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`
Project Page: `https://yifan-yang.net/wandapp.github.io/`

Abstract

Large Language Models (LLMs) pruning seeks to remove unimportant weights for inference speedup with minimal accuracy impact. However, existing methods often suffer from accuracy degradation 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. Moreover, despite updating weights with regional optimization, Wanda++ remains orthogonal to sparsity-aware fine-tuning, further reducing perplexity with LoRA in great extend. Our approach is lightweight, pruning a 7B LLaMA model in under 10 minutes on a single H100 GPU.

1 Introduction

The growing size of Large Language Models (LLMs) improves performance (Touvron et al., 2023; Intelligence, 2024a,b) at the cost of memory consumption and inference latency. For example, loading the weights of LLaMA-2 70B requires 140 GB of GPU memory in FP16 format. Hosting a such model even with a batch size of 1 and 512 input tokens 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; Ghiasvand et al., 2024), quantization (Lin et al., 2024; Zhou et al., 2025), and pruning (Sun et al., 2023; Frantar and Alistarh, 2023; Zhang et al., 2024), have

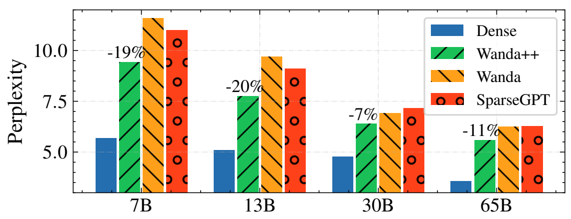


Figure 1: Wanda++ mitigates 2:4 pruning-induced degradation more effectively, with relative perplexity improvement over Wanda shown on Wikitext using LLaMA-1 models across four different sizes.

been explored. Similar to LLM quantization, many recent LLM pruning methods have shifted from in-training approaches (Han et al., 2015b; Frankle and Carbin, 2018) to post-training methods, as seen in SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2023). However, unlike post-training quantization like AWQ (Lin et al., 2024), which compresses model weights almost losslessly by $4\times$ (from 16-bit to 4-bit), these pruning methods have yet to achieve comparable levels of performance.

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 (BP). These studies highlight the importance of gradient information, demonstrating that it provides valuable insights even though Wanda assumes that the gradients of a fully trained network are small and contribute little when higher-order terms are considered. Meanwhile, other approaches focus on recovering model performance through full-model sparsity-aware tuning (Sun et al., 2023) or distillation (Liang et al., 2023). However, all these methods, which heavily depend on full-model loss, suffer from impractical memory requirements and excessive pruning time due to the high computational cost of full-model backpropagation. This raises the 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 sacrificing the pruning efficiency. Compared with traditional full model BP, the regional gradient can be obtained by only loading and computing gradients per single decoder block, which largely reduce the GPU hours required. Compare with previous gradient-free method like Wanda, the regional gradient can improve the pruning performance by accessing the gradient information. Wanda++ prune each decoder block by iteratively prune the model based on a Regional Gradient Score (RGS) and slightly update the weights with our proposed Regional Optimizer (RO).

To effectively obtain the regional gradient in the RGS score, we compute the regional gradient through a backward process on a regional loss, which is defined as the ℓ_2 norm of each decoder's output hidden states. To involve the regional gradient into the pruning score, we follow the design of GBLM score (Das et al., 2023). Note that the regional gradient is only computed once during the iterative pruning process of each decoder block and reused for all RO iterations to further reduce the computation. As a result, the regional gradient may become less accurate after weights being updated during the RO process. The design of GBLM score provide an effective way to reflect changes by fetch the input per in-block layer and blend it into the RGS along with regional gradients.

For the RO in Wanda++, we construct a simple loss function between the outputs of the dense and pruned decoder blocks to update the model weights within each block. The concept of regional optimization traces back to previous local optimization approaches in convolutional neural networks (CNNs), which aimed to optimize models by focusing on individual convolutional layers (Wang et al., 2021). Differently, RO in Wanda++ extends this idea for mitigating pruning-induced loss on a small portion of calibration data at each LLM decoder-block, instead of optimizing the cross-entropy loss for classification tasks.

Our proposed framework achieves up to a 32% reduction in WikiText perplexity at 2:4 sparsity, compared to the Wanda method under the same experimental setup, as shown in Figure. 1. Our contributions can be summarized as follows:

- We propose Wanda++, a lightweight yet effective framework that prunes LLMs using regional gradients, achieving performance improvements without requiring access to full model gradients.

- Wanda++ effectively lowers the pruning-induced degradation in a non-incremental way and generalize well in zero-shot downstream tasks.

- The RO method in Wanda++ is orthogonal to previous full-model sparsity-aware fine-tuning methods and has been shown to achieve a similar perplexity improvement as Wanda after applying LoRA fine-tuning.

2 Related Work

Network Pruning: The concept of pruning neural networks has been explored for decades, beginning with foundational works such as (LeCun et al., 1989; Hassibi et al., 1993). In addition to widely studied unstructured pruning approaches, structured pruning methods, which remove entire subnets of a network or rows/columns within weight matrices, are more easily supported on hardware for inference speedup (Liu et al., 2017; Shen et al., 2022). Research in this area has focused on analyzing input/output activation statistics to identify the most suitable neurons for pruning (Bai et al., 2021; Molchanov et al., 2022). However, even at 50% sparsity, structured pruning often results in non-trivial performance degradation (Ashkboos et al., 2024). Beyond these methods, semi-structured pruning (Pool et al., 2021; Fang et al., 2022, 2023), such as 2:4 sparsity, shows greater resilience at 50% sparsity and can effectively reduce runtime latency with throughput improvement on NVIDIA's recent hardware. Therefore, this work primarily focuses on unstructured and semi-structured patterns, although the proposed Wanda++, RO in particular, is sparsity pattern agnostic.

LLM Pruning: As LLMs continue to grow in size, scaling traditional pruning methods to accommodate them presents significant challenges. Traditional pruning methods, which typically require full model retraining, demand substantial computational resources, making them impractical in the era of LLMs. A notable trend in LLM pruning is the adoption of post-training pruning methods (Frantar and Alistarh, 2023; Sun et al., 2023; Das et al., 2023; Zhang et al., 2025), which develop specific pruning scores to determine the importance of different weight elements. Additionally, SliceGPT

(Ashkboos et al., 2024) focuses on structured pruning by slicing rows or columns of weight matrices based on input eigenvectors and eigenvalues.

Pruning with Gradient Information: The use of gradient information during the training process has been extensively studied in two ways. The first set of methods directly incorporates gradient information to refine their pruning scores at a finer granularity. Recent developments, such as GBLM and Pruner-Zero (Das et al., 2023; Dong et al., 2024) use gradients obtained through the full model backward with respect to cross entropy loss to achieve better performance, though they are time and memory infeasible for large-scale model.

The second set of methods focuses more on incorporating gradient information during sparsity-aware distillation or fine-tuning. For example, (Liang et al., 2023) explores a task-agnostic distillation approach combined with iterative pruning to achieve strong performance on natural language understanding tasks. Additionally, full-model or LoRA fine-tuning has been considered as an auxiliary step in various prior pruning works that emphasize single-shot performance to further reduce the pruning degradation, such as Wanda (Sun et al., 2023) and SliceGPT (Ashkboos et al., 2024).

3 Prior Solutions

Existing solvers for post-training LLM pruning often start with magnitude pruning (Han et al., 2015b), which removes individual weights based on their magnitudes. This approach has been widely adopted as a baseline method in both vision (Han et al., 2015a) and language processing (Gale et al., 2019). Magnitude pruning is built on the assumption that neurons with larger weight elements contribute higher gradients when input features have similar magnitudes. However, this assumption does not always hold true for LLMs, where input features can differ significantly in scale, as observed in (Dettmers et al., 2022).

To address this challenge, Sun et al. propose the Wanda method (Sun et al., 2023), which introduces a pruning criterion that multiplies the weight magnitude by the input activation on an element-wise basis. Specifically, given a linear layer with a weight matrix $\boldsymbol{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ and input $\boldsymbol{X} \in \mathbb{R}^{(B\times L)\times d_{in}}$ assuming the batch size is B, sequence length is L, the Wanda pruning score for the weight element connect input j and output i is defined as:

$$\boldsymbol{S}_{ij} = |\boldsymbol{W}_{ij}| \cdot \|\boldsymbol{X}_j\|_2, \tag{1}$$

where $\|\boldsymbol{X}_j\|_2$ represent the L2 norm of the j-th input channel $\boldsymbol{X}_j$. The Wanda score is further improved by (Das et al., 2023), which shows the first order gradient is still crucial even though Wanda score consider the higher-order terms and gives the GBLM score:

$$\boldsymbol{S}_{ij} = (\alpha \boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot |\boldsymbol{W}_{ij}|, \tag{2}$$

where $\boldsymbol{G}$ is full model gradient with respect to the cross-entropy loss between the prediction and labels. Next, we introduce how to involve the regional gradient into the pruning score and perform a effective and lightweight regional optimization.

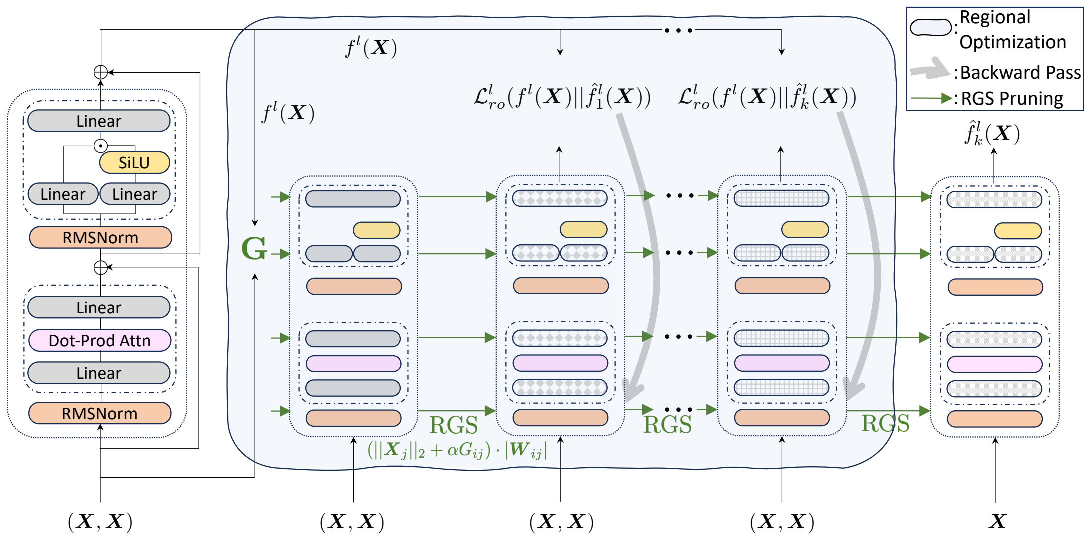
4 The Wanda++ Framework

In this section, we use the Wanda method to highlight the drawbacks of the linearity assumption in previous layer-wise post-training pruning approaches. We then introduce our Wanda++ framework, which efficiently reduces pruning degradation with a two-stage process for each decoder. This involves layer-wise pruning within blocks based on a new pruning criterion called the Regional Gradient Score (RGS) and Regional Optimization (RO), as shown in Figure 2. Wanda++ is efficient as it operates at each decoder block where its RO requires only 5 iterations.

4.1 Regional Gradient Score

As the first stage of Wanda++, we obtain the Regional Gradient Score (RGS) for in-block layer-wise 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 BP through a certain decoder block regarding the regional loss $\mathcal{L}^l$, we can efficiently obtain the stochastic estimated gradient magnitude for each weight matrix by taking the absolute value for the estimated gradient $\nabla_{\boldsymbol{W}_{ij}} \mathcal{L}_{RGS}^l(\boldsymbol{X}_n^l)$.

By performing the element-wise multiplication between the weight and estimated gradient magnitude, we have the initial regional gradient-based



: Regional Optimization
: Backward Pass
: RGS Pruning
$f^l(\boldsymbol{X})$
$\mathcal{L}^l_{ro}(f^l(\boldsymbol{X})\|\hat{f}^l_1(\boldsymbol{X}))$
$\mathcal{L}^l_{ro}(f^l(\boldsymbol{X})\|\hat{f}^l_k(\boldsymbol{X}))$
$\hat{f}^l_k(\boldsymbol{X})$
Linear
SiLU
Linear
Linear
RMSNorm
Linear
Dot-Prod Attn
Linear
RMSNorm
G
RGS
RGS
RGS
$(\|\boldsymbol{X}_j\|_2+\alpha G_{ij})\cdot|\boldsymbol{W}_{ij}|$
$(\boldsymbol{X},\boldsymbol{X})$
$(\boldsymbol{X},\boldsymbol{X})$
$(\boldsymbol{X},\boldsymbol{X})$
$(\boldsymbol{X},\boldsymbol{X})$
$\boldsymbol{X}$

Figure 2: Illustration of Wanda++, which leverages regional gradients in two ways: first, we estimate the block-level gradient **G**, used in Regional-Gradient Scoring (RGS) for layer-wise pruning in each block. Next, we calculate the pruning-induced loss $\mathcal{L}ro$ for lightweight Regional Optimization (RO). RO can be iterated multiple times to infer better pruning masks for each decoder block.

score for each weight element W_{ij}:

$$\boldsymbol{S}_{ij} = \Big(\sqrt{\frac{\sum_{n=1}^{N} \nabla\mathcal{L}^l_{RGS}(\boldsymbol{X}^l_n)^2}{N}}\Big)_{ij} \cdot |\boldsymbol{W}_{ij}|, \quad (3)$$

where N represents the total number of input samples used for pruning. For simplicity, we record the score in Eq. (3) as $\boldsymbol{S}_{ij} = \boldsymbol{G}_{ij} \cdot |\boldsymbol{W}_{ij}|$.

To limit the overhead of computing the pruning criterion, we estimate the regional gradient (Eq. 4) only once per decoder block. This can be suboptimal because pruning one layer in the l-th block affects the regional gradients of other layers. To address this, we blend in the layer-wise Wanda score, which tracks how pruning in one layer impacts the input of the next. Our RGS criterion is summarized as follows:

$$\boldsymbol{S}_{ij} = (\alpha\boldsymbol{G}_{ij} + \|\boldsymbol{X}_j\|_2) \cdot |\boldsymbol{W}_{ij}|, \quad (4)$$

where the scaling factor α is a constant to balance the magnitude of the gradient and input activation terms. For simplicity, we adopt the same scaling value of 100 as in (Das et al., 2023) although it can be model specific (see the ablation study on α in Appendix B.2).

4.2 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:

$$\mathcal{L}^{l,k}_{ro}(\hat{\boldsymbol{X}}^l_m) = (f^l(\hat{\boldsymbol{X}}^l_m) - \hat{f}^l_k(\hat{\boldsymbol{X}}^l_m))^2. \quad (5)$$

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.

4.3 Algorithms

We summarize the Wanda++ algorithm flow in Alg. 1, as also shown in Figure 2. For each decoder block, we perform layer-wise RGS pruning (step 5) followed by a round of the RO process (steps 6-8) for K iterations. An additional RGS pruning (step 11) is required to restore sparsity after RO. To detail how RGS is computed in steps 5 and 11, we provide PyTorch pseudo-code in Appendix A.

5 Experiment

We evaluate the proposed pruning method based on four criteria. The first is perplexity in next-token prediction, which is the main objective of LLM pre-training. A lower perplexity is often a strong indicator of superior performance in language understanding tasks. The second criterion is the performance on zero-shot/few-shot NLP tasks, such as text classification, question answering, and text generation. This is to ensure that the gradients involved in regional optimization do not lead to overfitting for downstream tasks. The third criterion is pruning time and memory consumption. Finally, we examine the actual latency reduction under both FP16 and FP8 quantization settings for various batch sizes and input/output lengths.

5.1 Experimental Setup

Regarding the model, we consider OpenLLaMA (3B/7B/70B), LLaMA-1 (7B/13B/30B/65B) and LLaMA-3.1 (8B). We follow the same settings as in (Sun et al., 2023) to examine pruning performance on unstructured sparsity, 2:4 sparsity, and 4:8 sparsity. By default, we randomly select 128 samples from the C4 training data for regional optimization and evaluate perplexity on both the C4 validation and Wikitext test datasets. For zero-shot evaluation, we use the Harness evaluation toolkit (Gao et al., 2024). All experiments are conducted on the NVIDIA H100 GPU, where a single GPU is sufficient for models with 13B or fewer parameters and 4 GPUs are used for 65B and 70B models. Latency and model size are measured via TensorRT-LLM. Regarding the proposed pruning methods, by default, Wanda++ enables both RGS and RO; Wanda++ RGS excludes RO, using only RGS for pruning, while Wanda++ RO applies the Wanda score (Sun et al., 2023) for pruning and weight updates within each decoder block.

5.2 Perplexity

First, we consider OpenLLaMA-3B, a relatively small model, for a more contrastive perplexity comparison between Wanda and our proposed method. As shown in Figure 3, we begin by pruning the first 2 decoder blocks, then gradually apply N:M sparsity to two more decoder blocks until all 26 blocks are pruned. The LM head and embeddings are excluded from pruning. Perplexity results are reported on both the C4 validation dataset and the WikiText test dataset. When all decoder blocks are pruned with the N:M pattern, resulting in a 50% sparsity ratio, our method outperforms Wanda by a noticeable margin: on the C4 validation dataset, with 2:4 sparsity, perplexity is reduced from 36.5 to 23.9, a relative reduction of 34.4%; with 4:8 sparsity, perplexity is reduced from 23.1 to 16.4, a relative reduction of 29.0%. Similarly, on the WikiText test dataset, the relative perplexity reductions are 32.1% and 25.5% for 2:4 and 4:8 sparsity, respectively. The margin generally increases with the sparsity ratio, although in the lower sparsity ratio region, particularly on WikiText (right plot), the benefit is obscure. Note that, in higher sparsity ratios, our method can achieve comparable or superior perplexity results under 2:4 sparsity compared to Wanda for 4:8 sparsity, a much more relaxed sparsity setting.

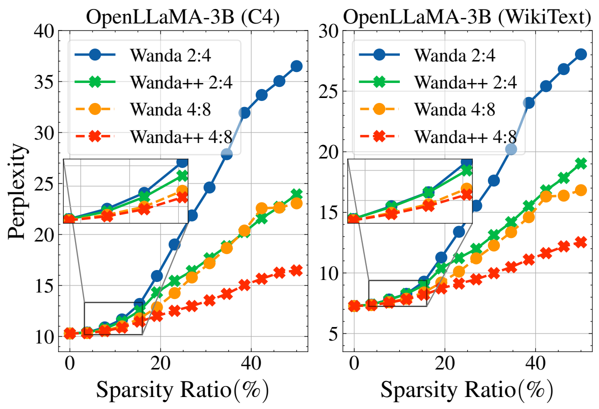


Figure 3: Perplexity on the C4 validation (left) and Wikitext test sets (right) as the sparsity ratio increases by gradually pruning two additional decoding blocks at a time, up to all 26 blocks. Wanda++ leverages regional gradients and significantly outperforms Wanda: our 2:4 results are comparable to Wanda's 4:8 counterparts.

A more comprehensive comparison of perplexity is summarized in Table 1, where LLaMA-1 models with four different sizes are included along with OpenLLaMA 7B and 70B. We consider SparseGPT and Wanda as two baseline pruning methods to compare with our method, Wanda++ w. RO. To understand the contribution of each enhancement in our method, we also report results from Wanda++ RO, where the Regional Optimizer (RO) is enabled for each decoder block, and Wanda++, where regional gradients are used along with the L2 norm of the input data for every layer in each decoder block. Regarding the sparsity patterns, we consider 50%

Algorithm 1 Pruning framework of Wanda++

```
Require: $\{\mathcal{X}^l\}^{l\in[1,L]}$                          ▷ Inputs set for each decoder block
Require: Scaling factor $\alpha$
 1: for $\ell = 1 \ldots L$ do
 2:     Calculating the RGS loss $\mathcal{L}^l_{RGS}$ with $\mathcal{X}^l$, backward, and record gradient $G$
 3:     for $k = 1 \ldots K$ do
 4:         Selecting RO samples $\hat{\mathcal{X}}^l$ from $\mathcal{X}^l$
 5:         Calculating and pruning with RGS by Eq. (4)                  ▷ Stage 1: Pruning
 6:         for $\hat{\boldsymbol{X}}^l_m \in \hat{\mathcal{X}}^l$ do            ▷ Stage 2: Regional Optimization
 7:             Calculating the RO loss $\mathcal{L}^{l,k}_{ro}(\hat{\boldsymbol{X}}^l_m)$, backward, and update weights
 8:         end for
 9:     end for
10:     Calculating the RGS loss $\mathcal{L}^l_{RGS}$ with $\mathcal{X}^l$, backward, and record gradient $G$
11:     Calculating and pruning with RGS by Eq. (4)
12: end for
13: return Pruned model
```

unstructured pruning, 2:4, and 4:8 semi-structured pruning methods, although 2:4 sparsity is the most commonly supported in hardware for runtime acceleration.

Between Wanda++ RO and Wanda++ RGS, leveraging regional gradients within the decoder block-level optimization proves more effective in mitigating pruning-induced degradation. Furthermore, applying RO to a better pruning criterion, as shown in Wanda++ w. RO compared to Wanda++ RO, further improves performance. We compare our method (the strongest among those three in italics) to Wanda and report the relative perplexity reductions in Table 1. The most noticeable benefit is observed with 2:4 sparsity: on LLaMA-1, the relative reductions are 19%, 20%, 7%, and 11% for model sizes 7B, 13B, 30B, and 65B, respectively. Our method is most effective in reducing pruning degradation for smaller models, where higher pruning degradations usually occur, which is also the case for OpenLLaMA models. In all experiments, our method consistently shows superior perplexity compared to the baseline methods. However, for 4:8 sparsity and unstructured pruning, where perplexity values are generally lower than those with 2:4 sparsity, the benefits of our method become less salient. Nonetheless, the improvement remains more substantial compared to what Wanda achieves on top of SparseGPT.

Sparsity-aware model fine-tuning may further improve the performance. We further show our Wanda++ method is orthogonal to LoRA fine-tuning (Hu et al., 2021) method in Section. 5.6. Also Another method distills smaller LLMs from a 15B pre-trained model but requires 16 DGX A100 nodes (Muralidharan et al., 2024) (128 80GB GPUs in total), whereas our method only needs one for the similar model size.

5.3 Zero-Shot Accuracy

We measure the zero-shot accuracy via Harness (Gao et al., 2024). Along with Wanda, we examine two of our proposed methods both leveraging regional gradients: *Wanda++ RGS* only uses regional gradients to compute the pruning criterion, while *Wanda++* applies them to both weight pruning (RGS) and in-block weight recompute (RO). We aim to understand if the observed benefit of improved perplexity from *Wanda++* can pan out in downstream NLP tasks, and that the integration of RO does not negatively impact the generality of the pruned model. Results from the LLaMA 7B model on 9 tasks are listed in Table 2. All pruned models are conformed to 2:4 sparsity.

Although no consistent pattern shows, *Wanda++* in general yields the best performance among the three pruning methods 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%, respectively, close to the accuracy of the dense baseline.

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++ 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%)**
SparseGPT*	2:4	11.00	9.11	7.16	6.28	15.91	11.62	-	-
Wanda*		11.53	9.58	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++ 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%)**
SparseGPT*	4:8	8.61	7.40	6.17	5.38	12.20	9.79	-	-
Wanda*		8.57	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++ 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 1: Wikitext perplexity comparison on LLaMA-1, OpenLLaMA, and LLaMA-3.1 model families. * indicates results from either the previous paper (Sun et al., 2023). - means results are not applicable from running their source code directly or out-of-memory issue. Bold highlights relative perplexity improvements over Wanda of 5% or more.

Method	Wic	Mrpc	Hellaswag	Arc_easy	Arc_challenge	Winogrande	BoolQ	RTE	MMLU
Baseline	49.84	69.12	56.96	75.29	41.80	70.00	75.02	66.43	35.10
Wanda	48.75	46.81	41.66	59.34	27.47	61.96	69.60	49.82	25.85
GBLM	49.32	65.31	41.80	61.43	30.45	63.24	**71.20**	57.43	26.34
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%)	26.40 (-2%)
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%)**	**27.52 (6%)**

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

5.4 Sensitivity Analysis

While Wanda's complexity is $O(d_{\text{hidden}}^2)$, 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 4 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.

5.5 Pruning Time and Memory Consumption

We further discuss the memory and time efficiency of our proposed method. 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

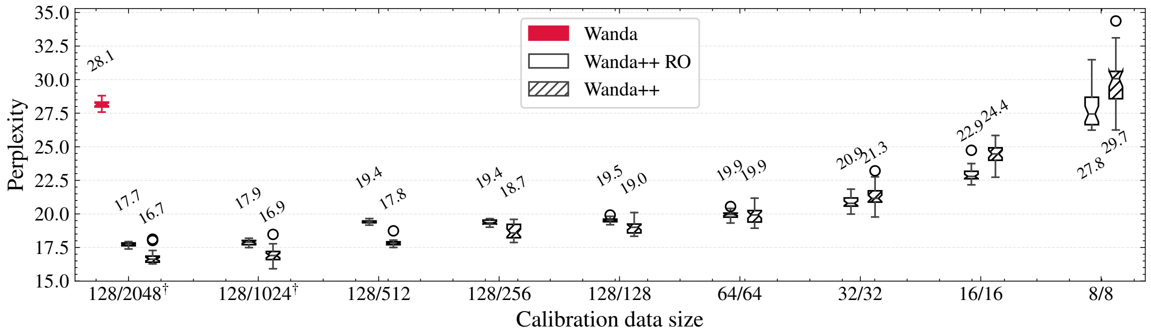


Figure 4: Box plot of perplexity on the Wikitext test set, based on 30 runs from 2:4 sparse OpenLLaMA-3B model.

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. For GBLM, we combined the time for both gradient computation and pruning, based on the provided 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.

Unlike conventional BP involved methods that require loading the full model into memory, Wanda++ significantly reduces memory overhead by performing regional optimization at the decoder block level—loading only one decoder block at a time. This design decouples memory cost from the overall model size, which scales with both hidden dimension and the total number of decoder blocks. Instead, Wanda++'s memory usage depends solely on the hidden size. For instance, in the Megatron-Turing NLG 530B model (Smith et al., 2022)—the largest model to our knowledge—each decoder block contains approximately 5.03 billion parameters with a hidden size of 20,480. The estimated theoretical memory required for optimizing a single block is approximately 40.24 GB (assuming we using Adam (Kingma, 2014) optimizer). While actual usage may be higher due to activations and implementation overhead, it remains within twice this estimate and can be accommodated by a single 80GB GPU with proper optimization. This modest memory cost is particularly favorable given the up to 32% performance improvement achieved by Wanda++, especially when contrasted with the 16–20 80GB GPUs typically needed just for inference of such large-scale models.

Method	**Time (Sec.)**			**Memory (GB)**		
	7B	13B	65B	7B	13B	65B
SparseGPT	322	594	-	23	38	-
GBLM	5801	10733	49663	26	50	269
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 3: Memory and pruning time comparison: For Wanda++, we consider two calibration settings that differ in the nunmber of tokens per input sample. *Wanda++ (M)* uses an input length of 128, while *Wanda++ (L)* uses 2048 like others.

5.6 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 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. Note that even we fine-tune the model with LoRA, the process is still time consuming, which takes around 12 hours to improve the performance. Also, the LoRA fine-tuning method requires much more memory compared with regional optimization, which makes it infeasible for the weights update of large scale

Methods	Dense	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 LLaMA-7B model with 2:4 sparsity.

pruned models.

5.7 Experiments on Higher Sparsity

In this section, we evaluate the performance of our proposed method under higher unstructured sparsity levels. Specifically, we compare Wanda++ against baselines such as GBLM and the original Wanda, with results summarized in Table 5. As shown, Wanda++ consistently outperforms prior methods across varying sparsity levels. However, we note that unstructured pruning—even at high sparsity—may not yield inference speedups comparable to structured formats such as 2:4 sparsity on modern GPUs. As a result, the 2:4 and 4:8 sparsity settings are of greater practical relevance for deployment.

Methods	0.6	0.7	0.8
GBLM	10.37	54.60	2550.10
Wanda	9.71	76.17	1942.53
Wanda++	9.50	55.52	1586.69

Table 5: Performance comparison of different methods under varying sparsity ratios.

6 Extending Wanda++ to Structured Pruning

While our main focus is on unstructured and semi-structured pruning, we further evaluate the applicability of our method to structured pruning to inspire the future work. Specifically, we conduct experiments on a naive row-wise structured pruning (SP) for each weight matrices in the model, where the pruning score for each row is the average score of all paramters in the row. Prior work has shown that naively adapt Wanda methods to structured pruning (Wanda-SP) tend to degrade sharply under structured settings when applied naively (An et al., 2024). Nonetheless, we adapt our approach (Wanda++-SP) to this setting and compare it against a baseline Wanda-SP method across varying pruning ratios.

As shown in Table 6, Wanda++-SP consistently achieves lower perplexity than Wanda-SP, particularly at higher pruning ratios. These results highlight the robustness and generalizability of our method beyond the unstructured and semi-structured regimes originally explored in this work. We further emphasize that the regional optimization component of Wanda++ is orthogonal to most existing structured pruning methods, such as SliceGPT (Ashkboos et al., 2024) and LLM-Pruner (Ma et al., 2023). In practice, a lightweight regional optimization using only a few hundred calibration samples is sufficient to yield substantial improvements in model performance.

Method	0.1	0.3	0.5
Wanda-SP	9.10	118.72	8804.84
Wanda++-SP	7.84	63.45	5428.97

Table 6: Perplexity of row-wise structured pruning (SP) on LLMs at different sparsity levels. Lower is better.

7 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. Wanda++ defines a region as each decoder block. The method includes a pruner based on regional gradient scores (RGS) and a Regional Optimizer (RO) for in-block, sparsity-aware calibration. By utilizing regional gradients, it outperforms Wanda on various LLaMA models. 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.

We also note that the regional optimization component of Wanda++ is **orthogonal** to most existing LLM pruning approaches, as well as to works exploring dense-to-MoE transformations (Wu et al., 2024). Incorporating regional optimization from Wanda++ provides a strong initialization for the foundation model, which can be further combined with downstream optimization techniques such as LoRA fine-tuning (Hu et al., 2021), Direct Preference Optimization (DPO) (Rafailov et al., 2023), or Group Relative Policy Optimization (GRPO) (Rafailov et al., 2023). An exciting direction for future work is to explore the potential of using Wanda++ as a general tool to enhance model performance following architectural adaptations.

Limitations

While Wanda++ demonstrates significant improvements in mitigating pruning-induced degradation and achieves state-of-the-art results under unstructured, 2:4, and 4:8 sparsity patterns, we have not fully tested Wanda++ for structured pruning, as it is not the main focus of this paper. Although the RO method has been tested with structured pruning approaches like SliceGPT, the full Wanda++ framework is not yet fully compatible with structured pruning. Future work could focus on adapting the Wanda++ method to better support structured pruning scenarios.

Acknowledgment

We thank Denis Filimonov for his insights in developing the initial idea and refining the RGS score.

References

Megha Agarwal. 2023. Llm inference performance engineering: Best practices.

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.

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. *arXiv preprint arXiv:2401.15024*.

Xiao Bai, Xiang Wang, Xianglong Liu, Qiang Liu, Jingkuan Song, Nicu Sebe, and Been Kim. 2021. Explainable deep learning for efficient and robust pattern recognition: A survey of recent developments. *Pattern Recognition*, 120:108102.

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

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. 2024. Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In *Forty-first International Conference on Machine Learning*.

Chao Fang, Wei Sun, Aojun Zhou, and Zhongfeng Wang. 2023. Efficient n: M sparse dnn training using algorithm, architecture, and dataflow co-design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(2):506–519.

Chao Fang, Aojun Zhou, and Zhongfeng Wang. 2022. An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(11):1573–1586.

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. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*.

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. 2024. A framework for few-shot language model evaluation.

Sajjad Ghiasvand, Yifan Yang, Zhiyu Xue, Mahnoosh Alizadeh, Zheng Zhang, and Ramtin Pedarsani. 2024. Communication-efficient and tensorized federated fine-tuning of large language models. *arXiv preprint arXiv:2410.13097*.

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

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

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.

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

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*.

Amazon Artificial General Intelligence. 2024a. The amazon nova family of models: Technical report and model card. *Amazon Technical Reports*.

Amazon Artificial General Intelligence. 2024b. Amazon nova sonic: Technical report and model card. *Amazon Technical Reports*.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

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

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

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

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744.

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.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2022. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. 2024. Compact language models via pruning and knowledge distillation. *arXiv preprint arXiv:2407.14679*.

NVIDIA Developer Blog. 2023. Sparsity in int8 training: Workflow and best practices for tensorrt acceleration. Accessed: 2024-10-04.

Jeff Pool, Abhishek Sawarkar, and Jay Rodge. 2021. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt. *NVIDIA Developer Technical Blog, https://developer. nvidia. com/blog/accelerating-inference-with-sparsityusing-ampere-and-tensorrt*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

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

Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M Alvarez. 2022. Structural pruning via latency-saliency knapsack. *Advances in Neural Information Processing Systems*, 35:12894–12908.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

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*.

Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. 2021. Revisiting locally supervised learning: an alternative to end-to-end training. In *International Conference on Learning Representations*.

Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. 2024. Parameter-efficient sparsity crafting from dense to mixture-of-experts for instruction tuning on general tasks. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 737–749.

Yifan Yang, Jiajun Zhou, Ngai Wong, and Zheng Zhang. 2024. 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)*, pages 3161–3176.

Qiaozhe Zhang, Ruijie Zhang, Jun Sun, and Yingzhuang Liu. 2024. How sparse can we prune a deep network: A fundamental limit perspective. *Advances in Neural Information Processing Systems*, 37:91337–91372.

Zhen Zhang, Yifan Yang, Kai Zhen, Nathan Susanj, Athanasios Mouchtaris, Siegfried Kunzmann, and Zheng Zhang. 2025. Mazo: Masked zeroth-order optimization for multi-task fine-tuning of large language models. *arXiv preprint arXiv:2502.11513*.

Jiajun Zhou, Yifan Yang, Kai Zhen, Ziyue Liu, Yequan Zhao, Ershad Banijamali, Athanasios Mouchtaris, Ngai Wong, and Zheng Zhang. 2025. Quzo: Quantized zeroth-order fine-tuning for large language models. *arXiv preprint arXiv:2502.12346*.

Algorithm 2 PyTorch pseudo-code for computing RGS criterion for a single decoder block

```
# block: DecoderBlock
# inps: Input hidden states of decoder block
# sq_grad: empty gradient dictionary
# alpha: hyperparameter
def wanda_plus_plus_pruner(block: torch.nn.module, inps: List, sq_grad: Dict, alpha: Constant):
    for inp in inps:
        loss = torch.norm(block.forward(inp))
        loss.backward()
        with torch.no_grad(): # Compute aggregate gradient for each weight
            for name, param in block.named_parameters():
                sq_grad[name] += param.grad ** 2
    for key, value in sq_grad.items():
        sq_grad[key] = torch.sqrt(value / inps_len)
    for n, par in layer.named_parameters(): # Compute the pruning scores
        layer_inps = torch.stack(get_signal(inps, n)) # Obtaining and stacking layer-wise inputs
        score = par.weight.abs() * (alpha * sq_grad[n] + layer_inps.norm(p=2, dim=0))
```

A Pytorch pseudo-code for Calculating RGS Criterion

For the pruning process, we first perform the forward and construct the loss function with the L2 norm of the decoder output hidden state. Then, a backward pass is conducted to obtain the gradient for each parameter weight matrices and the scaled stochastic gradients regarding each calibration data sample are aggregated to reduce the noise included with each single data sample. The aggregated gradient magnitude is normalized with the number of calibration data samples and used to calculate the designed pruning score with the weight magnitude given, as given in Eq. (4). Also, we summarize the Pytorch pseudo-code for calculating our RGS criterion here.

B Additional Experimental Results

B.1 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 7 using TensorRT-LLM-0.9.0 with the Sparse Tensor Core support (NVIDIA Developer Blog, 2023). 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 B.3 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		47	11	
	2048		47	10	
	4096		46	10	
4	128	64	45	11	
	1024		47	11	
	2048		45	9	
	4096		43	7	

Table 7: Relative reduction (%) for latency and weight memory from 2:4 sparsity under FP16 format.

B.2 Ablation Study on RGS Scaling Factor

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

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 8: Perplexity with different α values in RGS on LLaMA-3 8B model for 2:4 sparsity.

B.3 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).

Batch Size	Token Length		Latency		Weight Memory
	Input	Output	TTFT	TPOT	
1	128	64	15	15	22
	1024		4	13	
	2048		7	15	
	4096		8	14	
4	128	64	7	16	
	1024		7	14	
	2048		0	13	
	4096		-13	1	

Table 9: Relative reduction (%) for latency and weight memory from 2:4 sparsity under FP8 format.