ROSHIFT: LAZY SHIFT AND WEIGHT-AWARE ROTATION FOR ACTIVATION SPARSITY IN LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Activation sparsity accelerates LLM decoding by skipping unnecessary weight reads and computation. However, SiLU-based LLMs exhibit little natural sparsity, and imposing high sparsity can substantially degrade accuracy. We propose RoShift, a training-free method that transforms activations into sparsity-friendly representations. RoShift introduces lazy shift, which recenters SiLU activation distributions around zero and defers correction to the end of the FFN to obtain additional sparsity. It also reduces sparsification-induced output error through rotation that jointly accounts for activation and weight information. By fusing the corrections required for shifting and rotation, RoShift translates these gains into decoding speedup. At 50% and 60% sparsity, RoShift improves average accuracy over TEAL by 1.60 and 3.29 percentage points, respectively, across five models. It achieves up to $1.72\times$ decoding speedup over the dense model on an RTX 5090 at 50% sparsity and yields a better accuracy-speedup Pareto frontier than TEAL.

1 INTRODUCTION

Large language models (LLMs) achieve strong performance across diverse tasks, but their large parameter counts make inference expensive. Autoregressive decoding generates tokens sequentially within each sequence and repeatedly reads model weights at every step. In particular, at batch size 1, limited parallelism and weight reuse often make memory bandwidth the performance bottleneck. These characteristics make efficient decoding challenging in edge and on-device settings, where a single device serves one user's requests, as well as on Physical AI platforms.

Activation sparsity offers a way to reduce these weight reads and computations. In a linear layer, a zero activation makes the corresponding weights irrelevant to the output, allowing their memory accesses and computations to be skipped. ReLU-based models naturally produce many zero activations and can therefore readily exploit this property, whereas modern LLMs using SiLU or GELU rarely produce exact zeros. Methods such as CATS and TEAL approximate small-magnitude activations as zero to induce sparsity (Lee et al., 2024; Liu et al., 2025a). At high sparsity, however, this can remove components that contribute substantially to the output, leading to greater accuracy degradation.

Recent studies use rotation to concentrate activation energy in a subset of dimensions (Ashkboos et al., 2024; Liu et al., 2025b). In particular, LaRoSA applies such rotations to activation sparsification to reduce sparsification error. Meanwhile, WAS and WINA select components for removal using both activation magnitudes and the importance of the corresponding weights (Wang et al., 2025; Chen et al., 2026). Combining these approaches introduces a new challenge: rotation changes both activations and their corresponding weights, affecting not only which components are removed but also how their removal changes the output. A sparsity-friendly rotation should therefore account for both the activation distribution and the influence of weights on removal error.

We also observe that the channel-wise activation distributions in SiLU-based models are centered away from zero, limiting the components eligible for magnitude-based sparsification around zero. Shifting each channel by a fixed offset recenters them near zero, placing more elements within the sparsification threshold, enabling higher activation sparsity at comparable quality. SCAP exploits this property through mode-centering in non-GLU FFNs (Chua et al., 2024). In gated FFNs, however, restoring the original gate activations before multiplying them by the Up outputs prevents the Up and Down projections from exploiting the additional sparsity obtained by recentering. Moreover, preserving the original computation under representation transformations such as shifting and rotation may require

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

additional corrections, whose cost can offset the savings from sparsification. We therefore need transformations that reduce sparsification-induced output error while limiting correction overhead sufficiently to deliver practical inference acceleration.

To address these challenges, we propose RoShift, a training-free activation sparsification method. RoShift recenters post-SiLU activations, while lazy shift defers restoration to the end of the FFN so that the Up and Down projections can exploit the additional sparsity. RoShift also constructs rotations that jointly account for activation statistics and the influence of weights on removal error, reducing sparsification-induced output error. Finally, it fuses the corrections required for shifting and rotation and optimizes sparse kernels to translate the resulting sparsity into practical decoding acceleration.

At 50% and 60% non-uniform sparsity, RoShift improves average accuracy over TEAL by 1.60 and 3.29 percentage points, respectively, across five models. It achieves up to $1.72\times$ decoding speedup over the dense model on an RTX 5090 at 50% sparsity and yields a better accuracy-speedup Pareto frontier than TEAL.

Our main contributions are as follows.

1. Lazy shift: We recenter post-SiLU activations and defer the correction to the end of the FFN, providing additional sparsity that the Up and Down projections can exploit.
2. Activation- and weight-aware rotation: We jointly incorporate the activation distribution and weight information to construct rotations that reduce sparsification-induced output error.
3. Correction fusion and efficient GPU execution: We effectively fuse the two correction operations to reduce overhead and use efficient sparse kernels to translate these savings into practical decoding acceleration.

2 Background & Observation

2.1 Background: Activation Sparsity

Activation sparsity can accelerate inference by skipping weight reads and computations associated with components that do not contribute to the output. We review prior work on activation sparsity in Appendix A.1.

Activation sparsity is primarily exploited through input sparsity and output sparsity. Input sparsity exploits zeros in the activations entering a projection. For a linear projection with input activations $X \in \mathbb{R}^{1\times d_{\text{in}}}$ and weights $W \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$, the output is $Y = XW \in \mathbb{R}^{1\times d_{\text{out}}}$. If $X_i = 0$, the corresponding weight row $W_{i,:}$ does not contribute to the output, so its memory accesses and computations can be skipped. In SiLU-based models, where natural zeros are rare, thresholding or Top-K selection induces sparsity by approximating small-magnitude activations as zero.

Output sparsity, in contrast, identifies output dimensions expected to contribute little or nothing to subsequent computations and skips the corresponding weight columns $W_{:,j}$. Let d denote the model dimension and k the FFN intermediate dimension. A gated FFN computes the following for input X:

$$h_1 = \sigma(XW_{\text{gate}}), \quad h_2 = XW_{\text{up}}, \quad h_3 = h_1 \odot h_2, \quad Y_{\text{FFN}} = h_3 W_{\text{down}}, \tag{1}$$

Here, $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d\times k}$ and $W_{\text{down}} \in \mathbb{R}^{k\times d}$. Here, σ is an activation function such as SiLU and h_1 gates the channel-wise Up outputs. If $h_{1,j} = 0$, the computation for Up output $h_{2,j}$ and the contribution from the corresponding Down input $h_{3,j}$ can both be skipped; thus, Up output sparsity becomes Down input sparsity. CATS exploits this output sparsity by selecting channels according to the magnitudes of the gate activations h_1 (Lee et al., 2024).

Magnitude-based input sparsification, such as TEAL, selects components for removal based on their input activation magnitudes $|X_i|$ (Liu et al., 2025a). Removing a component X_i, however, removes the output contribution $X_i W_{i,:}$, whose magnitude is $\|X_i W_{i,:}\|_2 = |X_i| \cdot \|W_{i,:}\|_2$. Consequently, components with identical activation magnitudes can induce different sparsification errors depending on their corresponding weights. WAS and WINA incorporate both activation and weight information into their sparsification criteria (Wang et al., 2025; Chen et al., 2026).

Another approach to sparsification is to change the activation basis to obtain a sparsity-friendly representation. Concentrating activation energy in a subset of dimensions through rotation can reduce

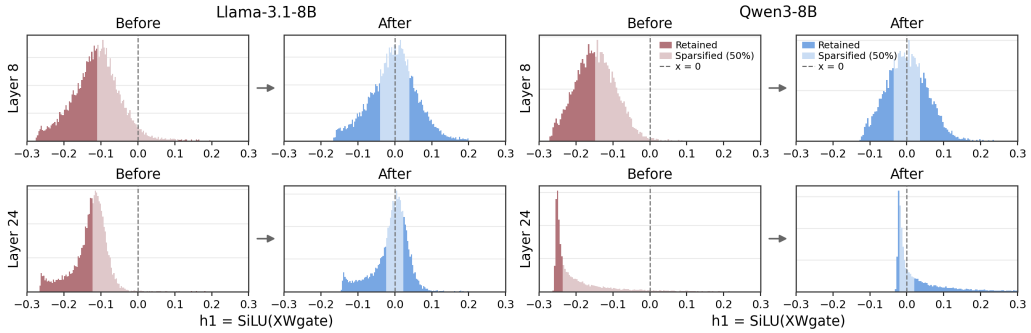


Figure 1: Activation distributions of post-SiLU h_1 before and after recentering. Light shading denotes the 50% of components removed by magnitude in each representation; dark shading denotes retained components. Dashed lines indicate zero.

the energy discarded by Top-K at a fixed sparsity. For an orthogonal matrix $Q \in \mathbb{R}^{d\times d}$ that defines this change of basis, Eq. (2) shows that jointly transforming activations and weights preserves the original linear projection output.

$$Y = XW = (XQ)(Q^\top W) \tag{2}$$

Within a Transformer block, the channel-wise RMSNorm scales and rotation can be absorbed into the relevant weights offline. Maintaining the residual in the form XQ allows computation in the rotated basis without a separate runtime rotation wherever the same Q is used. LaRoSA constructs layer-wise PCA rotations from calibration activation statistics to concentrate activation energy in a subset of dimensions. When different layers use different Q_l, however, the bases must be aligned between layers. Transforming XQ_l into XQ_{l+1} requires an additional operation with the transition matrix $Q_l^\top Q_{l+1} \in \mathbb{R}^{d\times d}$ (Liu et al., 2025b).

2.2 Motivation 1: Off-center SiLU Activation Distributions

Unlike ReLU, SiLU produces nonzero outputs for most inputs and therefore exhibits little natural sparsity. Nevertheless, these activations have asymmetric, off-center distributions. To center each channel's distribution at zero, we compute the channel-wise median $h_{1\text{med}} \in \mathbb{R}^{1\times k}$ from calibration activations and subtract it from h_1, yielding $\widetilde{h}_1 = h_1 - h_{1\text{med}}$. These channel-wise medians vary little across calibration samples, allowing the estimates to serve as fixed shifts that reliably approximate the centers of activation distributions for unseen inputs. Figure 1 shows that, even when the same 50% of components are removed, recentering maps values in densely populated regions of the original distribution to small residuals near zero, making them natural candidates for removal.

Preserving the original FFN output, however, requires a correction for the shift. If h_1 is restored before multiplication by h_2, the decisions to skip Up and Down computations are again based on the original h_1, preventing the additional sparsity from being exploited. To use the sparsity obtained through recentering in both Up and Down, we therefore need a correction that preserves the original FFN output without restoring the shifted activations midway through the computation.

2.3 Motivation 2: The Need for Weight-aware Rotation

Weight-aware sparsification selects components for removal using both activations and their corresponding weights, whereas rotation transforms activations into a sparsity-friendly representation. Applying a rotation Q changes the activations to XQ and the weights to $Q^\top W$, so both the removed components and their influence on the output depend on Q.

The output error after applying weight-aware Top-K to the rotated activations is

$$Y_{\text{error}} = \left\| \left(\text{TopK}_{\text{WA}}(XQ) - XQ\right)(Q^\top W) \right\|_2^2. \tag{3}$$

where TopK_{WA} retains components based on activation and weight importance. Thus, concentrating activation energy alone does not necessarily minimize sparsification error, motivating a rotation that considers both activation statistics and weights.

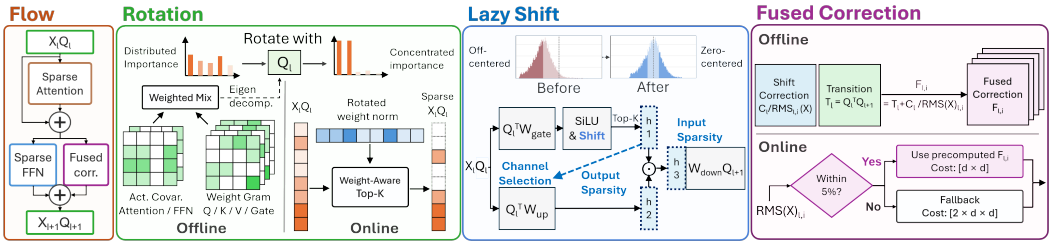


Figure 2: Overview of RoShift, which introduces lazy shift and weight-aware rotation, tightly integrating both with efficient correction fusion for activation sparsification.

3 RoShift

3.1 Overview

As illustrated in Figure 2, RoShift is a training-free method that transforms activations through rotation or shifting into sparsity-friendly representations and sparsifies them to reduce weight reads and computations. Both transformations require corresponding corrections to preserve the original computations. By minimizing their overhead, RoShift translates activation sparsity into practical decoding acceleration.

During inference, the rotated residual representation enables rotation-based input sparsity for Q/K/V, Gate, and Up, but nonlinear operations prevent propagation beyond these projections. RoShift therefore uses rotation-based input sparsity for Q/K/V and Gate, while lazy shift provides Up output sparsity that becomes Down input sparsity. For Up, lazy-shift output sparsity empirically achieves higher sparsity than rotation-based input sparsity, so we use the former.

3.2 Lazy Shift

Lazy shift makes activation distributions more amenable to sparsification and defers the correction to the end of the FFN, enabling Up and Down to exploit higher sparsity. Using the per-layer channel-wise median $h_{1\text{med}}$ precomputed during calibration, RoShift forms $\widetilde{h}_1 = h_1 - h_{1\text{med}}$. The gated FFN in Eq. (1) can then be decomposed as follows.

$$\begin{aligned} Y_{\text{FFN}} &= ((\widetilde{h}_1 + h_{1\text{med}}) \odot h_2) W_{\text{down}} \\ &= (\widetilde{h}_1 \odot h_2) W_{\text{down}} + XC, \qquad (4) \\ C &= W_{\text{up}}\, \text{diag}(h_{1\text{med}}) W_{\text{down}}. \qquad (5) \end{aligned}$$

The first term in Eq. (4) evaluates the FFN with shifted activations, and the second corrects for the shift. The correction matrix $C \in \mathbb{R}^{d\times d}$ is input-independent and can therefore be precomputed. The FFN can thus operate on shifted activations without restoring h_1 midway through the computation, and adding XC to its output exactly compensates for the shift.

We first apply input sparsity to Gate, then select Up output channels using the recentered gate magnitudes $|\widetilde{h}_1|$, skipping Up computations for unselected channels. Because these channels are also unused as inputs to Down, Up output sparsity becomes Down input sparsity. If Down is assigned a higher target sparsity, we apply additional input sparsification. We do not sparsify the correction term XC itself, because the resulting accuracy loss is large relative to the speedup obtained.

Computing XC requires reading and processing an additional d^2 weights. In typical gated FFNs, k is several times larger than d, so the d^2 entries of C are fewer than the $3dk$ weights in the three FFN projections. Appendix A.2 details this overhead for each model. Nevertheless, additional sparsity does not yield decoding acceleration if its savings are smaller than the correction cost. RoShift reduces this overhead by combining the correction with the inter-layer transition and computing it with optimized kernels, as described in Section 3.4.

3.3 Activation- and Weight-aware Rotation

RoShift constructs sparsity-friendly layer-wise rotations $Q_l \in \mathbb{R}^{d\times d}$ by jointly considering activation statistics and the influence of weights on sparsification-induced output error. Since Top-K selection changes discretely with the rotation, RoShift instead uses each axis's contribution variance to the projection output as a surrogate.

Consider the output contribution of a single rotation axis q. In Eq. (2), this contribution is $Y_q = (Xq)(q^\top W)$. Let S denote the activation covariance and $M = WW^\top$ the weight Gram matrix. The variance of this axis's output contribution is

$$e(q) = \mathrm{Var}(Xq)\|q^\top W\|_2^2 = (q^\top Sq)(q^\top Mq). \tag{6}$$

Maximizing $e(q)$ under the unit-norm constraint $q^\top q = 1$ yields the Lagrange stationarity condition:

$$\left[(q^\top Mq)S + (q^\top Sq)M\right] q = \lambda q. \tag{7}$$

Because the coefficients $q^\top Mq$ and $q^\top Sq$ depend on q, unlike standard PCA, no single fixed eigensystem directly solves the problem. RoShift approximates these axis-dependent coefficients with layer-shared mixture coefficients. Accordingly, the resulting eigensystem is a tractable surrogate for the axis-dependent objective rather than its exact closed-form optimizer. Appendix A.3 provides the full derivation.

To apply a single layer-wise rotation across Transformer projections, we combine six statistics: attention and FFN input covariances and Q/K/V/Gate weight Gram matrices. Activations are inverse-RMS normalized, with learned RMSNorm channel scales absorbed into the corresponding weights. After unit-trace normalization, the matrices $B_{l,i}$ are combined as $B_l = \sum_{i=1}^{6} \theta_{l,i} B_{l,i}$, $\theta_{l,i} \geq 0$. We eigendecompose B_l and use its eigenvectors in descending eigenvalue order as the columns of Q_l.

After applying this rotation, we impose input sparsity on the Q/K/V and Gate projections using Top-K selection that jointly considers rotated activation magnitudes and the corresponding rotated weight row norms. Because adjacent layers use different rotations, the residual representation is transformed between layers using the transition matrix $Q_l^\top Q_{l+1} \in \mathbb{R}^{d\times d}$. Section 3.5 describes the search for the layer-wise mixture coefficients $\theta_{l,i}$.

Rotation-Enabled LM-Head Sparsification. The rotation framework enables additional LM-head sparsification. By concentrating final hidden activations in leading dimensions, it improves early partial-logit estimates, enabling staged evaluation and early termination of unlikely candidates to skip LM-head weight reads and computation. The rotation is absorbed into the LM-head weights offline; details are provided in Appendix A.4.

3.4 Correction Fusion and Efficient GPU Execution

Correction Fusion. Lazy shift and layer-wise rotation each require an additional operation with a $d \times d$ matrix for shift correction and inter-layer basis conversion, respectively. Let X_l denote the rotated FFN input residual, T_l the inter-layer transition matrix, and C_l the shift correction matrix in the corresponding rotated basis. The two operations can be combined as

$$X_l \left[T_l + \frac{C_l}{\mathrm{RMS}(X_l)}\right]. \tag{8}$$

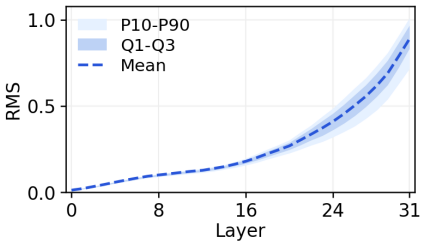


Figure 3: Layer-wise RMS distributions of FFN inputs before RMSNorm in Llama-3.1-8B, measured over 32×4096 tokens.

Although T_l and C_l are fixed matrices, $\mathrm{RMS}(X_l)$ varies across tokens, preventing a single fused matrix from being precomputed and stored. However, as Figure 3 shows, activation RMS varies relatively little across tokens within a layer. RoShift exploits this property by approximating calibration RMS values with a small set of representatives $a_{l,r}$ and precomputing $T_l + C_l/a_{l,r}$ for each representative. At inference time, it uses a fused matrix if the relative difference between the actual RMS and its representative is within 5%; otherwise, it falls back to computing the two operations separately using the actual RMS. Section 4.5 analyzes the memory-speedup trade-off as the number of representatives varies.

Efficient GPU Execution. To translate correction fusion into decoding acceleration, RoShift executes the sparse FFN and correction path in parallel on separate CUDA streams, synchronizing when their outputs are added. It also arranges weights offline according to the sparsification direction to enable coalesced memory access, and its sparse kernels compute only the retained components. Blockwise Top-K reduces selection overhead. Appendix A.5 describes the kernel implementation and its impact on quality.

3.5 Sparsity and Rotation Optimization

Sparsity Optimization: Table 1 compares the sparsity-allocation granularity of existing methods. TEAL assigns different sparsities to individual projections within each layer while keeping the total sparsity identical across layers (Liu et al., 2025a). WAS extends non-uniform allocation across layers, but constrains sparsity to be monotonically non-decreasing with depth (Wang et al., 2025).

Table 1: Sparsity allocation flexibility.

Scope	LaRoSA	TEAL	WAS	**RoShift**
Projection-wise sparsity	△	✓	✓	✓
Layer-wise sparsity	×	×	△	✓

△: LaRoSA shares projection-wise allocation across layers; WAS supports layer-wise sparsity only under a monotonic non-decreasing constraint.

Because early and late LLM layers are particularly sensitive to pruning and quantization (Ma et al., 2023; Wang et al., 2026), the monotonic constraint in WAS prevents assigning lower sparsity to sensitive late layers. RoShift initializes its allocation using TEAL-style within-layer allocation and WAS-style inter-layer search, then refines it through pairwise coordinate descent without such a constraint. Specifically, each update transfers sparsity between two layer-projection coordinates while preserving overall MAC sparsity. MAC sparsity denotes the fraction of linear-projection MACs eliminated by sparsification, following the same model-level sparsity definition as TEAL and WINA. We first screen candidates using inexpensive local error and PPL on a short calibration set, then evaluate the remaining candidates using PPL on the full calibration set. We repeatedly accept the update that yields the largest PPL improvement.

Rotation Optimization: After fixing the sparsity allocation, we search for the layer-wise coefficients $\theta_{l,i}$ that combine the six matrices in Section 3.3. Starting from the same initial mixture in every layer, we perform a greedy pairwise search by transferring a fixed amount between coefficients. Each resulting candidate rotation Q_l is evaluated using the relative L2 error of the block output after the attention and the FFN with respect to the dense output. We accept the best transfer if it reduces the error. However, locally accepted rotations may not remain beneficial when combined across layers. Rather than immediately adopting the layer-wise coefficient candidates obtained independently, we test them sequentially from shallow to deep layers and retain each Q_l only if it improves the full-model PPL on the full calibration set. Appendix A.6 details both search procedures.

4 Experiments

4.1 Experimental Setup

We evaluate RoShift's accuracy preservation and practical decoding acceleration. Our primary speed baseline is TEAL, which provides a publicly available end-to-end decoding acceleration implementation. WAS, WINA, and LaRoSA do not provide end-to-end decoding implementations. We additionally include LaRoSA and WINA as accuracy baselines, representing rotation-based and weight-aware activation sparsification, respectively (Liu et al., 2025b; Chen et al., 2026).

Table 2: Accuracy under non-uniform sparsity at matched Transformer MAC-reduction targets, excluding the LM-head. Each model entry reports mean accuracy over 11 benchmarks. Full results are in Appendix A.9.

Method / Model	Llama-3.1-8B	Llama-2-7B	Mistral-7B	Qwen2.5-7B	Qwen3-8B	Avg
Dense (0%)	63.87	47.14	59.67	73.28	76.43	64.08
TEAL 40%	61.83(-2.05)	45.89(-1.26)	58.56(-1.11)	**71.97(-1.32)**	74.54(-1.89)	62.56(-1.52)
LaRoSA 40%	61.21(-2.66)	44.99(-2.15)	57.83(-1.84)	71.04(-2.25)	74.02(-2.40)	61.82(-2.26)
WINA 40%	61.71(-2.16)	**46.39(-0.75)**	58.60(-1.07)	71.26(-2.02)	73.97(-2.45)	62.39(-1.69)
RoShift 40%	**62.26(-1.62)**	46.03(-1.12)	**59.24(-0.43)**	71.86(-1.42)	**75.16(-1.27)**	**62.91(-1.17)**
TEAL 50%	59.21(-4.67)	44.34(-2.81)	56.95(-2.72)	70.46(-2.82)	73.49(-2.94)	60.89(-3.19)
LaRoSA 50%	56.77(-7.10)	43.09(-4.05)	54.77(-4.90)	68.13(-5.16)	70.90(-5.52)	58.73(-5.35)
WINA 50%	58.52(-5.35)	45.49(-1.65)	56.44(-3.23)	68.95(-4.33)	71.94(-4.49)	60.27(-3.81)
RoShift 50%	**61.47(-2.40)**	**45.90(-1.25)**	**58.54(-1.13)**	**71.60(-1.68)**	**74.93(-1.50)**	**62.49(-1.59)**
TEAL 60%	53.44(-10.43)	41.46(-5.69)	54.42(-5.25)	67.19(-6.09)	70.41(-6.01)	57.38(-6.70)
LaRoSA 60%	47.49(-16.38)	39.77(-7.37)	50.58(-9.09)	61.39(-11.90)	60.35(-16.08)	51.92(-12.16)
WINA 60%	50.62(-13.26)	42.17(-4.97)	52.41(-7.26)	63.42(-9.86)	66.47(-9.96)	55.02(-9.06)
RoShift 60%	**58.54(-5.33)**	**43.41(-3.73)**	**57.82(-1.85)**	**70.53(-2.75)**	**73.07(-3.36)**	**60.68(-3.40)**

Table 3: Batch-size-1 decoding throughput (tokens/s; speedup over the dense baseline in parentheses) of RoShift with uniform sparsity across Transformer layers and projections. Measurement details are in Appendix A.5.

GPU	Sparsity	Llama-3.1 8B	Llama-2 7B	Mistral 7B	Qwen2.5 7B	Qwen3 8B
RTX 5090	Dense	100.58(1.00×)	104.62(1.00×)	94.66(1.00×)	96.72(1.00×)	97.43(1.00×)
	25%	120.03(1.19×)	121.64(1.16×)	121.63(1.28×)	129.81(**1.34**×)	113.52(1.17×)
	50%	159.86(1.59×)	159.20(1.52×)	162.66(**1.72**×)	159.37(1.65×)	146.86(1.51×)
	75%	238.18(2.37×)	235.79(2.25×)	246.03(**2.60**×)	239.96(2.48×)	217.42(2.23×)
Jetson AGX Orin 64GB	Dense	11.49(1.00×)	12.58(1.00×)	12.12(1.00×)	12.08(1.00×)	11.34(1.00×)
	25%	14.09(**1.23**×)	14.78(1.17×)	14.25(1.18×)	14.62(1.21×)	13.45(1.19×)
	50%	19.38(**1.69**×)	20.16(1.60×)	19.69(1.63×)	20.42(**1.69**×)	18.18(1.60×)
	75%	30.86(2.69×)	31.65(2.52×)	31.87(2.63×)	32.98(**2.73**×)	28.34(2.50×)

We evaluate five LLM models on 11 benchmarks: MMLU, ARC-Easy, ARC-Challenge, BoolQ, PIQA, WinoGrande, CommonsenseQA, BBH, GSM8K, MATH-500, and MBPP. End-to-end latency and decoding throughput, including correction operations, are measured on an RTX 5090 desktop GPU and a Jetson AGX Orin edge device. Appendix A.7 provides the detailed experimental settings.

4.2 Accuracy

The target MAC sparsity in Table 2 covers Transformer blocks only, excluding LM-head sparsity. Among the methods in Table 2, RoShift achieves the highest mean accuracy in 13 of the 15 configurations spanning five models and three sparsity levels. At both 50% and 60% sparsity, it outperforms TEAL, LaRoSA, and WINA on all five models. Averaged across models, the gains over TEAL are 1.60 and 3.29 percentage points at 50% and 60% sparsity, respectively; the corresponding gains are 3.76 and 8.76 percentage points over LaRoSA, and 2.22 and 5.66 percentage points over WINA. Note that the measured MAC sparsity of TEAL's fixed-threshold implementation is, on average across models, 0.83–1.68 percentage points below its target, which may partly explain its relatively strong accuracy compared with WINA. See Appendix A.7.2 for details. These results show that RoShift's accuracy preservation advantage becomes more pronounced as sparsity increases.

Subsequent speedup and accuracy-speedup Pareto analyses use uniform sparsity for controlled runtime comparisons. The corresponding accuracy results are reported in Appendix A.9.2, while Appendix A.9.3 compares uniform and non-uniform sparsity for RoShift.

4.3 Speedup

Table 3 reports RoShift's end-to-end decoding throughput with FP16, batch size 1, and uniform sparsity. RoShift accelerates decoding over Dense for every evaluated model and sparsity setting on both the RTX 5090 and Jetson AGX Orin. At 50% sparsity, it achieves speedups of up to 1.72× and 1.69× on the RTX 5090 and Orin, respectively; at 75% sparsity, these increase to 2.60× and 2.73×.

RoShift additionally stores a transition matrix $Q_l^\top Q_{l+1}$, a lazy-shift correction matrix, and four fused matrices per layer. For Llama-3.1-8B at 50% sparsity and a 4K context length, these additions increase peak VRAM usage by 29.31% over standard dense execution. Correction and transition add an average computational overhead of 14.04%, while LM-head sparsification saves 10.49%, resulting in a net overhead of 3.55%. The end-to-end decoding speedups in Table 3 include all these overheads.

4.4 Accuracy–Speedup Trade-off

Figure 4 compares the end-to-end accuracy-speedup trade-offs of RoShift and TEAL using measured end-to-end decoding speedups. Because speed is measured under uniform sparsity, we evaluate accuracy with the same uniform sparsity settings to construct each method's Pareto frontier. RoShift additionally sparsifies the LM-head, and its effect is included in both decoding speedup and generation benchmark evaluations. LM-head sparsification is a RoShift-specific component enabled by its rotation.

RoShift not only preserves higher accuracy at the same sparsity, but also offers a better accuracy-speedup trade-off than TEAL when compared using measured end-to-end decoding speedups. Specif-

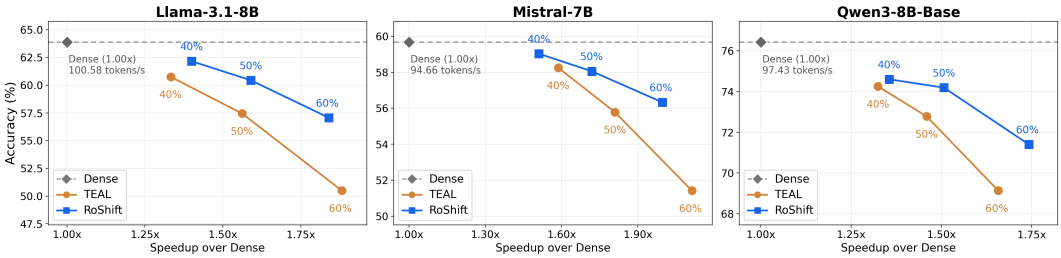


Figure 4: Accuracy-speedup trade-offs for three models on the RTX 5090. Accuracy is averaged over 11 benchmarks at uniform sparsity; speedup is the end-to-end decoding speedup over each model's Dense baseline.

ically, it achieves higher speedup at comparable accuracy or higher accuracy at comparable speedup, yielding a better Pareto frontier. This shows that lazy shift and weight-aware rotation reduce sparsification-induced accuracy loss, while correction fusion and efficient GPU execution translate these benefits into greater end-to-end speedup for a given accuracy loss.

4.5 Memory–Speedup Trade-off

RoShift uses a 5% tolerance for the relative difference between the actual RMS(X) and its representative. Figure 5 shows end-to-end decoding latency versus fused-matrix storage under this tolerance. Storing more fused matrices increases RMS(X) matches, reducing exact-correction fallbacks and latency. Beyond a certain point, however, most inputs already use fused matrices, so additional memory yields diminishing returns. Considering this trade-off, RoShift uses an average of four fused matrices per layer by default. Appendix A.8 provides the tolerance calibration and detailed analysis.

Fused-matrix storage can be adjusted to deployment memory constraints. With limited VRAM, a smaller cache reduces additional memory use; with sufficient VRAM, storing more fused matrices reduces fallbacks and increases speedup.

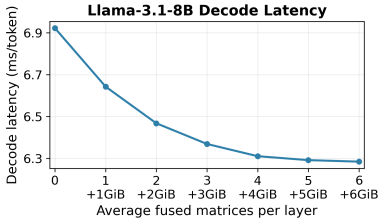


Figure 5: Decoding latency versus fused-matrix cache size for Llama-3.1-8B at 50% sparsity. The x-axis shows the average number of fused matrices per layer. The GiB annotations exclude the fixed 2 GiB for transition and correction matrices.

4.6 Ablation Studies

Accuracy. Table 4 compares mean accuracy over 11 benchmarks on Llama-3.1-8B under uniform sparsity for variants of rotation, input selection, and lazy shift. Top-K uses activation magnitudes; WA-Top-K also uses weight norms. A-Rot. and AW-Rot. denote activation-only and activation- and weight-aware rotations. The LM-head remains dense in all configurations to isolate Transformer-layer sparsification; the effect of LM-head sparsification is evaluated separately in Appendix A.4.

Without rotation, WA-Top-K achieves higher accuracy than Top-K at 50% and 60% sparsity, with a larger gain at 60%. In contrast, A-Rot. + Top-K does not consistently outperform Top-K without rotation. Replacing Top-K with WA-Top-K under A-Rot. does not produce a consistent gain and lowers accuracy at 50% and 60% sparsity. However, when all configurations use WA-Top-K, AW-Rot. consistently achieves higher accuracy than both A-Rot. and no rotation, with the gains over A-Rot. becoming more pronounced at higher sparsity. These results highlight a limitation of simply combining activation-only rotation with weight-aware selection; RoShift addresses this limitation by incorporating weight information into the rotation itself.

Lazy shift provides substantial gains even without rotation. Notably, lazy shift also achieves substantial gains with standard Top-K, indicating that its effectiveness

Table 4: Accuracy ablation on Llama-3.1-8B at uniform sparsity. Top-K uses activation magnitudes; WA-Top-K also uses weight norms. A-Rot. and AW-Rot. denote activation-only and activation- and weight-aware rotations.

Method	**40%**	**50%**	**60%**
Dense	63.87	63.87	63.87
Top-K	58.81	52.44	41.28
WA-Top-K	58.75	52.97	42.57
A-Rot. + Top-K	58.55	53.21	39.40
A-Rot. + WA-Top-K	58.69	52.63	37.60
AW-Rot. + WA-Top-K	59.47	54.34	44.92
Lazy shift + Top-K	61.87	58.59	51.79
Lazy shift + WA-Top-K	61.70	58.95	53.28
RoShift	**62.09**	**60.41**	**57.08**

does not depend on weight-aware selection. Adding lazy shift to WA-Top-K yields higher accuracy than adding AW-Rot., indicating that lazy shift is a major contributor to RoShift's accuracy gains. Moreover, RoShift, which carefully integrates AW-Rot. and lazy shift, achieves the highest accuracy at every sparsity level. These results show that lazy shift provides substantial standalone gains, with activation- and weight-aware rotation offering complementary benefits that further improve accuracy.

Speedup. Figure 6 compares decoding speedup over Dense as RoShift's optimizations are added sequentially at 50% sparsity. Step 0 uses sparse kernels but computes rotation transition and shift correction separately, keeping the LM-head dense. Step 1 introduces fused matrices associated with RMS representatives, and Step 2 executes the sparse FFN and correction on separate CUDA streams. Finally, Step 3 adds LM-head sparsification.

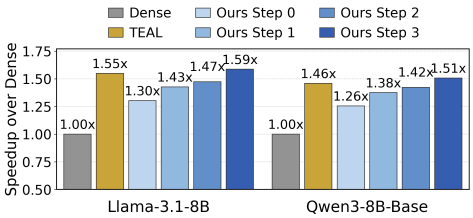


Figure 6: Cumulative decoding speedup over Dense at 50% uniform sparsity.

In this cumulative ordering, correction fusion provides a substantial additional speedup. When a fused matrix is selected, it replaces the two $d \times d$ matrix-vector products for rotation transition and shift correction with one, eliminating one matrix's weight reads and computations. LM-head sparsification also provides substantial acceleration because the LM-head uses a large weight matrix to project from the model dimension to the entire vocabulary. The incremental speedups differ across models, reflecting differences in factors such as the LM-head's share of computation and the frequency of fused-matrix use.

4.7 Compatibility with Quantization

To evaluate compatibility with weight quantization, we apply 8-bit and 4-bit quantization to RoShift using RTN, GPTQ, AWQ, and OmniQuant (Frantar et al., 2022; Lin et al., 2024; Shao et al., 2024). Table 5 reports PPL with OmniQuant. We quantize the model weights and inter-layer transition matrices $Q_l^\top Q_{l+1}$ to either 8 or 4 bits, or use 4-bit weights while retaining the transition matrices at 8 or 16 bits. All sparse models use 50% uniform sparsity.

Table 5: PPL of RoShift under OmniQuant. W and T denote the bit widths of the model weight and transition matrix $Q_l^\top Q_{l+1}$, respectively. Baseline uses FP16.

Method	Baseline	W8T8	W4T16	W4T8	W4T4
0%	4.99	5.07	5.38	5.52	392.60
50%	5.82	5.73	6.04	6.14	282.01

With 8-bit quantization (W8T8), both the dense and 50%-sparse models retain essentially the same PPL as baseline models, indicating that RoShift is highly compatible with 8-bit quantization. In contrast, jointly quantizing both the weights and the transition matrices $Q_l^\top Q_{l+1}$ to 4 bits causes severe degradation, suggesting that the transition matrices are particularly sensitive to low-bit quantization.

Keeping transition matrices at 8 or 16 bits while quantizing the weights to 4 bits largely restores the PPL. Combining RoShift with weight quantization could thus provide additional inference acceleration, but requires kernels jointly optimized for sparse and quantized computation, which we leave to future work. Appendix A.10 provides complete results, including RTN, GPTQ, and AWQ.

5 Conclusion

We proposed RoShift, a training-free activation sparsification method for efficient LLM inference. Lazy shift recenters activation-function outputs and defers the required correction, enabling more activations to be sparsified while maintaining comparable accuracy. Activation- and weight-aware rotation further reduces the loss of important information during sparsification. Finally, correction fusion and optimized GPU execution mitigate the overhead introduced by these transformations, translating the resulting activation sparsity into practical end-to-end decoding acceleration.

Our experiments show that RoShift preserves higher accuracy than the strong baselines TEAL, LaRoSA and WINA at high sparsity and achieves decoding speedups over dense models on real hardware. Its accuracy-speedup Pareto frontier provides a better trade-off than that of TEAL, showing that RoShift's benefits extend beyond accuracy preservation to practical execution efficiency. These results highlight RoShift's potential for latency- and efficiency-sensitive environments, including edge and on-device settings and Physical AI platforms.

AI USE STATEMENT

Generative AI tools assisted with research-methodology discussion, experiment design, software implementation, result interpretation, literature discovery, and manuscript preparation, including translation. The research ideas and mathematical derivations were primarily developed by the authors, with AI tools providing assistance in reviewing the mathematical proofs. The authors review and validate all AI-assisted code, experiments, mathematical statements, citations, and manuscript text and take responsibility for the final submission.

REPRODUCIBILITY STATEMENT

We describe the experimental setup and evaluation protocols in Section 4.1 and Appendix A.7, the kernel design and decoding measurement procedure in Appendix A.5, and the sparsity allocation and rotation search algorithms in Appendix A.6. Code and experimental configurations, including those for reproducing the reported accuracy and decoding speedup results, are provided in an anonymized ZIP archive as supplementary material. We will release an official public code repository upon acceptance.

REFERENCES

Saleh Ashkboos, Maximilian Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*, pp. 11682–11701, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/316648eb8b4ffb6010f531b07848c300-Abstract-Conference.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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

Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca, 2023.

Sihan Chen, Dan Zhao, Jongwoo Ko, Colby Banbury, Huiping Zhuang, Luming Liang, Pashmina Cameron, and Tianyi Chen. WINA: Weight informed neuron activation for accelerating large language model inference. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=l7Vb3yxmuz.

Vui Seng Chua, Yujie Pan, and Nilesh Jain. Post-training statistical calibration for higher activation sparsity. In *Fourth NeurIPS Workshop on Efficient Natural Language and Speech Processing (ENLSP-IV)*, 2024. URL https://neurips2024-enlsp.github.io/papers/paper_7.pdf.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Donghyun Lee, Je-Yong Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. CATS: Contextually-aware thresholding for sparsity in large language models. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=v3w2a7EInO.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

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

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. Training-free activation sparsity in large language models. In *International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2408.14690.

Kai Liu, Bowen Xu, Shaoyu Wu, Xin Chen, Hao Zhou, Yongliang Tao, and Lulu Hu. La RoSA: Enhancing LLM efficiency via layerwise rotated sparse activation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 39968–39986. PMLR, 2025b. URL https://proceedings.mlr.press/v267/liu25cj.html.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. 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*, pp. 22137–22176. PMLR, 2023. URL https://proceedings.mlr.press/v202/liu23am.html.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21702–21720. Curran Associates, Inc., 2023. doi: 10.52202/075280-0950. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/44956951349095f74492a5471128a7e0-Paper-Conference.pdf.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. URL https://arxiv.org/abs/1609.07843.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. ReLU strikes back: Exploiting activation sparsity in large language models. In *International Conference on Learning Representations*, pp. 25378–25400, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6cf669c222ad13f60d503736fb2bd15b-Abstract-Conference.html.

PyTorch Team. gpt-fast: Simple and efficient PyTorch-native transformer text generation. GitHub repository, 2023. URL https://github.com/meta-pytorch/gpt-fast.

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Gao Peng, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 45472–45496, 2024.

Jiho Shin, Hoeseok Yang, and Youngmin Yi. Grasp: Group-based prediction of activation sparsity for fast LLM inference. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7. IEEE, 2025a. doi: 10.1109/DAC63849.2025.11132899. URL https://doi.org/10.1109/DAC63849.2025.11132899.

Jiho Shin, Hoeseok Yang, and Youngmin Yi. SparseInfer: Training-free prediction of activation sparsity for fast LLM inference. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7. IEEE, 2025b. doi: 10.23919/DATE64628.2025.10992997. URL https://doi.org/10.23919/DATE64628.2025.10992997.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. ProSparse: Introducing and enhancing intrinsic activation sparsity within large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 2626–2644. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.coling-main.180/.

Geunsoo Song, Hoeseok Yang, and Youngmin Yi. oFFN: Outlier and neuron-aware structured FFN for fast yet accurate LLM inference. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1301–1315. ACM, 2026. doi: 10.1145/3779212.3790194. URL https://doi.org/10.1145/3779212.3790194.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. PowerInfer: Fast large language model serving with a consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 590–606. ACM, 2024. doi: 10.1145/3694715.3695964. URL https://doi.org/10.1145/3694715.3695964.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421/.

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

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

Ming Wang, Miao Zhang, Xuebo Liu, and Liqiang Nie. Weight-aware activation sparsity with constrained Bayesian optimization scheduling for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1086–1098. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.57. URL https://aclanthology.org/2025.emnlp-main.57/.

Shigeng Wang, Chao Li, Yangyuxuan Kang, Jiawei Fan, Zhonghong Ou, and Anbang Yao. SliderQuant: Accurate post-training quantization for LLMs. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2603.25284.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. ReLU^2 wins: Discovering efficient activation functions for sparse LLMs. *arXiv preprint arXiv:2402.03804*, 2024. URL https://arxiv.org/abs/2402.03804.

Zhenyu Zhang, Zechun Liu, Yuandong Tian, Harshit Khaitan, Zhangyang Wang, and Steven Li. R-Sparse: Rank-aware activation sparsity for efficient LLM inference. In *International Conference on Learning Representations*, pp. 77452–77466, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c0c165157df7e3082f9c6d70d3a4b6e9-Abstract-Conference.html.

A Appendix

A.1 Related Work

LLM Activation Sparsity: DejaVu observes contextual sparsity in LLMs, where a subset of neurons and attention heads can approximate the output for a given input, and predicts this subset using lightweight predictors (Liu et al., 2023). Building on this prediction-based approach to exploiting output sparsity, SparseInfer, Grasp, and oFFN use low-bit activation and weight information to reduce prediction cost while improving accuracy (Shin et al., 2025b;a; Song et al., 2026). Other work optimizes execution based on neuron activation patterns. PowerInfer and oFFN distinguish hot and cold neurons by activation frequency and assign suitable hardware resources and execution paths to each (Song et al., 2024; 2026).

Sparsifying Non-ReLU LLMs: Modern LLMs using SiLU exhibit little natural activation sparsity. ReLU Strikes Back and ProSparse replace the activation function with ReLU and use additional training to obtain sparsity while preserving model performance (Mirzadeh et al., 2024; Song et al., 2025). Training-free alternatives induce sparsity by approximating small-magnitude activations as zero. CATS applies this approach to FFNs, while TEAL extends it to attention projections for model-wide sparsification (Lee et al., 2024; Liu et al., 2025a). R-Sparse incorporates SVD-based low-rank approximation to compensate for sparsification-induced output error (Zhang et al., 2025).

Weight-aware Sparsification: Beyond activation magnitude, several methods assess component importance using the corresponding weights. ReLU^2 Wins evaluates neuron importance through its output contribution and measures the cumulative error from removal using CETT (Zhang et al., 2024). From this output-contribution perspective, WAS and WINA improve sparsification criteria by jointly considering activation magnitudes and the corresponding weight norms (Wang et al., 2025; Chen et al., 2026).

Sparsity-friendly Transformations: An alternative to improving removal criteria is to transform activations into representations more amenable to sparsification. SliceGPT uses PCA-based rotation to concentrate important activation information in a subset of dimensions, then compresses the model by removing less important dimensions and their corresponding weights (Ashkboos et al., 2024). LaRoSA applies rotation-based representation transformations to activation sparsification, combining layer-wise PCA rotation with Top-K selection of rotated activations (Liu et al., 2025b). SCAP applies mode-centering to activation-function outputs in non-GLU FFNs, shifting the distribution toward zero to increase the number of small-magnitude activations and facilitate sparsification (Chua et al., 2024).

Unlike prior transformation-based approaches, RoShift introduces lazy shift to exploit recentering-induced sparsity throughout the gated FFN, constructs activation- and weight-aware rotations tailored to sparsification error, and carefully integrates the two transformations through correction fusion to realize practical end-to-end decoding acceleration.

A.2 Lazy Shift Correction Overhead

Lazy shift adds a correction term XC to each FFN. Let d be the model dimension, k the FFN intermediate dimension, and L the number of layers. Each correction matrix C contains d^2 entries per layer, whereas the Gate, Up, and Down projections together contain $3dk$ parameters. The additional number of stored matrix entries ΔP across all correction matrices and its ratio to the FFN parameter count are therefore

$$\Delta P = Ld^2, \qquad \frac{\Delta P}{P_{\mathrm{FFN}}} = \frac{Ld^2}{3Ldk} = \frac{d}{3k}.$$

Because both correction and FFN parameter counts scale linearly with the number of layers, the per-layer ratio also applies to the full FFN stack. The model-wide ratio is lower, however, because the full model also contains attention, embedding, LM-head, and normalization parameters. Table 6 compares these two ratios using each model's actual parameter count. Here, additional parameters refer to entries in correction matrices precomputed from existing weights, not newly learned parameters. The table reports the storage overhead of C only; the rotation transition matrix introduces the same additional storage.

Table 6: Storage overhead of lazy-shift correction matrices. Added parameters sum C across all layers. Model-relative and FFN-relative increases use the full Dense model parameter count and the combined Gate, Up, and Down parameter count as denominators, respectively. Ratios are computed before rounding.

Model	Dense params (A)	Added C params (B)	Model increase (B/A)	FFN increase ($d/(3k)$)
Llama-3.1-8B	8.03B	0.54B	6.69%	9.52%
Llama-2-7B	6.74B	0.54B	7.97%	12.40%
Mistral-7B	7.24B	0.54B	7.41%	9.52%
Qwen2.5-7B	7.62B	0.36B	4.72%	6.31%
Qwen3-8B	8.19B	0.60B	7.37%	11.11%

A.3 Rotation Derivations

This section derives RoShift's rotation construction. We begin with activation-only PCA rotation, extend it to incorporate the corresponding weights for a single linear projection, and then generalize to a layer-wise rotation shared by multiple projections in a Transformer block.

A.3.1 Activation-only Rotation

Let $X \in \mathbb{R}^{N\times d}$ denote activation samples collected during calibration and $\mu \in \mathbb{R}^d$ their mean across samples. The centered activations X_c and empirical covariance matrix S are defined as follows.

$$X_c = X - \mathbf{1}\mu^\top \tag{9}$$

$$S = \frac{1}{N} X_c^\top X_c \in \mathbb{R}^{d\times d} \tag{10}$$

For a unit direction $q \in \mathbb{R}^d$ with $q^\top q = 1$, the variance of the projected activations is

$$\mathrm{Var}(Xq) = \frac{1}{N}\|X_c q\|_2^2 = q^\top S q \tag{11}$$

The first principal component is therefore obtained by maximizing

$$\max_{q^\top q=1} q^\top S q. \tag{12}$$

Defining the Lagrangian as in Eq. (13), the stationarity condition yields Eq. (14).

$$\mathcal{L}(q,\lambda) = q^\top S q - \lambda(q^\top q - 1) \tag{13}$$

$$Sq = \lambda q \tag{14}$$

The eigenvector associated with the largest eigenvalue therefore solves Eq. (12). Requiring each subsequent axis to be orthogonal to the previously selected axes yields an orthogonal rotation formed from the eigenvectors of S in descending eigenvalue order:

$$Q = [q_1, \ldots, q_d], \qquad Q^\top Q = I \tag{15}$$

Thus, in the activation-only case, a single covariance matrix S suffices to construct the rotation.

A.3.2 Activation- and Weight-aware Rotation for a Single Projection

For activation sparsification, both activation variance and its influence on the projection output through the corresponding weights matter. Consider a linear projection with weights W and an orthogonal rotation $Q = [q_1, \ldots, q_d]$. We have

$$Y = XW, \qquad W \in \mathbb{R}^{d\times d_{\text{out}}} \tag{16}$$

$$Y = (XQ)(Q^\top W) = \sum_{j=1}^{d} (Xq_j)(q_j^\top W) \tag{17}$$

The contribution of a single rotation axis q to the projection output is therefore

$$Y_q = (Xq)(q^\top W) \tag{18}$$

We measure this axis's output contribution by the sum of variances across all output dimensions:

$$e(q) = \mathbb{E}\left[\|Y_q - \mathbb{E}[Y_q]\|_2^2\right] = \operatorname{tr}\left(\operatorname{Cov}(Y_q)\right). \tag{19}$$

Since W is fixed and $q^\top W$ is independent of the sample for a given q,

$$e(q) = \operatorname{Var}(Xq)\|q^\top W\|_2^2 \tag{20}$$

$$= (q^\top S q)(q^\top W W^\top q). \tag{21}$$

Defining the weight Gram matrix M as follows rewrites Eq. (21) as Eq. (23).

$$M = WW^\top \tag{22}$$

$$e(q) = (q^\top S q)(q^\top M q) \tag{23}$$

Accounting for projection weights thus extends the activation-only PCA objective $q^\top S q$ to a product of two quadratic forms.

We maximize Eq. (23) subject to $q^\top q = 1$. Define

$$\alpha(q) = q^\top M q, \qquad \beta(q) = q^\top S q \tag{24}$$

Then

$$\nabla_q e(q) = 2\left[\alpha(q) S + \beta(q) M\right] q. \tag{25}$$

The Lagrange stationarity condition is therefore

$$\left[\alpha(q) S + \beta(q) M\right] q = \lambda q \tag{26}$$

Unlike PCA, Eq. (26) does not directly provide a single fixed eigensystem, because $\alpha(q)$ and $\beta(q)$ depend on the rotation axis q being sought. For the full rotation $Q = [q_1, \ldots, q_d]$, the coefficients

$$\alpha_j = q_j^\top M q_j, \qquad \beta_j = q_j^\top S q_j \tag{27}$$

vary across axes, so the matrices $\alpha_j S + \beta_j M$ generally differ as well. Optimizing the axes independently does not automatically guarantee mutual orthogonality.

To obtain a single fixed eigensystem, RoShift approximates these axis-dependent coefficients with mixture coefficients shared across all axes. In the following, we use trace-normalized S and M to reduce scale differences between the statistics and construct

$$B = \theta_S S + \theta_M M, \qquad \theta_S, \theta_M \geq 0 \tag{28}$$

Since B is a fixed symmetric matrix, its eigendecomposition

$$BQ = Q\Lambda \tag{29}$$

provides a complete orthogonal rotation. For a single projection, this extends PCA based on one activation covariance matrix to a construction that jointly considers two statistics: activation covariance and the weight Gram matrix.

A.3.3 Extension to Multiple Transformer Projections

In a Transformer block, a single layer-wise rotation Q_l affects multiple projections. Since the Q/K/V and Gate projections to which RoShift applies input sparsity share the same rotation, their output contributions must be considered jointly.

Let S_l^{att} and S_l^{ffn} denote the attention-input and FFN-input covariance matrices at layer l, respectively. Q/K/V share the attention input, while Gate uses the FFN input. The projection-specific weight Gram matrices are defined as

$$M_l^Q = W_l^Q (W_l^Q)^\top, \qquad M_l^K = W_l^K (W_l^K)^\top, \tag{30}$$

$$M_l^V = W_l^V (W_l^V)^\top, \qquad M_l^G = W_l^G (W_l^G)^\top \tag{31}$$

Extending the single-projection objective in Eq. (23) to these four projections gives the following joint output-contribution objective for an axis q:

$$E_l(q) = (q^\top S_l^{\text{att}} q) \left[q^\top (M_l^Q + M_l^K + M_l^V) q \right] + (q^\top S_l^{\text{ffn}} q)(q^\top M_l^G q). \tag{32}$$

Defining

$$M_l^{\text{att}} = M_l^Q + M_l^K + M_l^V \tag{33}$$

the gradient of Eq. (32) is

$$\nabla_q E_l(q) = 2 \Big[\alpha_l^{\text{att}}(q) S_l^{\text{att}} + \beta_l^{\text{att}}(q) M_l^{\text{att}} + \alpha_l^{\text{ffn}}(q) S_l^{\text{ffn}} + \beta_l^{\text{ffn}}(q) M_l^G \Big] q, \tag{34}$$

where

$$\alpha_l^{\text{att}}(q) = q^\top M_l^{\text{att}} q, \qquad \beta_l^{\text{att}}(q) = q^\top S_l^{\text{att}} q, \tag{35}$$

$$\alpha_l^{\text{ffn}}(q) = q^\top M_l^G q, \qquad \beta_l^{\text{ffn}}(q) = q^\top S_l^{\text{ffn}} q. \tag{36}$$

Expanding M_l^{att} into the Q/K/V Gram matrices expresses Eq. (34) as a q-dependent linear combination of six statistics:

$$S_l^{\text{att}}, \quad S_l^{\text{ffn}}, \quad M_l^Q, \quad M_l^K, \quad M_l^V, \quad M_l^G \tag{37}$$

Thus, the two statistics for a single projection, activation covariance and the weight Gram matrix, become six statistics when a layer-wise rotation is shared across multiple Transformer projections: two activation covariance matrices and four weight Gram matrices.

As in the single-projection case, the coefficients still depend on the rotation axis q, so they do not directly yield a fixed eigensystem. RoShift approximates these axis-dependent coefficients with mixture coefficients shared within each layer.

Denote the six statistics by $A_{l,i}$, $i = 1, \ldots, 6$. To reduce differences in scale, we apply trace normalization:

$$B_{l,i} = \frac{A_{l,i}}{\operatorname{tr}(A_{l,i})} \tag{38}$$

We then construct

$$B_l = \sum_{i=1}^{6} \theta_{l,i} B_{l,i}, \qquad \theta_{l,i} \geq 0 \tag{39}$$

and use the eigenvectors of B_l in descending eigenvalue order as the columns of Q_l.

To account for RMSNorm, activation covariances are computed from activations normalized by inverse RMS. The learned channel-wise scales are absorbed into the corresponding projection weights before constructing their Gram matrices.

In summary, RoShift's rotation construction extends progressively as follows.

$$\underbrace{S}_{\text{Activation-only PCA}} \longrightarrow \underbrace{\{S, M\}}_{\text{Single projection}} \longrightarrow \underbrace{\{S_l^{\text{att}}, S_l^{\text{ffn}}, M_l^Q, M_l^K, M_l^V, M_l^G\}}_{\text{Transformer layer}}. \tag{40}$$

A.4 Sparse LM-head

A.4.1 LM-head Sparsification

The LM-head maps the final activations to logits over the entire vocabulary to determine the next token. This does not require completing the computation for every candidate. Greedy decoding selects the candidate with the largest logit, while Top-K and Top-P sampling primarily use a subset of high-logit candidates. Instead of applying the same input sparsity to every candidate, RoShift reduces LM-head weight reads and computations by terminating computation early for candidates unlikely to be selected.

A.4.2 Rotation for Candidate Selection

RoShift constructs a dedicated orthogonal LM-head rotation from final-activation statistics, concentrating energy in the leading dimensions. This rotation is also absorbed into the corresponding LM-head weights, so the full logit computation before candidate pruning is algebraically equivalent to the original computation. In this representation, partial logits computed from the leading dimensions provide rough estimates of final logits. Computation for a vocabulary candidate is therefore terminated if its intermediate logit is too small.

A.4.3 Staged Candidate Selection

Figure 7 illustrates the computations skipped through staged candidate selection. We divide the rotated input dimensions into three segments. The first stage computes partial logits for all vocabulary candidates. Let m and σ be the maximum and standard deviation of the accumulated partial logits among the current candidates. Only candidates with partial logits at least $m - z\sigma$ advance to the next stage. The stage-specific coefficients z and segment boundaries are determined through calibration. We accumulate contributions from the next segment for the surviving candidates and repeat the selection process. Only final survivors complete the remaining dimensions to obtain full logits. Subsequent computations for pruned candidates are skipped, and their final logits are set to $-\infty$. Finer segmentation can eliminate candidates earlier and skip more computation, but incurs overhead from inspecting accumulated logits and selecting candidates after each segment. We therefore use the three-stage configuration that empirically provides the best trade-off.

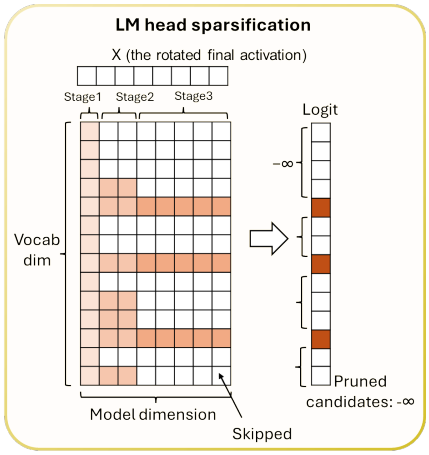


Figure 7: Three-stage LM-head candidate selection. Only shaded regions are computed. Subsequent computations for pruned candidates are skipped, and their logits are set to $-\infty$.

A.4.4 Experimental Validation

For all models, we calibrate the selection criteria so that candidate pruning changes the Top-1 prediction, the candidate with the largest logit, for fewer than 0.1% of calibration samples. Table 7 reports the resulting effective LM-head sparsity for each model. We define effective LM-head sparsity as the fraction of dense LM-head MACs skipped through candidate termination.

Table 8 compares effective LM-head sparsity with and without rotation. RoShift's rotation achieves higher sparsity for all five models, with particularly large gains for Llama-2-7B and Qwen2.5-7B. For Qwen3, the aggregate calibration Top-1 change is matched at 0.05371% relative to each basis's no-pruning reference, without a per-domain cap; no-rotation inputs are reconstructed from stored rotated activations.

Table 7: Effective LM-head sparsity and pruning-induced Top-1 prediction changes on the calibration set. Changes are measured relative to a no-pruning path with the same rotation, three-stage partition, and rounding.

Model	Effective sparsity (%)	Top-1 change (%)
Llama-3.1-8B	83.03	0.049
Llama-2-7B	75.82	0.034
Mistral-7B	71.64	0.048
Qwen2.5-7B	75.06	0.031
Qwen3-8B	71.42	0.054

Table 8: Effective LM-head sparsity (%) with and without RoShift's LM-head rotation. Increase is the difference between the two sparsity percentages (RoShift rotation minus no rotation).

Model	No rotation	RoShift rotation	Increase
Llama-3.1-8B	56.02	**83.03**	+27.01
Llama-2-7B	0.65	**75.82**	+75.17
Mistral-7B	55.17	**71.64**	+16.47
Qwen2.5-7B	1.84	**75.06**	+73.22
Qwen3-8B-Base	43.66	**71.42**	+27.76

TEAL-style input sparsification. We also evaluate simple TEAL-style input sparsification (Liu et al., 2025a) applied to the LM-head without rotation. Under a Top-1 prediction change constraint of less than 0.1%, the resulting effective LM-head sparsity ranges from 1.98% to 2.93% across the five models (Table 9). Thus, this simple input sparsification offers limited scope for reducing LM-head computation under the evaluated constraint.

Table 9: TEAL-style LM-head input sparsification without rotation, subject to a Top-1 prediction change below 0.1%. Effective sparsity is the fraction of dense LM-head MACs skipped by input sparsification.

Model	Effective sparsity (%)	Top-1 change (%)
Llama-3.1-8B	1.98	0.098
Llama-2-7B	2.12	0.090
Mistral-7B	2.93	0.099
Qwen2.5-7B	2.62	0.099
Qwen3-8B-Base	2.03	0.098

Downstream generation accuracy. We further evaluate LM-head sparsification on MBPP, GSM8K, and MATH-500 using Llama-3.1-8B, keeping the same RoShift Transformer at 50% non-uniform sparsity and changing only the LM-head. In addition to greedy decoding, we test two sampling configurations using temperature scaling and Top-P/Top-K filtering (Table 10). Average accuracy is largely preserved across these decoding settings, with changes ranging from -0.39 to $+0.47$ percentage points relative to the matched dense LM-head.

Table 10: Dense versus sparse LM-head accuracy (%) on Llama-3.1-8B with the same RoShift Transformer at 50% non-uniform sparsity. Parentheses show differences in percentage points from the dense head under the same decoding setting. Avg is the unweighted mean of the three benchmarks.

Method	MBPP	GSM8K	MATH-500	Avg
Dense head (Greedy)	44.60	45.03	18.20	35.94
Sparse head (Greedy)	44.60 (+0.00)	44.66 (-0.38)	18.00 (-0.20)	**35.75 (-0.19)**
Dense head (Sampling A)	41.73	42.43	15.27	33.14
Sparse head (Sampling A)	40.07 (-1.67)	43.72 (+1.29)	14.47 (-0.80)	**32.75 (-0.39)**
Dense head (Sampling B)	42.53	43.06	16.20	33.93
Sparse head (Sampling B)	43.20 (+0.67)	43.62 (+0.56)	16.40 (+0.20)	**34.41 (+0.47)**

Sampling A: $T = 0.6$, Top-P $= 0.9$, Top-K $= 0$ (disabled). Sampling B: $T = 0.7$, Top-P $= 0.8$, Top-K $= 20$. Generation is capped at 256 tokens. Sampling scores use each benchmark's full evaluation set and are averaged over seeds 0, 1, and 2.

A.5 Kernel Details

RoShift's kernels translate activation sparsity into actual reductions in weight reads and computations. Blockwise Top-K reduces selection cost, and sparse GEMV operates only on selected components. Sparsity-aware weight layouts and 2D tiling improve execution, while kernel fusion and multiple streams reduce the overhead of additional operations.

A.5.1 BLOCKWISE TOP-K

Global Top-K requires a global reduction to merge results from multiple GPU thread blocks, potentially incurring substantial selection overhead. RoShift uses B128 Blockwise Top-K to select components independently within blocks of 128 activation dimensions. A fixed retained count per block keeps the subsequent sparse GEMV workload consistent. At 25%, 50%, and 75% sparsity, each block retains 96, 64, and 32 components, respectively.

If important components concentrate in a few blocks, however, accuracy can be lower than with Global Top-K. To mitigate this, we form selection blocks by applying a fixed random permutation to the rotated activation dimensions. This permutation can be incorporated into the corresponding rotation and weights offline. As Table 11 shows, the PPL increase over Global Top-K is small under the evaluated conditions.

Table 11: PPL comparison of Dense, Global Top-K, and B128 Blockwise Top-K on Llama-3.1-8B using the calibration set.

Sparsity	Global Top-K	B128 Top-K
Dense	4.99	4.99
40%	5.32	5.34
50%	5.63	5.70
60%	6.33	6.40

A.5.2 INDEX-BASED SPARSE GEMV AND 2D TILING

RoShift adapts oFFN's selective weight loading and index-based sparse execution to its input and output sparsity patterns (Song et al., 2026). Instead of assigning work to all dimensions and skipping pruned components, the kernels collect retained indices and perform weight reads and computations only for those components.

Input-sparse GEMV uses 2D tiling across groups of selected inputs and output segments. Each CUDA thread block computes the partial contribution of an input group to an output segment, and partial sums from different input groups are combined using atomic additions. This exposes parallelism along both the input and output dimensions. For output-sparse Up, work is assigned only to selected output channels, and each channel's dot product is computed.

Weight layouts are prepared offline to match these access directions. For input-sparse projections, weights associated with one input component are contiguous along the output dimension. For output-sparse Up, weights associated with one output channel are contiguous along the input dimension. Thus, even though selected indices are irregular, neighboring threads access contiguous weights for each selected component.

A.5.3 MULTI-STREAM EXECUTION

The sparse FFN and correction path share the same FFN input but do not depend on each other's outputs. RoShift therefore executes them on separate CUDA streams once the input is ready, using CUDA events to synchronize immediately before summing their outputs.

A.5.4 DECODING MEASUREMENTS

Dense, TEAL, and RoShift are evaluated in a gpt-fast-based execution environment (PyTorch Team, 2023). Dense uses dense projections, TEAL uses threshold-based Triton sparse GEMV, and RoShift integrates the kernels described above into the same environment. Dense and TEAL use dense LM-heads, whereas RoShift uses a separately calibrated sparse LM-head.

The results in Table 3 are measured with FP16 and batch size 1. For each of WikiText-2 (Merity et al., 2016), Alpaca, GSM8K, and OpenBookQA (OBQA) (Mihaylov et al., 2018), we use one fixed prompt and teacher-forced continuation. Following a 1,024-token dense prefill, CUDA events measure only the 128-token decoding interval. All methods use Math SDPA, `torch.compile`, and CUDA Graphs. We average per-token latency over four repeated measurements per dataset and across all datasets.

A.6 Sparsity Allocation and Rotation Search

RoShift first determines the sparsity allocation under a target MAC sparsity, then fixes it before searching for rotations. Both searches screen candidates using local reconstruction and select the final configuration based on full-model calibration PPL.

A.6.1 Sparsity Allocation

TEAL assigns different sparsities to projections within each block but keeps total sparsity uniform across blocks (Liu et al., 2025a). WAS permits inter-block allocation but restricts sparsity to be monotonically non-decreasing with depth (Wang et al., 2025). These constraints reduce the search space but cannot express allocations with lower sparsity in later layers. RoShift is designed to explore such allocations while preserving the overall MAC sparsity constraint.

Jointly searching all layer and projection sparsities, however, leads to a rapidly growing number of candidate configurations. We therefore initialize with TEAL-style within-layer allocation and WAS-style inter-layer allocation, then refine the result using pairwise coordinate descent that transfers equal MAC budgets between two coordinates. This limits the variables changed in each update while allowing both layer-wise total sparsity and within-layer allocation to be adjusted.

Algorithm 1: Non-uniform Sparsity Allocation

Require: Target sparsity r, calibration data D, short calibration data D_s, transfer grid G

$s \leftarrow$ TEAL-style allocation followed by WAS-style allocation at target r
$p \leftarrow \mathrm{PPL}(s, D)$
while search budget remains **do**
  $(\mathcal{D}, \mathcal{R}) \leftarrow$ LocalErrorShortlist(s, D)
  $C \leftarrow$ FeasiblePairTransfers$(s, \mathcal{D}, \mathcal{R}, G)$ *Preserve MAC budget*
  $C \leftarrow$ ShortPPLScreen(C, D_s) *Retain each pair's best grid point and neighbors.*
  if C is empty **then**
    break
  $s_{\mathrm{new}} \leftarrow \arg\min_{c \in C} \mathrm{PPL}(c, D)$
  $p_{\mathrm{new}} \leftarrow \mathrm{PPL}(s_{\mathrm{new}}, D)$
  if the improvement criterion relative to p is not met **then**
    break
  $(s, p) \leftarrow (s_{\mathrm{new}}, p_{\mathrm{new}})$
return s

To reduce the cost of evaluating full calibration PPL for every candidate, we screen candidates successively using local error and short calibration PPL. Because local error alone cannot adequately capture error accumulation across layers, the final update is accepted only if it improves full calibration PPL. Algorithm 1 gives the procedure.

A.6.2 Rotation Search

An optimal sparsity-friendly rotation would vary across projections and depend on the sparsity assigned to each projection. We therefore search for layer-specific mixture coefficients under the fixed sparsity allocation to derive a single rotation for each layer. Determining allocation and rotation sequentially reduces the search space and ensures that rotation candidates are compared under identical sparsity settings.

Rather than evaluating all mixtures of the six components, rotation search redistributes mass between pairs of coefficients while preserving their sum. For each mixture, we eigendecompose the weighted sum of the statistic matrices and evaluate the candidate using the block output's relative L2 error against the dense output, $E_l(\theta)$. We repeatedly update the mixture while local error improves, reducing the step size when improvement stops or the update limit is reached.

Rotations that improve individual blocks need not provide the same benefit when combined. We therefore validate the candidates from shallow to deep layers, retaining each only if it improves full-model calibration PPL with all previously accepted changes in place. This sequential selection is performed exactly once.

Algorithm 2: Layer-wise Rotation Search

Require: Fixed s, statistics B, calibration D, decreasing steps Δ, update limit K

$\theta_0 \leftarrow (0.25, 0.25, 0.125, 0.125, 0.125, 0.125)$
$Q \leftarrow$ Eigenbases(B, θ_0)
for each layer l **do**
  $\theta \leftarrow \theta_0$; $e \leftarrow E_l(\theta)$
  for $\delta \in \Delta$ **do** ▷ Coarse to fine
    for $t = 1, \ldots, K$ **do**
      $H \leftarrow$ FeasiblePairTransfers(θ, δ)
      if $H = \varnothing$ **then break**
      $\theta' \leftarrow \arg\min_{h \in H} E_l(h)$; $e' \leftarrow E_l(\theta')$
      if $e' \geq e - 10^{-12}$ **then break**
      $(\theta, e) \leftarrow (\theta', e')$
  $\widehat{Q}_l \leftarrow$ Eigenbasis(B_l, θ)
$p \leftarrow \mathrm{PPL}(s, Q, D)$
for $l = 1, \ldots, L$ **do** ▷ One sequential pass
  $Q_{\mathrm{trial}} \leftarrow Q$ with $Q_l \leftarrow \widehat{Q}_l$
  $p_{\mathrm{trial}} \leftarrow \mathrm{PPL}(s, Q_{\mathrm{trial}}, D)$
  if $p_{\mathrm{trial}} < p - 10^{-12}$ **then**
    $(Q, p) \leftarrow (Q_{\mathrm{trial}}, p_{\mathrm{trial}})$
return Q

A.6.3 SEARCH TIME

On a single RTX 5090, sparsity allocation and rotation search take approximately 4.6 and 0.7 hours per sparsity setting, respectively, averaged across the five models. Table 12 reports the per-model estimates.

Table 12: Approximate search time (hours) per sparsity setting on a single RTX 5090, averaged over 40%, 50%, and 60% sparsity using combined local and server run records.

Model	Sparsity allocation	Rotation search
Llama-3.1-8B	5.2	0.8
Llama-2-7B	6.3	0.6
Mistral-7B	4.0	0.7
Qwen2.5-7B	4.8	0.7
Qwen3-8B	2.6	0.8
Model average	4.6	0.7

A.7 DETAILED EXPERIMENTAL SETTINGS

A.7.1 GENERAL EXPERIMENTAL SETTINGS

We evaluate five SiLU-based base LLMs: **Llama-3.1-8B, Llama-2-7B, Mistral-7B-v0.1, Qwen2.5-7B, and Qwen3-8B-Base** (Grattafiori et al., 2024; Touvron et al., 2023; Jiang et al., 2023; Yang et al., 2024; 2025). Accuracy experiments use target sparsities of 40%, 50%, and 60% and report the unweighted mean over 11 downstream benchmarks. All RoShift experiments use the same calibration set of 32 sequences of 4,096 tokens each: 16 from C4, six from Alpaca, five from MetaMathQA, and five from CodeAlpaca-20K (Raffel et al., 2020; Taori et al., 2023; Yu et al., 2023; Chaudhary, 2023). For the baselines, we use the calibration datasets and settings specified by their respective public implementations, following each method's intended calibration recipe rather than forcing a common calibration distribution.

Throughout our experiments, MAC sparsity denotes the fraction of linear-projection MACs eliminated by sparsification. This definition follows the same model-level sparsity formulation used by TEAL and WINA, accounting for the computational cost of each projection and thus directly reflecting the amount of computation skipped through activation sparsification.

Benchmark accuracy evaluation is based on EleutherAI's lm-evaluation-harness (lm-eval) (Gao et al., 2023). We evaluate MMLU, ARC-Easy, ARC-Challenge, BoolQ, PIQA, WinoGrande, CommonsenseQA, BBH, GSM8K, MATH-500, and MBPP (Hendrycks et al., 2021; Clark et al., 2018; 2019; Bisk et al., 2020; Sakaguchi et al., 2020; Talmor et al., 2019; Suzgun et al., 2022; Cobbe et al., 2021; Lightman et al., 2023; Austin et al., 2021). For log-likelihood evaluation, we use 0-shot prompting on PIQA, BoolQ, ARC-Easy, and CommonsenseQA, 5-shot prompting on WinoGrande and MMLU, and 25-shot prompting on ARC-Challenge. For generation, we use 3-shot prompting on MBPP and BBH, 8-shot prompting on GSM8K, and 4-shot prompting on MATH-500.

For log-likelihood benchmarks, Transformer sparsity is applied at every input-token position, while the LM-head remains dense. For generation benchmarks, the entire prompt prefill is dense, and Transformer sparsity is applied at every decoding step. RoShift uses a separately calibrated sparse LM-head during decoding, with its own sparsity setting independent of Transformer sparsity, as reported in Table 7. TEAL, LaRoSA, and WINA use the same prefill and decoding sparsity scope but keep the LM-head dense. The target MAC sparsity reported in Table 2 is defined over the Transformer blocks only and excludes LM-head sparsity; thus, RoShift's sparse LM-head does not contribute to the sparsity target used for the accuracy comparison.

A.7.2 Baseline Implementations and Modifications

We use TEAL as the primary speed baseline because it applies to SiLU-based LLMs without additional training and provides public FP16 sparse kernels and an end-to-end decoding implementation. We additionally include LaRoSA and WINA as accuracy baselines (Liu et al., 2025b; Chen et al., 2026). LaRoSA uses layer-wise rotations to concentrate activation energy. WINA combines activation magnitudes with the corresponding weight-column ℓ_2 norms and uses an offline SVD-based orthogonal transformation to enforce column orthogonality for its theoretical analysis. Thus, WINA is also related to RoShift through its use of weight information and rotation for activation sparsification.

We follow each baseline's public implementation and settings as closely as possible, with adjustments to token scope, runtime precision, and sparsity allocation. Regarding token scope, TEAL (Liu et al., 2025a) and WINA (Chen et al., 2026) exclude some initial tokens from sparsification in their original prefill evaluation settings because effects such as attention sinks (Xiao et al., 2024) can cause greater degradation at these positions. In our log-likelihood evaluation, we instead apply sparsity at every input-token position to evaluate robustness under the common token scope described in Section A.7.1. For generation benchmarks, the entire prompt prefill remains dense for all methods.

LaRoSA's public implementation applies the rotation matrix Q immediately before Top-K and restores the original basis after sparsification, rather than retaining activations in the rotated basis throughout execution. These runtime rotations use FP64; we change them to FP16 to reduce experimental runtime. We retain the original precision for the offline statistics used to construct the rotations and leave other operations unchanged. For comparison, RoShift absorbs its layer-wise rotations into the model weights offline; its online rotation-related operation is the transition between consecutive rotated bases, $Q_l^\top Q_{l+1}$, which is likewise executed in FP16.

For sparsity allocation, LaRoSA's original policy measures sparsity by counting activation elements. Because projections have different computational costs, this can yield a MAC reduction below the target sparsity. We therefore modify its allocation policy to match the target MAC reduction within the Transformer, consistent with the comparison criterion in Section A.7.1. TEAL and RoShift also use MAC-sparsity-based allocation. Table 26 separately reports benchmark accuracy and MAC sparsity under LaRoSA's original allocation policy.

WINA's public evaluation setup (Chen et al., 2026) uses activation-magnitude-based Top-K for its TEAL baseline, whereas we follow the original TEAL paper and official implementation (Liu et al., 2025a) by using fixed thresholds determined from calibration data. Under fixed thresholds, the achieved MAC sparsity may deviate from the target depending on the activation distribution of the evaluation data. Table 13 therefore reports TEAL's achieved MAC sparsity averaged over the 11 benchmarks. When comparing the relative performance of TEAL and WINA reported in our study and the original WINA paper, these differences in baseline implementation and achieved sparsity should be taken into account.

Table 13: Measured Transformer MAC sparsity (%) of TEAL, averaged over 11 benchmarks. Column headers indicate the target MAC sparsity.

Model	40%	50%	60%
Llama-3.1-8B	38.78	48.24	57.89
Llama-2-7B	40.00	49.43	59.14
Mistral-7B-v0.1	39.89	49.42	59.25
Qwen2.5-7B	38.81	48.18	57.94
Qwen3-8B-Base	38.39	47.89	57.39
Model average	**39.17**	**48.63**	**58.32**

A.8 Fused-matrix Cache and RMS Tolerance

For each layer, RoShift stores fused matrices corresponding to a small set of RMS representatives obtained during calibration. At inference time, a fused matrix is used only if the relative difference between the current RMS and its representative is at most 5%; otherwise, execution falls back to exact correction using the current RMS.

The 5% relative tolerance was selected through calibration experiments to avoid degrading model PPL. Figure 8 shows how PPL changes with the allowed relative error at 0% and 50% sparsity. Performance remains essentially unchanged at 5% in nearly all evaluated cases, supporting our choice of this tolerance. We select the 5% tolerance on Llama-3.1-8B and then fix it across all models.

Both the representatives and their counts per layer are determined from the calibration distribution. For each layer, we consider a set of candidate counts. For each fixed count, we select representative values that maximize the number of calibration samples covered within the tolerance. We then use these values in dynamic programming to allocate representative counts across layers so as to minimize the total number of fallbacks subject to the overall fused-matrix storage budget.

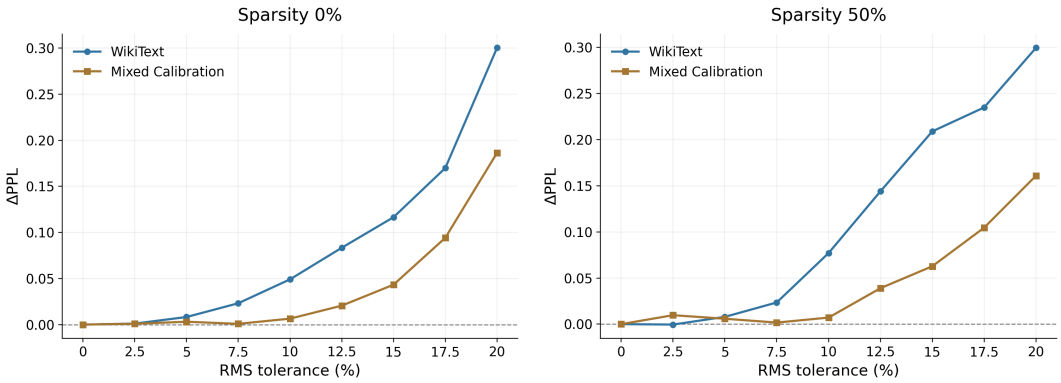


Figure 8: PPL changes versus relative RMS tolerance for Llama-3.1-8B at 0% (left) and 50% (right) sparsity, evaluated on WikiText (Merity et al., 2016) and Mixed Calibration. ΔPPL is the difference from the 0%-tolerance result in each setting.

A.9 Complete Experimental Results

A.9.1 Non-uniform Sparsity

Tables 14–18 provide the per-model results underlying Table 2. Sparsity denotes the target MAC reduction for linear projections in the Transformer. Entries report benchmark accuracy; bold indicates the highest score among the sparse methods compared at the same sparsity. Avg is the unweighted mean over 11 benchmarks.

RoShift improves average accuracy over TEAL by 0.35 percentage points across five models at 40% sparsity, with the gap increasing to 1.60 points at 50% and 3.29 points at 60%. Individual benchmark results show the same trend. Of the 55 model-benchmark combinations, RoShift outperforms TEAL in 33/55 at 40% sparsity, increasing to 50/55 at 50% and 54/55 at 60%. Thus, RoShift's accuracy preservation advantage becomes both larger and more consistent at higher sparsity.

Although our log-likelihood evaluation applies sparsity to all input tokens, unlike TEAL's original setting, RoShift's accuracy advantage is also evident on generation benchmarks, where TEAL, WINA, and RoShift all use dense prompt prefill. At 50% and 60% sparsity, RoShift's mean gains over TEAL are 1.82 and 3.70 percentage points on generation benchmarks, respectively, compared with 1.47 and 3.06 points on log-likelihood benchmarks. Similarly, at 50% and 60% sparsity, RoShift's mean gains over WINA are 3.20 and 7.98 percentage points on generation benchmarks, respectively, compared with 1.66 and 4.33 points on log-likelihood benchmarks.

Table 14: Llama-3.1-8B: complete non-uniform accuracy results. Avg is the unweighted mean of 11 benchmarks.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	65.57	82.49	58.28	83.12	81.12	78.22	70.60	64.52	48.90	21.60	48.20	63.87
TEAL 40%	62.95	80.22	**56.83**	**82.51**	**80.20**	76.87	68.47	61.91	46.32	19.40	44.40	61.83
LaRoSA 40%	62.38	79.21	55.72	81.65	79.60	77.03	**69.53**	60.34	43.90	16.80	**47.20**	61.21
WINA 40%	63.49	**81.57**	56.14	81.93	79.60	76.24	68.88	61.22	44.96	**19.60**	45.20	61.71
RoShift 40%	**63.70**	80.85	**56.83**	82.35	79.43	**77.19**	68.71	**62.92**	**47.23**	18.60	47.00	**62.26**
TEAL 50%	59.98	78.49	55.46	81.50	79.05	74.43	64.13	58.52	40.94	15.20	43.60	59.21
LaRoSA 50%	57.43	77.06	52.82	80.34	78.89	72.22	63.80	54.71	35.63	12.00	39.60	56.77
WINA 50%	59.28	78.79	55.03	81.71	79.43	75.30	63.80	56.80	36.77	15.00	41.80	58.52
RoShift 50%	**62.27**	**81.02**	**57.00**	**82.32**	**79.87**	**77.51**	**69.45**	**59.75**	**44.58**	**17.80**	**44.60**	**61.47**
TEAL 60%	52.81	74.20	49.15	78.10	76.71	70.40	56.51	52.14	30.02	11.00	36.80	53.44
LaRoSA 60%	46.24	69.74	43.69	75.75	74.97	64.48	48.48	46.92	17.51	7.60	27.00	47.49
WINA 60%	48.99	72.39	47.95	78.72	76.22	68.98	53.32	49.02	21.00	7.20	33.00	50.62
RoShift 60%	**58.88**	**78.66**	**52.47**	**81.93**	**78.62**	**73.80**	**66.83**	**56.17**	**39.80**	**15.00**	**41.80**	**58.54**

Table 15: Llama-2-7B: complete non-uniform accuracy results. Avg is the unweighted mean of 11 benchmarks.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	45.96	73.86	52.65	79.33	78.73	74.66	32.76	39.69	14.56	3.20	23.20	47.14
TEAL 40%	44.26	72.52	51.11	77.37	78.62	72.61	30.79	**39.58**	**14.48**	2.40	21.00	45.89
LaRoSA 40%	42.28	72.94	50.17	77.16	77.09	71.59	28.83	38.78	12.51	3.60	20.00	44.99
WINA 40%	**44.97**	**73.86**	**52.56**	77.65	**78.73**	73.95	**31.20**	38.15	13.04	**3.80**	**22.40**	**46.39**
RoShift 40%	43.48	73.27	52.30	**79.20**	78.51	**74.19**	28.67	39.20	13.87	3.00	20.60	46.03
TEAL 50%	41.31	71.55	50.00	75.35	77.69	71.43	27.44	37.61	11.52	**4.40**	19.40	44.34
LaRoSA 50%	38.81	72.22	47.27	75.84	76.82	68.67	27.35	37.26	10.39	3.80	15.60	43.09
WINA 50%	42.88	**72.69**	**52.39**	76.97	**78.73**	**73.88**	**30.06**	37.37	11.68	4.00	19.80	45.49
RoShift 50%	**43.28**	72.52	51.28	**78.72**	77.75	73.01	28.91	**38.72**	**13.87**	**4.40**	**22.40**	**45.90**
TEAL 60%	35.81	68.43	45.82	73.76	76.22	68.19	25.47	35.54	7.96	3.00	15.80	41.46
LaRoSA 60%	32.35	67.05	41.81	72.32	75.03	68.03	21.87	35.69	6.37	**4.40**	12.60	39.77
WINA 60%	**38.33**	69.53	47.53	75.44	76.61	67.88	**26.54**	34.08	6.75	**4.40**	16.80	42.17
RoShift 60%	37.86	**72.43**	**49.06**	**76.06**	**77.75**	**71.27**	23.75	**36.32**	**10.61**	4.00	**18.40**	**43.41**

Table 16: Mistral-7B: complete non-uniform accuracy results. Avg is the unweighted mean of 11 benchmarks.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	62.65	79.38	61.26	84.40	82.54	78.93	59.13	56.70	39.58	13.00	38.80	59.67
TEAL 40%	61.07	**79.46**	60.75	83.70	81.66	**78.93**	56.10	55.66	35.86	12.80	**38.20**	58.56
LaRoSA 40%	60.78	79.25	59.13	83.46	81.72	78.22	55.28	55.28	35.18	11.20	36.60	57.83
WINA 40%	60.98	79.34	61.01	83.91	**82.48**	77.11	56.76	55.68	36.92	**13.20**	37.20	58.60
RoShift 40%	**61.61**	78.75	**61.60**	**84.74**	**82.48**	78.69	**59.87**	**56.14**	**37.76**	12.80	37.20	**59.24**
TEAL 50%	59.64	77.86	60.15	83.55	81.07	76.56	54.14	54.12	33.59	**12.00**	33.80	56.95
LaRoSA 50%	57.56	77.90	56.57	81.68	80.25	75.69	48.98	52.39	29.26	8.60	33.60	54.77
WINA 50%	58.92	77.90	59.90	82.97	**81.56**	76.24	54.95	52.50	32.30	10.00	33.60	56.44
RoShift 50%	**60.25**	**78.87**	**60.41**	**84.65**	81.39	**77.82**	**59.87**	**55.31**	**37.23**	11.40	**36.80**	**58.54**
TEAL 60%	55.14	76.89	56.40	82.57	80.74	74.90	49.06	51.08	28.58	9.00	34.20	54.42
LaRoSA 60%	50.78	73.61	51.71	80.09	78.89	72.69	41.20	47.26	23.96	8.00	28.20	50.58
WINA 60%	54.09	77.19	55.55	80.76	79.76	74.03	47.99	47.14	22.44	8.00	29.60	52.41
RoShift 60%	**58.28**	**78.11**	**58.36**	**84.86**	**81.01**	**78.06**	**60.28**	**55.01**	**34.65**	**11.00**	**36.40**	**57.82**

Table 17: Qwen2.5-7B: complete non-uniform accuracy results. Avg is the unweighted mean of 11 benchmarks.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	74.23	77.40	63.74	84.68	79.76	76.01	85.50	68.96	82.64	49.60	63.60	73.28
TEAL 40%	72.82	77.15	62.46	**84.86**	79.33	**76.09**	83.78	66.32	**80.82**	**45.00**	**63.00**	**71.97**
LaRoSA 40%	72.67	**78.45**	61.01	83.73	79.38	74.35	83.37	**66.93**	79.30	41.40	60.80	71.04
WINA 40%	72.46	77.57	61.95	84.68	78.78	73.95	83.29	66.72	79.45	**45.00**	60.00	71.26
RoShift 40%	**73.58**	77.53	**63.31**	84.53	**80.14**	75.30	**84.93**	**66.93**	77.86	44.40	62.00	71.86
TEAL 50%	71.30	77.82	61.43	**84.50**	78.89	74.03	82.80	64.52	**78.01**	42.00	59.80	70.46
LaRoSA 50%	69.07	76.89	59.81	82.20	78.29	72.14	79.77	63.40	74.60	36.80	56.40	68.13
WINA 50%	69.72	77.57	58.19	83.09	78.84	72.77	81.82	63.43	76.42	39.60	57.00	68.95
RoShift 50%	**73.12**	**79.29**	**63.05**	84.25	**79.87**	**75.93**	**83.62**	**65.87**	77.03	**44.60**	**61.00**	**71.60**
TEAL 60%	68.59	75.42	59.30	81.74	78.67	69.22	79.03	61.59	73.92	37.60	54.00	67.19
LaRoSA 60%	61.95	72.52	52.56	78.81	76.33	70.56	72.65	57.07	65.20	25.60	42.00	61.39
WINA 60%	65.87	73.23	54.95	80.83	78.35	68.82	77.40	56.54	66.03	29.20	46.40	63.42
RoShift 60%	**72.07**	**78.03**	**60.92**	**83.73**	**79.38**	**74.27**	**82.72**	**65.20**	**78.92**	**42.00**	**58.60**	**70.53**

Table 18: Qwen3-8B: complete non-uniform accuracy results. Avg is the unweighted mean of 11 benchmarks.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	76.86	80.01	67.58	83.03	79.38	77.66	85.91	77.64	85.44	57.40	69.80	76.43
TEAL 40%	75.54	79.88	**67.24**	82.60	79.00	**76.24**	83.95	73.89	82.56	**52.00**	67.00	74.54
LaRoSA 40%	75.47	80.85	66.30	82.26	79.05	75.69	83.05	75.36	81.43	48.00	66.80	74.02
WINA 40%	75.30	78.41	66.47	81.35	78.89	75.06	83.13	75.29	83.02	50.00	66.80	73.97
RoShift 40%	**75.73**	**81.36**	**67.24**	**83.24**	**79.71**	75.61	**84.77**	**75.87**	**83.47**	**52.00**	**67.80**	**75.16**
TEAL 50%	74.16	**80.51**	65.70	82.29	79.76	74.66	81.82	72.11	81.96	51.00	64.40	73.49
LaRoSA 50%	71.52	80.22	63.82	79.97	78.02	72.69	78.38	71.29	78.62	42.40	63.00	70.90
WINA 50%	72.74	78.62	64.59	81.25	77.75	72.69	80.84	72.17	80.89	46.80	63.00	71.94
RoShift 50%	**75.07**	80.47	**66.38**	**83.39**	**80.20**	**75.14**	**84.03**	**75.04**	**83.93**	**52.00**	**68.60**	**74.93**
TEAL 60%	70.70	80.51	63.82	77.89	78.40	71.82	77.89	68.81	78.32	44.60	61.80	70.41
LaRoSA 60%	60.50	75.13	56.83	74.19	74.54	64.48	65.11	60.19	62.85	29.00	41.00	60.35
WINA 60%	66.44	77.15	59.30	77.95	75.79	69.38	72.73	64.89	73.54	37.80	56.20	66.47
RoShift 60%	**73.13**	**81.19**	**65.70**	**82.81**	**78.62**	**75.69**	**80.84**	**71.96**	**81.65**	**48.00**	**64.20**	**73.07**

A.9.2 UNIFORM SPARSITY

Table 19 summarizes accuracy under uniform sparsity, and Tables 20–24 provide the per-model benchmark results. All sparse results cover the same 11 benchmarks, including BBH, at 40%, 50%, and 60% sparsity across five models. Bold indicates the highest score among the sparse methods compared at the same sparsity.

Table 19: Downstream task accuracy under uniform sparsity. Each model entry reports the mean over 11 benchmarks, including BBH; Avg averages the five models. Parentheses indicate changes from Dense.

Method / Model	Llama-3.1-8B	Llama-2-7B	Mistral-7B	Qwen2.5-7B	Qwen3-8B	Avg
Dense (0%)	63.87	47.14	59.67	73.28	76.43	64.08
TEAL 40%	60.75(-3.13)	45.37(-1.78)	58.25(-1.42)	71.52(-1.77)	74.26(-2.17)	62.03(-2.05)
RoShift 40%	**62.15(-1.72)**	**46.12(-1.02)**	**59.03(-0.64)**	**71.60(-1.69)**	**74.61(-1.82)**	**62.70(-1.38)**
TEAL 50%	57.44(-6.43)	43.79(-3.35)	55.78(-3.89)	69.43(-3.85)	72.78(-3.65)	59.84(-4.24)
RoShift 50%	**60.44(-3.43)**	**45.47(-1.68)**	**58.05(-1.62)**	**70.73(-2.55)**	**74.19(-2.23)**	**61.78(-2.30)**
TEAL 60%	50.48(-13.39)	39.82(-7.33)	51.42(-8.25)	64.13(-9.15)	69.13(-7.29)	55.00(-9.08)
RoShift 60%	**57.06(-6.81)**	**43.31(-3.83)**	**56.33(-3.34)**	**67.37(-5.91)**	**71.40(-5.03)**	**59.09(-4.99)**

Table 20: Llama-3.1-8B: complete uniform accuracy results. Avg is the unweighted mean of 11 benchmarks, including BBH.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	65.57	82.49	58.28	83.12	81.12	78.22	70.60	64.52	48.90	21.60	48.20	63.87
TEAL 40%	62.33	79.17	56.23	**82.20**	79.54	75.69	65.93	60.41	44.12	17.80	44.80	60.75
RoShift 40%	**63.81**	**79.92**	**56.91**	81.62	**79.71**	**78.37**	**68.39**	**62.28**	**46.47**	**19.20**	**47.00**	**62.15**
TEAL 50%	57.52	78.24	52.90	79.54	77.69	72.14	61.51	57.47	39.42	15.20	40.20	57.44
RoShift 50%	**61.63**	**79.34**	**56.48**	**80.80**	**79.11**	**75.37**	**65.85**	**59.98**	**44.88**	**17.20**	**44.20**	**60.44**
TEAL 60%	48.63	73.15	43.86	75.50	76.06	65.75	50.45	50.64	26.69	10.40	34.20	50.48
RoShift 60%	**56.52**	**76.81**	**52.90**	**79.72**	**77.97**	**72.61**	**61.59**	**55.58**	**36.54**	**16.60**	**40.80**	**57.06**

Table 21: Llama-2-7B: complete uniform accuracy results. Avg is the unweighted mean of 11 benchmarks, including BBH.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	45.96	73.86	52.65	79.33	78.73	74.66	32.76	39.69	14.56	3.20	23.20	47.14
TEAL 40%	43.53	**72.18**	50.68	76.79	77.53	**71.82**	31.29	38.67	12.13	**4.20**	20.20	45.37
RoShift 40%	**45.08**	72.05	**51.45**	**78.56**	**78.62**	71.59	**33.01**	**39.04**	**12.96**	3.40	**21.60**	**46.12**
TEAL 50%	40.78	71.63	49.32	74.50	77.20	69.93	27.35	36.51	**12.28**	**4.40**	17.80	43.79
RoShift 50%	**42.84**	**72.22**	**51.11**	**76.12**	**78.51**	**72.69**	**30.63**	**39.21**	12.21	4.20	**20.40**	**45.47**
TEAL 60%	33.49	67.72	42.15	71.04	75.14	67.09	23.26	33.88	6.44	**4.20**	13.60	39.82
RoShift 60%	**38.22**	**70.58**	**47.18**	**75.17**	**76.33**	**70.96**	**27.76**	**37.00**	**11.22**	3.60	**18.40**	**43.31**

Table 22: Mistral-7B: complete uniform accuracy results. Avg is the unweighted mean of 11 benchmarks, including BBH.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	62.65	79.38	61.26	84.40	82.54	78.93	59.13	56.70	39.58	13.00	38.80	59.67
TEAL 40%	60.89	**79.04**	60.49	83.82	81.77	76.95	57.00	55.44	34.72	12.40	**38.20**	58.25
RoShift 40%	**61.18**	78.96	**60.84**	**84.46**	**81.94**	**77.98**	**58.72**	**55.86**	**38.36**	**13.40**	37.60	**59.03**
TEAL 50%	58.70	77.15	57.94	83.09	81.23	75.06	53.07	53.22	30.71	9.60	33.80	55.78
RoShift 50%	**60.06**	**78.28**	**58.11**	**84.43**	**81.77**	**76.48**	**59.71**	**55.40**	**34.95**	**11.60**	**37.80**	**58.05**
TEAL 60%	52.40	73.86	52.13	81.01	79.38	71.03	47.01	49.32	23.65	6.40	29.40	51.42
RoShift 60%	**57.16**	**77.74**	**56.57**	**84.10**	**80.25**	**76.01**	**56.67**	**52.85**	**33.06**	**9.40**	**35.80**	**56.33**

Table 23: Qwen2.5-7B: complete uniform accuracy results. Avg is the unweighted mean of 11 benchmarks, including BBH.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	74.23	77.40	63.74	84.68	79.76	76.01	85.50	68.96	82.64	49.60	63.60	73.28
TEAL 40%	72.48	76.89	61.95	83.88	78.94	**75.06**	83.29	**66.87**	**80.52**	**46.20**	60.60	71.52
RoShift 40%	**73.16**	**77.82**	**62.37**	**84.89**	**79.33**	**75.06**	**84.60**	66.06	77.86	44.20	**62.20**	**71.60**
TEAL 50%	70.03	77.19	60.24	83.67	**79.33**	**74.11**	80.18	64.00	76.80	39.60	58.60	69.43
RoShift 50%	**71.28**	**78.16**	**60.75**	**84.34**	79.05	73.40	**84.28**	**65.20**	**77.56**	**43.20**	**60.80**	**70.73**
TEAL 60%	64.46	74.66	54.35	80.31	77.20	68.59	74.45	56.77	70.89	30.40	53.40	64.13
RoShift 60%	**68.41**	**76.89**	**58.11**	**83.09**	**77.37**	**72.30**	**79.69**	**60.56**	**74.91**	**35.20**	**54.60**	**67.37**

Table 24: Qwen3-8B: complete uniform accuracy results. Avg is the unweighted mean of 11 benchmarks, including BBH.

	Log-Likelihood-Based Tasks							Generation Tasks				
Method	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	BBH	GSM8K	MATH-500	MBPP	Avg
Dense (0%)	76.86	80.01	67.58	83.03	79.38	77.66	85.91	77.64	85.44	57.40	69.80	76.43
TEAL 40%	75.54	**81.61**	65.61	82.60	**79.05**	**76.09**	83.05	73.31	**81.58**	**51.40**	67.00	74.26
RoShift 40%	**75.79**	80.39	**66.81**	**82.69**	**79.05**	**76.09**	**84.93**	**76.09**	80.82	49.80	**68.20**	**74.61**
TEAL 50%	73.81	80.39	64.16	81.35	**79.11**	73.56	81.08	72.60	80.74	49.40	64.40	72.78
RoShift 50%	**74.65**	**81.73**	**66.30**	**83.06**	78.94	**75.53**	**82.72**	**74.03**	**81.96**	**49.60**	**67.60**	**74.19**
TEAL 60%	68.94	79.46	60.92	75.54	76.77	70.17	78.05	68.71	76.12	43.60	62.20	69.13
RoShift 60%	**71.12**	**80.35**	**62.37**	**82.66**	**77.53**	**72.77**	**79.44**	**70.65**	**80.14**	**45.20**	**63.20**	**71.40**

A.9.3 UNIFORM VS. NON-UNIFORM SPARSITY

Table 25 compares uniform and non-uniform sparsity for RoShift across the same five models and 11 benchmarks. At 40%, 50%, and 60% sparsity, non-uniform allocation reduces the mean accuracy loss relative to Dense from 1.38, 2.30, and 4.99 to 1.17, 1.59, and 3.40 percentage points, respectively.

Table 25: RoShift with uniform and non-uniform sparsity. Each model entry averages 11 benchmarks; Avg averages all 55 model-benchmark combinations. Parentheses show differences from Dense in percentage points. Bold indicates the higher accuracy within each sparsity pair.

Method / Model	Llama-3.1-8B	Llama-2-7B	Mistral-7B	Qwen2.5-7B	Qwen3-8B	Avg
Dense (0%)	63.87	47.14	59.67	73.28	76.43	64.08
Uniform 40%	62.15(-1.72)	**46.12(-1.02)**	59.03(-0.64)	71.60(-1.69)	74.61(-1.82)	62.70(-1.38)
Non-uniform 40%	**62.26(-1.62)**	46.03(-1.12)	**59.24(-0.43)**	**71.86(-1.42)**	**75.16(-1.27)**	**62.91(-1.17)**
Uniform 50%	60.44(-3.43)	45.47(-1.68)	58.05(-1.62)	70.73(-2.55)	74.19(-2.23)	61.78(-2.30)
Non-uniform 50%	**61.47(-2.40)**	**45.90(-1.25)**	**58.54(-1.13)**	**71.60(-1.68)**	**74.93(-1.50)**	**62.49(-1.59)**
Uniform 60%	57.06(-6.81)	43.31(-3.83)	56.33(-3.34)	67.37(-5.91)	71.40(-5.03)	59.09(-4.99)
Non-uniform 60%	**58.54(-5.33)**	**43.41(-3.73)**	**57.82(-1.85)**	**70.53(-2.75)**	**73.07(-3.36)**	**60.68(-3.40)**

A.9.4 LAROSA WITH ORIGINAL ALLOCATION

Table 26 reports LaRoSA results under its original allocation policy, which measures sparsity by the number of activation dimensions, separately from the MAC-sparsity-based allocation search used in the main paper. Both configurations use FP16 runtime rotations. BBH results are not included in this table. Note that in RoShift, the layer-wise rotations are absorbed into the model weights offline, so no runtime rotation is required; only the transition matrix $Q_l^\top Q_{l+1}$ is applied online in FP16. Averaged over five models and ten benchmarks excluding BBH, RoShift still outperforms LaRoSA with its original allocation policy by 0.81, 2.56, and 4.59 percentage points at nominal sparsities of 40%, 50%, and 60%, respectively.

Table 26: LaRoSA with its original activation-count allocation (FP16 rotation). Target sparsity is the requested activation sparsity; MAC sparsity is the recipe-implied MAC sparsity estimate (%). Avg averages the 10 available benchmarks, excluding BBH.

Method	Target sparsity (%)	MAC sparsity (%)	MMLU	ARC-E	ARC-C	BoolQ	PIQA	Wino Grande	CSQA	GSM8K	MATH-500	MBPP	Avg
Llama-3.1-8B													
Dense	0	0.00	65.57	82.49	58.28	83.12	81.12	78.22	70.60	48.90	21.60	48.20	63.81
LaRoSA	40	34.47	63.18	80.85	55.89	81.62	80.20	77.19	69.78	45.72	17.00	45.00	61.64
LaRoSA	50	45.59	59.42	77.19	54.44	79.76	79.16	72.85	66.01	38.89	15.00	42.60	58.53
LaRoSA	60	54.71	51.35	72.98	47.44	75.54	77.37	67.72	56.59	26.38	9.40	31.40	51.62
Llama-2-7B													
Dense	0	0.00	45.96	73.86	52.65	79.33	78.73	74.66	32.76	14.56	3.20	23.20	47.89
LaRoSA	40	37.77	42.35	73.11	50.09	76.79	78.18	71.67	30.47	12.81	4.00	21.00	46.05
LaRoSA	50	47.21	40.56	72.26	48.21	76.94	76.55	70.24	27.35	11.22	3.20	18.60	44.51
LaRoSA	60	56.66	33.21	69.15	46.25	73.18	75.84	66.77	22.93	8.95	3.80	12.60	41.27
Mistral-7B													
Dense	0	0.00	62.65	79.38	61.26	84.40	82.54	78.93	59.13	39.58	13.00	38.80	59.97
LaRoSA	40	35.24	60.98	79.04	59.13	83.55	81.28	77.66	54.46	35.48	12.40	35.20	57.92
LaRoSA	50	44.04	58.11	77.65	57.76	82.29	81.12	75.30	48.32	33.28	11.00	32.40	55.72
LaRoSA	60	54.71	52.56	75.63	55.12	80.34	79.49	73.80	40.87	27.60	7.40	30.20	52.30
Qwen2.5-7B													
Dense	0	0.00	74.23	77.40	63.74	84.68	79.76	76.01	85.50	82.64	49.60	63.60	73.72
LaRoSA	40	34.91	73.02	78.11	62.37	83.64	79.71	74.90	83.95	80.44	46.00	60.60	72.27
LaRoSA	50	42.22	71.62	76.68	60.15	81.80	78.51	74.03	82.72	76.57	40.60	60.40	70.31
LaRoSA	60	51.23	68.63	76.26	59.39	79.69	78.29	73.40	78.13	74.15	36.20	57.20	68.13
Qwen3-8B													
Dense	0	0.00	76.86	80.01	67.58	83.03	79.38	77.66	85.91	85.44	57.40	69.80	76.31
LaRoSA	40	35.49	75.49	80.18	66.47	82.08	78.73	74.59	83.78	81.65	51.60	65.20	73.98
LaRoSA	50	43.49	73.32	80.81	64.68	80.15	78.62	74.11	80.43	79.30	50.00	61.80	72.32
LaRoSA	60	52.97	69.25	77.40	62.80	77.77	78.18	71.27	73.96	74.00	45.60	59.80	69.00

A.10 Complete Quantization Results

Table 27 reports the 8-bit quantization results for RoShift. Tables 28 and 29 report RoShift's 4-bit weight quantization results with the transition matrices $Q_l^\top Q_{l+1}$ retained in 8 and 16 bits, respectively. Table 30 compares TEAL, LaRoSA, and RoShift under 4-bit quantization. For Table 30, unlike the public implementation used in the accuracy experiments, we implement LaRoSA following its paper formulation, retaining activations in the rotated basis and applying $Q_l^\top Q_{l+1}$ between layers. All tables report PPL on Llama-3.1-8B using RTN, GPTQ, AWQ, and OmniQuant, measured on the datasets specified in Appendix A.5. We evaluate all sparsified models under uniform sparsity.

With 8-bit quantization, the PPL gap between sparse RoShift and Dense remains nearly unchanged across all quantization methods, demonstrating high compatibility between sparsification and 8-bit quantization.

In contrast, compatibility with 4-bit quantization depends strongly on the precision of the transition matrices $Q_l^\top Q_{l+1}$. TEAL, which does not use transition matrices, retains a similar PPL gap and is robust to quantization. LaRoSA and RoShift, however, degrade substantially across all quantization methods when both weights and transition matrices are quantized to 4 bits. We attribute this sensitivity to quantization errors in the transition matrices, which act on the entire rotated FFN input residual X_l, allowing errors to accumulate across layers.

Keeping RoShift's transition matrices at 8 bits while quantizing the weights to 4 bits yields PPL comparable to that obtained without quantizing the transition matrices across all evaluated quantizers. This demonstrates that 8-bit precision for the transition matrices largely preserves model performance under 4-bit weight quantization.

Table 27: PPL comparison of RoShift across 8-bit quantization methods.

Method	Baseline (16-bit)	RTN	GPTQ	AWQ	OmniQuant
RoShift 0%	4.99	5.07	5.07	5.05	5.07
RoShift 50%	5.82	5.73	5.75	5.72	5.73

Table 28: PPL comparison of RoShift under 4-bit weight quantization with transition matrices retained in 8-bit.

Method	Baseline (16-bit)	RTN	GPTQ	AWQ	OmniQuant
RoShift 0%	4.99	5.57	5.56	5.37	5.52
RoShift 50%	5.82	6.24	6.19	5.97	6.14

Table 29: PPL comparison of RoShift under 4-bit weight quantization with transition matrices retained in 16-bit.

Method	Baseline (16-bit)	RTN	GPTQ	AWQ	OmniQuant
RoShift 0%	4.99	5.50	5.48	5.29	5.38
RoShift 50%	5.82	6.15	6.09	5.91	6.04

Table 30: PPL comparison of TEAL, LaRoSA, and RoShift under 4-bit quantization.

Method	Baseline (16-bit)	RTN	GPTQ	AWQ	OmniQuant
TEAL 0%	4.99	5.50	5.48	5.29	5.38
TEAL 50%	6.22	6.62	6.78	6.92	6.68
LaRoSA 0%	4.99	5528.44	336.47	4447.38	1464.26
LaRoSA 50%	6.16	7368.24	1415.94	7593.46	6201.85
RoShift 0%	4.99	331.88	231.45	2331.75	392.60
RoShift 50%	5.82	481.92	349.68	1187.89	282.01