Weight-Aware Activation Sparsity with Constrained Bayesian Optimization Scheduling for Large Language Models

Ming Wang, Miao Zhang*, Xuebo Liu, Liqiang Nie
Harbin Institute of Technology (Shenzhen)
190110509@stu.hit.edu.cn, {zhangmiao, liuxuebo, nieliqiang}@hit.edu.cn

Abstract

Activation sparsity provides a dynamic, input-dependent alternative to weight pruning for accelerating inference in large language models (LLMs), effectively reducing unnecessary computations and memory accesses during the forward pass. Despite its promise, existing activation sparsification methods suffer from two major limitations: (1) solely relying on activation magnitude for sparsification, ignoring the coupling influence with the corresponding weights, (2) applying uniform sparsity rates across all blocks without considering block-wise sparsity sensitivity. To address these issues, this paper proposes a novel training-free weight-aware activation sparsity framework, called **WAS**. Firstly, with analyzing the coupling relationship between weight and activation, we introduce a weight-aware scoring method to measure the activation importance in sparsification. Then, a novel constrained Bayesian optimization algorithm is further devised to set a suitable sparsity ratio for all blocks based on the sparsity sensitivity. Finally, we implement a custom GPU sparsity kernel to support the resulting sparsity patterns for wall-clock decoding speed-ups. Our **WAS** achieves competitive performance at 60% model-level sparsity and significantly outperforms prior methods at higher sparsity levels, achieving up to 1.68× inference speed-up—at no retraining or weight update. Codes are available at https://github.com/HITSZ-Miao-Group/WAS.

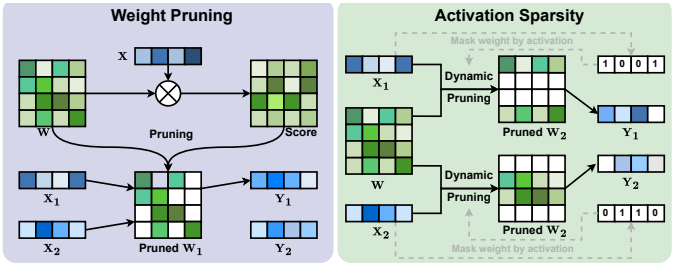
1 Introduction

Large language models (LLMs) have demonstrated remarkable performance across a wide range of natural language processing tasks (Brown et al., 2020). However, the enormous computational and memory demands associated with deploying these models, which often contain billions of parameters, pose significant challenges for real-world applications. Model compression has emerged as a promising approach to tackle this issue, with quantization (Frantar et al., 2022; Shao et al., 2023; Zhao et al., 2024) and pruning (Kim et al., 2024; Ashkboos et al., 2024; Men et al., 2024) being two widely adopted techniques. Quantization accelerates inference by representing weights and activations using lower bits, while pruning removes redundant parts of the model to reduce computational cost. As illustrated in Figure 1a, activation sparsity (Raihan and Aamodt, 2020; Grimaldi et al., 2023; Chen et al., 2023) offers a more dynamic and input-dependent alternative compared to weight pruning. Instead of statically pruning the model prior to inference, activation sparsity identifies and skips unimportant weights at inference time based on the sparsity of the intermediate activations. This allows the model to selectively omit computations based on the input, making activation sparsity a more flexible and adaptive approach for inference acceleration.

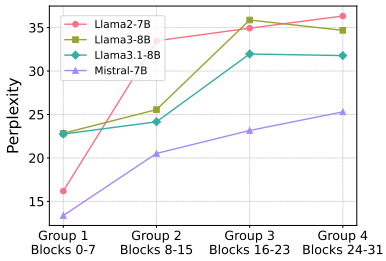
Previous work on activation sparsity has primarily focused on older ReLU-based large language models such as OPT (Zhang et al., 2022), where the inherent sparsity of the ReLU function naturally leads to highly sparse activations. This property can be effectively leveraged to accelerate inference. However, modern LLMs have largely transitioned to GLU-based activation functions (Shazeer, 2020), such as SwiGLU (Chowdhery et al., 2023), which no longer exhibit this natural sparsity. To reintroduce sparsity into these models, some approaches (Mirzadeh et al., 2023; Song et al., 2024) have replaced the activation functions with ReLU and performed continued pre-training, incurring significant computational costs. More recently, several studies (Liu et al., 2024; Lee et al., 2024) have observed that even without additional pre-training, modern models can still exhibit activation patterns that resemble sparsity, suggesting the potential for training-free sparsification techniques. However, these methods face two major limitations. Firstly,

* Corresponding author

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 1087–1099
November 4-9, 2025



(a) Differences between weight pruning and activation sparsity



(b) Sensitivity to sparsity across transformer blocks

Figure 1: (a) Comparison of weight pruning and activation sparsity, where the latter one is dynamic and input-dependent. (b) Sensitivity analysis of transformer blocks under different sparsity configurations. We divide the transformer into four contiguous block groups (8 blocks each) and evaluate perplexity when sparsifying one group to 60% while others remain at 80%.

they determine the sparsification threshold solely based on the magnitude of the activations, without accounting for the role of the corresponding weights in the sparsification process. Secondly, they apply a uniform sparsity rate across all blocks, neglecting the fact that blocks at different depths in the model contribute differently to its overall performance.

To address the two aforementioned aspects, we propose a novel activation sparsity framework, termed as **WAS**. Firstly, we analyze the coupling effect of weights and activations on the sparsification error and reveal that both activation values and weights play equally important roles. Based on this insight, we propose a new thresholding strategy that jointly considers the magnitudes of both activations and weights to determine whether a value should be sparsified. In addition, as shown in Figure 1b, we observe that blocks at different depths exhibit varying sensitivities to sparsity and we introduce a constrained Bayesian optimization approach to assign block-wise sparsity rates, instead of applying a uniform sparsity rate across all blocks. Finally, we design an optimized GPU kernel that directly integrates weight-aware sparsity logic into the computation pipeline, enabling efficient, structure-sensitive inference acceleration with minimal overhead.

Our key contributions can be summarized as:

- A novel activation sparsity framework called WAS is proposed, which incorporates weight information into the thresholding decision by analyzing the mathematical formulation of the error introduced during sparsification.
- We introduce a constrained Bayesian optimization algorithm that dynamically allocates sparsity rates across different blocks, motivated by empirical observation (Figure 1b). Without additional training, our method achieves up to $1.52\times$ inference acceleration with acceptable performance degradation in 60% sparsity level, and significantly outperforms the baselines in 75% sparsity.
- Furthermore, we develop a custom GPU kernel that integrates weight information into the computation pipeline, supports non-uniform sparsity and enables structure-specific thresholding across blocks. Our kernel is compatible with general Transformer architecture and supports practical inference deployment.

2 Related Work

Activation sparsity can be broadly categorized into training-based and training-free approaches. We briefly review both lines of research below.

2.1 Training-based Activation Sparsity

Several training-based methods have been proposed to induce activation sparsity in neural networks. (Kurtz et al., 2020) first identified the natural sparsity phenomenon caused by ReLU activations in CNNs, and further enhanced this sparsity through regularization-based training, ultimately enabling faster convolution by exploiting the sparse structure of activations. DejaVu (Liu et al., 2023) observed that, due to residual connections in deep neural networks, token embeddings across adjacent layers change slowly, and neurons with larger activation norms dominate the forward computation. To take advantage of this, DejaVu trains a lightweight two-layer MLP to predict neurons with smaller activation norms and selectively prunes

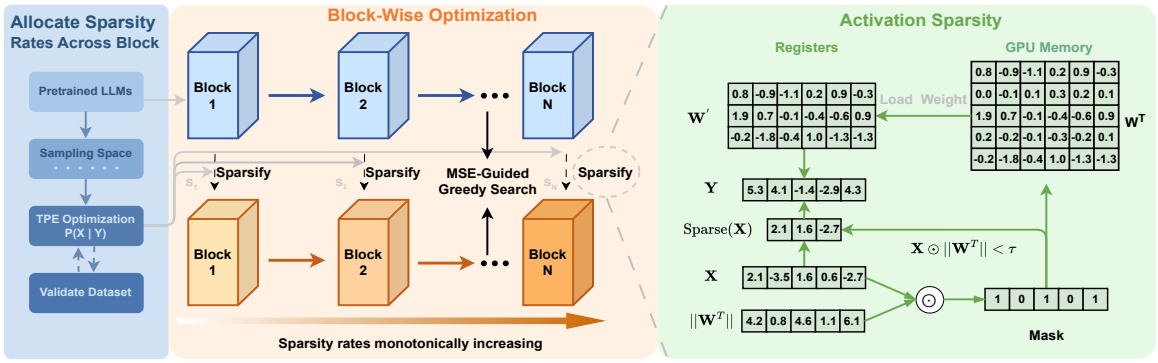


Figure 2: An overview of our WAS. WAS assigns inter-block sparsity using constrained Bayesian optimization, and intra-block sparsity by greedy search, while incorporating weight information into the sparsification process.

them, achieving significant inference acceleration. While earlier work largely focused on ReLU-based models, ReLUfication (Mirzadeh et al., 2023) and ProSparse (Song et al., 2024) aim to recover similar sparsity in modern LLMs by replacing the original activation functions with ReLU and performing continued pre-training to restore ReLU-like behavior. (Zhang et al., 2024) further explores the design space of activation functions by retraining small models with ReLU2 activation, which has been shown to induce even higher activation sparsity than standard ReLU.

While effective, these approaches typically require additional training or modification to the model architecture, which limits their applicability in deployment scenarios where retraining is not feasible or desirable.

2.2 Training-free Activation Sparsity

To circumvent the limitations of training-based methods, several studies have proposed training-free activation sparsity techniques. By applying a top-k sparsification to the MLP layers in T5 (Raffel et al., 2020a) and ViT (Dosovitskiy et al., 2020), (Li et al., 2022) achieves notable performance improvements. Griffin (Dong et al., 2024) identifies a flocking effect, where different tokens within the same sentence tend to activate similar neurons, while different sentences activate distinct sets of neurons. Based on this observation, they use the activations from prompt tokens to predict which neurons will be activated in the remainder of the sentence, thereby enabling inference acceleration. Similarly, (Ma et al., 2024) leverages the activations from the prefill phase to generate sparsity masks used during the generation phase, improving efficiency without modifying the model. CATS (Lee et al., 2024) proposes a relative magnitude-based sparsification technique by analyzing the distribution of gate activations in MLP layers on a calibration set to determine cutoff thresholds. This enables selective computation of W_{up} and W_{down}, reducing the cost of MLP inference. However, CATS only sparsify gate outputs, limiting its impact to a subset of the MLP computation. In contrast, TEAL (Liu et al., 2024) applies sparsification to the inputs of all components in the model, achieving model-level activation sparsity and significantly improving overall computational efficiency.

Nevertheless, most of these methods are still in their infancy, relying on straightforward heuristics without deeper modeling or optimization.

3 Method

In this section, we provide a detailed overview of WAS, as illustrated in Figure 2. We first introduce the preliminaries and motivation in Section 3.1. Next, we introduce our sparsification method in Section 3.2, describe the sparsity allocation strategies across and within transformer blocks in Sections 3.3 and 3.4, and detail our custom GPU kernel for efficient inference acceleration in Section 3.5.

3.1 Preliminaries and Motivation

The core idea of activation sparsity lies in identifying and zeroing out the activations that have minimal impact on model performance, and inference acceleration can be achieved by skipping the loading of weight columns that would be multiplied with sparsified activations. This is typically achieved through threshold-based methods, which

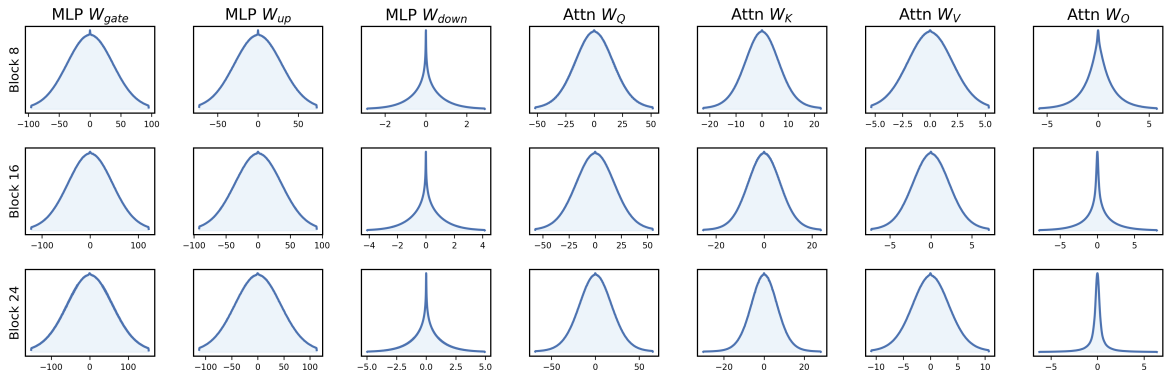


Figure 3: Distributions of activations (after being scaled by the corresponding weight norm) in Blocks 8, 16, and 24 of LLaMA-3-8B exhibit symmetric and unimodal around zero. This property enables reliable threshold selection for a given sparsity level.

can be formally expressed as:

$$sp(x_i) = \begin{cases} 0 & \text{if } |x_i| < \tau \\ x_i & \text{otherwise} \end{cases}, \tag{1}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_m]$ is the input. $sp(\cdot)$ is the sparse operation, and τ denotes the cutoff threshold. The error of the sparsification process can be expressed as:

$$\mathcal{L} = ||\mathbf{x}\mathbf{W}^T - \mathbf{x}^{'}\mathbf{W}^T||, \tag{2}$$

$$\mathbf{x}^{'} = sp(\mathbf{x}), \tag{3}$$

where $\mathbf{W} \in \mathbb{R}^{n \times m}$ is the model weight. According to Equation 2, we observe that weights also play a significant role in the process of model sparsity. An intuitive idea is to incorporate weights into the decision-making process of whether an activation value should be set to zero.

Secondly, Transformer-based LLMs exhibit varying degrees of sensitivity to sparsity across different blocks. Formally, for a LLM with L blocks, the model sparsity is defined as a weighted average across all blocks in the model:

$$S = \frac{1}{L}\sum_{l=1}^{L} p_l \tag{4}$$

where p_l is the sparsity ratio of the l-th block. Typically, information flows hierarchically from shallow to deep blocks, where shallow blocks primarily capture fundamental token-level semantics, serving as the foundation for subsequent high-level feature extraction and propagation. Intuitively, these shallower blocks, being critical to the overall architecture, are more sensitive to sparsity and thus require lower sparsity levels. Previous weight pruning approaches have addressed similar considerations: LLM_Pruner (Ma et al., 2023) selectively prunes intermediate transformer blocks, while OWL (Yin et al., 2023) dynamically adjusts sparsity ratios based on feature outliers. Motivated by these insights, we explore the necessity of assigning block-wise activation sparsity ratios, instead of applying a uniform sparsity rate across all blocks as in prior work (Liu et al., 2024).

3.2 Weight-aware Activation Sparsification

Inspired by WANDA (Sun et al., 2023), we integrate weight information into our sparsification framework, with a key distinction being our focus on activation pruning rather than weight pruning. Specifically, by employing the L1 norm, Equation 2 can be extended as follows:

$$\begin{aligned} \mathcal{L} &= \sum_{j=1}^{n} |\sum_{i=1}^{m} (x_i - x_i^{'}) w_{i,j}| \\ &\leq \sum_{i=1}^{m} |x_i - x_i^{'}| \times \sum_{j=1}^{n} |w_{i,j}| \\ &= \sum_{i=1}^{m} |x_i - x_i^{'}| \times \|\mathbf{W}_{i,:}^T\|_1, \end{aligned} \tag{5}$$

where $w_{i,j}$ denote the element at the i-th row and j-th column of $\mathbf{W}^T$. Assume we select one of the two activation values x_p and x_q, which are close to each other numerically, for sparsification. According to Equation 5, the corresponding sparsification errors are $|x_p| \times \|\mathbf{W}_{p,:}^T\|_1$ and $|x_q| \times \|\mathbf{W}_{q,:}^T\|_1$. To minimize the sparsification loss, the decision should be determined by the relative magnitudes of $\|\mathbf{W}_{p,:}^T\|_1$ and $\|\mathbf{W}_{q,:}^T\|_1$. Consequently, the sparsification process depends not only on activation but

also incorporates weight norm, as mathematically formulated below:

$$\text{mask}(x_i, \mathbf{W}) = \begin{cases} 0, & \text{if } |x_i| \times \|\mathbf{W}_{i,:}^T\|_1 < \tau \\ 1, & \text{otherwise} \end{cases}, \tag{6}$$

$$\mathbf{x}' = \mathbf{x} \odot \text{mask}(\mathbf{x}, \mathbf{W}). \tag{7}$$

Notably, the column-wise L1 norm of weights can be precomputed prior to sparsification as an m-dimensional vector, introducing negligible computational overhead.

In determining the sparsity threshold τ, existing approaches generally fall into two categories: (1) online computation via top-k selection of activation values, which introduces additional runtime overhead, and (2) offline estimation through cumulative distribution function (CDF) modeling of activations using calibration data, which requires precise distribution modeling to guarantee target sparsity rates. Building upon the foundational observation by (Liu et al., 2024) that LLM activations typically follow zero-mean Gaussian or Laplacian distributions, our analysis in Figure 3 demonstrates that this statistical property remains remarkably consistent when activations are modulated by weight information across different layers and model architectures. This statistical regularity enables threshold selection based on the distribution of activations scaled by weight norms. Formally, given a target sparsity ratio $p \in [0, 1]$, the threshold τ_p is determined by:

$$\frac{1}{m} \sum_{i=1}^{m} \mathbb{P}\left(|x_i| \times \|\mathbf{W}_{i,:}^T\|_1 \leq \tau_p\right) = p \tag{8}$$

3.3 Inter-Block Sparsity Allocation

Determining appropriate sparsity ratios across blocks is critical for optimizing overall model performance. As discussed in Section 3.1, transformer blocks exhibit varying sensitivity to sparsity, particularly under high global sparsity. Figure 1b illustrates this positional sensitivity: reducing sparsity in shallow blocks yields substantial performance gains, while reducing sparsity in deeper blocks has limited effect. This trend reveals a monotonic decrease in sparsity sensitivity from shallow to deep layers. Motivated by this observation, we propose a constrained Bayesian optimization framework for inter-block sparsity allocation.

Specifically, we utilize a variant of Bayesian optimization, the Tree-structured Parzen Estimator (TPE) (Bergstra et al., 2011), to search for the optimal sparsity ratios. Traditional Bayesian optimization typically relies on Gaussian processes to model the objective function which struggles to scale in high-dimensional spaces. TPE employs kernel density estimation for modeling, which not only reduces computational complexity but also exhibits better performance in high-dimensional optimization tasks.

Algorithm 1 Constrained Bayesian Optimization

Input:
Model $\mathcal{M}$ with L blocks, validation dataset $\mathcal{D}_{\text{val}}$, target sparsity level r, iteration counts N_{trials}, greedy lookup table T with L blocks

Output:
Optimal sparsity ratios $s \in \mathbb{R}^L$ for all blocks

Initialize Optuna study with TPESampler
for $t \leftarrow 1$ to N_{trials} **do**
  $s_1 \leftarrow \text{TPESampler}(r - 0.05, r)$ ▷ Initialize sparsity for the first block
  for $i \leftarrow 2$ to L **do**
    $s_i \leftarrow \text{TPESampler}(s_{i-1}, r + 0.05)$
  end for
  if $|\frac{1}{L}\sum_{i=1}^{L} s_i - r| > \epsilon$ **then**
    continue ▷ Resample
  end if
  $\text{ApplySparsity}(\mathcal{M}, \{s_i\}_{i=1}^{L}, T)$
  $\text{ppl} \leftarrow \text{Evaluate}(M, \mathcal{D}_{\text{val}})$
  $\text{UpdateResult}(\{s_i\}_{i=1}^{L}, \text{ppl})$
end for
return s

As detailed in Algorithm 1, for each trial, we uniformly sample the sparsity ratio within a narrow range for the first block, while imposing a constraint on the subsequent blocks, using the sampled value from the previous block as the lower bound for the current block. This constraint not only aligns with our empirical observations but also effectively reduces the search space. We will discuss the impact of applying constraints on the final results in detail in Section 4.5. For each trial, we use the model's perplexity on the WikiText2 (Merity et al., 2016) as the evaluation metric.

3.4 Intra-Block Sparsity Allocation

Given the overall sparsity ratio of a block, we further optimize the sparsity ratios for its internal components (e.g., Q/K/V matrices) to achieve the best performance. Due to the non-differentiable nature of threshold-based methods, we initially employed straight-through gradient estimator (STE) (Bengio et al., 2013) to optimize component-wise sparsity allocation within each block. However, we identi-

Dataset	Sparsity	Methods / Models	LLaMA-2			LLaMA-3		LLaMA-3.1		Mistral
			7B	13B	70B	8B	70B	8B	70B	7B
WikiText2	0%	Baseline	5.47	4.88	3.32	6.14	2.85	6.27	2.83	3.12
	40%	CATS	46.87	48.63	54.22	502.19	71.84	162.33	66.95	4.6e4
		WANDA	6.07	5.38	3.74	7.49	4.38	7.59	4.44	5.68
		TEAL	5.74	5.02	3.47	6.60	3.48	6.61	3.41	5.40
		WAS	**5.68**	**4.99**	**3.47**	**6.52**	**3.39**	**6.61**	**3.38**	**5.40**
	60%	WANDA	10.84	8.49	5.27	26.40	9.54	25.54	8.18	11.44
		TEAL	6.80	5.66	4.09	10.04	5.70	8.21	5.74	6.10
		WAS	**6.56**	**5.54**	**4.09**	**8.30**	**5.60**	**8.14**	**5.59**	**6.09**
	75%	TEAL	42.15	12.17	6.37	87.48	10.42	27.73	9.49	13.02
		WAS	**12.76**	**8.16**	**6.19**	**28.33**	**9.91**	**26.61**	**9.27**	**10.34**
C4	0%	Baseline	7.26	6.73	5.71	9.44	7.16	9.53	7.10	3.12
	40%	CATS	43.96	51.10	55.73	325.24	102.20	136.29	95.64	4.4e4
		WANDA	8.12	7.40	6.03	11.91	8.32	12.00	8.27	8.91
		TEAL	7.55	6.90	5.82	10.45	7.56	10.29	7.47	8.57
		WAS	**7.52**	**6.88**	**5.82**	**10.18**	**7.52**	**10.26**	**7.44**	**8.57**
	60%	WANDA	14.09	11.79	7.81	39.71	16.03	39.20	12.98	16.25
		TEAL	9.03	7.80	6.34	15.93	9.47	13.17	9.43	9.50
		WAS	**8.78**	**7.68**	**6.34**	**13.45**	**9.35**	**13.07**	**9.25**	**9.47**
	75%	TEAL	51.34	16.38	8.54	86.68	15.73	47.99	14.49	17.65
		WAS	**15.56**	**11.10**	**8.38**	**43.54**	**14.93**	**41.29**	**13.99**	**14.03**

Table 1: Comparative analysis of perplexity performance(**lower is better**) across LLaMA and Mistral model families at 40%, 60% and 75% sparsity levels. We omit WANDA 75% as it degenerates seriously.

fied critical gradient backpropagation failures to the sparsity parameters. While alternative approaches like Bayesian optimization were explored, they exhibited prohibitively slow convergence and susceptibility to local optima. We ultimately follow (Liu et al., 2024) to determine the optimal sparsity configuration for intra-block components by greedy search. The detailed procedure is provided in Algorithm 2 at Appendix D.

3.5 Hardware Acceleration

We implement a custom Triton kernel (Tillet et al., 2019) specifically tailored to the sparsity patterns produced by WAS, aiming to achieve practical inference acceleration. Building upon DejaVu (Liu et al., 2023), our kernel integrates the mask generation process for activation sparsification directly into the computation pipeline. It performs FP16 accumulation along the external SplitK dimension to reduce memory overhead, and employs a cache-aware scheduling strategy that prioritizes activations reused across thread blocks while deprioritizing block-specific weights. On top of this foundation, we introduce several key enhancements: (1) We incorporate weight norms, as described in Section 3.2, directly into the mask generation process; (2) We extend the kernel to support non-uniform sparsity, allowing different sparsity rates across transformer blocks; (3) We generalize the thresholding mechanism to support distinct thresholds for different substructures (e.g., Q, K, V), enabling more fine-grained mask generation under varying sparsity levels.

4 Experiments

4.1 Experimental Settings

Models and Datasets. We conduct comprehensive evaluations of WAS on several leading open-source LLMs, including the META's LLaMA-2 (Touvron et al., 2023), LLaMA-3, LLaMA-3.1 (Grattafiori et al., 2024) and Mistral (Jiang et al., 2023) families. We assess the generative capabilities of the sparsified models on the WikiText2 (Merity et al., 2016) and C4 (Raffel et al., 2020b) datasets. For reasoning performance, we evaluate all models on five zero-shot tasks, including ARC-Easy, ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA(Bisk et al., 2020), and Winogrande (Sakaguchi et al., 2021). Furthermore, we conduct evaluations on the 5-shot MMLU (Hendrycks et al., 2020) and GSM8K (Cobbe et al., 2021) tasks. All evaluations are performed using the lm-evaluation-harness (Gao et al., 2024) framework.

Baseline and Implement Details. Given the relatively limited research on training-free model sparsification, we primarily compare our method

Models	Sparsity	Methods	ARC-C	ARC-E	HellaSwag	PIQA	Winogrande	GSM8K	MMLU	AVG
LLaMA-2-7B	0%	Baseline	43.43	76.35	57.14	78.07	69.14	13.57	45.87	54.80
	60%	WANDA	30.80	64.94	43.51	71.49	65.75	2.43	28.07	43.86
		TEAL	38.73	72.60	52.63	75.24	63.22	6.14	36.12	49.24
		WAS	**39.59**	**73.11**	**54.42**	**77.20**	**66.54**	**7.43**	**38.93**	**51.03**
	75%	TEAL	27.05	57.79	35.22	68.17	54.78	0.0	26.19	38.46
		WAS	**32.85**	**63.09**	**44.86**	**70.35**	**60.62**	**1.36**	**28.14**	**43.04**
LLaMA-2-13B	0%	Baseline	48.46	79.42	60.03	79.16	72.22	23.05	55.20	59.65
	60%	WANDA	37.29	69.23	48.30	75.08	68.90	4.17	35.42	48.34
		TEAL	46.08	77.19	57.83	77.26	67.01	15.31	47.79	55.50
		WAS	**47.35**	**78.24**	**59.11**	**78.24**	**69.38**	**16.83**	**49.29**	**56.92**
	75%	TEAL	32.59	67.30	42.66	71.27	59.75	0.99	26.54	43.01
		WAS	**39.08**	**71.17**	**52.23**	**73.56**	**62.67**	**4.25**	**38.04**	**48.71**
LLaMA-3-8B	0%	Baseline	50.43	80.09	60.17	79.71	72.85	50.04	65.34	65.52
	60%	WANDA	26.71	59.51	36.38	67.08	58.80	1.97	27.92	39.77
		TEAL	38.91	71.34	52.45	75.63	64.17	11.83	47.56	51.70
		WAS	**43.69**	**74.33**	**54.07**	**75.95**	**67.48**	**21.61**	**53.81**	**55.85**
	75%	TEAL	24.66	45.16	36.16	64.25	54.54	0.0	23.98	35.54
		WAS	**30.80**	**58.92**	**39.83**	**67.41**	**59.43**	**1.44**	**27.25**	**40.73**
LLaMA-3.1-8B	0%	Baseline	51.37	81.44	60.04	80.25	73.40	49.43	65.20	65.88
	60%	WANDA	25.43	57.83	36.87	67.90	58.56	2.20	29.16	39.71
		TEAL	41.89	**74.92**	53.95	77.26	66.14	22.44	53.31	55.70
		WAS	**43.17**	74.71	**54.55**	**77.75**	**68.59**	**23.43**	**55.00**	**56.74**
	75%	TEAL	29.01	**61.66**	37.84	68.93	55.88	1.36	27.39	40.30
		WAS	**33.11**	60.10	**40.97**	**69.48**	**58.41**	**2.43**	**28.08**	**41.80**
Mistral-7B	0%	Baseline	50.34	80.85	61.20	80.58	73.80	38.44	62.63	63.98
	60%	WANDA	32.25	66.96	44.33	71.76	65.27	4.55	38.56	46.24
		TEAL	**46.50**	**78.20**	58.97	**78.89**	69.61	25.93	55.65	59.11
		WAS	46.33	77.95	**59.14**	78.56	**70.32**	**26.54**	**56.49**	**59.33**
	75%	TEAL	36.18	**71.72**	49.05	74.54	62.90	1.82	33.16	47.05
		WAS	**39.85**	71.55	**51.97**	**74.81**	**64.96**	**4.02**	**39.34**	**49.50**

Table 2: Comparative analysis of zero-shot and few-shot performance(**higher is better**) across LLaMA and Mistral model families at 60% and 75% sparsity levels.

with TEAL, the current state-of-the-art in this category. We additionally include CATS as a baseline, which achieves up to 40% sparsity. To broaden the comparison, we also evaluate against WANDA. We randomly select ten 2,048-token sequences from the Alpaca (Taori et al., 2023) training set to profile activation value distributions for threshold determination as detailed in Equation 8. The intra-block sparsity ratios were optimized through a greedy algorithm minimizing MSE loss between original and sparsified models, while the inter-block sparsity allocation was systematically determined via 50 trials of TPE-based Bayesian optimization using WikiText2 perplexity as the optimization objective.

4.2 Experiments on Language Generation Tasks

The fundamental competency of large language models resides in their generative performance. To evaluate the generative capability of the sparsified models, we conducted comprehensive perplexity experiments across different sparsity levels, primarily focusing on 40%, 60% and 75% sparsity.

As demonstrated in Table 1, CATS suffers from significant performance degradation at 40% sparsity. This is because it only sparsifies the output of the W_{gate} layer in the MLP, requiring an extremely high sparsity (up to 90%) within the MLP itself to achieve 40% overall sparsity. In contrast, at both 40% and 60% sparsity levels, WAS achieves comparable or superior performance to TEAL on both WikiText2 and C4 tasks. The results also show that both WAS and TEAL outperform WANDA, demonstrating the superiority of dynamic activation sparsity over static weight pruning. Most notably, under an extreme 75% sparsity level, our method achieves significant improvements, reducing perplexity by **29.39** and **35.78** on LLaMA-2-7B for WikiText2 and C4, respectively. These results underscore the effectiveness of weight-aware sparsification and constrained bayesian optimization in determining optimal sparsity ratios for different blocks, particularly in high-sparsity settings.

4.3 Experiments on Reasoning Benchmarks

While perplexity reflects generative capability, reasoning and knowledge tasks offer a complementary lens into model understanding. Zero- and few-shot evaluations assess whether sparsified models retain knowledge and in-context learning ability. We evaluate model performance under 60% and 75% sparsity to examine this capability. The results under the 40% sparsity setting can be found in Appendix C.

As evidenced in Table 2, both WAS and TEAL consistently outperform WANDA across all evaluated models at 60% sparsity in reasoning tasks, with WAS achieving the best results. Most notably, under the challenging 75% sparsity condition, WAS achieves substantial improvements ranging from **1.50%** to **5.70%** absolute gains over TEAL. These results not only validate the efficacy of WAS but also establish new state-of-the-art performance in the training-free sparsification domain.

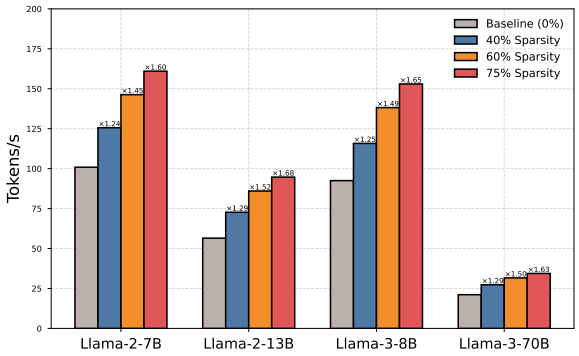


Figure 4: End-to-end single-batch inference speed (tokens per second).

4.4 Experiments on Hardware Acceleration

We benchmark the end-to-end, single-batch decoding speed of WAS on the A800 GPU. Following the standard inference benchmarking setup in (PyTorch, 2024), we report the averaged speedup results over five input sentences. Our experiments primarily focus on LLaMA-2 (7B, 13B) and LLaMA-3 (8B, 70B) models across 40%, 60%, and 75% sparsity levels, with Tensor Parallelism (TP2) enabled for LLaMA-3-70B. As shown in Figure 4, WAS achieves up to $1.52\times$ and $1.68\times$ speedups at 60% and 75% sparsity, respectively. These results demonstrate the practical effectiveness of WAS in accelerating LLM inference.

Greedy	Weight Norm	TPE	Wiki-2	C4
✓			42.15	51.34
✓		✓	21.13	24.80
✓	✓		19.21	21.24
✓	✓	✓	**12.76**	**15.56**

Table 3: Ablation results of LLaMA-2-7B at 75% sparsity levels.

Model	Method	Wiki-2	C4
LLaMA-2-7B	w/o constraint	13.79	17.28
	constraint	**12.76**	**15.56**
LLaMA-2-13B	w/o constraint	8.57	11.60
	constraint	**8.16**	**11.10**

Table 4: Results with and without constrained optimization at 75% sparsity levels for LLaMA-2 (7B, 13B).

4.5 Ablation Experiments

To comprehensively validate our proposed Weight-Aware Activation Sparsity and the constrained TPE optimization for inter-block sparsity allocation, we conduct ablation studies on LLaMA2-7B at 75% sparsity levels and results are presented in Table 3. The first row of the table replicates TEAL's approach, employing a uniform sparsity distribution with greedy intra-block optimization. The results in the second and third rows demonstrate the benefits of our proposed enhancements: using constrained TPE for inter-block sparsity alone reduces perplexity by 21.02 and 26.54 on WikiText2 and C4, respectively, while applying WAS under uniform sparsity achieves reductions of 22.94 and 30.10. These findings highlight the individual advantages of each point. Notably, the combined approach, integrating both techniques, yields the most substantial improvement, underscoring the necessity of hierarchical sparsity allocation and reaffirming the importance of weight information in the sparsification process.

Furthermore, we conducted experiments to validate the effectiveness of our constrained TPE approach. As shown in Table 4, at 75% sparsity level, we compared unconstrained TPE (allowing arbitrary sparsity distribution across blocks) with our constrained version on both LLaMA-2-7B and LLaMA-2-13B. While unconstrained TPE theoretically offers greater optimization potential, empirical results demonstrate our constrained method achieves superior performance. This phenomenon may stem from the inherent limitations of Bayesian optimization in high-dimensional search spaces,

where the algorithm becomes increasingly prone to convergence at local optima rather than the global optimum as the dimensionality expands. Crucially, the unconstrained TPE significantly slows convergence due to difficulty maintaining target sparsity - LLaMA-2-13B required over 100 trials versus just 50 for the constrained version. This improvement suggests our constraints effectively reduce optimization dimensionality, addressing Bayesian optimization's instability in high-dimensional spaces while simultaneously accelerating convergence and enhancing final performance.

5 Conclusion

In this paper, we propose WAS, a novel training-free activation sparsity framework. WAS introduces a new sparsification strategy that incorporates weight importance into the thresholding decision, addressing a key limitation of prior magnitude-only approaches. To further improve sparsity allocation, we employ a constrained Bayesian optimization method that assigns block-wise sparsity rates in a monotonically increasing fashion. Finally, we design a custom GPU kernel to realize end-to-end inference acceleration under the WAS sparsity patterns. Experimental results demonstrate that WAS achieves performance comparable to strong baselines at sparsity levels below 60%, and establishes new state-of-the-art results under the more challenging 75% sparsity setting.

Limitations

While WAS incorporates weight information into the activation sparsification process and leverages constrained Bayesian optimization to assign block-wise sparsity rates, our current design primarily targets LLMs whose activation distributions are symmetric and unimodal around zero. In future work, we plan to extend our method to a broader range of model architectures. Additionally, due to limited hardware resources, our focus has been on training-free scenarios. Incorporating training into the WAS framework remains a promising direction for further improving performance.

Ethics Statement

This paper presents a training-free activation sparsity framework designed to improve the inference efficiency of LLMs, with the broader aim of enabling more accessible and sustainable deployment of LLMs in real-world settings. We are aware of ongoing ethical discussions surrounding LLMs, particularly regarding fairness, bias amplification, and environmental impact. Our proposed method focuses solely on sparsifying activation patterns during inference without modifying model weights or introducing new training data. As such, we believe that WAS does not exacerbate existing biases nor introduce additional ethical risks.

Acknowledgments

Miao Zhang was partially sponsored by the National Natural Science Foundation of China under Grant 62306084 and U23B2051, Shenzhen College Stability Support Plan under Grant GXWD20231128102243003, and Shenzhen Science and Technology Program under Grant ZDSYS20230626091203008 and KJZD20230923115113026.

References

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*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Xuanyao Chen, Zhijian Liu, Haotian Tang, Li Yi, Hang Zhao, and Song Han. 2023. Sparsevit: Revisiting activation sparsity for efficient high-resolution vision transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2061–2070.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul

Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Harry Dong, Beidi Chen, and Yuejie Chi. 2024. Prompt-prompted adaptive structured pruning for efficient llm generation. *arXiv preprint arXiv:2404.01365*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and 1 others. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

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

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, and 5 others. 2024. A framework for few-shot language model evaluation.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Matteo Grimaldi, Darshan C Ganji, Ivan Lazarevich, and Sudhakar Sah. 2023. Accelerating deep neural networks via semi-structured activation sparsity. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1179–1188.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lelio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothee Lacroix, and William El Sayed. 2023. Mistral 7b.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*, 11.

Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. 2020. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning*, pages 5533–5543. PMLR.

Donghyun Lee, Je-Yong Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. 2024. Cats: Contextually-aware thresholding for sparsity in large language models. *arXiv preprint arXiv:2404.08763*.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, and 1 others. 2022. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. *arXiv preprint arXiv:2210.06313*.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. 2024. Training-free activation sparsity in large language models. *Preprint*, arXiv:2408.14690.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and 1 others. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

Chi Ma, Mincong Huang, Ying Zhang, Chao Wang, Yujie Wang, Lei Yu, Chuan Liu, and Wei Lin. 2024. First activations matter: Training-free methods for dynamic activation in large language models. *arXiv preprint arXiv:2408.11393*.

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.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

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

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. Relu strikes back: Exploiting activation

sparsity in large language models. *arXiv preprint arXiv:2310.04564*.

Team PyTorch. 2024. Accelerating generative ai with pytorch ii: Gpt, fast. https://pytorch.org/blog/accelerating-generative-ai-2/. Accessed: 2024-04-29.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020a. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020b. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Md Aamir Raihan and Tor Aamodt. 2020. Sparse weight activation training. *Advances in Neural Information Processing Systems*, 33:15625–15638.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and 1 others. 2024. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. *arXiv preprint arXiv:2402.13516*.

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*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pechenizkiy, Yi Liang, and 1 others. 2023. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. *arXiv preprint arXiv:2310.05175*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, and 1 others. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. 2024. Relu^2 wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209.

Appendix

A Sensitivity Across Transformer Blocks

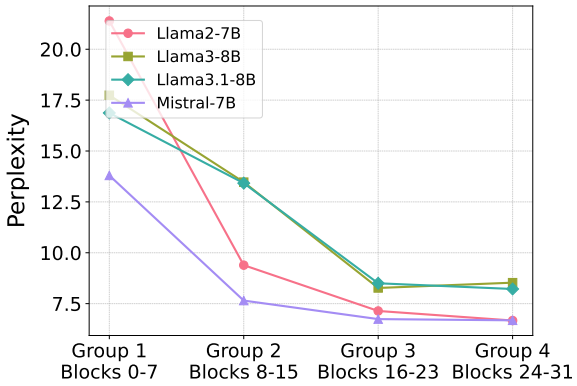


Figure 5: We divide the transformer into four contiguous block groups (8 blocks each) and evaluate perplexity when sparsifying one group to 80% while others remain at 60% .

We partition the transformer into four contiguous block groups (8 blocks each) and evaluate perplexity by setting one group's sparsity to 80% while keeping the remaining groups at 60%. As shown in Figure 5, the trend in perplexity changes is opposite to that observed when sparsifying one group to 60% with others at 80%. Specifically, increasing sparsity in shallow blocks substantially deteriorates performance, whereas increasing sparsity in deeper blocks has a limited effect. This further confirms that transformer blocks at different positions exhibit varying sensitivities to sparsity, with shallower blocks being notably more sensitive.

B Effect of Weight Norm on Activation Distributions

We visualize the activation distributions of additional models after incorporating weight norm. As shown in Figures 6, 7 and 8, we observe that the distribution shape of activations remains largely unchanged after being scaled by weight norms. This may be due to the weight norm applying only a scale transformation without altering the distribution's shape. This stability facilitates threshold determination at a given sparsity level.

C Extended Results at 40% Sparsity

As demonstrated in Table 5, both WAS and TEAL surpass WANDA at 40% sparsity level, reconfirming the superiority of activation sparsity over conventional weight pruning. The marginal improvement of WAS over TEAL at this sparsity level suggests the model's intrinsic low-sparsity property may inherently limit the optimization headroom. It is worth noting that we omit the results of CATS under 40% sparsity, as its performance has completely degraded at this level.

Models	Methods	ARC-C	ARC-E	HellaSwag	PIQA	Winogrande	GSM8K	MMLU	AVG
LLaMA-2-7B	WANDA	42.66	75.59	55.59	77.26	**69.93**	8.34	38.16	52.50
	TEAL	43.00	75.63	56.40	77.69	68.59	12.05	43.79	53.87
	WAS	**43.69**	**75.84**	**56.92**	**77.75**	67.64	**12.36**	**44.50**	**54.10**
LLaMA-2-13B	WANDA	46.59	78.54	59.27	78.45	**72.22**	18.35	50.90	57.76
	TEAL	47.95	**78.62**	60.02	78.45	71.67	20.55	**54.25**	58.79
	WAS	**48.81**	78.58	**60.25**	**79.22**	72.06	**22.52**	54.18	**59.37**
LLaMA-3-8B	WANDA	48.16	75.93	56.53	77.80	**73.09**	31.31	59.03	60.26
	TEAL	48.89	**79.17**	**59.08**	**79.87**	72.30	43.82	**63.03**	**63.74**
	WAS	**49.66**	78.41	59.04	78.89	71.59	**44.20**	62.98	63.54
LLaMA-3.1-8B	WANDA	48.04	77.90	56.81	77.80	**72.45**	32.15	60.52	60.81
	TEAL	**49.83**	79.88	59.00	**79.38**	71.98	43.82	63.59	63.93
	WAS	48.72	**80.60**	**59.16**	79.00	71.35	**45.56**	**63.79**	**64.03**
Mistral-7B	WANDA	46.76	78.24	59.10	79.76	72.53	26.61	58.78	60.25
	TEAL	49.15	79.59	61.20	79.98	72.85	**36.85**	**61.57**	63.03
	WAS	**49.74**	**80.39**	**61.22**	**79.98**	**73.56**	35.41	61.37	**63.10**

Table 5: Comparative analysis of zero-shot and few-shot performance(**higher is better**) across LLaMA and Mistral model families at 40% sparsity levels.

D Intra-Block Sparsity Allocation By Greedy Search

Algorithm 2 Layer-wise Greedy Sparsity Assignment

Input: Block B, sparsity increment α, input $\mathbf{X} \in \mathbb{R}^{B\times seq\times d}$ with n matrices, target sparsity t

for $i = 1$ to n **do**
  $f_i \leftarrow \text{size}(W_i)$
end for
$F \leftarrow \sum_{i=1}^{n} f_i$
$\mathbf{p} \leftarrow \mathbf{0}_n$, $P \leftarrow 0$ ▷ Initialize sparsity settings
$\mathbf{Y}_{\text{target}} \leftarrow B(\mathbf{X})$
while $P < t$ **do**
  for $i = 1$ to n **do**
    $\Delta_i \leftarrow \alpha \cdot \left(\frac{F}{f_i}\right)$
    $\mathbf{p}' \leftarrow \mathbf{p}$
    $p'_i \leftarrow p_i + \Delta_i$
    $\widehat{\mathbf{Y}}_i \leftarrow B(\mathbf{X}, \mathbf{p}')$
    $E_i \leftarrow \|\mathbf{Y}_{\text{target}} - \widehat{\mathbf{Y}}_i\|_2$
  end for
  ▷ Update sparsity for the least-error matrix
  $j \leftarrow \arg\min_i E_i$
  $p_j \leftarrow p_j + \Delta_j$
  $P \leftarrow \sum_{i=1}^{n} (p_i \cdot f_i) / F$ ▷ Record sparsity configuration
end while

As shown in Algorithm 2, our greedy algorithm for intra-block sparsity optimization operates through three key phases: (1) Initialization with component-wise parameter ratios relative to the block, (2) Iterative sparsity increment by step size α, where each step allocates sparsity to the component yielding minimal MSE degradation, and (3) Convergence to the target sparsity with optimal component-level distribution.

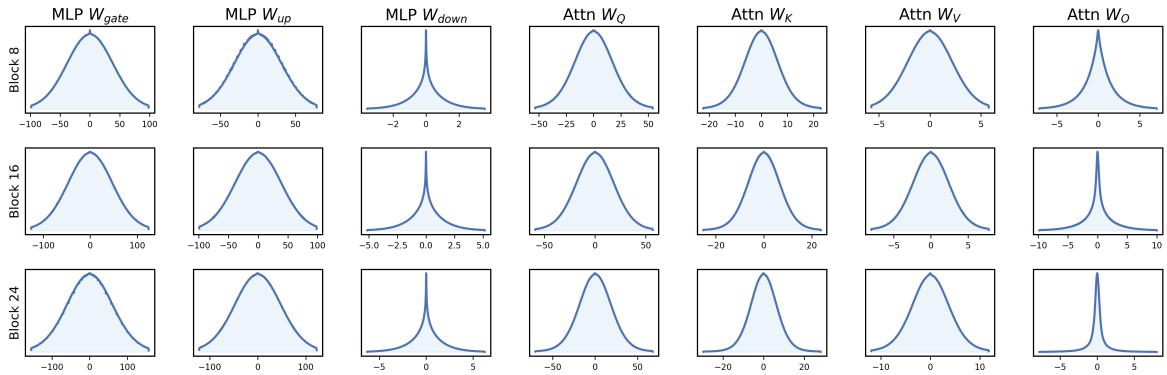


Figure 6: Distributions of activations (after being scaled by the corresponding weight norm) in Blocks 8, 16, and 24 of LLaMA-3.1-8B exhibit symmetric and unimodal around zero.

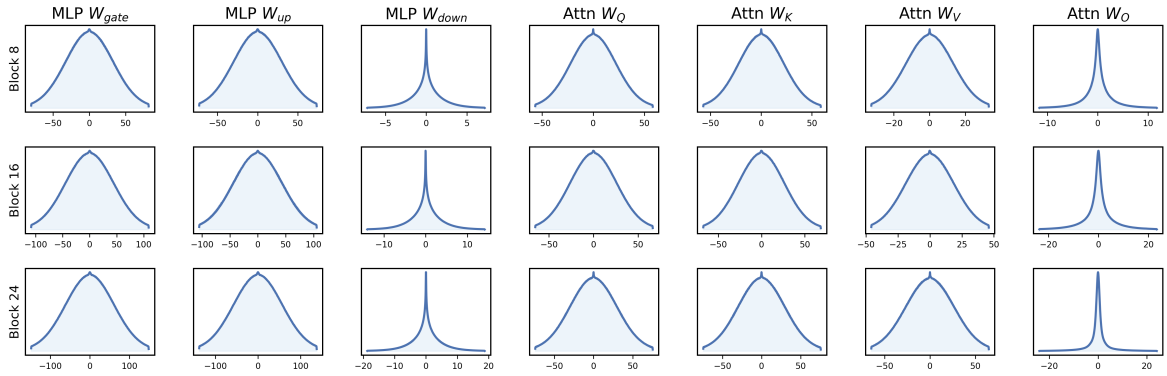


Figure 7: Distributions of activations (after being scaled by the corresponding weight norm) in Blocks 8, 16, and 24 of LLaMA-2-7B exhibit symmetric and unimodal around zero.

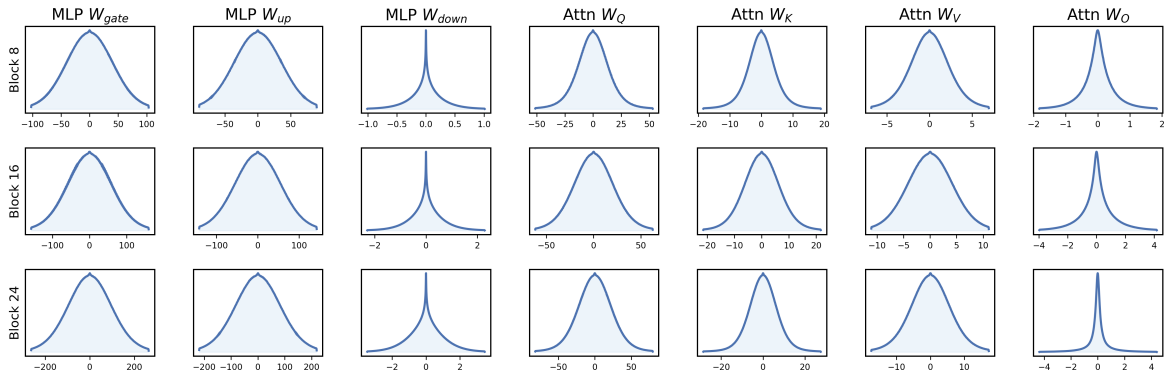


Figure 8: Distributions of activations (after being scaled by the corresponding weight norm) in Blocks 8, 16, and 24 of Mistral-7B exhibit symmetric and unimodal around zero.