasible to implement SpMM kernels for high-sparsity unstructured sparse matrices, we believe WRP might offer a certain level of speedup.

Blocked-ELL could be a potential solution for acceleration. Therefore, we extend our method to block recovery. However, we discover that, although block recovery offers some restoration of model accuracy, it results in a notable decline in performance compared to the effects of unstructured recover weights format. As a result, we believe that block recovery does not achieve the optimal trade-off between accuracy and compression. Overall, in pursuit of hardware acceleration, we introduce additional constraints, which adversely affect the model's performance.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant 62372366. We thank the ACL anonymous reviewers for helpful suggestions.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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

Jesse Cai. 2023. (prototype) accelerating bert with semi-structured (2:4) sparsity. https://pytorch.org/tutorials/prototype/semi_structured_sparse.html.

Together Computer. 2023. Redpajama: an open dataset for training large language models.

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*.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

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. 2022. Optq: Accurate quantization for generative pre-trained transformers. In *The Eleventh 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. 2023. A framework for few-shot language model evaluation.

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.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

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

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*.

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

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

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 *EMNLP*.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

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.

NVIDIA. 2020. cusparselt: A high-performance cuda library for sparse matrix-matrix multiplication. `https://docs.nvidia.com/cuda/cusparselt/index.html`.

NVIDIA. 2022. cusparse. `https://docs.nvidia.com/cuda/cusparse/index.html`.

Jeff Pool and Chong Yu. 2021. Channel permutations for n: M sparsity. *Advances in neural information processing systems*, 34:13316–13327.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

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. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

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

Takuma Yamaguchi and Federico Busato. 2021. Accelerating matrix multiplication with block sparse format and nvidia tensor cores. `https://developer.nvidia.com/blog/accelerating-matrix-multiplication-with-block-sparse-format-and-nvidia-tensor-cores/`.

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*.

Mingyang Zhang, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, Bohan Zhuang, et al. 2023. Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*.

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

A OPT Zero-shot Tasks

We evaluate 5 zero-shot tasks on OPT models, with scales ranging from 2.7B to 30B parameters. The results, as shown in Table 8, indicate that WRP outperforms Wanda across nearly all tasks, and surpasses SparseGPT at the 2.7B and 30B levels. For models scale at 6.7B and 13B parameters, WRP shows a slight decrease in performance compared to SparseGPT.

B Perplexity of Block Recover

We evaluate the block recovery on the LLAMA2-7B and 13B models across various block sizes and numbers of columns. The results are illustrated in Tables 6 and 7. Generally, a smaller block size and a larger number of columns are associated with enhanced model performance. Moreover, under equivalent sparsity levels, an unstructured recovery matrix format demonstrates superior performance compared to block recovery.

Block	Column	PPL(wikitext2)	PPL(c4)
64	4	10.72	12.84
64	8	9.86	11.86
32	8	10.66	12.80
32	16	9.74	11.78
16	16	10.60	12.7
16	32	9.57	11.58

Table 6: Perplexity of Block recover on LLAMA2-7B

Block	Column	PPL(wikitext2)	PPL(c4)
64	5	8.20	10.49
64	10	7.62	9.81
32	10	8.18	10.46
32	20	7.57	9.74
16	20	8.05	10.33
16	40	7.48	9.59

Table 7: Perplexity of Block recover on LLAMA2-13B

C Model Size of Block Recover

We evaluate the compression efficacy on OPT models. The results are presented in Table 9. Despite the Blocked-ELL data format typically storing more non-zero values, its requirement for fewer indices results in great compression performance.

D Compatible with 4:8 Sparsity

Wanda and SparseGPT also provide the results of 4:8 sparsity. They typically adjust the group size to 8, that is, retaining 4 out of every 8 elements. Although such 4:8 sparsity cannot be compressed and accelerated like the 2:4 pattern, considering their compatibility, we also implement 4:8 sparsity. Results are show in Table 10.

One thing can be certain is that when adopting 4:8 sparsity, the constraints of the group will be weakened, hence the gains from WRP will decrease than 2:4. It is foreseeable that as the group size increases, such as to 16, 32, 128, the effectiveness of the pruning algorithm will increasingly approach the unstructured 50%.

Size	Method	HellaSwag	PIQA	WinoGrande	OpenBookQA	RTE	Average
2.7B	Dense	60.6	74.8	61.0	35.2	55.2	57.4
	SparseGPT	49.2	70.5	58.1	31.6	51.6	52.2
	Wanda	45.7	68.9	55.6	**32.4**	52.7	51.1
	WRP	**49.7**	**70.8**	**59.0**	32.0	**54.2**	**53.1**
6.7B	Dense	67.2	76.6	65.4	37.4	55.2	60.4
	SparseGPT	57.0	**73.5**	61.8	**36.6**	**54.2**	**56.6**
	Wanda	54.2	71.8	58.8	34.4	52.3	54.3
	WRP	**58.1**	**73.5**	**62.0**	35.4	52.7	56.3
13B	Dense	72.3	76.8	65.0	39.0	58.1	62.2
	SparseGPT	59.5	**73.8**	62.5	**37.2**	53.8	**57.4**
	Wanda	58.0	72.4	61.6	33.2	53.8	55.8
	WRP	**60.7**	73.7	**62.8**	34.4	**54.5**	57.2
30B	Dense	72.3	78.1	68.2	40.4	57.8	63.4
	SparseGPT	64.8	**77.1**	**65.3**	36.8	54.2	59.6
	Wanda	63.4	75.5	63.5	36.2	54.9	58.7
	WRP	**66.7**	76.2	**65.3**	**37.6**	**56.3**	**60.4**

Table 8: Accuracies (%) for 5 zero-shot tasks with 2:4 pattern on OPT-family. For HellaSwag, PIQA, and OpenBookQA tasks, we present the normalized accuracy (acc_norm).

Average Density	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B
6.25%	1.6GB	3.2GB	7.8GB	15GB	35GB
12.5%	1.7GB	3.4GB	8.3GB	16GB	37GB

Table 9: OPT model size for block recovery, where block size = 32.

Models	Method	PPL(Wikitext2)	PPL(c4)	Recover Density
LLAMA2-7B	SparseGPT	8.32	10.70	
	Wanda	8.58	10.40	
	WRP(α=0.25)	8.03	9.78	0.56%
	WRP(α=0.35)	7.32	9.01	1.92%
LLAMA2-13B	SparseGPT	6.87	9.32	
	Wanda	6.95	8.91	
	WRP(α=0.25)	6.61	8.50	0.59%
	WRP(α=0.35)	6.20	7.99	1.93%
LLAMA2-70B	SparseGPT	4.87	7.20	
	Wanda	4.77	6.73	
	WRP(α=0.25)	4.59	6.56	0.78%
	WRP(α=0.35)	4.36	6.33	1.98%

Table 10: WRP can also recover the model perplexity of 4:8 pattern pruning on LLAMA2-family.