# WANDA++: PRUNING LARGE LANGUAGE MODELS VIA REGIONAL GRADIENTS

**Yifan Yang**[◇,†,*,‡] **Kai Zhen**[♣,†,‡] **Bhavana Ganesh**[♣] **Aram Galstyan**[♣] **Goeric Huybrechts**[♣]
**Markus Müller**[♣] **Jonas M. Kübler**[♣] **Rupak Vignesh Swaminathan**[♣] **Athanasios Mouchtaris**[♣]
**Sravan Babu Bodapati**[♣] **Nathan Susanj**[♣] **Zheng Zhang**[◇] **Jack FitzGerald**[♣] **Abhishek Kumar**[♣]
[◇] University of California, Santa Barbara [♣] Amazon AGI
[†]Equal contributions [*] Work done at Amazon
[‡] Corresponding authors: yifanyang@cs.ucsb.edu, kaizhen@amazon.com

## ABSTRACT

Large Language Models (LLMs) pruning seeks to remove unimportant weights for inference speedup with minimal performance impact. However, existing methods often suffer from performance loss without full-model sparsity-aware fine-tuning. This paper presents Wanda++, a novel pruning framework that outperforms the state-of-the-art methods by utilizing decoder-block-level **regional** gradients. Specifically, Wanda++ improves the pruning score with regional gradients for the first time and proposes an efficient regional optimization method to minimize pruning-induced output discrepancies between the dense and sparse decoder output. Notably, Wanda++ improves perplexity by up to 32% over Wanda in the language modeling task and generalizes effectively to downstream tasks. Further experiments indicate our proposed method is orthogonal to sparsity-aware fine-tuning, where Wanda++ can be combined with LoRA fine-tuning to achieve a similar perplexity improvement as the Wanda method. The proposed method is lightweight, pruning a 7B LLaMA model in under 10 minutes on a single NVIDIA H100 GPU.

## 1 INTRODUCTION

The growing size of Large Language Models (LLMs) improves performance (Devlin et al., 2018; Touvron et al., 2023) at the cost of memory consumption and inference latency. For example, hosting an LLaMA-2-70B model needs at least four A100-40GB GPUs with the time to first token (TTFT) exceeding 100 milliseconds (Agarwal, 2023). To address these challenges, various model compression approaches, including weight decomposition (Hsu et al., 2022; Yang et al., 2024), quantization (Lin et al., 2024; Tian et al., 2023), and pruning (Sun et al., 2023; Xu et al., 2024), have been explored. Among pruning methods, post-training LLM pruning approaches, such as SparseGPT (Sun et al., 2023) and Wanda (Sun et al., 2023), have gained attention as they circumvent the prohibitive memory overhead associated with traditional in-training pruning techniques (Han et al., 2015; Frankle & Carbin, 2018). However, post-training pruning often leads to substantial performance degradation, limiting its effectiveness for efficient LLM deployment.

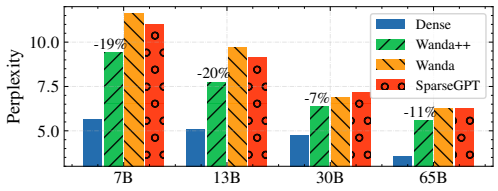


Figure 1: Wanda++ mitigates 2:4 pruning-induced degradation more effectively, with relative perplexity improvement over Wanda shown on Wikitext using LLaMA-1 models across four different sizes.

To mitigate the performance degradation, GBLM and Pruner-Zero (Das et al., 2023; Dong et al., 2024) propose improved pruning criteria that enhance the layer-wise Wanda score by incorporating gradient information obtained through full-model backpropagation. Meanwhile, other approaches focus on recovering model performance through sparsity-aware tuning (Sun et al., 2023) or distillation (Liang et al., 2023). Although these methods effectively reduce pruning-induced degradation, they suffer from impractical memory requirements and excessive pruning time due to the high computational cost of full-model backpropagation. This raises an important question:

*Is there a way to effectively involve gradient information while still in a lightweight manner?*

In this paper, we propose Wanda++ pruning framework to leverage gradients at the "decoder-block" level, termed regional gradients, which shows significant improvement compared to Wanda Sun et al. (2023) without largely sacrificing the pruning efficiency. Compared to the Wanda method, the proposed approach efficiently incorporates crucial gradient information during pruning, which has been shown to provide non-trivial insights in prior works such as Optimal Brain Surgeon (Hassibi et al., 1993) and GBLM (Das et al., 2023). Notably, the regional gradient can be computed by loading and processing backpropagation for only a single decoder block at a time, making gradient calculation feasible regardless of the LLM's size.

Our proposed pruning framework consists of two components that leverage regional gradients: Regional Gradient Score (RGS) and Regional Optimizer (RO). The RGS demonstrates the effectiveness of replacing the full-model gradient used in the GBLM method with a regional gradient, computed via backpropagation on a loss function defined as the $\ell_2$ norm of the decoder output. Meanwhile, the RO method performs lightweight, decoder-block-level weight updates after each pruning step, mitigating pruning-induced loss by adjusting weights based on a small calibration dataset. This is achieved by considering the difference between the outputs of dense and pruned decoder blocks, ensuring efficient local recovery. Our contributions can be summarized as follows:

- We propose Wanda++, a lightweight yet effective framework that prunes LLMs using regional gradients, achieving significant performance improvements without requiring full-model backward.
- As shown in Fig. 1, Wanda++ effectively mitigates pruning-induced degradation in a non-incremental manner and generalizes well to zero-shot downstream tasks.
- The proposed method is orthogonal to previous sparsity-aware fine-tuning approaches and achieves a similar perplexity improvement as Wanda when combined with LoRA fine-tuning.

## 2 THE WANDA++ FRAMEWORK

In this section, we introduce Wanda++, a framework designed to efficiently reduce pruning-induced degradation through a two-stage process. The algorithm flow is summarized in Algorithm 1, which iteratively performs pruning based on the regional pruning score followed by rapid weight updates using the RO process. Finally, a last pruning step is performed to obtain the pruned weight for each decoder blocks.

### 2.1 REGIONAL GRADIENT SCORE

As the first stage of Wanda++, we obtain the Regional Gradient Score (RGS) for in-block pruning. We start with constructing an RGS loss function for obtaining the gradient of each weight matrix. Given a model with $L$ decoder blocks, we represent the set of input hidden states for the $l$-th decoder block specifically as $\mathcal{X}^l = \{\boldsymbol{X}_1^l, \cdots, \boldsymbol{X}_N^l\}$ and define the decoder block function as $f^l(\boldsymbol{X}_n^l)$ with input $\boldsymbol{X}_n^l \in \mathcal{X}^l$. The RGS loss for $l$-th block is defined as $\mathcal{L}_{RGS}^l(\boldsymbol{X}_n^l) = \|f^l(\boldsymbol{X}_n^l)\|_2$. By performing a single backpropagation through a certain decoder block regarding the $\mathcal{L}^l$, we can efficiently obtain the regional gradient as $\nabla_{\boldsymbol{W}_{ij}} \mathcal{L}_{RGS}^l(\boldsymbol{X}_n^l)$.

We compute the regional gradient $\nabla \mathcal{L}_{RGS}^l(\boldsymbol{X}_n^l)$ only once during the iteratively pruning and weights update process. Here, we replace the full-model gradient utilized in GBLM score with our regional gradient to obtained the RGS score. Note that both RGS and GBLM score only compute the gradient once. Thus, to further capture the interdependence of layers within the same block, the Wanda score is integrated, which tracks how pruning a single linear layer affects other layers in the block. In conclusion, ourRGS criterion is summarized as follows:

$$\boldsymbol{S}_{ij} = \left( \frac{\alpha}{N} \sqrt{\sum_{n=1}^{N} (\nabla \mathcal{L}_{RGS}^l(\mathbf{X}_n^l)^2)_{ij}} + \|\mathbf{X}_j\|_2 \right) \cdot |\boldsymbol{W}_{ij}| \tag{1}$$

where $N$ represents the total number of input samples used for pruning and the scaling factor $\alpha$ is a constant to balance the magnitude of the gradient and input activation terms. We choose the value of $\alpha$ to be 100 based on our ablation study in Appendix D.

**Algorithm 1** Pruning framework of Wanda++

**Require:** $\{\mathcal{X}^l\}^{l\in[1,L]}$ ▷ Inputs set for each decoder block
**Require:** Scaling factor $\alpha$
**for** $\ell = 1 \dots L$ **do**
  Calculating the RGS loss $\mathcal{L}^l_{RGS}$ with $\mathcal{X}^l$, backward, and record gradient $G$
  **for** $k = 1 \dots K$ **do**
    Selecting RO samples $\hat{\mathcal{X}}^l$ from $\mathcal{X}^l$
    Calculating and pruning with RGS by Eq. (1) ▷ Stage 1: Pruning
    **for** $\hat{\boldsymbol{X}}^l_m \in \hat{\mathcal{X}}^l$ **do** ▷ Stage 2: Regional Optimization
      Calculating the RO loss $\mathcal{L}^{l,k}_{ro}(\hat{\boldsymbol{X}}^l_m)$, backward, and update weights
    **end for**
  **end for**
  Calculating the RGS loss $\mathcal{L}^l_{RGS}$ with $\mathcal{X}^l$, backward, and record gradient $G$
  Calculating and pruning with RGS by Eq. (1)
**end for**
**return** Pruned model

### 2.2 Regional Optimization

The second stage for our Wanda++ framework is the Regional Optimization (RO). During this process, we slightly update the model weights within each decoder block to minimize the difference between the output from dense and pruned decoding blocks. Specifically, for the $l$-th decoder block, the output of the dense output can be represented as $f^l(\boldsymbol{X}^l_n)$ and the pruned output at $k$-th round is defined as $\hat{f}^{l,k}(\boldsymbol{X}^l_n)$, respectively. To further reduce the time of the RO process, we randomly select $M$ inputs from the inputs set $\mathcal{X}^l$ of each decoder block to construct an RO inputs set $\hat{\mathcal{X}}^l = \{\hat{\boldsymbol{X}}^l_1, \cdots, \hat{\boldsymbol{X}}^l_M\}$, without replacement. Then, the RO loss with input $\hat{\boldsymbol{X}}^l_m \in \hat{\mathcal{X}}^l$ for the $l$-th decoder in the $k$-th round can be defined as an MSE loss between the dense and pruned outputs, which gives:

$$\mathcal{L}^{l,k}_{ro}(\hat{\boldsymbol{X}}^l_m) = (f^l(\hat{\boldsymbol{X}}^l_m) - \hat{f}^l_k(\hat{\boldsymbol{X}}^l_m))^2. \tag{2}$$

For each RO sample $\hat{\boldsymbol{X}}^l_m$, we perform a forward pass within the decoder block to compute the RO loss, followed by backpropagation and a weight update. This process takes place after the pruning stage in each iteration of our Wanda++ framework. Typically, we randomly select 32 RO inputs from the 128 inputs used in the pruning stage at the start of each RO iteration. RMSprop optimizer (Ruder, 2016) is used with the learning rate of 3e-7.

## 3 Experiment

We closely follow the experimental setup of previous work like Wanda Sun et al. (2023) on unstructured sparsity, 2:4 sparsity, and 4:8 sparsity. Regarding the model, we consider OpenLLaMA (3B/7B/70B), LLaMA-1 (7B/13B/30B/65B) and LLaMA-3.1 (8B). By default, we randomly select 128 samples from the C4 training data for regional optimization and evaluate perplexity on both the C4/Wikitext test datasets. For zero-shot evaluation, we use the Harness evaluation toolkit (Gao et al., 2024). We test Wanda++ RGS for the performance of RGS criteria itself. Further experiments on the sensitivity analysis regarding sample sizes , latency reduction, and the ablation studies of the RO component can be found in Appendix A, B, E. All experiments are conducted on the H100 GPU.

**Perplexity:** We report our results in Table 1. We consider SparseGPT, Wanda and GBLM as baselines to compare with our proposed method. In all experiments, our method consistently shows superior perplexity compared to the baseline methods. For the OpenLLaMA-3B, the relative perplexity reductions are 32.1% and 25.5% for 2:4 and 4:8 sparsity, respectively. On LLaMA-1-7B model with 2:4 sparsity, the relative reductions are 19%, 20%, 7%, and 11% for model sizes 7B, 13B, 30B, and 65B, respectively. Compared to the GBLM score, the Wanda++ RGS achieves similar performance, demonstrating that regional gradients serve as a strong replacement for full-model gradients, especially for the 30B model. We were unable to obtain results for the 65B/70B models using the GBLM and SparseGPT methods due to computational resource limitations.

**Zero-Shot Accuracy:** We compare the downstream performance with Wanda. Although no consistent pattern shows, *Wanda++* in general yields the best performance among the three pruning methods

| Method | Sparsity | LLaMA-1 | | | | OpenLLaMA | | | LLaMA-3.1 |
|---|---|---|---|---|---|---|---|---|---|
| | | 7B | 13B | 30B | 65B | 3B | 7B | 70B | 8B |
| Baseline | - | 5.68 | 5.09 | 4.77 | 3.56 | 7.27 | 6.49 | 4.30 | 6.39 |
| SparseGPT | 0.5 | 7.22 | 6.21 | 5.31 | 4.57 | 10.41 | 8.57 | - | - |
| Wanda | | 7.26 | 6.15 | 5.24 | 4.57 | 12.37 | 9.15 | 5.25 | 9.99 |
| GBLM | | 7.15 | 6.11 | 5.18 | - | 10.75 | 8.49 | - | 9.90 |
| *Wanda++ RGS* | | 7.18 | 6.12 | 5.15 | 4.48 | 10.78 | 8.50 | 5.19 | 9.92 |
| *Wanda++* | | 7.02 (-3%) | 6.00 (-2%) | 5.10 (-3%) | 4.43 (-3%) | **9.25 (-25%)** | **7.82 (-15%)** | 5.11 (-3%) | **9.22 (-7%)** |
| SparseGPT | 2:4 | 11.00 | 9.11 | 7.16 | 6.28 | 15.91 | 11.62 | - | - |
| Wanda | | 11.59 | 9.69 | 6.90 | 6.25 | 28.04 | 15.35 | 6.47 | 24.83 |
| GBLM | | 11.33 | 9.16 | 6.87 | - | 24.75 | 13.19 | - | 24.34 |
| *Wanda++ RGS* | | 11.46 | 9.44 | 6.93 | 6.23 | 24.77 | 13.27 | 6.40 | 24.54 |
| *Wanda++* | | **9.43 (-19%)** | **7.75 (-20%)** | **6.39 (-7%)** | **5.59 (-11%)** | **19.03 (-32%)** | **11.30 (-26%)** | 6.35 (-2%) | **18.32 (-26%)** |
| SparseGPT | 4:8 | 8.61 | 7.40 | 6.17 | 5.38 | 12.20 | 9.79 | - | - |
| Wanda | | 8.61 | 7.40 | 5.97 | 5.30 | 16.83 | 11.38 | 5.73 | 14.63 |
| GBLM | | 8.48 | 7.26 | 5.89 | - | 14.86 | 10.38 | - | 14.29 |
| *Wanda++ RGS* | | 8.58 | 7.33 | 5.90 | 5.17 | 14.92 | 10.42 | 5.70 | 14.32 |
| *Wanda++* | | **7.88 (-8%)** | **6.75 (-9%)** | **5.65 (-5%)** | 5.07 (-4%) | **12.54 (-25%)** | **9.42 (-17%)** | 5.65 (-1%) | **12.55 (-14%)** |

Table 1: WikiText perplexity comparison across baselines. Bold highlights relative perplexity improvements over Wanda of 5% or more.

including Wanda which leads in `BoolQ` task. Compared to the margin from perplexity evaluation, the improvement from *Wanda++* against *Wanda++ RGS* is less salient. This is reasonable as RO is conducted on C4 dataset without optimizing any downstream tasks. Note that for `Mrpc` and `RTE` tasks, *Wanda++* outperforms Wanda by 46% and 24%, close to the accuracy of the dense baseline.

| Method | Wic | Mrpc | Hellaswag | Arc_easy | Arc_challenge | Winogrande | BoolQ | RTE | OBQA | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Baseline | 49.84 | 69.12 | 56.96 | 75.29 | 41.80 | 70.00 | 75.02 | 66.43 | 34.40 | 59.87 |
| Wanda | 48.75 | 46.81 | 41.66 | 59.34 | 27.47 | 61.96 | **69.60** | 49.82 | 23.80 | 47.69 |
| *Wanda++ RGS* | 49.37 (1%) | 64.46 (38%) | 41.43 (-1%) | 62.42 (5%) | **31.06 (13%)** | 62.83 (1%) | 67.95 (-2%) | 58.48 (17%) | 23.40 (-2%) | 51.27 |
| *Wanda++* | **50.00 (2%)** | **68.38 (46%)** | **45.31 (8%)** | **63.72 (7%)** | 29.27 (6%) | **65.04 (4%)** | 67.80 (-2%) | **62.09 (24%)** | **24.80 (4%)** | **52.93** |

Table 2: Zero-shot accuracy (%) from Llama-1 7B across various tasks under 2:4 sparsity.

**Pruning Efficiency:** We compare the pruning time of different methods to highlight the efficiency of Wanda++. Experiments for 7B/13B models are conducted with one 80G GPU and 65B model use four 80G GPUs. The results are summarized in Table 3, where Wanda++ (M) uses an input length of 128, while Wanda++ (L) follows the standard length of 2048, like other methods. Note that Wanda++ (M) is enough to achieve the performance in Table 1 based on App. A even though we report Wanda++ (L) results as a reference. We can observe that our method enables efficient pruning, requiring 10 minutes or less for 7B/13B models and 30 minutes for the 65B model.

| Method | Time (Sec.) | | |
|---|---|---|---|
| | 7B | 13B | 65B |
| SparseGPT | 322 | 594 | - |
| GBLM | 5801 | 10733 | - |
| Wanda | 55 | 95 | 628 |
| *Wanda++ RGS* | 147 | 190 | 1461 |
| *Wanda++ (M)* | 290 | 574 | 1821 |
| *Wanda++ (L)* | 2381 | 5569 | 22409 |

Table 3: Pruning time comparison.

**Sparsity-aware Fine-tuning:** We conducted experiments using LoRA Hu et al. (2021) to fine-tune both Wanda and Wanda++ pruned LLaMA-1 7B models. We followed the same experimental settings as the original Wanda paper, where LoRA is applied to the q and v modules in all transformer blocks. Both models were trained for 30k steps on the C4 dataset. As shown in Table 4, our method achieves similar 27% improvements with LoRA compared to the Wanda method. This demonstrates our proposed method is orthogonal to the fine-tuning approach, further strengthening the fairness of our comparison with weight-update-free methods like Wanda in Table 1.

| Methods | Dense Model | Pruned Model | After LoRA-tuned |
|---|---|---|---|
| Wanda | 5.68 | 11.59 | 8.23(-29%) |
| Wanda++ | 5.68 | 9.43 | 6.88(-27%) |

Table 4: Perplexity comparison on Wikitext with LoRA. All experiments are conducted on Lllama-7B model with 2:4 sparsity.

## 4 CONCLUSION

In this paper, we proposed Wanda++, a lightweight post-training LLM pruning method that leverages regional gradients to effectively mitigate pruning-induced performance degradation. By utilizing regional gradients, it outperforms Wanda on various LLaMA models, where Wanda only uses layer-wise weight and activation information. Wanda++ is efficient, especially compared to full gradient fine-tuning methods, pruning 7B LLMs in 10 minutes, as it operates only within each decoder block.

## ACKNOWLEDGEMENT

We thank Denis Filimonov for his valuable contributions, particularly in shaping the initial idea and refining the RGS score. His insights and discussions significantly enriched this work.

## REFERENCES

Megha Agarwal. Llm inference performance engineering: Best practices, 2023. URL https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices.

Rocktim Jyoti Das, Liqun Ma, and Zhiqiang Shen. Beyond size: How gradients shape pruning decisions in large language models. *arXiv preprint arXiv:2311.04902*, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=1tRLxQzdep.

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

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. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

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

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112*, 2022.

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

Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. Fp8 quantization: The power of the exponent. *Advances in Neural Information Processing Systems*, 35:14651–14662, 2022.

Xinjin Li, Yu Ma, Yangchen Huang, Xingqi Wang, Yuzhen Lin, and Chenxi Zhang. Integrated optimization of large language models: Synergizing data utilization and compression techniques. *Preprints. 10.20944/preprints202409. 0662. vl*, 2024.

Chen Liang, Haoming Jiang, Zheng Li, Xianfeng Tang, Bin Yin, and Tuo Zhao. Homodistil: Homotopic task-agnostic distillation of pre-trained transformers. *arXiv preprint arXiv:2302.09632*, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100, 2024.

NVIDIA Developer Blog. Sparsity in int8 training: Workflow and best practices for tensorrt acceleration, 2023. URL https://developer.nvidia.com/blog/sparsity-in-int8-training-workflow-and-best-practices-for-tensorrt-acceleration/. Accessed: 2024-10-04.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

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

Jiayi Tian, Chao Fang, Haonan Wang, and Zhongfeng Wang. Bebert: Efficient and robust binary ensemble bert. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

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

Han Xu, Yuhong Shao, Kareem Benaissa, and Yutong Li. Sparsebf: Enhancing scalability and efficiency for sparsely filled privacy-preserving record linkage. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 4143–4147, 2024.

Yifan Yang, Jiajun Zhou, Ngai Wong, and Zheng Zhang. Loretta: Low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3161–3176, 2024.

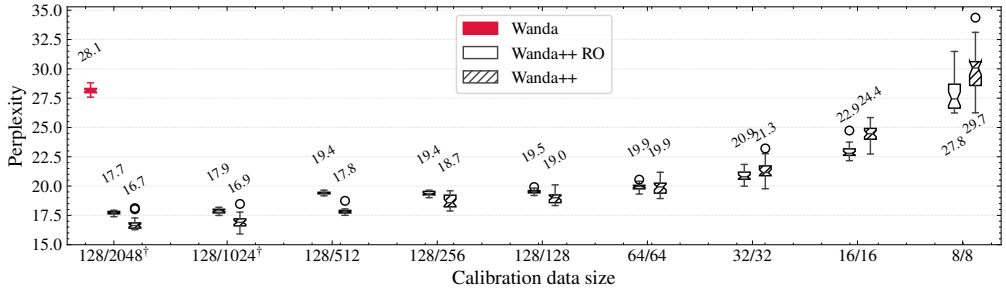


Figure 2: Box plot of perplexity on the Wikitext test set, based on 30 runs from 2:4 sparse OpenLLaMA-3B model.

## A Sensitivity Analysis

While Wanda's complexity is $O(d^2_{\text{hidden}})$, the pruning time and memory consumption both depend linearly on the amount of calibration data. This is also the case for our proposed methods. We alternate the number of samples and context length of each sample in C4 training data and compare the corresponding perplexities in each calibration dataset setting in Figure 2 as the box plot. OpenLLaMA-3B is used in this sensitivity analysis on the size of calibration data. We run each experiment 30 times. Each box extends from the lower to the upper quartile with a 95% confidence interval (the notch) of the median. The outliers are also shown in the black circles. For Wanda, we stick to the default setting with 128 calibration samples and 2048 context length each. For both Wanda++ RO and Wanda++, we consider nine calibration settings (number of samples/context length): from a tiny calibration set of 8/8 up to the 128/2048 case. In both $128/2048^{\dagger}$ and $128/1024^{\dagger}$ settings, each epoch uses 32 random samples to avoid out-of-memory issues.

Compared to Wanda, which shows stable perplexity across various numbers of calibration samples Sun et al. (2023), our methods favor larger calibration sizes, particularly for Wanda++, which only starts to outperform Wanda++ RO beyond the 64/64 setting. However, even at the 16/16 setting, both of our proposed methods yield lower perplexities than Wanda. Both Wanda and Wanda++ RO are more stable overall than Wanda++. The comparison is less contrastive with larger calibration datasets.

## B Model Size and Latency Reduction

We measure the Time to First Token (TTFT), Time Per Output Token (TPOT) and total model weight memory consumption to examine 2:4 sparsity's actual impact on a dummy 7B LLaMA-akin model in Table 5 using TensorRT-LLM-0.9.0 with the Sparse Tensor Core support NVIDIA Developer Blog (2023), which are widely adapted in other work Li et al. (2024). Only the multi-layer perceptron (MLP) modules are pruned, with both tensor parallelism and pipeline parallelism set to 1. Under FP16 format, we observe a TTFT reduction of 33% or more, while the TPOT reduction is around 10%. Total weight memory is reduced by 28% for FP16 (from 12.8 GB to 9.2 GB), which are also reflected in the size of compiled TensorRT engines. See Appendix F for FP8 format Kuzmin et al. (2022) results.

| Batch Size | Token Length | | Latency | | Weight Memory |
|---|---|---|---|---|---|
| | Input | Output | TTFT | TPOT | |
| 1 | 128 | 64 | 33 | 10 | 28 |
| | 1024 | 64 | 47 | 11 | |
| | 2048 | 64 | 47 | 10 | |
| | 4096 | 64 | 46 | 10 | |
| 4 | 128 | 64 | 45 | 11 | |
| | 1024 | 64 | 47 | 11 | |
| | 2048 | 64 | 45 | 9 | |
| | 4096 | 64 | 43 | 7 | |

Table 5: Relative reduction (%) for latency and weight memory from 2:4 sparsity under FP16 format.

## C Pruning Time and Memory Consumption

We further discuss the memory and time efficiency of our proposed method. The result is summarized in Table. 6. As mentioned earlier, integrating gradient information into the pruning score poses a significant computational challenge. Wanda avoids any gradient approximation and backpropagation, and therefore achieves simple and efficient LLM pruning compared with other post-training pruning methods like SparseGPT.

Here, we evaluate the time and memory costs during the pruning process to demonstrate that our proposed method maintains similar computational advantages, especially when compared with previous LLMs pruning methods that utilize full model gradient information. It is important to note that the memory and pruning results were obtained by running the source code of each method, under the assumption that the released code has been fully optimized for that particular method. For GBLM, we combined the time for both gradient computation and pruning, as they are handled separately in the code.

Without model weight updates, Wanda has the shortest pruning time. Wanda++ RGS (without RO) comes in second. When RO is added, as shown in the Wanda++ (M) row, the pruning time approaches that of SparseGPT. We also report metrics for Wanda++ (L) as a reference, though in practice, Wanda++ (M) is sufficient to achieve the performance shown in Table. 1. It takes 10 minutes or less to prune the 7B and 13B models, and about 30 minutes for the 65B model. For 7B and 13B, one 80 GB GPU is enough, while the 65B model requires 4 H100 GPUs.

| **Method** | **Time (Sec.)** | | | **Memory (GB)** | | |
|---|---|---|---|---|---|---|
| | 7B | 13B | 65B | 7B | 13B | 65B |
| SparseGPT | 322 | 594 | - | 23 | 38 | - |
| GBLM | 5801 | 10733 | - | 26 | 50 | - |
| Wanda | 55 | 95 | 628 | 22 | 36 | 320 |
| *Wanda++ RGS* | 147 | 190 | 1461 | 29 | 49 | 320 |
| *Wanda++ (M)* | 290 | 574 | 1821 | 25 | 49 | 280 |
| *Wanda++ (L)* | 2381 | 5569 | 22409 | 31 | 49 | 335 |

Table 6: Memory and pruning time comparison: For Wanda++, we consider two calibration settings that differ in the number of tokens per input sample. *Wanda++ (M)* uses an input length of 128, while *Wanda++ (L)* uses 2048 like others.

## D Ablation Study on RGS Scaling Factor

We examine the hyperparameter $\alpha$ in the RGS criterion (Eq. 1), which balances the regional gradient score and the layer-wise Wanda score. An ablation study is conducted, testing $\alpha$ values from 1 to 1,000,000, to assess their effect on perplexity. Results from LLaMA-3 8B models are presented in Table 7. The lowest perplexity for LLaMA-3 8B with 2:4 sparsity occurs at $\alpha = 50$, indicating that the optimal choice of $\alpha$ is model-specific. However, with $\alpha = 100$, the perplexity remains close to that at $\alpha = 50$. For simplicity, we reuse the setting of $\alpha = 100$ as in (Das et al., 2023).

## E Ablation Study on the RO component

Here, we further evaluate the effectiveness of the RO component, providing extended results from Table 1. In Table 8, we present Wanda++ RO, which integrates the RO method in Wanda++ with Wanda. For better comparison, we also include results for Wanda++ RGS, which uses only the RGS score. We observe that RO is compatible with the Wanda method, yielding an 8% improvement over Wanda alone with only a few additional seconds of pruning time. However, the best results are achieved when combining the RGS score with the RO process.

| RGS Criterion | Perplexity |
|---|---|
| $(1 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 24.55 |
| $(10 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 24.62 |
| $(50 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 23.99 |
| $(100 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 24.68 |
| $(500 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 25.66 |
| $(1000 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 26.25 |
| $(5000 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 29.07 |
| $(10000 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 29.90 |
| $(1000000 \cdot \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot \|\boldsymbol{W}_{ij}\|$ | 31.14 |

Table 7: Perplexity with different $\alpha$ values in RGS on LLaMA-3 8B model for 2:4 sparsity.

| **Method** | **Sparsity** | **LLaMA-1** | | | | **OpenLLaMA** | | | **LLaMA-3.1** |
|---|---|---|---|---|---|---|---|---|---|
| | | 7B | 13B | 30B | 65B | 3B | 7B | 70B | 8B |
| Baseline | - | 5.68 | 5.09 | 4.77 | 3.56 | 7.27 | 6.49 | 4.30 | 6.39 |
| Wanda | | 7.26 | 6.15 | 5.24 | 4.57 | 12.37 | 9.15 | 5.25 | 9.99 |
| *Wanda++ RO* | | 7.07 | 6.08 | 5.12 | 4.43 | 9.86 | 8.27 | 5.14 | 9.34 |
| *Wanda++ RGS* | | 7.18 | 6.12 | 5.15 | 4.48 | 10.78 | 8.50 | 5.19 | 9.92 |
| *Wanda++* | | 7.02 (-3%) | 6.00 (-2%) | 5.10 (-3%) | 4.43 (-3%) | **9.25 (-25%)** | **7.82 (-15%)** | 5.11 (-3%) | **9.22 (-7%)** |
| Wanda | | 11.59 | 9.69 | 6.90 | 6.25 | 28.04 | 15.35 | 6.47 | 24.83 |
| *Wanda++ RO* | | 10.78 | 7.89 | 6.51 | 5.86 | 19.41 | 11.69 | 6.37 | 19.43 |
| *Wanda++ RGS* | | 11.46 | 9.44 | 6.93 | 6.23 | 24.77 | 13.27 | 6.40 | 24.54 |
| *Wanda++* | | **9.43 (-19%)** | **7.75 (-20%)** | **6.39 (-7%)** | **5.59 (-11%)** | **19.03 (-32%)** | **11.30 (-26%)** | 6.35 (-2%) | **18.32 (-26%)** |
| Wanda | | 8.61 | 7.40 | 5.97 | 5.30 | 16.83 | 11.38 | 5.73 | 14.63 |
| *Wanda++ RO* | | 8.34 | 7.18 | 5.73 | 5.11 | 13.10 | 9.52 | 5.67 | 12.88 |
| *Wanda++ RGS* | | 8.58 | 7.33 | 5.90 | 5.17 | 14.92 | 10.42 | 5.70 | 14.32 |
| *Wanda++* | | **7.88 (-8%)** | **6.75 (-9%)** | **5.65 (-5%)** | 5.07 (-4%) | **12.54 (-25%)** | **9.42 (-17%)** | 5.65 (-1%) | **12.55 (-14%)** |

Table 8: Wikitext perplexity comparison on LLaMA-1, OpenLLaMA, and LLaMA-3.1 model families. Bold highlights relative perplexity improvements over Wanda of 5% or more.

## F LATENCY / MODEL SIZE REDUCTION FOR FP8

When the weight, activation and KV-cache are quantized to the FP8 format, the TTFT latency reduction from 2:4 sparsity is smaller, particularly when the batch size and input length increase compared to that under FP16. One explanation is that the model leans towards being compute-bound, where reducing weight memory load becomes less meaningful. TPOT reduction under FP8 is 13% or greater, except when the batch size is 4 and the output length is 4096. Total weight memory is reduced by 22% with 2:4 sparsity under the FP8 format (from 6.8 GB to 5.3 GB).