Motivating Next-Gen Accelerators with Flexible $N:M$ Activation Sparsity via Benchmarking Lightweight Post-Training Sparsification Approaches

Shirin Alanova♣ [7], Kristina Kazistova♣ [7], Ekaterina Galaeva♣ [7], Alina Kostromina♣ [4,3],
Vladimir Smirnov [6], Dmitry Redko [1], Alexey Dontsov [1],
Maxim Zhelnin [5], Evgeny Burnaev [1,2], Egor Shvetsov [1]
[1] Applied AI [2] Artificial Intelligence Research Institute [3] HSE University [4] Sb AI Lab
[5] MWS AI [6] Yandex [7] Independent Researcher
♣ indicates equal contribution.

Abstract

The demand for efficient large language model inference has spurred interest in sparsification, yet current hardware support remains narrowly focused on 2:4 weight sparsity. In this work, we argue that activation sparsity despite being overlooked in hardware design offers a promising path for dynamic, input-adaptive compression with significant I/O and memory benefits. We present a comprehensive post-training study of N:M activation pruning across four LLMs (Llama2-7B-chat, Llama3.1-8B-Instruct, Qwen2.5-7B-Instruct, Gemma3-4B-Instruct), demonstrating that activation pruning consistently outperforms weight pruning at matched sparsity levels. We evaluate lightweight, plug-and-play error mitigation and selection strategies that require minimal or no calibration data across four sparsity patterns: 2:4, 4:8, 8:16, and 16:32. Among these, 16:32 approaches the performance of unstructured 50% sparsity and is approximately 2.7× better than 2:4, while 8:16 offers an optimal balance of accuracy and practicality. Our results provide evidence that next-generation accelerators should consider native support for N:M activation sparsity and can serve as a strong baseline for the future methods. The code is available here.

1 Introduction

Large Language Models (LLMs) have intensified demand for efficient inference. A common rule of thumb suggests serving speed for a dense N-parameter model scales as $\propto 1/\sqrt{N}$ (Erdil, 2025). Inference is often accelerated via quantization (Frantar et al., 2022; Shvetsov et al., 2024) or sparsification (Frantar and Alistarh, 2023; Maximov et al., 2025), with sparsity reducing both compute and memory I/O.

Weights vs. Activations. Although weight and activation sparsity yield the same theoretical FLOP count, they differ in practice. Weight sparsity enables static compression but can irreversibly degrade model quality, whereas activation sparsity is input-adaptive and better preserves model capacity though it requires an additional pruning step for each input.

Accelerating LLMs with Sparse Activations. Naturally, pruning is most effective when the values to be pruned already have low magnitudes or are zero, which is often the case for some LLMs' intermediate representations (Liu et al., 2023; Li et al., 2022). Activation sparsity was employed to accelerate the decoding stage by enabling faster *sparse vector–dense matrix* multiplications up to 2× speedup using specialized kernels (Song et al., 2024b,a; Liu et al., 2024; Lee et al., 2024). These gains are most pronounced for batch size 1 and diminish as batch size increases (Shrestha et al., 2025). The speedup primarily arises because rows of the dense matrix corresponding to zero elements in the sparse vector can be skipped during computation. Moreover, these methods often require some predictive mechanism to upload required weight indices into memory ahead of time (Liu et al., 2023).

Semi-structured N:M activation sparsity, keeping N non-zeros per block of size M, can extend benefits beyond single vector–matrix products by employing hardware support. Although 2:4 activation sparsity has been explored for training (Haziza et al., 2025; Wang et al., 2024), and post-training weight sparsification (Maximov et al., 2025; Han et al., 2015; Frantar and Alistarh, 2023; Kurtić et al., 2023) post-training N:M activation pruning is largely unexplored because it is not supported on commercial hardware.

Hardware and N:M Sparsity. Current commercial hardware provides native support only for 2:4 structured sparsity in weights, which can deliver ∼1.5–1.7× inference speedup and ∼1.5× energy reduction for models like 7B LLMs, primarily by halving memory bandwidth demand (Fang et al., 2024; Lin et al., 2023). However, this fixed pattern offers limited flexibility: a 2:4 block has only $\binom{4}{2} = 6$ valid configurations. In contrast, larger

patterns like 8:16 provide the same 2× bandwidth reduction but with $\binom{16}{8} = 12{,}870$ possible layouts nearly 10× more than four concatenated 2:4 blocks ($6^4 = 1{,}296$)—at a modest increase in metadata cost (from ≈0.75 to ≈0.875 bits per element). This highlights a significant opportunity for more flexible sparsity. Although the dynamic nature of activation sparsity requires additional computations and is impractical on existing hardware, recent developments point towards its future feasibility. We estimate potential hardware overhead to support dynamic sparsity and feasibility of development in Appendix A.

Focus and Motivation. The primary goal of this work is twofold: to motivate the development of hardware that natively supports semi-structured activation sparsity, and to benchmark existing approaches for inducing such sparsity in activations. Generally, these approaches address two key challenges:

(P1) Selection Strategy. As with weight sparsification, the choice of which activations to retain critically affects model accuracy (Zhelnin et al., 2025).

(P2) Error Mitigation. While post-training fine-tuning can recover lost performance, it is often impractical due to computational cost or risks to safety alignment (Kharinaev et al., 2025). We therefore mainly focus on lightweight, *plug-and-play* methods that require minimal (e.g., WikiText) or zero calibration data.

Our Contributions:

- First, we demonstrate that **activation sparsity outperforms weight sparsity**. At matched sparsity levels, activation pruning consistently yields higher accuracy than weight pruning across four diverse large language models Llama2-7B-chat, Llama3.1-8B-Instruct, Qwen2.5-7B-Instruct and Gemma3-4B-Instruct highlighting activations as a more promising target for future sparse accelerators.

- Second, we benchmark four plug-and-play error mitigation techniques, three of which are applied to semi-structured activation sparsity for the first time, including statistical approaches such as median shift and variance correction. We also evaluate three selection criteria. These methods establish strong, retraining-free baselines that require minimal metadata, aligning well with hardware constraints.

- Finally, we analyze a wide range of structured sparsity patterns and show that larger patterns dramatically improve model fidelity: the 16:32 pattern achieves performance close to unstructured 50% sparsity and retains over two times more accuracy than the conventional 2:4 pattern. Nevertheless, considering implementation trade-offs, we advocate for **8:16** as the optimal balance it offers twice the accuracy retention of 2:4 while remaining highly practical for near-term hardware adoption.

Together, these results provide concrete evidence that expanding hardware support beyond static 2:4 weight sparsity can unlock significant efficiency gains without compromising model quality, thereby motivating the next generation of sparsity-aware architectures.

2 Methods

In the sections below, we formally define the pruning criteria and error mitigation strategies evaluated in this work. A brief summary of these methods is provided in Table 1.

2.1 Preliminaries

For a linear layer with weights $\mathbf{W}$, input activations $\mathbf{X}$, and output $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^\top. \tag{1}$$

We construct a sparsity mask $\mathbf{M}$ using a metric S and threshold t:

$$\mathbf{M}_{ij} = \begin{cases} 1, & S(\mathbf{X}_{ij}) \geq t, \\ 0, & S(\mathbf{X}_{ij}) < t, \end{cases} \tag{2}$$

yielding

$$\mathbf{Y}_p = (\mathbf{X} \odot \mathbf{M})\mathbf{W}^\top. \tag{3}$$

Unstructured sparsity applies a global threshold over all elements. Semi-structured $N{:}M$ sparsity partitions rows (or columns) into non-overlapping blocks of size M and keeps the top-N elements per block by S (e.g., 2:4 removes 50%) (Hu et al., 2024). We also study 8:16, which retains the same 50% density with higher flexibility and modest metadata overhead (Maximov et al., 2025).

Table 1: Brief description of evaluated activation *pruning metrics* (top) and *transformations* (bottom). Abbreviations in **bold** with an asterisk (*) denote methods proposed here or first evaluated with sparse activations.

Short Name	Method	Key Mechanism
Pruning metrics		
ACT	Magnitude Pruning	Selects based on activation magnitude
WT	Weight-based Pruning	Selects by corresponding weight magnitude
CLACT*	Cosine Loss Activation	A metric inspired by cosine similarity from (Mi et al., 2025)
Amber-Pruner (2025)	Weights Importance	A metric which accounts for important weights after outlier removal and normalization
Transformations		
D-PTS*	Dynamic Per-Token Shift	Batch-wise dynamic centering of activations before sparsification
S-PTS (2024)	Static Per-Token Shift	Fixed centering of activations before sparsification using a per-token bias value **pre-collected on WikiText-2**
L-PTS*	Learnable Per-Token Shift	Fixed centering of activations before sparsification using per-token bias value **learned on WikiText-2**
VAR*	Variance Correction	Token-wise variance normalization after sparsification
VAR+L-PTS*	Scaling + Learnable Shift	Apply VAR scaling, then add per-token bias value **learned on WikiText-2**
R-Sparse (2025)	Rank-Aware Sparsity	Combines sparse activations with weight low-rank SVD factors **learned on WikiText-2**

2.2 Pruning Criterion

ACT: This is the magnitude activation pruning metric (**ACT**), defined as the absolute value of the element $\mathbf{X}_{ij}$: $S_{ACT}(\mathbf{X}_{ij}) = |\mathbf{X}_{ij}|$.

WT: This is the weight-based pruning metric (**WT**), defined as the absolute value of the corresponding weight $\mathbf{W}_{ij}$: $S_{WT}(\mathbf{W}_{ij}) = |\mathbf{W}_{ij}|$.

CLACT: Inspired by output cosine similarity in (Mi et al., 2025), we propose Cosine Loss ACTivation (CLACT), a context-aware score that emphasizes activations aligned with their row/column energy:

$$S_{CLACT}(\mathbf{X}_{ij}) = \frac{|\mathbf{X}_{ij}|}{\sqrt{\sum_{k=1}^{h} \mathbf{X}_{ik}^2}} \sqrt{\sum_{p=1}^{l} \mathbf{X}_{pj}^2}, \quad (4)$$

where h is the hidden dimension and l is the sequence length.

Amber-Pruner: Following (An et al., 2025), we (i) remove weight outliers outside the 0.5–99.5 percentiles, (ii) standardize the remaining weights, and (iii) score each activation as $S_{Amb.\text{-}Pr.}(\mathbf{X}_{ij}) = |\mathbf{X}_{ij}| \cdot \mathcal{L}(\hat{\mathbf{W}}_{:,j})$, where $\mathcal{L}(\cdot)$ is the channel-wise ℓ_2 norm.

CLACT is context-aware (and for l=1 reduces to an ℓ_1-type criterion), whereas Amber-Pruner leverages weight magnitudes but is not context-aware.

2.3 Transformations for Error Mitigation

D-/S-/L-PTS (dynamic/ static/ learnable per-token shift) centers activations near zero: $\hat{\mathbf{X}} = \mathbf{X} - \boldsymbol{\eta}$, and uses the compensated form $\mathbf{Y}_p = ((\hat{\mathbf{X}} \odot \mathbf{M}) + \boldsymbol{\eta})\mathbf{W}^\top$ (Chua et al., 2024), where $\boldsymbol{\eta} = \overline{\mathbf{X}}$ for D-PTS; S-PTS uses a fixed $\boldsymbol{\eta}$ collected in a short warm-up; L-PTS learns $\boldsymbol{\eta}$.

VAR applies per-token variance correction after pruning: $\mathbf{Y}_p = \boldsymbol{\nu}(\mathbf{X} \odot \mathbf{M})\mathbf{W}^\top$, with

$$\nu = \sqrt{\frac{\mathrm{Var}[\mathbf{X}]}{\mathrm{Var}[\mathbf{X} \odot \mathbf{M}]}}. \quad (5)$$

VAR+L-PTS combines VAR scaling with the learnable shift. **R-Sparse** combines activation sparsity with a low-rank weight approximation (Zhang et al., 2025) (details in Appendix B).

For methods requiring calibration/learning, we use WikiText-2: S-PTS stores a fixed bias vector $\boldsymbol{\eta}$, L-PTS (and VAR+L-PTS) learns $\boldsymbol{\eta}$, and R-Sparse learns low-rank factors; all other methods require no learned parameters.

2.4 Evaluation & Models

We evaluate the methods in Table 1 in two stages. First, we use **Core Datasets** (BoolQ, WinoGrande, PIQA, ARC-Easy) to screen all methods. We then focus on the most promising approaches on Qwen2.5-7B-Instruct and Llama3.1-8B-Instruct, and extend evaluation to **Extended Datasets** (HellaSwag, OpenBookQA, RTE, MMLU, Lambada_standard, Lambada_openai, IFEval). When calibration is required, we use WikiText-2, following common compression practice (Egiazarian et al., 2024; Frantar et al., 2022; Van Baalen et al., 2024). All results are obtained with LM Eval Harness (Gao et al., 2023); dataset details are in Table 9. We report results for Llama2-7B-chat, Llama3.1-8B-Instruct, Qwen2.5-7B-Instruct and Gemma3-4B-Instruct. For Qwen2.5-7B-Instruct, we do not

sparsify key/query/value activations due to severe degradation observed in preliminary experiments.

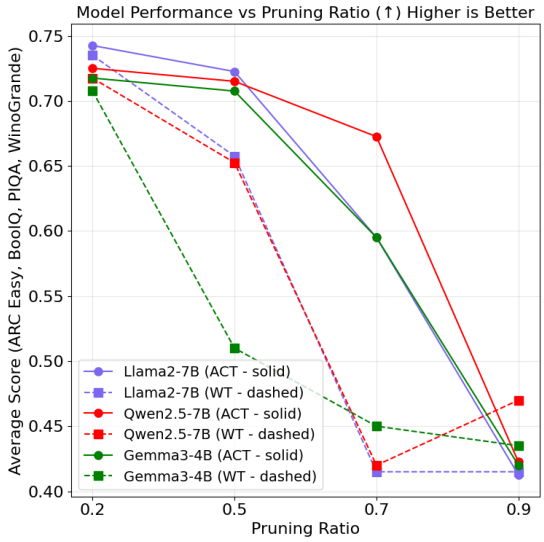

Figure 1: Comparison of **unstructured sparsity** in **activations** (**ACT**) and **weights** (**WT**) averaged across four datasets at varying sparsity ratios. **Higher is Better.** More detailed results are presented in Appendix Table 10.

3 Results

3.1 Sparse Weights vs. Activations

In Figure 1 and Table 10, we demonstrate that unstructured **weight sparsification causes greater model degradation** than unstructured activation sparsification at the same sparsity levels: {20%, 50%, 70%, 90%}. For this evaluation, we specifically use unstructured magnitude-based sparsification, as it is less damaging than semi-structured sparsification and thus provides a lower bound on performance degradation.

3.2 Optimal semi-structured sparsity patterns

Our preliminary investigation demonstrates that while the 16:32 pattern achieves performance close to unstructured sparsity (a 5.4% drop versus 4.5% for 50% unstructured), it requires more metadata and greater resources for gather operations, as discussed in Section 1. Therefore, we focus on the 8:16 pattern, despite its higher performance drop of 7.38%. For comparison, the 2:4 pattern results in a 14.35% drop. These results are shown in Figure 2 and Table 7 in Appendix, we used only magnitude pruning to obtain these results. By demonstrating the superior model quality of 8:16 sparsity, our

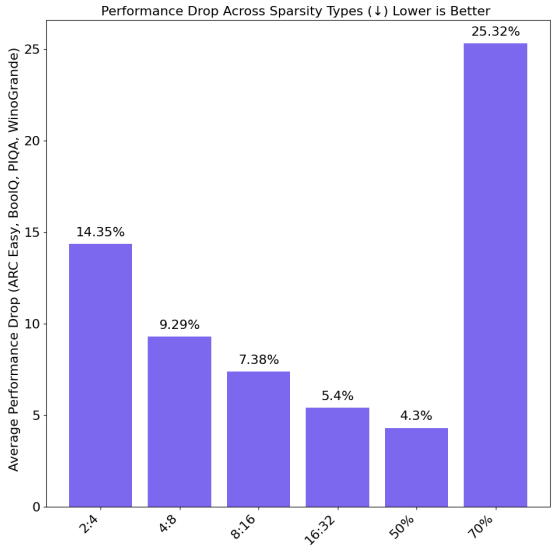

Figure 2: Comparison of sparsity patterns with unstructured sparsity. 50% and 70% correspond to unstructured sparsity. **Lower is Better.** More detailed results are presented in Appendix Table 7.

work incentivizes hardware designers to invest in or at least consider supporting the 8:16 pattern.

3.3 Results on Single/Multi-choice Datasets

3.3.1 Evaluation of Pruning Criteria

We evaluate CLACT, Amber-Pruner, and magnitude pruning as a baseline. The main results for the 2:4 and 8:16 sparsity patterns are presented in Table 2. On average, both CLACT and Amber-Pruner outperform magnitude pruning by at least 2%, however, we observe no clear winner between them. As noted in Section 2.2, these criteria are designed for different purposes: CLACT adjusts based on context, while Amber-Pruner adjusts based on weight magnitudes. Notably, for Llama3.1-8B-Instruct under the 2:4 sparsity pattern, simple magnitude pruning outperforms both advanced criteria, underscoring model and architecture-specific sensitivities to pruning strategies.

3.3.2 Evaluation of Transformations

Our main results are presented in Table 2. Surprisingly, we find that simple methods such as dynamic and static per-token shifts (D-PTS, S-PTS) outperform most other approaches. The second most effective methods are VAR and R-SPARSE. We also observe that increasing the number of dimensions in R-SPARSE (from 64 to 128) leads to worse performance, which may indicate overfitting on the calibration data. Finally, we note that L-

Table 2: Average relative performance drop (%, lower is better; negative = improvement) across four datasets, averaged over Llama2-7B-chat, Llama3.1-8B-Instruct, Qwen2.5-7B-Instruct and Gemma3-4B-Instruct. Act/Wt: activation/weight sparsity. Selection: ACT/CLACT/Amber-Pruner; transformations: VAR, D-PTS, S-PTS, L-PTS, R-SPARSE. Methods marked with an asterisk (*) are proposed in this paper. Full results: Appendix 11 and 12.

Target	Pattern	Method	Avg drop (↓)
Act	u50	ACT	3.82
Wt	2:4	WT	24.49
Act	2:4	ACT	9.67
Act	2:4	CLACT*	**7.79**
Act	2:4	Amber-Pruner	7.85
Act	2:4	VAR*	6.09
Act	2:4	D-PTS*	5.84
Act	2:4	S-PTS	**4.29**
Act	2:4	L-PTS*	8.79
Act	2:4	R-SPARSE (64)	7.70
Act	2:4	R-SPARSE (128)	8.05
Wt	8:16	WT	17.68
Act	8:16	ACT	5.47
Act	8:16	CLACT*	2.29
Act	8:16	Amber-Pruner	**1.56**
Act	8:16	VAR*	3.30
Act	8:16	D-PTS*	2.07
Act	8:16	S-PTS	**0.61**
Act	8:16	L-PTS*	5.32
Act	8:16	R-SPARSE (64)	1.52
Act	8:16	R-SPARSE (128)	2.63

PTS, the approach with learnable per-token shifts, significantly underperforms compared to its static counterpart, S-PTS.

3.4 Instruction-Following Tasks

Table 3 presents instruction-following performance on the IFEval benchmark for Llama3.1-8B-Instruct and Qwen2.5-7B-Instruct, evaluated under two semi-structured sparsity patterns (2:4 and 8:16) and four activation transformation methods: S-PTS, D-PTS, R-Sparse, and VAR. First of all, we observe a strong model degradation on generative tasks. Second of all, we see that VAR is the strongest performer overall, especially for Llama3.1-8B-Instruct. S-PTS/D-PTS are competitive and lightweight, and R-Sparse lags significantly, particularly at 2:4. We speculate that while semi-structured patterns are good for prefill stage in LLMs they significantly degrade performance during decode stage. However, as discussed in Section 1 decode stage for single vector can be accelerated with more flexible approaches.

3.5 Results for Unstructured Pruning

We evaluate D-PTS, VAR, and two selection criteria in this experiment: CLACT and Amber-Pruner

Table 3: Instruction-following (IFEval) prompt-level accuracy reported as PS/PL (prompt-level strict/loose accuracy).

Model	Method	2:4	8:16
Llama3.1-8B	ORIG	0.4455/0.4861	0.4455/0.4861
	S-PTS	0.1682/0.1904	0.2995/0.3327
	D-PTS	0.1941/0.2015	0.2828/0.3198
	R-Sparse	0.0869/0.0979	0.2089/0.2311
	VAR	**0.2237/0.2458**	**0.3161/0.3586**
Qwen2.5-7B	ORIG	0.7135/0.7394	0.7135/0.7394
	S-PTS	0.4325/0.5176	0.5194/0.5804
	D-PTS	0.4399/0.5120	**0.5434/0.5989**
	R-Sparse	0.2736/0.3457	0.3697/0.4196
	VAR	**0.4565/0.5342**	0.5249/0.5896

Table 4: Performance comparison of pruning methods under 50% and 70% unstructured sparsity for Llama3.1-8B-Instruct model.

Method	ArcE	BoolQ	PIQA	Wino Grande	(%) Avg. Drop
Original	0.821	0.839	0.800	0.734	—
Unstructured 50%					
ACT	0.777	0.820	0.771	0.686	4.450
D-PTS	0.786	0.825	0.781	0.690	3.600
VAR	0.784	0.819	0.776	0.705	**3.470**
CLACT	0.780	0.825	0.766	0.700	3.890
Amber	0.768	0.820	0.763	0.702	4.450
Unstructured 70%					
ACT	0.558	0.631	0.647	0.548	25.320
D-PTS	0.565	0.624	0.651	0.534	25.680
VAR	0.614	0.651	0.676	0.532	**22.660**
CLACT	0.555	0.604	0.627	0.524	27.670
Amber	0.487	0.594	0.590	0.539	30.680

using the Llama3.1-8B-Instruct model. Results in Table 4 indicate that VAR is the most effective transformation under unstructured sparsity. Moreover, CLACT outperforms Amber-Pruner by a wider margin here than in our semi-structured pruning experiments. These findings suggest two key insights: (1) No single method emerges as optimal for both unstructured and semi-structured sparsity. (2) The methods proposed in this work, VAR and CLACT, demonstrate strong generalization and are well-suited for both semi-structured and unstructured activation pruning.

3.6 Combination of Methods

Next, we evaluated combinations of multiple approaches to explore potential performance gains. These combinations and their results are presented in Table 8. As shown, none of the evaluated combinations outperforms any single method, highlighting the challenges of naively combining them.

Table 5: Llama3.1-8B-Instruct with 8:16 activation sparsity: aggregated results across layer subsets. **ORIG. AVG.** = 0.6726. Drop is computed w/o perplexity. Full results: Appendix 13.

Method	Layers	PPL	AVG.	Drop ↓
LS+L-PTS	all	9.6036	0.6047	10.90%
LS+L-PTS	key,out,gate,down	8.3483	0.6385	5.43%
LS+L-PTS	key,value,gate,down	8.0821	0.6503	**3.56%**
LS+L-PTS+VAR	all	9.4983	0.6056	10.60%
LS+L-PTS+VAR	key,out,gate,down	8.2930	0.6422	4.64%
LS+L-PTS+VAR	key,value,gate,down	8.0259	0.6516	**3.36%**

3.7 Layer Sensitivity

We study layer sensitivity to 8:16 activation sparsity on Llama3.1-8B-Instruct using the Extended Datasets (Table 5). This analysis focuses on learnable methods, mainly L-PTS and LS (learnable diagonal scaling). Although learnable approaches underperform on average in our broader experiments, the best 8:16 configurations for Llama3.1-8B-Instruct rely on learnable parameters (Table 5). We observe that the FFN up projection and the attention out projection[1] are the most sensitive: pruning them causes the largest drops, suggesting they should be preserved or handled with extra care. While this may not generalize to all layers, it indicates that layer importance under activation sparsity is highly non-uniform.

3.8 Analysis of Qwen2.5-7B-Instruct Anomalous Improvements

Several configurations for Qwen2.5-7B-Instruct under 8:16 sparsity outperform dense baseline on certain benchmarks (e.g., D-PTS: -8.28%, R-SPARSE(64): -6.90%. Unlike the other three models, Qwen2.5-7B-Instruct required excluding key/query/value projections from sparsification due to severe degradation observed in preliminary experiments. This means that only a subset of linear layers is actually pruned, reducing the overall perturbation applied to the network. With fewer layers affected, the risk of cascading errors is lower, and the remaining pruned layers may benefit from an implicit regularization effect without being offset by damage elsewhere.

4 Discussion

The performance gap between multiple/single-choice benchmarks (e.g., BoolQ, PIQA) and IFEval likely stems from differences in the inference stages they emphasize. Core QA benchmarks primarily stress the prefill phase, whereas IFEval evaluates both prefill and autoregressive generation. Our evaluation remains valid: semi-structured patterns like 2:4 and 8:16 are especially effective at accelerating the prefill stage, which often dominates inference latency.

4.1 Implications of IFEval Degradation for Generative Deployment

While activation sparsity preserves multiple-choice QA performance remarkably well, our IFEval results reveal a substantially different picture for generative, instruction-following tasks. A $\sim 26\%$ degradation in instruction adherence under 8:16 sparsity is likely unacceptable for many such use cases without additional mitigation. We hypothesize that the gap between QA and generative performance stems from how sparsity interacts with different inference stages. Multiple-choice QA benchmarks primarily stress the *prefill* phase, where the model processes the entire prompt in parallel and produces a single-token or few-token classification response. Semi-structured sparsity patterns are well-suited to this regime because the sparsification is applied to large activation matrices with favorable statistics. In contrast, IFEval evaluates both prefill and *autoregressive generation*, where errors introduced by sparsification compound across hundreds of generated tokens. During the decode stage, each token's representation is a single vector, and block-structured sparsity patterns impose rigid constraints on which elements can be zeroed. We emphasize that this limitation is not unique to our approach; the performance gap between generative and QA tasks under compression is well-documented across both sparsification and quantization methods (Ding et al., 2026).

5 Conclusion

This work establishes that post-training activation pruning is significantly more accuracy-preserving than weight pruning in large language models. Across four diverse architectures (Llama2-7B, Llama3.1-8B, Qwen2.5-7B, and Gemma3-4B), we demonstrate that activation sparsity consistently retains model capabilities better than weight sparsity at matched sparsity levels.

Our evaluation reveals that lightweight error mitigation techniques particularly CLACT, D-PTS, and VAR establish strong, hardware friendly base-

[1]The output projection of the attention mechanism. It combines outputs from all attention heads and projects them back to the model's hidden dimension.

lines requiring minimal calibration data. Through systematic analysis of semi-structured patterns, we find that 16:32 approaches unstructured 50% sparsity in fidelity, while 8:16 emerges as the optimal near term target.

6 Limitations

Key limitations: First, all evaluations use software emulation without hardware measurements of speedup or energy efficiency. Second, layer sensitivity analysis remains preliminary; while FFN up-projections and attention out-projections appear most vulnerable, broader architectural studies are needed. Third, generative performance (IFEval) degrades significantly more than multiple-choice QA under sparsity, revealing an evaluation bias toward prefill-dominated workloads. The anomalous improvements on Qwen2.5-7B-Instruct for some benchmarks further highlight dataset-specific artifacts rather than genuine capability preservation. Our hardware overhead analysis in Appendix A is a rough estimate and more precise analysis is required.

Acknowledgment: The work was supported by the grant for research centers in the field of AI provided by the Ministry of Economic Development of the Russian Federation in accordance with the agreement 000000C313925P4F0002 and the agreement with Applied AI Institute №139-10-2025-033.

References

Tai An, Ruwu Cai, Yanzhe Zhang, Yang Liu, Hao Chen, Pengcheng Xie, Sheng Chang, Yiwu Yao, and Gongyi Wang. 2025. Amber pruner: Leveraging n:m activation sparsity for efficient prefill in large language models. *Preprint*, arXiv:2508.02128.

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.

Vui Seng Chua, Yujie Pan, and Nilesh Jain. 2024. Post-training statistical calibration for higher activation sparsity. *arXiv preprint arXiv:2412.07174*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

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. *Preprint*, arXiv:1803.05457.

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

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pages 177–190. Springer.

Longwei Ding, Anhao Zhao, Fanghua Ye, Ziyang Chen, and Xiaoyu Shen. 2026. From llms to lrms: Rethinking pruning for reasoning-centric models. *arXiv preprint arXiv:2601.18091*.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. 2024. Extreme compression of large language models via additive quantization. *Preprint*, arXiv:2401.06118.

Ege Erdil. 2025. Inference economics of language models. *Preprint*, arXiv:2506.04645.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. 2024. Maskllm: Learnable semi-structured sparsity for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 7736–7758. Curran Associates, Inc.

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

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. 2023. A framework for few-shot language model evaluation.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7, pages 785–794.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Daniel Haziza, Timothy Chou, Dhruv Choudhary, Luca Wehrstedt, Francisco Massa, Jiecao Yu, Geonhwa Jeong, Supriya Rao, Patrick Labatut, and Jesse Cai. 2025. Accelerating transformer inference and training with 2:4 activation sparsity. *Preprint*, arXiv:2503.16672.

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*.

Yuezhou Hu, Kang Zhao, Weiyu Huang, Jianfei Chen, and Jun Zhu. 2024. Accelerating transformer pre-training with 2:4 sparsity. *Preprint*, arXiv:2404.01847.

Aaron Jarmusch and Sunita Chandrasekaran. 2025. Microbenchmarking nvidia's blackwell architecture: An in-depth architectural analysis. *arXiv preprint arXiv:2512.02189*.

Artyom Kharinaev, Viktor Moskvoretskii, Egor Shvetsov, Kseniia Studenikina, Bykov Mikhail, and Evgeny Burnaev. 2025. Investigating the impact of quantization methods on the safety and reliability of large language models. *arXiv preprint arXiv:2502.15799*.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2023. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*.

Eldar Kurtić, Elias Frantar, and Dan Alistarh. 2023. Ziplm: Inference-aware structured pruning of language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 65597–65617. Curran Associates, Inc.

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*.

Bin Lin, Ningxin Zheng, Lei Wang, Shijie Cao, Lingxiao Ma, Quanlu Zhang, Yi Zhu, Ting Cao, Jilong Xue, Yuqing Yang, and Fan Yang. 2023. Efficient gpu kernels for n:m-sparse weights in deep learning. In *Proceedings of Machine Learning and Systems*, volume 5, pages 513–525. Curan.

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.

Changxi Liu, Miao Yu, Yifan Sun, and Trevor E. Carlson. 2025. The sparsity-aware lazygpu architecture. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, page 1020–1034, New York, NY, USA. Association for Computing Machinery.

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

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. 2023. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR.

Egor Maximov, Yulia Kuzkina, Azamat Kanametov, Alexander Prutko, Aleksei Goncharov, Maxim Zhelnin, and Egor Shvetsov. 2025. From 2:4 to 8:16 sparsity patterns in llms for outliers and weights with variance correction. *Preprint*, arXiv:2507.03052.

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

Zhendong Mi, Zhenglun Kong, Geng Yuan, and Shaoyi Huang. 2025. Ace: Exploring activation cosine similarity and variance for accurate and calibration-efficient llm pruning. *Preprint*, arXiv:2505.21987.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1525–1534.

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

Susav Shrestha, Brad Settlemyer, Nikoli Dryden, and Narasimha Reddy. 2025. Polar sparsity: High throughput batched llm inferencing with scalable contextual sparsity. *arXiv preprint arXiv:2505.14884*.

Egor Shvetsov, Dmitry Osin, Alexey Zaytsev, Ivan Koryakovskiy, Valentin Buchnev, Ilya Trofimov, and Evgeny Burnaev. 2024. Quantnas for super resolution: Searching for efficient quantization-friendly architectures against quantization noise. *IEEE Access*, 12:117008–117025.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024a. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, page 590–606, New York, NY, USA. Association for Computing Machinery.

Yixin Song, Haotong Xie, Zhengyan Zhang, Bo Wen, Li Ma, Zeyu Mi, and Haibo Chen. 2024b. Turbo sparse: Achieving llm sota performance with minimal activated parameters. *arXiv preprint arXiv:2406.05955*.

Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. 2024. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*.

Hongyu Wang, Shuming Ma, Ruiping Wang, and Furu Wei. 2024. Q-sparse: All large language models can be fully sparsely-activated. *Preprint*, arXiv:2407.10969.

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*.

Zhenyu Zhang, Zechun Liu, Yuandong Tian, Harshit Khaitan, Zhangyang Wang, and Steven Li. 2025. R-sparse: Rank-aware activation sparsity for efficient llm inference. *arXiv preprint arXiv:2504.19449*.

Maxim Zhelnin, Viktor Moskvoretskii, Egor Shvetsov, Maria Krylova, Venediktov Egor, Zuev Aleksandr, and Evgeny Burnaev. 2025. Gift-sw: Gaussian noise injected fine-tuning of salient weights for llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6463–6480.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

Appendix

A Hardware Implications and Computational Overhead Analysis

While our empirical results demonstrate significant accuracy benefits from flexible N:M activation sparsity, the practical value of these techniques depends critically on whether hardware implementations can overcome the computational overhead of dynamic sparsification. This section provides a comprehensive break-even analysis to determine the hardware conditions required for activation sparsity to deliver net performance and efficiency gains.

The fundamental challenge for activation sparsity is that the theoretical memory bandwidth reduction must overcome the overhead introduced by dynamic sparsification operations. Based on literature and performance models, we estimate break-even thresholds where benefits begin to outweigh costs.

A.1 Energy-Delay Product Analysis

The Energy-Delay Product (EDP) provides a comprehensive metric for evaluating whether activation sparsity delivers net efficiency benefits. For 8:16 sparsity to be worthwhile from an EDP perspective, it must overcome both computational overhead and energy costs of the sparsification process itself.

We model EDP improvement for semi-structured sparsity patterns as:

$$\text{EDP}_{\text{improvement}} = \frac{\text{EDP}_{\text{dense}}}{\text{EDP}_{\text{sparse}}} \approx \frac{r \cdot \eta}{1 + \alpha}$$

Where:

- $r = 2.0$ is the theoretical bandwidth reduction ratio for 8:16 sparsity
- $\eta = 0.85$ is the hardware utilization efficiency (representative of practical implementations)
- $\alpha = 0.3$ is the overhead factor from sparsification operations

This overhead factor α is calibrated from real measurements (Fang et al., 2024) demonstrated that dynamic activation sparsification incurs 30–35% additional latency on current hardware without native support, broken down as:

- Activation magnitude computation and block-wise selection
- Mask application and metadata handling
- Error mitigation techniques (D-PTS, VAR), which is not included in (Fang et al., 2024).

Solving for the minimum hardware acceleration factor k required for net EDP benefits:

$$r \cdot \eta > k \cdot (1 + \alpha)$$

$$2.0 \cdot 0.85 > k \cdot (1 + 0.3)$$

$$k > \frac{1.7}{1.3} \approx 1.31$$

However, due to our imprecise estimations we will consider a higher *amortized* $k > 1.6\times$ required for speedup.

A.2 Hardware Implementation Requirements

To achieve the required $>1.6\times$ speedup and cross the break-even threshold, hardware must include:

- **Dedicated sparsity controllers**: On-chip circuitry that can generate sparsity masks with minimal latency, reducing the 15–20% selection overhead
- **Hardware-supported statistical units**: Specialized units for variance/mean calculation required by error mitigation techniques, eliminating their computational overhead
- **Hierarchical sparsity support**: Different patterns for different layer types based on sensitivity analysis (Section 3.7)
- **Bandwidth-optimized gather operations**: Specialized memory controllers that maintain high efficiency despite irregular access patterns

Recent hardware developments show promising directions: NVIDIA's Blackwell architecture includes a dedicated hardware decompression engine (Jarmusch and Chandrasekaran, 2025), while the LazyGPU microarchitecture enables lazy memory request issuing (Liu et al., 2025).

This analysis provides concrete targets for hardware designers: 8:16 activation sparsity can deliver significant accuracy preservation while achieving net performance gains, but only if hardware can deliver $>1.6\times$ speedup for sparse operations and efficiently support the statistical computations required by error mitigation techniques.

A.3 Microarchitectural Implementation Costs & Complexity Analysis

Precise implementation cost estimates are inherently challenging to formalize, as commercial GPU/TPU vendors typically do not disclose detailed microarchitectural specifications; many of these design parameters are governed by strict NDAs. The figures presented here are therefore engineering estimates grounded in publicly available microarchitectural analyses. While the absolute index width increases, the control logic scales sub-linearly because the combinatorial encoder/decoder can be implemented via lightweight lookup tables and bit-packing circuits rather than full-width arithmetic units. Drawing on published sparse-accelerator design studies (Lin et al., 2023) and microarchitectural analyses of Blackwell-class decompression engines, we conservatively estimate that extending an existing 2:4 pipeline to support 8:16 will incur an incremental die area overhead of $< 2\%$. The 8:16 pattern incurs a **16.7% higher metadata bandwidth** relative to 2:4 ($0.875/0.75 \approx 1.167$). To provide a structured overview of implementation trade-offs, we summarize the relative complexity across four key dimensions for 2:4 vs. 8:16 activation sparsity in Table 6. Ratings reflect incremental cost relative to a baseline dense tensor core. We assess implementation complexity across four dimensions, including estimated Non-Recurring Engineering (NRE) cos the one-time design, validation, and integration effort required to extend a tensor core to support a new sparsity pattern, excluding per-unit manufacturing costs.

B R-Sparse Details

Finally, we include **R-Sparse** (Zhang et al., 2025), which combines activation sparsity with a low-rank approximation of the weight matrix. Instead of pruning solely by magnitude, R-Sparse decomposes the computation into two parts: (i) sparse channels with high-magnitude activations, and (ii) a low-rank

Table 6: Qualitative complexity comparison across four microarchitectural dimensions for 2:4 vs. 8:16 activation sparsity. NRE = Non-Recurring Engineering cost (one-time design/validation effort).

Dimension	2:4	8:16	Justification & References
Metadata Overhead	Low (0.75 bits/elt)	Low–Med (0.875 bits/elt)	Combinatorial encoding scales logarithmically; 16.7% increase is marginal
Controller Logic	Low (2-bit decoders)	Medium (14-bit unpacking)	Requires wider LUTs & dynamic gather scheduling, but shares base sparse pipeline (Lin et al., 2023; Fang et al., 2024)
Memory Bandwidth	Low (halves fetches)	Low–Med (+16.7% metadata)	Net bandwidth drops due to 2× activation pruning; metadata fits HBM3 headroom
NRE Cost Tier	Low (mature IP)	Medium (index + gather opt.)	Validates dynamic mask generation without full tensor-core redesign (Liu et al., 2025)

component obtained via SVD of $\mathbf{W}$ that approximates the contribution of pruned activations.

Formally, the linear layer

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^\top \tag{6}$$

is approximated as

$$\mathbf{Y} \approx \mathbf{Y}_s + \mathbf{Y}_r, \tag{7}$$

where

$$\mathbf{Y}_s = \sigma_{t(s)}(\mathbf{X})\mathbf{W}^\top,$$

$$\mathbf{Y}_r = (\mathbf{X} - \sigma_{t(s)}(\mathbf{X}))(A_r B_r)^\top.$$

Here $\sigma_{t(s)}(\cdot)$ denotes sparsification of activations with threshold $t(s)$, and $A_r B_r^\top$ is the rank-r approximation of $\mathbf{W}$ obtained from its truncated SVD. The trade-off between $\mathbf{Y}_s$ and $\mathbf{Y}_r$ is determined by a sparsity budget s and rank r, which can be optimized via evolutionary search.

C Main extended results

Here, we present un-aggregated results. Comparisons between different sparsity patterns is presented in Table 7. In Table 11 and Table 12 we present the results of the error mitigation strategies and the selection criteria. Table 11 reports the results of the combined methods. Table 10 compares unstructured and semi-structured sparsity. Finally, in Table 13 we demonstrate results when some of the layers are excluded to evaluate layer sensitivity.

D Datasets

Detailed description of the datasets is given in Table 9.

E Comparison with Quantization Baselines

A key question for practitioners is whether activation sparsity offers competitive accuracy retention compared to quantization, the dominant compression technique in production LLM serving. In Table 14, we compare our post-training activation sparsity results against quantization baselines from Zhelnin et al. (2025) on Llama3.1-8B-Instruct.

Several observations emerge from this comparison. First, 8-bit quantization with fine-tuning (GIFT-SW (Zhelnin et al., 2025)) achieves strong results, in some cases exceeding the dense baseline (e.g., WinoGrande: 0.738 vs. 0.734), but it requires gradient-based stochastic training, which incurs significant computational cost and risks degrading safety alignment (Kharinaev et al., 2025). In contrast, our activation sparsity methods are entirely **post-training** and require **no fine-tuning**, making them immediately deployable without retraining infrastructure.

Second, unstructured 50% activation sparsity with VAR achieves competitive performance across all four benchmarks, with an average drop of only 3.47%, while providing a 2× theoretical FLOP reduction. The semi-structured 8:16 variants incur somewhat larger drops (6–8%), but offer hardware-friendly regularity that can be exploited by future accelerators. Finally, we note that this comparison is limited to a single model and a subset of benchmarks. A comprehensive comparison would require evaluating additional quantization methods (GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2024), SqueezeLLM (Kim et al., 2023)).

Table 7: Performance comparison of different sparsity patterns on Llama3.1-8B-Insruct across various benchmarks. Values represent accuracy scores, with the last column showing the average performance drop relative to the original model.

	ARC Easy	**BoolQ**	**PIQA**	**WinoGrande**	**Avg Drop (↓)**
Original	0.8207	0.8391	0.8003	0.7340	
2:4	0.6837	0.7261	0.7163	0.6110	14.35%
4:8	0.7272	0.7810	0.7529	0.6393	9.29%
8:16	0.7525	0.7969	0.7568	0.6551	7.38%
16:32	0.7698	0.8082	0.7688	0.6771	5.40%
50% unstructured	0.7820	0.8198	0.7714	0.6858	4.30%
70% unstructured	0.5580	0.6311	0.6474	0.5477	25.32%

Table 8: **A comparison of combined approaches with 8:16 semi-structured sparsity.** Average relative performance (%) across four datasets. Values indicate performance drops (lower is better), negative values signify performance improvement. Full, non-aggregated results are available in Appendix 12.

	Models				
Method	**Llama2-7B-chat**	**Qwen2.5-7B-Instruct**	**Gemma3-4B-Instruct**	**Llama3.1-8B-Instruct**	**Average Drop (↓)**
CLACT + PTS	5.63%	−5.06%	0.50%	8.55%	**2.40**%
CLACT + VAR	5.07%	−2.90%	0.54%	8.59%	2.82%
Amber-Pruner + PTS	6.16%	−3.47%	0.17%	7.42%	2.57%
Amber-Pruner + VAR	4.74%	−3.63%	−0.16%	8.39%	**2.34**%
L-PTS + VAR	6.87%	2.86%	3.41%	7.15%	5.07%

Table 9: Datasets used to evaluate hypotheses. *Prompt-level strict accuracy* is the fraction of prompts for which all verifiable instructions in the prompt are followed exactly as stated. *Instruction-level strict accuracy* is the fraction of individual instructions that are followed exactly as stated, averaged across all instructions.

Dataset	**Description**	**Metric**
WikiText-2 (Merity et al., 2016)	A collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia.	Perplexity
ARC-Easy (Clark et al., 2018)	QA benchmark for genuine grade-school level, multiple-choice science questions. The dataset contains 2251 examples for training, 570 for development and 2376 for testing.	Accuracy
ARC_Challenge (Clark et al., 2018)	QA benchmark for more difficult grade-school level science questions, part of the AI2 Reasoning Challenge. Designed to require deeper reasoning than ARC-Easy.	Accuracy
BoolQ (Clark et al., 2019)	QA benchmark for yes/no questions. The dataset contains 9427 examples for training and 3270 for testing.	Accuracy
PIQA (Bisk et al., 2020)	Physical commonsense QA benchmark for choosing the right answer between two options. Contains 16K train, 2K dev, and 3K test examples.	Accuracy
WinoGrande (Sakaguchi et al., 2021)	QA benchmark for pronoun resolution with adversarial filtering. Contains 40K train, 1267 dev, and 1767 test examples.	Accuracy
HellaSwag (Zellers et al., 2019)	Commonsense reasoning benchmark for sentence completion, designed to be easy for humans but hard for models. Contains 70K train and 10K validation examples.	Accuracy
OpenBookQA (Mihaylov et al., 2018)	Open-book question answering dataset requiring retrieval of elementary science facts. Contains 5957 4-way multiple-choice questions.	Accuracy
RTE (Dagan et al., 2005; Haim et al., 2006)	Recognizing Textual Entailment datasets from PASCAL challenges. Task is to classify if a hypothesis is entailed by a premise.	Accuracy
MMLU (Hendrycks et al., 2020)	Massive Multitask Language Understanding benchmark covering 57 subjects across STEM, humanities, and social sciences. Measures multitask accuracy.	Accuracy
Lambada_Standard (Paperno et al., 2016)	Word prediction task requiring broad discourse context. Target word is unpredictable from local context alone.	Accuracy
Lambada_OpenAI (Paperno et al., 2016)	LAMBADA test set preprocessed by OpenAI for standardized evaluation. Task remains final word prediction with long-range dependencies.	Accuracy
GSM8K (Cobbe et al., 2021)	Grade school math word problems requiring multi-step reasoning. Contains 7.5K train and 1.3K test examples.	Accuracy (Strict) Accuracy (Flexible)
IFEval (Zhou et al., 2023)	Benchmark with 541 prompts containing verifiable instructions to measure instruction-following fidelity.	Accuracy (Prompt-level) Accuracy (Instruct-level)

Table 10: The performance of models with applied unstructured activation pruning. We show that even with severe sparsity (70-90%) models were able to perform decently on our benchmarks. **ACT** stands for activations pruning, **WT** — for weight pruning. **OUT** denotes values more than 10^3, according accuracy scores most likely correspond to random.

Pruning	WikiText2 ↓	ARC Easy	BoolQ	PIQA	WinoGrande	Drop (↓)%
Llama2-7B-chat						
Base	6.94	0.74	0.80	0.76	0.66	-
0.2 **ACT**	6.96	0.74	0.80	0.77	0.66	**-0.33**%
0.2 **WT**	7.49	0.72	0.80	0.76	0.66	0.68%
0.5 **ACT**	7.53	0.70	0.78	0.75	0.66	**2.32**%
0.5 **WT**	18.72	0.60	0.72	0.70	0.61	11.10%
0.7 **ACT**	20.11	0.56	0.64	0.65	0.53	**19.62**%
0.7 **WT**	**OUT**	0.27	0.38	0.54	0.47	43.44%
0.9 **ACT**	**OUT**	0.26	0.38	0.52	0.49	43.39%
0.9 **WT**	**OUT**	0.27	0.38	0.53	0.48	43.39%
Qwen2.5-7B-Instruct						
Base	7.46	0.69	0.86	0.75	0.60	-
0.2 **ACT**	7.48	0.69	0.86	0.74	0.61	**2.37**%
0.2 **WT**	8.03	0.67	0.86	0.74	0.60	3.42%
0.5 **ACT**	8.3	0.67	0.87	0.74	0.58	3.87%
0.5 **WT**	43.6	0.56	0.80	0.68	0.57	**3.42**%
0.7 **ACT**	18.7	0.6	0.81	0.70	0.58	**3.87**%
0.7 **WT**	**OUT**	0.28	0.38	0.54	0.48	12.12%
0.9 **ACT**	**OUT**	0.25	0.38	0.54	0.52	44.22%
0.9 **WT**	**OUT**	0.25	0.58	0.54	0.51	**36.35**%
Gemma3-4B-Instruct						
Base	17.29	0.72	0.84	0.72	0.62	-
0.2 **ACT**	17.60	0.71	0.84	0.72	0.60	**3.35**%
0.2 **WT**	18.93	0.68	0.84	0.72	0.59	4.74%
0.5 **ACT**	22.39	0.71	0.83	0.72	0.57	**4.80**%
0.5 **WT**	273	0.36	0.55	0.61	0.52	30.89%
0.7 **ACT**	88	0.55	0.63	0.66	0.54	**19.57**%
0.7 **WT**	**OUT**	0.27	0.49	0.53	0.51	38.81%
0.9 **ACT**	**OUT**	0.26	0.38	0.54	0.50	42.64%
0.9 **WT**	**OUT**	0.25	0.45	0.52	0.52	**40.60**%

Table 11: **Semi-Structured 2:4 Sparsification** - performance Metrics, for calibration, when it is required, and perplexity we use WikiText2. Average Drop is computed without accounting for perplexity.

Pruning	WikiText2 ↓	ARC Easy	BoolQ	PIQA	WinoGrande	Average Drop %
Llama2-7B-chat	6.94	0.74	0.80	0.76	0.66	-
ACT	10.23	0.66	0.71	0.71	0.60	9.43%
WT	42.40	**0.57**	0.65	**0.69**	0.56	16.52%
D-PTS	9.38	0.64	0.68	0.71	0.61	10.67%
S-PTS	9.36	0.66	0.68	0.71	0.60	10.37%
VAR	**8.31**	0.67	0.69	0.72	0.59	9.76%
CLACT	8.23	0.65	0.72	0.71	0.63	8.32%
Amber-Pruner	9.24	0.64	0.68	**0.69**	0.60	11.70%
LPTS	8.89	0.65	**0.60**	0.72	**0.59**	13.13%
LPTS + VAR	8.39	0.67	0.63	0.72	0.60	11.47%
R-SPARSE (64)	9.19	0.66	0.63	0.69	0.59	12.90%
R-SPARSE (128)	9.29	0.65	0.65	0.70	0.59	12.23%
Llama3.1-8B-Instruct	7.21	0.82	0.84	0.80	0.73	-
ACT	16.61	0.68	0.73	0.72	0.61	14.35%
WT	20.14	0.41	0.57	0.60	0.54	33.63%
PTS	16.4	0.69	0.73	0.72	0.60	14.59%
S-PTS (N-100)	16.5	0.67	0.74	0.72	0.60	14.61%
S-PTS (N-200)	16.5	0.68	0.73	0.72	0.61	14.31%
VAR	14.17	0.70	0.73	0.73	0.62	13.11%
CLACT	19.49	0.65	0.71	0.69	0.59	17.27%
WANDA	15.86	0.66	0.74	0.69	0.61	15.01%
L-PTS	12.77	0.71	0.71	0.73	0.59	14.13%
L-PTS + VAR	12.40	0.73	0.71	0.73	0.60	13.49%
R-SPARSE (64)	15.07	0.69	0.72	0.71	0.61	15.28%
R-SPARSE (128)	16.09	0.67	0.71	0.70	0.61	16.34%
Qwen2.5-7B-Instruct	7.46	0.69	0.86	0.75	0.60	-
ACT	10.06	0.65	0.86	0.72	0.54	4.95%
WT	35.37	0.53	0.78	0.68	0.54	12.96%
D-PTS	10.07	0.79	0.86	0.76	0.66	-6.46%
S-PTS	10.74	0.78	0.84	0.74	0.65	-4.43%
VAR	13.95	0.74	0.83	0.74	0.61	-1.48%
CLACT	11.16	0.73	0.84	0.71	0.67	-2.45%
Amber-Pruner	10.64	0.74	0.84	0.70	0.64	-1.23%
LPTS	9.13	0.67	0.81	0.72	0.58	3.66%
LPTS + VAR	9.10	0.68	0.81	0.73	0.56	3.97%
R-SPARSE (64)	9.03	0.79	0.76	0.75	0.64	-2.55%
R-SPARSE (128)	9.12	0.77	0.77	0.75	0.63	-1.51%
Gemma3-4B-Instruct	17.29	0.72	0.84	0.72	0.62	-
ACT	35.62	0.65	0.76	0.70	0.51	9.94%
WT	421.95	0.35	0.44	0.58	0.49	34.86%
D-PTS	35.94	0.70	0.76	0.70	0.60	4.58%
S-PTS	35.84	0.71	0.77	0.70	0.60	3.93%
VAR	33.25	0.60	0.76	0.63	0.54	5.04%
CLACT	39.22	0.66	0.74	0.67	0.59	8.01%
Amber-Pruner	35.56	0.67	0.76	0.68	0.61	5.91%
LPTS	19.55	0.65	0.73	0.70	0.55	9.19%
LPTS + VAR	19.13	0.65	0.74	0.71	0.53	9.82%
R-SPARSE (64)	17.04	0.69	0.76	0.69	0.60	5.17%
R-SPARSE (128)	16.17	0.68	0.75	0.70	0.61	5.16%

Table 12: **Semi-Structured 8:16 Sparsification** - performance Metrics, for calibration, when it is required, and perplexity we use WikiText2. Average Drop is computed without accounting for perplexity.

Pruning	WikiText2 ↓	ARC Easy	BoolQ	PIQA	WinoGrande	Average Drop %
Llama2-7B-chat	6.94	0.74	0.80	0.76	0.66	-
ACT	8.12	0.69	0.75	0.73	0.63	5.37%
WT	20.47	0.64	0.76	0.72	0.61	7.84%
D-PTS	6.92	0.70	0.73	0.75	0.64	4.63%
S-PTS	6.93	0.70	0.73	0.75	0.66	3.87%
VAR	6.67	0.69	0.72	0.75	0.65	4.85%
CLACT	6.54	0.71	0.74	0.75	0.64	3.98%
CLACT + PTS	7.00	0.69	0.72	0.73	0.64	5.63%
CLACT + VAR	6.72	0.69	0.73	0.75	0.64	5.07%
R-SPARSE (64)	7.75	0.69	0.71	0.73	0.64	5.91%
R-SPARSE (128)	7.82	0.68	0.69	0.74	0.61	7.93%
Amber-Pruner	8.10	0.66	0.75	0.73	0.66	5.32%
Amber-Pruner + PTS	6.90	0.68	0.72	0.72	0.65	6.16%
Amber-Pruner + VAR	6.66	0.70	0.72	0.74	0.65	4.74%
LPTS	7.50	0.69	0.66	0.74	0.63	8.15%
LPTS + VAR	7.52	0.69	0.67	0.74	0.64	6.87%
Llama3.1-8B-Instruct	7.21	0.82	0.84	0.80	0.73	-
ACT	10.32	0.75	0.80	0.76	0.66	7.38%
WT	22.56	0.51	0.64	0.63	0.54	27.26%
D-PTS	10.34	0.76	0.80	0.76	0.66	6.79%
S-PTS	10.31	0.76	0.80	0.75	0.66	7.30%
VAR	10.67	0.74	0.79	0.75	0.66	8.30%
CLACT	10.67	0.73	0.79	0.74	0.66	8.60%
CLACT + PTS	10.68	0.74	0.79	0.74	0.65	8.55%
CLACT + VAR	10.15	0.74	0.79	0.75	0.64	8.59%
R-SPARSE (64)	11.42	0.75	0.77	0.75	0.66	8.44%
R-SPARSE (128)	10.43	0.75	0.78	0.74	0.66	8.49%
Amber-Pruner	10.16	0.73	0.80	0.75	0.68	7.13%
Amber-Pruner + PTS	10.17	0.75	0.80	0.75	0.66	7.42%
Amber-Pruner + VAR	9.94	0.74	0.80	0.75	0.64	8.39%
LPTS	10.04	0.76	0.79	0.77	0.65	7.19%
LPTS + VAR	10.26	0.77	0.78	0.76	0.66	7.15%
Qwen2.5-7B-Instruct	7.46	0.69	0.86	0.75	0.60	-
ACT	8.61	0.66	0.87	0.73	0.53	4.38%
WT	40.79	0.59	0.82	0.67	0.52	9.54%
D-PTS	8.61	0.80	0.87	0.77	0.68	-8.28%
S-PTS	8.84	0.80	0.86	0.76	0.67	-7.24%
VAR	11.91	0.69	0.72	0.75	0.65	1.93%
CLACT	8.94	0.77	0.85	0.73	0.65	-4.02%
CLACT + PTS	8.94	0.77	0.86	0.83	0.67	-5.06%
CLACT + VAR	8.87	0.76	0.84	0.71	0.65	-2.90%
R-SPARSE (64)	8.12	0.82	0.79	0.77	0.69	-6.90%
R-SPARSE (128)	8.24	0.80	0.79	0.77	0.67	-5.40%
Amber-Pruner	8.80	0.77	0.86	0.74	0.69	-6.20%
Amber-Pruner + PTS	8.79	0.77	0.85	0.73	0.64	-3.40%
Amber-Pruner + VAR	8.73	0.75	0.85	0.74	0.65	-3.60%
LPTS	8.23	0.69	0.83	0.75	0.57	1.70%
LPTS + VAR	8.21	0.70	0.83	0.73	0.56	2.80%
Gemma3-4B-Instruct	17.29	0.72	0.84	0.72	0.62	-
ACT	25.31	0.70	0.81	0.71	0.55	4.76%
WT	198.53	0.39	0.60	0.62	0.52	26.11%
D-PTS	25.17	0.70	0.81	0.71	0.54	5.16%
S-PTS	25.40	0.75	0.82	0.74	0.63	-1.54%
VAR	23.93	0.75	0.81	0.73	0.65	-1.87%
CLACT	25.85	0.75	0.81	0.71	0.61	0.60%
CLACT + PTS	26.03	0.73	0.81	0.70	0.63	0.50%
CLACT + VAR	24.78	0.74	0.81	0.70	0.63	0.54%
R-SPARSE (64)	15.39	0.76	0.80	0.74	0.64	-1.36%
R-SPARSE (128)	14.55	0.74	0.80	0.73	0.63	-0.44%
Amber-Pruner	25.11	0.74	0.82	0.70	0.63	0.08%
Amber-Pruner + PTS	25.28	0.75	0.81	0.70	0.63	0.17%
Amber-Pruner + VAR	23.97	0.74	0.81	0.71	0.64	-0.16%
LPTS	15.73	0.70	0.79	0.73	0.56	4.21%
LPTS + VAR	15.68	0.71	0.79	0.72	0.57	3.41%

Table 13: Llama3.1-8B-Instruct with 8:16 activation sparsity. LS+L-PTS indicates Learnable Diagonal Scale + Learnable Shift, "Layers" indicates the subset of linear layers where the method was applied. Drop is computed without accounting for perplexity.

Method	Layers	PPL	BoolQ	WinoGrande	PIQA	ARC Easy	ARC Chal.	HellaSwag	OpenBookQA	RTE	MMLU	Lambada stand.	Lambada (OpenAI)	Drop % ↓
ORIGINAL	–	–	0.8391	0.7340	0.8003	0.8207	0.5196	0.5905	0.3420	0.6859	0.6790	0.6569	0.7308	–
LS+L-PTS	all	9.6036	0.7841	0.6638	0.7715	0.7647	0.4514	0.5294	0.2720	0.6318	0.5521	0.5686	0.6625	10.90%
LS+L-PTS	k,o,gate,down	8.3483	0.8205	0.7111	0.7889	0.7887	0.4659	0.5591	0.3200	0.6462	0.6060	0.6123	0.7046	5.43%
LS+L-PTS	k,v,gate,down	8.0821	0.8352	0.7174	0.7867	0.7992	0.4898	0.5651	0.3260	0.6643	0.6322	0.6262	0.7108	3.56%
LS+L-PTS + VAR	all	9.4983	0.7872	0.6606	0.7606	0.7601	0.4334	0.5372	0.2880	0.6390	0.5532	0.5729	0.6689	10.60%
LS+L-PTS + VAR	k,o,gate,down	8.2930	0.8116	0.7135	0.7851	0.7908	0.4838	0.5634	0.3300	0.6498	0.6095	0.6189	0.7079	4.64%
LS+L-PTS + VAR	k,v,gate,down	8.0259	0.8306	0.7269	0.7851	0.7955	0.4863	0.5673	0.3260	0.6715	0.6327	0.6317	0.7143	3.36%

Table 14: Comparison of activation sparsity and quantization on Llama3.1-8B-Instruct. Quantization results are from Zhelnin et al. (2025). Note that GIFT-SW uses stochastic fine-tuning (STE), whereas our activation sparsity methods require **no fine-tuning**.

Method	BoolQ	WinoGrande	PIQA	ARC-Easy
Baseline (dense)	0.839	0.734	0.800	0.821
Quantization (requires fine-tuning)				
8-bit GIFT-SW (STE)	—	0.738	0.810	0.798
Activation sparsity (no fine-tuning)				
50% unstruct. + S-PTS	0.800	0.660	0.750	0.760
50% unstruct. + VAR	0.819	0.705	0.776	0.784
8:16 + ACT (magnitude)	0.797	0.655	0.757	0.753
8:16 + Amber-Pruner	0.800	0.680	0.750	0.730
8:16 + D-PTS	0.800	0.660	0.760	0.760
8:16 + VAR	0.790	0.660	0.750	0.740