PRISM: PRINCIPAL-RESIDUAL ACTIVATION SPARSITY VIA STRUCTURE-AWARE BUDGET ALLOCATION FOR EFFICIENT LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Training-free activation sparsity accelerates Large Language Model (LLM) inference by skipping computations on selected activation channels, without requiring additional training. Existing methods usually construct sparse computation patterns by selecting channels in the original activation coordinates. However, under high sparsity, the few retained channels may fail to cover sufficient output-relevant information, leading to large projection errors. Beyond the issue of channel selection, existing sparsity allocation methods commonly estimate module sensitivity with a single metric, overlooking structural dependencies inside Transformer blocks and the impact of error propagation. To address these limitations, we propose PRISM, a training-free activation sparsity framework with principal-residual computation and structure-aware sparsity allocation. PRISM first preserves low-dimensional principal components of linear-layer inputs through a compact dense path, and then applies sparse computation only to the residual activation. This dual-path design reduces the information loss caused by direct channel selection. PRISM further allocates sparsity budgets according to structural dependencies in Attention and MLP modules, avoiding overly aggressive sparsification of sensitive components. Experiments on representative LLMs show that PRISM better preserves model performance under high sparsity while providing practical inference speedup. At 75% sparsity, PRISM improves average reasoning accuracy by 3.7 points over the strongest sparse baseline and achieves up to 1.82× end-to-end generation speedup. Our code is available at `https://anonymous.4open.science/r/PRISM-84BC`.

1 INTRODUCTION

Large Language Models (LLMs) perform well across many natural language processing tasks (Achiam et al., 2023; Guo et al., 2025; Grattafiori et al., 2024), but their scale makes inference computationally and memory intensive. Existing compression methods, such as low-bit quantization (Frantar et al., 2022; Xiao et al., 2023; Lin et al., 2024; Chee et al., 2023) and weight pruning (Frantar & Alistarh, 2023; Sun et al., 2024; Ashkboos et al., 2024; Ma et al., 2023), reduce this cost by changing numerical precision or removing model weights. Activation sparsity (Kurtz et al., 2020; Liu et al., 2023; Lee et al., 2024) provides a complementary direction by skipping computations on selected activation channels during inference, while keeping the original model weights unchanged. Early activation sparsity studies exploit natural sparsity in ReLU-based Transformers (Li et al., 2022; Liu et al., 2023), whereas modern LLMs use SiLU (Elfwing et al., 2018), GELU (Hendrycks & Gimpel, 2016), or SwiGLU (Shazeer, 2020) activations, which are usually dense. Recent training-free activation sparsity methods induce sparsity by selecting activation channels at runtime (Lee et al., 2024; Liu et al., 2025a; Zhang et al., 2025).

For example, TEAL (Liu et al., 2025a) keeps channels with large activation magnitudes, while R-Sparse (Zhang et al., 2025) adds a low-rank weight approximation to compensate for discarded channels. Despite their effectiveness, these methods still face two limitations. First, their sparse computation is largely based on channel selection in the original activation coordinates of linear layers. When the sparsity is high, only a few original channels are retained, which may not cover enough output-relevant information and can lead to large projection errors. LaRoSA (Liu et al., 2025b) mitigates this issue through full-dimensional layerwise rotations, but can introduce considerable

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.

computational overhead. Second, existing non-uniform sparsity allocation strategies often estimate module sensitivity with a single metric, overlooking structural dependencies within Transformer modules and the true impact of error propagation.

In this work, we propose **PRISM**, a training-free activation sparsity framework featuring activation-side principal subspace preservation and structure-aware sparsity budget allocation. To address the first limitation, PRISM replaces direct original-coordinate channel selection with a principal-residual decomposition. It learns a low-dimensional principal subspace from a small calibration set and keeps it fixed during inference. For each target linear layer, PRISM decomposes the input activation into principal and residual components, computing the former through a compact dense path and applying sparse computation only to the latter. Unlike LaRoSA's full-dimensional inter-layer alignment, this path operates in a low-dimensional subspace with lower computational overhead. By preserving dominant activation information before sparsification, PRISM reduces the information loss from selecting only a few original channels and makes sparse computation more reliable at high sparsity. To address the second limitation, PRISM introduces a structure-aware sparsity allocation strategy. Rather than relying on a single metric, PRISM allocates residual sparsity based on local sensitivity and structural dependencies within Attention and MLP modules. Since errors from different projections may interact within coupled module computations, this allocation helps avoid overly aggressive sparsification of structurally sensitive components and reduces error propagation.

Experiments on representative LLMs show that PRISM better preserves performance at high sparsity while accelerating inference. At 75% sparsity, PRISM improves average reasoning accuracy by about 3.7 percentage points over the strongest sparse baseline and achieves up to 1.82× end-to-end generation speedup.

Our contributions are summarized as follows:

- We propose a training-free principal-residual activation sparsity framework for efficient LLM inference. Unlike original coordinate channel selection, PRISM preserves low-dimensional principal components first and shifts the sparsification target from the full activation to the residual activation.
- We introduce a structure-aware sparsity allocation strategy for Transformer modules. By considering structural dependencies inside Attention and MLP modules, PRISM assigns more suitable residual sparsity ratios and alleviates error propagation beyond a single sensitivity metric.
- We conduct extensive experiments on representative LLMs and reasoning benchmarks. PRISM outperforms strong training-free activation sparsity baselines under high sparsity, improving average reasoning accuracy by about 3.7 percentage points at 75% sparsity and achieving up to 1.82× end-to-end generation speedup without training or weight updates.

2 Related Work

LLM compression and activation sparsity. Efficient LLM inference has been widely studied due to the high computation and memory cost of large models. Post-training quantization methods, such as GPTQ (Frantar et al., 2022), SmoothQuant (Xiao et al., 2023), and AWQ (Lin et al., 2024), reduce numerical precision, while pruning methods such as SparseGPT (Frantar & Alistarh, 2023) remove redundant weights. Different from these weight-centric methods, activation sparsity reduces inference cost by skipping computations on inactive or pruned activation channels. Early methods mainly rely on natural sparsity in ReLU-based models, such as DejaVu (Liu et al., 2023). However, modern LLMs commonly use dense non-ReLU activations such as GELU, SiLU, or SwiGLU. Some works therefore modify activation functions or introduce sparse training, such as ReLU-based replacement (Mirzadeh et al., 2024) and Q-Sparse (Wang et al., 2024). These methods improve activation sparsity, but they require model modification or additional training, which limits their direct use on existing LLMs.

Training-free activation sparsity. Training-free activation sparsity aims to reduce inference cost without retraining the model. CATS (Lee et al., 2024) applies context-aware thresholding to gated MLP activations, enabling input-dependent sparsification during inference. TEAL (Liu et al., 2025a) extends activation sparsity to the main linear projections in Transformer blocks and keeps channels

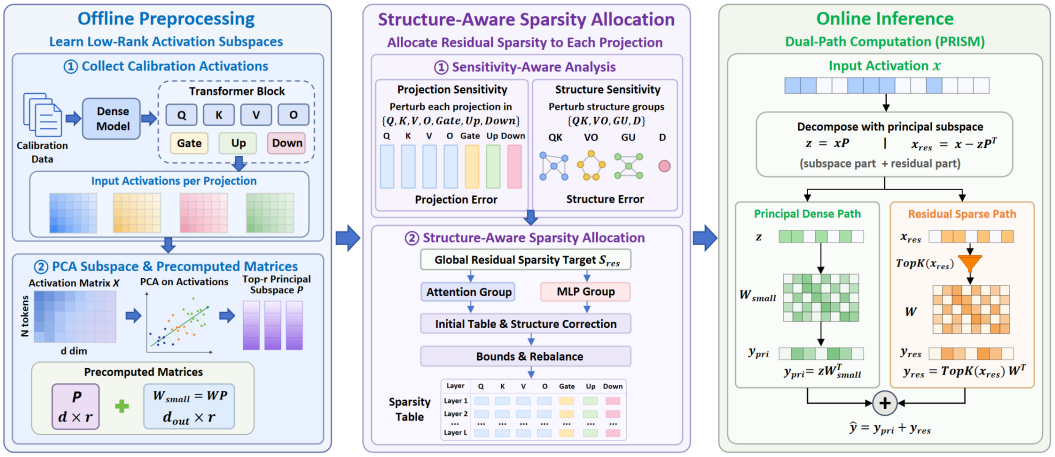

Figure 1: An overview of PRISM. PRISM learns PCA-based low-rank activation subspaces offline, allocates residual sparsity across projections in a structure-aware manner, and performs online dual-path inference with principal and residual activations.

according to activation magnitude. R-Sparse (Zhang et al., 2025) further combines activation sparsity with low-rank weight approximation, using the low-rank path to compensate for discarded activation channels. LaRoSA (Liu et al., 2025b) selects channels after full-dimensional layerwise rotations, enabling sparsification in transformed coordinates. These methods show that training-free activation sparsity can accelerate modern LLMs while preserving much of their performance. However, most of these methods still select channels in the original activation coordinates, which may lose output-relevant information under high sparsity. LaRoSA mitigates this issue, but aligning representations across layers requires repeated full-dimensional rotations and can introduce considerable computational overhead. Moreover, sparsity allocation based on local reconstruction errors may overlook structural dependencies within Transformer modules.

3 METHOD

We propose PRISM, a training-free principal-residual activation sparsity framework for efficient LLM inference. As shown in Figure 1, PRISM learns PCA-based principal subspaces offline, allocates residual sparsity with Transformer structures in mind, and performs dual-path inference with a compact principal path and a sparse residual path.

3.1 PRELIMINARIES AND MOTIVATION

Activation sparsity. Modern Transformer-based LLMs consist of repeated Transformer blocks with Attention and MLP modules. We focus on the linear projections in these modules. For a linear layer, the dense computation is

$$y = xW^\top, \tag{1}$$

where $x \in \mathbb{R}^{d_{\text{in}}}$ is the input activation, $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is the weight matrix, and $y \in \mathbb{R}^{d_{\text{out}}}$ is the output.

Activation sparsity accelerates this computation by applying a binary mask m to the input channels, giving $\widehat{y} = (m \odot x)W^\top$, where $m_i = 0$ means that channel i and its corresponding weight column $W_{:,i}$ can be skipped during inference. Most existing training-free methods construct m in the original activation coordinates. A representative magnitude-thresholding rule is:

$$m_i(x_i; \tau_s) = \begin{cases} 1, & \text{if } |x_i| \geq \tau_s \\ 0, & \text{if } |x_i| < \tau_s \end{cases}, \tag{2}$$

where τ_s is the threshold corresponding to the target sparsity ratio s.

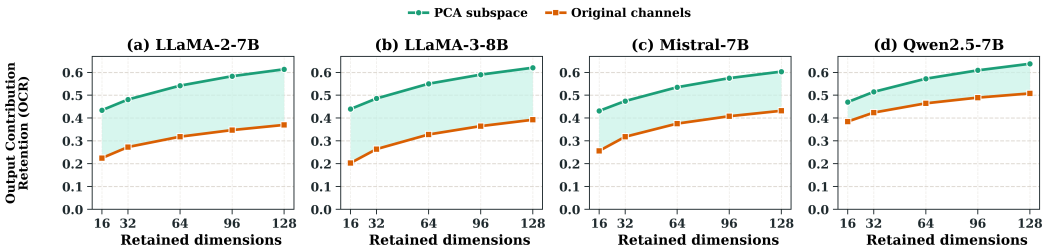

Figure 2: Linear-layer output contribution retention (OCR) of PCA subspaces and original-coordinate channel selection with the same number of retained dimensions.

This strategy is often effective at low sparsity, but becomes less reliable under high sparsity. Since channel selection is performed in the original coordinate space, retaining only a small number of original channels may fail to sufficiently preserve output-relevant information, leading to large linear-projection errors.

Motivation. This limitation motivates us to reconsider whether activations should be sparsified directly in the original coordinate space. To quantify output preservation, we use output contribution retention (OCR):

$$\mathrm{OCR} = 1 - \frac{\|Y_{\mathrm{dense}} - Y_{\mathrm{approx}}\|_F^2}{\|Y_{\mathrm{dense}}\|_F^2}, \tag{3}$$

where Y_{dense} is the dense linear-projection output, and Y_{approx} is the approximated output.

We compare PCA subspaces with original-coordinate channel selection under the same retained dimensions. For original-coordinate channel selection, we select the top-r channels estimated from offline activation samples to contribute most to the linear-layer output. As shown in Figure 2, PCA subspaces capture more output contribution across the evaluated models. This suggests that a compact principal subspace can preserve output-relevant information more efficiently than a small set of original activation channels. Based on this observation, PRISM decomposes the activation into a principal component x_{pri} and a residual component x_{res}, such that $x = x_{\mathrm{pri}} + x_{\mathrm{res}}$. It preserves the principal component through the principal path and sparsifies only the residual, helping reduce output error at high sparsity.

3.2 Principal-Residual Activation Sparsity

PRISM approximates each target linear projection through a principal-residual decomposition of its input activation. The procedure contains an offline calibration stage and an online inference stage.

Offline calibration. For each target projection, PRISM collects input activations from a small calibration set. We apply principal component analysis (PCA) (Jolliffe & Cadima, 2016) to these activations and obtain a low-dimensional orthogonal basis $P \in \mathbb{R}^{d_{\mathrm{in}} \times r}$:

$$P^\top P = I, \tag{4}$$

where $r \ll d_{\mathrm{in}}$ denotes the dimension of the preserved principal subspace.

The principal activation can be written as $x_{\mathrm{pri}} = xPP^\top$. Instead of applying the full weight matrix to this reconstructed activation, PRISM uses the following equivalent form:

$$x_{\mathrm{pri}}W^\top = (xPP^\top)W^\top = (xP)(WP)^\top. \tag{5}$$

We precompute $W_{\mathrm{small}} = WP$ as the compact projected weight matrix. After calibration, both P and W_{small} are fixed and reused during inference. This process does not update the original model weights. Details of PCA calibration are provided in Appendix A.

Online inference. Given an input activation x, PRISM first projects it onto the calibrated principal subspace:

$$z = xP, \tag{6}$$

where z is the low-dimensional principal representation. The principal-path output is then computed as $zW_{\text{small}}^{\top}$.

PRISM reconstructs the principal component as $zP^{\top}$ and obtains the residual activation:

$$x_{\text{res}} = x - zP^{\top}. \tag{7}$$

Sparse computation is applied only to x_{res}. The final approximation is obtained by summing the principal-path output and the residual-path output:

$$\widehat{y} = zW_{\text{small}}^{\top} + \mathcal{A}_{s_{\text{res}}}(x_{\text{res}})W^{\top}, \tag{8}$$

where $\mathcal{A}_{s_{\text{res}}}(\cdot)$ denotes residual Top-K sparsification, and s_{res} is the residual sparsity ratio determined by the compute budget and the sparsity allocation strategy.

Different from directly sparsifying the original activation x, PRISM first preserves the principal component and applies sparse computation only to the residual activation. This principal-residual formulation helps reduce output error under high sparsity in sparse inference. With $W_{\text{small}} = WP$ precomputed, PRISM adds $O((d + d_{\text{out}})r)$ dense computation, much less than the $O(d^2)$ cost of LaRoSA's full-dimensional inter-layer alignment. Here, d and d_{out} are the input and output dimensions of a linear projection, and $r \ll d$ is the principal subspace rank.

3.3 Compute-Constrained Rank Selection

The principal rank r controls both the capacity and the cost of the principal path. Under a fixed compute-equivalent budget, increasing r improves the principal approximation but reduces the budget available to the residual Top-K path. Thus, PRISM selects r by balancing principal-subspace preservation and residual sparsification capacity.

During offline calibration, PRISM evaluates a small candidate rank set $\mathcal{R}$. For each candidate rank, we estimate the cost of the principal path and remap the residual sparsity so that the two paths match the target compute budget. We then compute the normalized linear-projection approximation error:

$$D_{\text{proj}}(r) = \frac{\|Y_{\text{dense}} - Y_{\text{PRISM}}(r)\|_F^2}{\|Y_{\text{dense}}\|_F^2}, \tag{9}$$

where Y_{dense} is the dense output, and $Y_{\text{PRISM}}(r)$ is the PRISM approximation under rank r with remapped residual sparsity.

Let $D_{\min}$ be the minimum value of $D_{\text{proj}}(r)$ over $r \in \mathcal{R}$. Instead of simply choosing the rank with the minimum error, PRISM considers candidate ranks whose errors are within a small tolerance of $D_{\min}$. It then selects a low-cost rank from this near-optimal set and uses model-level output drift on calibration sequences as a secondary stability check. The selected rank is fixed before structure-aware residual sparsity allocation. Further rank-selection details and principal-rank ablation results are provided in Appendix G.8.

3.4 Structure-Aware Sparsity Allocation

After selecting the principal rank, PRISM allocates the remaining compute budget to the residual Top-K path. Uniform residual sparsity assigns the same sparsity to all projections, while an MSE-only greedy allocation based on TEAL (Liu et al., 2025a) uses local output errors. However, local errors may not fully reflect structural dependencies inside Transformer blocks. PRISM therefore uses Algorithm 1 to build an initial sparsity table from projection-level sensitivity and then apply bounded corrections based on structural damage.

For projection-level initialization, $e_{\ell,p}$ denotes the sensitivity of projection p in layer ℓ. Starting from uniform residual sparsity, ALLOCATE perturbs one projection at a time and measures the resulting change in relative layer-output error against a dense reference on calibration data. These sensitivities guide the updates to obtain the initial sparsity table $\mathbf{s}_\ell^0$ under the target residual sparsity S_{res}.

For structure-level correction, PRISM evaluates $\mathcal{G} = \{\text{QK}, \text{VO}, \text{GU}, \text{D}\}$, which correspond to attention probability, attention output, MLP intermediate, and MLP output perturbations, respectively. For each group, STRUCT-ERR temporarily increases the residual sparsity of its projections by δ under

Algorithm 1 Structure-Aware Sparsity Allocation

Input: Model $\mathcal{M}$, calibration set $\mathcal{D}_{\text{cal}}$, target residual sparsity S_{res}, perturbation step δ, projection set $\mathcal{P}$, correction bounds $\{B_g\}$
Output: Residual sparsity table $\mathbf{s} = \{s_{\ell,p}\}$

$\mathcal{G} \leftarrow \{\text{QK}, \text{VO}, \text{GU}, \text{D}\}$
for each layer ℓ **do**
  Compute dense reference signals on $\mathcal{D}_{\text{cal}}$
  $\mathbf{s}_\ell^{\text{unif}} \leftarrow S_{\text{res}}\mathbf{1}$ ▷ Uniform residual sparsity
  $\mathbf{s}_\ell^0 \leftarrow \textsc{Allocate}(\mathbf{s}_\ell^{\text{unif}}, \delta)$ ▷ Allocate by projection sensitivity
  for each group $g \in \mathcal{G}$ **do**
    $b_{\ell,g}^{\text{unif}} \leftarrow \textsc{Struct-Err}(\ell, g, \mathbf{s}_\ell^{\text{unif}}, \delta)$
    $b_{\ell,g}^0 \leftarrow \textsc{Struct-Err}(\ell, g, \mathbf{s}_\ell^0, \delta)$
    $r_{\ell,g} \leftarrow \textsc{Relative-Damage}(b_{\ell,g}^0, b_{\ell,g}^{\text{unif}})$ ▷ Estimate structural sensitivity
  $\boldsymbol{\Delta}_\ell \leftarrow \textsc{Bounded-Shift}(\mathbf{r}_\ell, \{B_g\})$ ▷ Protect sensitive groups
  $\widetilde{\mathbf{s}}_\ell \leftarrow \{s_{\ell,p}^0 + \Delta_{\ell,g(p)}\}_{p\in\mathcal{P}}$
  $\mathbf{s}_\ell \leftarrow \textsc{Apply-Bounds}(\widetilde{\mathbf{s}}_\ell)$
  $\mathbf{s}_\ell \leftarrow \textsc{Rebalance}(\mathbf{s}_\ell, S_{\text{res}})$ ▷ Rebalance residual sparsity budget
return $\mathbf{s}$

both the uniform configuration $\mathbf{s}_\ell^{\text{unif}}$ and the initial allocation $\mathbf{s}_\ell^0$. It measures the resulting structural errors using the same group-specific metric and dense reference signals. RELATIVE-DAMAGE then measures the increase in perturbed error relative to the uniform configuration. This within-group comparison avoids directly combining raw structural errors measured on different scales.

Based on these relative damage scores, BOUNDED-SHIFT preferentially reduces sparsity for groups with larger scores and adjusts the other groups to share the budget. For each group g, the shift satisfies $-B_g \le \Delta_{\ell,g} \le B_g$, where B_g is the maximum allowed correction magnitude. The corrected sparsity is:

$$s_{\ell,p} = s_{\ell,p}^0 + \Delta_{\ell,g(p)}. \tag{10}$$

Finally, APPLY-BOUNDS prevents extreme sparsity values, and REBALANCE restores the layer-level residual budget:

$$\frac{\sum_{p\in\mathcal{P}} w_{\ell,p} s_{\ell,p}}{\sum_{p\in\mathcal{P}} w_{\ell,p}} = S_{\text{res}}, \tag{11}$$

where $w_{\ell,p}$ denotes the relative computation size of projection p in layer ℓ. The final sparsity table is fixed after calibration and used during inference. See Appendix C for the necessity and details of structure-aware allocation.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models. We evaluate PRISM on several widely used LLMs, including LLaMA-2-7B (Touvron et al., 2023), LLaMA-3-8B (Grattafiori et al., 2024), Mistral-7B (Jiang et al., 2023), and Qwen2.5-7B (Qwen et al., 2025). All training-free sparsification methods use the same unlabeled Alpaca (Taori et al., 2023) dataset for calibration. The calibration data are only used for offline statistics and method-specific configuration generation. No evaluation data are used for calibration, and the original model parameters are not updated. Calibration sensitivity to dataset choice and sample size is reported in Appendix E.

Evaluation Tasks. We use LM Evaluation Harness (Gao et al., 2021) for evaluation. The main tasks include five zero-shot multiple-choice benchmarks: PIQA (Bisk et al., 2020), ARC-Easy (Clark et al., 2018), ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2021). We additionally evaluate MMLU (Hendrycks et al., 2020) and GSM8K (Cobbe et al., 2021) to assess general knowledge and mathematical reasoning after sparsification.

Baselines and sparsification settings. We compare PRISM with three representative training-free activation sparsity baselines: TEAL (Liu et al., 2025a), R-Sparse (Zhang et al., 2025), and LaRoSA (Liu et al., 2025b). The dense model denotes the original unsparsified model and is included only as a reference. To avoid actual sparsity drift caused by offline fixed thresholds, all sparse methods adopt a unified Top-K selection strategy, so that each forward pass retains a fixed number of channels. We analyze the token-level sparsity drift of fixed-threshold routing in Appendix B. TEAL and R-Sparse select Top-K channels in the original activation coordinates; LaRoSA selects channels after full-dimensional layerwise rotations; PRISM applies Top-K to the residual after removing the principal component.

For PRISM, the reported sparsity denotes *compute-equivalent global sparsity*. Given a target global sparsity s, we include both the principal path and the residual sparse path in the compute budget. We adjust the residual sparsity s_{res} so that the total computation of PRISM, including the principal path, matches that of TEAL and R-Sparse at the same target sparsity:

$$\begin{aligned} C_{\text{PRISM}}(s) &= C_{\text{pri}} + C_{\text{res}}(s_{\text{res}}) \\ &\approx (1-s)C_{\text{dense}} \end{aligned}, \tag{12}$$

where C_{pri} is the principal-path cost, $C_{\text{res}}(s_{\text{res}})$ is the residual sparse-path cost under residual sparsity s_{res}, and C_{dense} is the dense computation cost over the same sparsified projections. We follow LaRoSA's sparsity definition in the original paper (Liu et al., 2025b), so online rotation overhead is excluded when matching sparsity. Additional details of the compute-equivalent accounting are given in Appendix D.

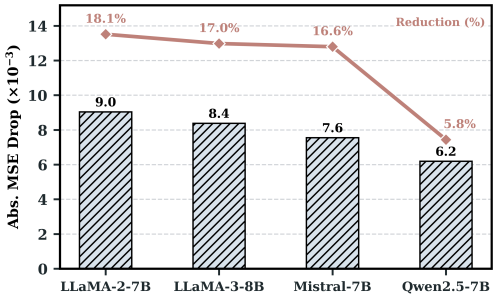

Figure 3: Linear-projection MSE reduction of PRISM over original-coordinate Top-K at 60% sparsity. Bars show the absolute MSE drop, and the line shows the relative reduction.

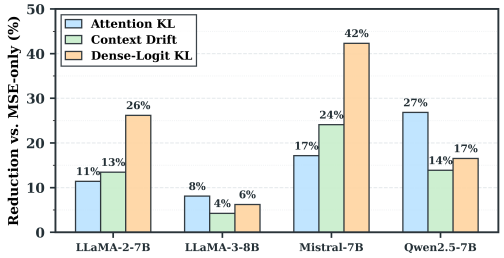

Figure 4: Reduction of structure-related drift metrics from structure-aware sparsity allocation over MSE-only allocation. Higher is better.

4.2 Effectiveness Analysis of PRISM

We analyze PRISM from two perspectives: approximation quality and structure preservation. Figure 3 compares PRISM with original-coordinate Top-K at 60% sparsity. The linear-projection MSE is measured on separate Alpaca samples (Taori et al., 2023) not used to compute the offline PCA subspaces. MSE is the mean squared error between dense and approximated projection outputs. Under the same compute-equivalent budget, PRISM consistently reduces relative MSE across all evaluated models, with up to 18.1% reduction. This indicates that the principal-residual decomposition better approximates dense linear-layer outputs than directly applying Top-K in the original activation coordinates.

Figure 4 evaluates the effect of structure-aware sparsity allocation on WikiText2 samples (Merity et al., 2016). We compare PRISM with an MSE-only PRISM variant that retains the principal-residual design. It allocates residual sparsity with TEAL's block-wise greedy optimization algorithm (Liu et al., 2025a), using local block-output MSE as the sensitivity metric. We report three structure-related drift metrics: Attention KL measures the divergence between dense and sparse attention probabilities, Context Drift measures the change in attention context outputs, and Dense-Logit KL measures the output-distribution drift from the dense model. Structure-aware allocation reduces the three metrics by up to 27%, 24%, and 42%, respectively. These results show that considering module dependencies helps preserve attention behavior and output distributions.

Table 1: Accuracy on reasoning benchmarks under different sparsity levels. For each model at each sparsity level, the best sparse-method result is shown in bold.

Model	Sparsity	Method	PIQA	HellaSwag	ARC-C	ARC-E	WinoGrande	GSM8K	MMLU	AVG
LLaMA-2-7B	0%	Dense	79.11	76.02	46.42	74.62	68.90	13.95	45.84	57.84
	60%	R-Sparse	73.29	63.82	36.43	64.06	**64.01**	2.88	33.59	48.30
		TEAL	72.96	63.10	34.81	62.63	62.04	2.58	30.49	46.94
		LaRoSA	73.46	65.05	36.61	64.85	62.56	4.12	32.71	48.48
		PRISM	**74.27**	**67.40**	**38.65**	**67.51**	63.22	**5.23**	**33.93**	**50.03**
	75%	R-Sparse	55.98	30.11	22.35	34.43	50.26	1.15	25.49	31.40
		TEAL	52.72	28.25	22.27	29.38	48.30	0.38	25.30	29.51
		LaRoSA	52.99	29.32	**23.98**	28.11	50.36	1.04	24.46	30.04
		PRISM	**60.17**	**34.14**	**23.98**	**42.63**	**51.07**	**1.44**	**26.29**	**34.25**
LLaMA-3-8B	0%	Dense	80.79	79.19	53.41	77.69	72.45	50.04	65.30	68.41
	60%	R-Sparse	73.18	63.71	38.74	67.72	62.83	6.75	44.97	51.13
		TEAL	73.94	65.06	40.61	66.58	63.61	7.13	42.46	51.34
		LaRoSA	75.68	67.33	41.89	67.97	62.75	9.33	44.42	52.77
		PRISM	**75.95**	**68.10**	**42.32**	**70.75**	**64.72**	**11.60**	**49.72**	**54.74**
	75%	R-Sparse	57.40	29.80	21.76	33.92	49.96	**2.43**	26.23	31.64
		TEAL	57.13	29.53	20.99	32.91	49.72	2.05	26.01	31.19
		LaRoSA	56.86	29.44	24.32	34.30	50.41	2.17	23.71	31.60
		PRISM	**62.30**	**37.77**	**27.82**	**48.06**	**51.46**	2.35	**27.30**	**36.72**
Mistral-7B	0%	Dense	82.15	81.06	54.01	79.59	73.95	38.59	62.65	67.43
	60%	R-Sparse	77.37	70.47	41.13	71.80	67.64	12.74	49.82	55.85
		TEAL	79.22	75.46	47.70	74.54	**70.24**	19.64	**52.51**	59.90
		LaRoSA	79.00	**77.23**	47.44	75.00	67.32	17.74	50.69	59.20
		PRISM	**79.60**	75.70	**47.87**	**75.84**	69.22	**21.83**	52.18	**60.32**
	75%	R-Sparse	61.75	38.32	24.49	44.74	52.41	1.82	25.99	35.65
		TEAL	69.59	52.95	32.00	59.22	58.25	**2.20**	28.33	43.22
		LaRoSA	66.93	50.02	27.56	57.60	56.14	1.29	26.57	40.87
		PRISM	**72.74**	**56.15**	**34.13**	**61.83**	**60.22**	1.97	**28.36**	**45.06**
Qwen2.5-7B	0%	Dense	81.77	79.06	58.53	81.82	72.61	80.67	73.43	75.41
	60%	R-Sparse	75.52	68.66	48.46	73.74	67.48	55.42	62.34	64.52
		TEAL	**78.18**	73.80	51.45	75.67	**70.01**	67.78	65.24	68.88
		LaRoSA	76.84	72.15	50.12	73.32	66.02	60.16	63.58	66.03
		PRISM	77.20	**73.90**	**53.41**	**76.77**	68.03	**73.69**	**66.36**	**69.91**
	75%	R-Sparse	64.04	37.04	27.82	49.16	50.28	1.14	32.13	37.37
		TEAL	72.52	57.19	39.42	61.62	58.09	13.72	46.57	49.88
		LaRoSA	61.14	40.06	29.03	53.65	53.24	7.12	44.85	41.30
		PRISM	**72.69**	**59.41**	**41.72**	**64.14**	**58.72**	**34.95**	**52.02**	**54.81**

4.3 Reasoning Benchmark Accuracy

Table 1 compares the accuracy of different methods on reasoning benchmarks under 60% and 75% sparsity. Overall, PRISM achieves the best average accuracy in all eight model-sparsity settings, showing that the principal-residual decomposition preserves reasoning performance more consistently. At 60% sparsity, PRISM improves the average score by about 1.2 points over the strongest sparse baseline. Under the more aggressive 75% sparsity setting, the average improvement increases to about 3.7 points. This indicates that the advantage of PRISM becomes larger when the compute budget is tighter. The results suggest that directly selecting channels in the original activation coordinates becomes less reliable under high sparsity, while PRISM better maintains reasoning ability by preserving principal activation components before sparsifying the residual.

While other sparse baselines lead on some tasks, PRISM achieves the best average accuracy across all evaluated settings. These results show that PRISM better preserves reasoning performance than the sparse baselines at the same target sparsity. Additional experiments on Gemma-2-9B (Team et al., 2024) and LLaMA-2-13B (Touvron et al., 2023) further support PRISM's applicability across model families and sizes. Detailed results are provided in Appendix F.

4.4 End-to-End Inference Speedup

To achieve efficient inference, we adapt TEAL's sparse GEMV kernel (Liu et al., 2025a) in Triton (Tillet et al., 2019) for PRISM's residual path. The kernel fuses residual Top-K selection with selective weight-column loading to reduce weight memory access and linear-layer computation. The principal path uses compact dense operations with the precomputed W_{small}.

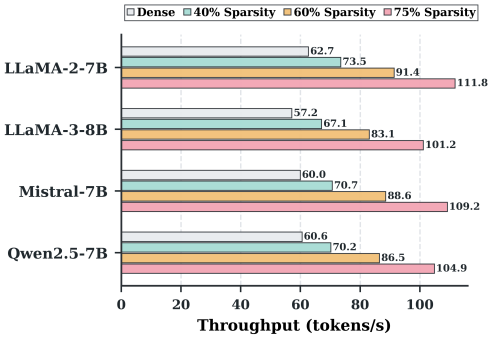

Figure 5: End-to-end generation throughput under different sparsity levels.

To evaluate the practical speedup of this implementation, we measure end-to-end generation throughput. Figure 5 compares the dense models with PRISM at 40%, 60%, and 75% sparsity, measured in tokens per second. PRISM accelerates generation across all evaluated models, with throughput increasing as sparsity rises. It achieves about $1.16\times$–$1.18\times$ speedup at 40% sparsity and $1.43\times$–$1.48\times$ at 60% sparsity. At 75% sparsity, the speedup reaches up to $1.82\times$. These end-to-end gains persist despite the overhead of the principal path and residual construction and hold across different model families. Principal-path overhead and memory usage are detailed in Appendix G.9.

4.5 Ablation Study

Table 2 reports the component ablation at 60% sparsity, with Full (Dense) serving as the original dense reference. DP and SA denote the principal-residual dual path and structure-aware sparsity allocation, respectively; Avg_7 is the mean accuracy over the seven reasoning benchmarks in Table 1. On Mistral-7B, compared with Original Top-K, enabling the dual path improves Avg_7 from 57.25 to 58.74. This confirms that preserving principal activation components before sparsifying the residual can reduce performance loss caused by direct Top-K selection. Adding structure-aware allocation further improves Avg_7 to 60.32, yielding a gain of 3.07 points over Original Top-K. On LLaMA-3-8B, enabling the dual path improves Avg_7 from 51.08 to 52.61, and adding structure-aware allocation further improves it to 54.74. This indicates that the two components provide complementary benefits across both models.

Table 2: Component ablation on LLaMA-3-8B and Mistral-7B at 60% sparsity. DP and SA denote dual path and structure-aware allocation, respectively. ✓ indicates that the component is enabled, and – indicates disabled or not applicable.

Variant	DP	SA	LLaMA-3-8B	Mistral-7B
Full (Dense)	–	–	68.41	67.43
Original Top-K	–	–	51.08	57.25
Dual Path Only	✓	–	52.61	58.74
PRISM	✓	✓	**54.74**	**60.32**

5 Conclusion

We present PRISM, a training-free activation sparsity framework for efficient LLM inference. Instead of directly sparsifying the full activation in the original coordinate space, PRISM decomposes the activation into principal and residual components. The principal component is preserved through a compact low-dimensional dense path, while Top-K sparse computation is applied only to the residual activation. This principal-residual design reduces projection errors and makes sparse computation more reliable at high sparsity. PRISM further selects an appropriate principal rank under a compute-equivalent budget and allocates residual sparsity with structure-aware corrections inside Transformer modules. Experiments on representative LLMs and reasoning benchmarks show that PRISM better preserves model performance across different tasks under high sparsity and achieves practical end-to-end generation speedup without training or weight updates.

USE OF AI ASSISTANTS

AI assistants were used to translate and polish author-provided passages, including improvements to English grammar, clarity, terminology consistency, and LaTeX presentation. They also helped check paragraph transitions and potentially ambiguous wording. The authors reviewed and edited the suggestions they adopted and verified the technical descriptions and experimental results. The research problem, method design, experiments, and final manuscript remain the authors' responsibility.

ETHICAL CONSIDERATIONS

PRISM uses existing model checkpoints and publicly available datasets to improve inference efficiency, without fine-tuning models or modifying their original weights. We collect no new human-subject or private user data. Unlabeled text is used only to collect offline activation statistics and determine sparsity configurations; we report aggregate results and do not release raw text examples. Public corpora may nevertheless contain personally identifying information or offensive content and cannot be assumed risk-free. Lower inference costs may also facilitate deployment of models with existing biases, privacy risks, or unsafe outputs. PRISM does not directly address these model-level risks, so appropriate evaluation and safeguards remain necessary before deployment.

REPRODUCIBILITY STATEMENT

The main text presents PRISM's principal-residual inference, rank selection, and structure-aware sparsity allocation. Appendices A, C, and D detail calibration, allocation, and compute-budget accounting. Appendix G provides the models, baselines, evaluation protocol, hyperparameters, and hardware environment. An anonymous code link is provided in the abstract.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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*, volume 2024, pp. 11682–11701, 2024.

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

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems*, 36: 4396–4429, 2023.

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, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 9, 2021.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

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, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

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.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

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

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

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

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

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

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, Yoon Kim, Ben Athiwaratkun, et al. Training-free activation sparsity in large language models. In *International Conference on Learning Representations*, volume 2025, pp. 98302–98322, 2025a.

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. *arXiv preprint arXiv:2507.01299*, 2025b.

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

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

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

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*, volume 2024, pp. 25378–25400, 2024.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

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

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

Mingjie Sun, Zhuang Liu, Anna Bair, and Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 4942–4964, 2024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

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

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.

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

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

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

A Details of PCA Calibration

This section describes how PRISM constructs the PCA bases used by the principal path. The calibration is performed offline on a small unlabeled calibration set and does not update the original model weights.

Activation collection. For each Transformer layer ℓ, PRISM collects input activations for target linear projections. Since several projections share the same input activation, we group the collected activations into four types:

$$\begin{aligned} &X_\ell^{\text{attn_in}}, \quad X_\ell^{\text{attn_out}}, \\ &X_\ell^{\text{mlp_in}}, \quad X_\ell^{\text{mlp_out}}. \end{aligned} \tag{A.1}$$

Here, $X_\ell^{\text{attn_in}}$ is shared by the q, k, and v projections; $X_\ell^{\text{attn_out}}$ is used by the o projection; $X_\ell^{\text{mlp_in}}$ is shared by the gate and up projections; and $X_\ell^{\text{mlp_out}}$ is used by the down projection. For each activation type, we flatten the batch and sequence dimensions into the token dimension, obtaining a calibration matrix $X \in \mathbb{R}^{N\times d}$, where N is the number of calibration tokens and d is the input dimension.

Uncentered PCA. PRISM uses uncentered second-moment PCA to construct the principal basis. For each calibration matrix X, we compute

$$M = \frac{1}{N} X^\top X \in \mathbb{R}^{d\times d}. \tag{A.2}$$

Let $u_1, \ldots, u_r$ denote the eigenvectors of M corresponding to its largest r eigenvalues. We form the PCA basis by stacking them column-wise:

$$P = [u_1, \ldots, u_r] \in \mathbb{R}^{d\times r}, \quad P^\top P = I_r. \tag{A.3}$$

Equivalently, P corresponds to the top-r right singular vectors of X. We use uncentered PCA because the online principal path directly computes $z = xP$. Centering would require storing the empirical mean and introducing additional correction terms during inference, while uncentered PCA keeps the offline calibration aligned with the online computation.

Projected weights and online computation. For a linear projection with weight matrix $W \in \mathbb{R}^{d_{\text{out}}\times d}$, PRISM precomputes the projected weight

$$W_{\text{small}} = WP \in \mathbb{R}^{d_{\text{out}}\times r}. \tag{A.4}$$

During inference, given an input activation x, PRISM computes the principal representation and residual activation as

$$\begin{aligned} z &= xP, \\ x_{\text{res}} &= x - zP^\top. \end{aligned} \tag{A.5}$$

The PRISM approximation is then

$$\widehat{y} = zW_{\text{small}}^\top + \mathcal{A}_{s_{\text{res}}}(x_{\text{res}})W^\top, \tag{A.6}$$

where $\mathcal{A}_{s_{\text{res}}}(\cdot)$ denotes residual sparsification under residual sparsity s_{res}. Since P is orthonormal, $PP^\top$ is an orthogonal projector. If the residual path were computed densely, the decomposition would exactly recover the dense linear output. Therefore, the approximation error comes only from sparsifying the residual activation:

$$y - \widehat{y} = \left(x_{\text{res}} - \mathcal{A}_{s_{\text{res}}}(x_{\text{res}})\right) W^\top. \tag{A.7}$$

Basis sharing. PRISM shares PCA bases across projections with the same input activation. For each layer ℓ, the basis-to-projection mapping is:

Basis	Projections
$P_\ell^{\text{attn_in}}$	q, k, v
$P_\ell^{\text{attn_out}}$	o
$P_\ell^{\text{mlp_in}}$	gate, up
$P_\ell^{\text{mlp_out}}$	down

Each projection still uses its own projected weight, such as $W_q P_\ell^{\text{attn_in}}$, $W_k P_\ell^{\text{attn_in}}$, and $W_v P_\ell^{\text{attn_in}}$. After calibration, PRISM stores the layer-wise PCA bases and the corresponding projected weights, which are reused during inference.

B Token-Level Sparsity Stability of Fixed-Threshold Routing

In the main experiments, we use Top-K selection for all sparse methods to ensure that each forward pass retains a fixed number of channels. This appendix further examines why this choice is necessary by analyzing the token-level sparsity produced by fixed-threshold routing.

Fixed-threshold methods such as the original TEAL first convert a target sparsity into a cutoff threshold using offline calibration statistics. During inference, the same threshold is applied to every token. For token t and channel i, the mask can be written as:

$$m_{t,i}(\tau) = \begin{cases} 1, & \text{if } |x_{t,i}| \geq \tau \\ 0, & \text{if } |x_{t,i}| < \tau \end{cases}, \tag{B.1}$$

where $x_{t,i}$ is the activation value of channel i at token t, and τ is the fixed threshold corresponding to a nominal target sparsity. However, runtime activation distributions vary across tokens. As a result, the same threshold can retain different numbers of channels for different tokens, leading to unstable realized sparsity.

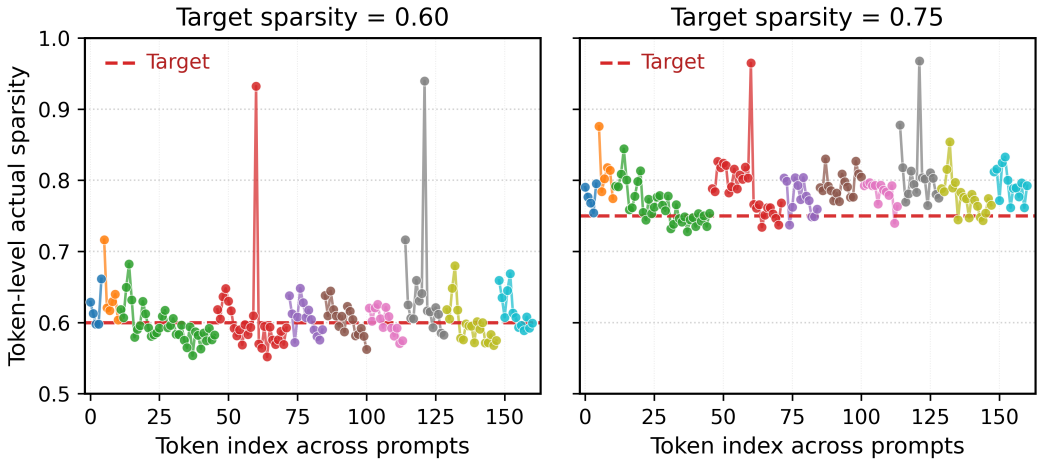

Figure 6: TEAL fixed-threshold routing causes token-level sparsity drift; Top-K avoids this by fixing the retained channel count.

Figure 6 shows the realized sparsity of TEAL fixed-threshold routing on LLaMA-2-7B under nominal sparsities of 0.6 and 0.75. This experiment is a lightweight runtime sparsity probe rather than a downstream task evaluation. We use short prompts covering simple continuation, question answering, and commonsense-style inputs, and apply fixed-threshold routing during the sparsified prefill tokens.

For each token t, we aggregate the masks over all layers and sparse projections, including q, k, v, o, gate, up, and down. The token-level realized sparsity is computed as

$$s_t = 1 - \frac{\text{kept}_t}{\text{total}_t}, \tag{B.2}$$

where kept_t is the number of retained activation channels for token t across all sparse projections, and total_t is the total number of activation channels involved in sparsity decisions.

If fixed-threshold routing precisely controlled the sparsity of each token, the points in Figure 6 would closely follow the dashed target line. Instead, the realized sparsity shows clear token-dependent fluctuations. Some tokens are sparsified much more or less than the nominal target, especially under the higher 0.75 sparsity setting. This indicates that fixed thresholds only approximate the desired sparsity on the calibration distribution, but do not guarantee stable sparsity for runtime tokens.

Algorithm 2 Block-wise Greedy Sparsity Optimization

Require: Transformer block B, calibration activations X, base step size α, n linear projections
Ensure: Recorded sparsity configurations $\{(\mathbf{p}, P)\}$

```
# Record size of each projection
f_i ← size(W_i) for i = 1, ..., n
F ← Σ_{i=1}^{n} f_i # Find size of block
# Initialize projection and block sparsities to zero
p ← 0_n, P ← 0
Y_gt ← B(X) # Dense reference output
while P < 1 do
    for i = 1 to n do
        δ_i ← α · (F / f_i)
# Error if we further sparsify this projection
        p_i ← p_i + δ_i
        Ŷ_i ← B_sparse(X; p)
        E_i ← ||Y_gt − Ŷ_i||_2
        p_i ← p_i − δ_i
    j ← arg min_i E_i
    p_j ← p_j + δ_j # Increment projection with lowest error
    P ← Σ_{i=1}^{n} p_i f_i / F
    Record (p, P)
```

Such sparsity drift affects fair comparison between sparse methods. Even with the same nominal sparsity, different tokens may use different effective compute budgets under fixed-threshold routing. Therefore, we adopt Top-K selection in our main experiments. Top-K directly fixes the number of retained channels in each forward pass, ensuring stable realized sparsity and better compute alignment across methods.

C Necessity and Details of Structure-Aware Sparsity Allocation

Necessity of structure-aware allocation. Existing training-free activation sparsity methods commonly allocate sparsity budgets according to local reconstruction errors. A representative example is the block-wise greedy optimization used in TEAL (Liu et al., 2025a). For each Transformer block, this procedure starts from a dense configuration and gradually increases sparsity inside the block. At each step, it temporarily sparsifies one projection, measures the resulting block-output error, and permanently increases the sparsity of the projection that introduces the smallest error. The projection size f_i is used to normalize the update step, so that each update contributes a comparable amount to the block-level sparsity budget P. Algorithm 2 summarizes this MSE-based greedy procedure.

Although block-output MSE provides a useful local approximation metric, it may not fully capture structural damage inside Transformer modules. This issue is particularly important in Attention, where perturbations to the q, k, and v projections can change the attention probability distribution before the final block output is formed. A small block-output MSE does not necessarily imply that the internal attention pattern is preserved.

To examine this mismatch, we analyze Mistral-7B under a target sparsity of $s = 0.6$. We focus on the q, k, and v projections in the Attention module and measure two perturbation metrics. The first metric, block-output MSE damage, measures the relative MSE increase in the Transformer block output caused by sparsifying a projection. The second metric, attention KL damage, measures the KL divergence (Kullback & Leibler, 1951) between the sparse and dense attention probability distributions. These two metrics allow us to compare whether a projection that appears safe under block-output MSE can still cause substantial drift in the internal attention structure.

For visualization, we convert both metrics into percentile ranks across all analyzed projections:

$$r_i^{\mathrm{MSE}} = \frac{\mathrm{rank}(\Delta_i^{\mathrm{MSE}})}{N-1} \times 100 \qquad r_i^{\mathrm{KL}} = \frac{\mathrm{rank}(\Delta_i^{\mathrm{KL}})}{N-1} \times 100 \tag{C.1}$$

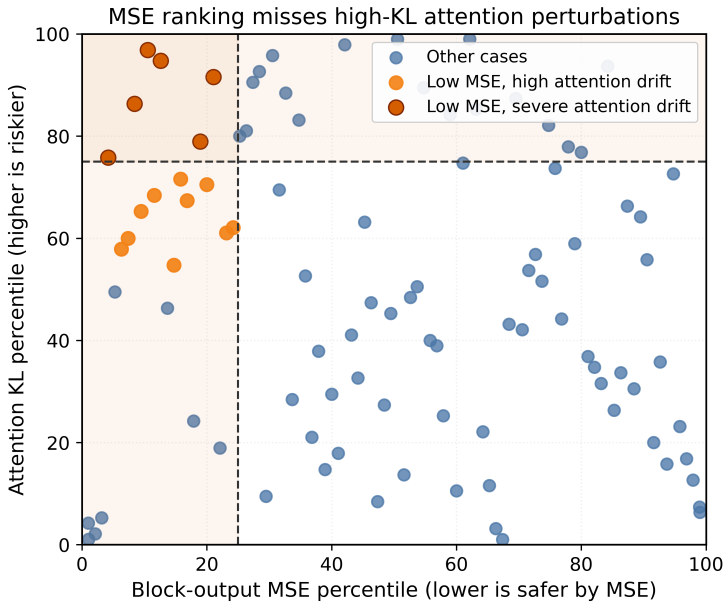

Figure 7: MSE-based allocation can miss high-KL attention perturbations.

where N is the number of analyzed projection cases. A lower MSE percentile means that the projection appears safer according to block-output MSE, while a higher KL percentile indicates stronger disruption to the attention distribution. The shaded upper-left region in Figure 7 marks cases with $r^{\mathrm{MSE}} \leq 25$ and $r^{\mathrm{KL}} \geq 75$. These are false-safe cases: they rank among the safest projections according to block-output MSE, but among the riskiest projections according to attention KL.

Figure 7 shows that such false-safe cases do exist. If sparsity allocation relies only on block-output MSE, these projections may be assigned overly aggressive sparsity because they appear locally safe. However, their high attention KL indicates that the internal attention distribution has already drifted substantially. Therefore, block-output MSE is useful for measuring local approximation error, but it is insufficient for characterizing structural damage inside Attention. This motivates PRISM to complement projection-level sensitivity with structure-aware signals, such as attention distribution drift, when allocating residual sparsity.

Detailed allocation procedure. Algorithm 1 produces a fixed residual sparsity table from unlabeled calibration data. For each layer, it first initializes projection sparsities and then performs one structure-aware correction. Let $w_{\ell,p}$ denote the relative dense computation cost of projection p, normalized such that $\sum_{p\in\mathcal{P}} w_{\ell,p} = 1$.

Phase I: Projection-level initialization. ALLOCATE starts from uniform residual sparsity, $s^{(0)}_{\ell,p} = S_{\mathrm{res}}$ for all p. For a sparsity table $\mathbf{s}$, let

$$E_\ell(\mathbf{s}) = \frac{\|Y_\ell^{\mathrm{dense}} - Y_\ell(\mathbf{s})\|_F^2}{\|Y_\ell^{\mathrm{dense}}\|_F^2 + \varepsilon} \tag{C.2}$$

be its relative layer-output error on fixed calibration inputs.

At iteration t, we temporarily increase only projection p's residual sparsity by δ and compute

$$e^{(t)}_{\ell,p} = \frac{E_\ell(\mathbf{s}^{(t)}_\ell + \delta \mathbf{1}_p) - E_\ell(\mathbf{s}^{(t)}_\ell)}{w_{\ell,p}\delta}. \tag{C.3}$$

Here, $\mathbf{1}_p$ selects projection p. The numerator is the error increase from the current sparsity table; the denominator accounts for the computation saved by the probe. The dense output remains the fixed reference throughout allocation.

With $\bar{e}_\ell^{(t)} = \sum_{p\in\mathcal{P}} w_{\ell,p} e_{\ell,p}^{(t)}$, ALLOCATE updates

$$s_{\ell,p}^{(t+1)} \leftarrow s_{\ell,p}^{(t)} - \eta\big(e_{\ell,p}^{(t)} - \bar{e}_\ell^{(t)}\big), \tag{C.4}$$

where $\eta > 0$ is the update step. Thus, projections with higher estimated error per unit of saved computation receive lower sparsity. After each update, we apply sparsity bounds and restore the compute-weighted budget $\sum_p w_{\ell,p} s_{\ell,p}^{(t+1)} = S_{\text{res}}$. We then reevaluate the sensitivities. The iterations stop when $\max_p |s_{\ell,p}^{(t+1)} - s_{\ell,p}^{(t)}| < \tau$, or when $T_{\max}$ iterations have been reached. The resulting table is denoted by $\mathbf{s}_\ell^0$.

Phase II: Structure-level correction. We consider $\mathcal{G} = \{\text{QK}, \text{VO}, \text{GU}, \text{D}\}$: q and k form QK, v and o form VO, gate and up form GU, and down forms D. For a base sparsity table $\mathbf{s}$, the notation $\mathbf{s}^{+\delta,g}$ means temporarily increasing the residual sparsity of projections in group g by δ while leaving other projections unchanged. This probe does not alter the final sparsity table.

Let $b_{\ell,g}(\mathbf{s})$ be the structural error measured after this group perturbation. For QK, we use attention-probability KL divergence:

$$b_{\ell,\text{QK}}(\mathbf{s}) = \text{Avg}\,\text{KL}\big(A_\ell^{\text{dense}} \,\big\|\, A_\ell(\mathbf{s}^{+\delta,\text{QK}})\big)\,. \tag{C.5}$$

The KL divergence is computed over causally visible key positions using the same masks for dense and perturbed attention, then averaged over valid queries, heads, and calibration examples.

For the remaining groups, we use the normalized error of their structural outputs:

$$b_{\ell,g}(\mathbf{s}) = \frac{\|Z_{\ell,g}^{\text{dense}} - Z_{\ell,g}(\mathbf{s}^{+\delta,g})\|_F^2}{\|Z_{\ell,g}^{\text{dense}}\|_F^2 + \varepsilon}, \quad g \in \{\text{VO}, \text{GU}, \text{D}\}. \tag{C.6}$$

Here, $Z_{\ell,g}$ denotes the attention output, gated MLP intermediate, or MLP output, respectively. The Frobenius norms aggregate over valid calibration tokens. Thus, $b_{\ell,g}(\mathbf{s})$ is the error *after* a temporary perturbation, not the difference between errors before and after it.

We apply the same group perturbation to the uniform table $\mathbf{s}_\ell^{\text{unif}} = S_{\text{res}}\mathbf{1}$ and the initial table $\mathbf{s}_\ell^0$. Writing their measured errors as $b_{\ell,g}^{\text{unif}}$ and $b_{\ell,g}^0$, respectively, RELATIVE-DAMAGE computes

$$r_{\ell,g} = \frac{[b_{\ell,g}^0 - b_{\ell,g}^{\text{unif}}]_+}{b_{\ell,g}^0 + b_{\ell,g}^{\text{unif}} + \varepsilon}, \qquad [x]_+ = \max(x, 0). \tag{C.7}$$

This score compares each group with its own uniform reference instead of directly averaging raw KL and output errors. It is a bounded allocation signal; equal scores across groups do not necessarily imply equal effects on downstream accuracy.

Let $\bar{r}_\ell = |\mathcal{G}|^{-1} \sum_{g\in\mathcal{G}} r_{\ell,g}$. The bounded correction is

$$\Delta_{\ell,g} = -B_g(r_{\ell,g} - \bar{r}_\ell), \qquad \widetilde{s}_{\ell,p} = s_{\ell,p}^0 + \Delta_{\ell,g(p)}. \tag{C.8}$$

Because $r_{\ell,g} \in [0,1]$, we have $|\Delta_{\ell,g}| \le B_g$. Groups with larger relative damage scores receive lower proposed residual sparsity. When all groups have the same score, no structure-level correction is applied.

Phase III: Feasibility projection. Budget rebalancing must preserve both the valid sparsity range and the correction bound. For each projection, set

$$L_{\ell,p} = \max(0, s_{\ell,p}^0 - B_{g(p)}), \qquad U_{\ell,p} = \min(0.98, s_{\ell,p}^0 + B_{g(p)}). \tag{C.9}$$

APPLY-BOUNDS first clips $\widetilde{s}_{\ell,p}$ to $[L_{\ell,p}, U_{\ell,p}]$; call the result $v_{\ell,p}$. REBALANCE then applies a shared offset λ_ℓ within these bounds:

$$s_{\ell,p} = \text{clip}(v_{\ell,p} + \lambda_\ell, L_{\ell,p}, U_{\ell,p}), \qquad \sum_{p\in\mathcal{P}} w_{\ell,p} s_{\ell,p} = S_{\text{res}}. \tag{C.10}$$

The offset is found by a one-dimensional search. A feasible result exists because the initial table $\mathbf{s}_\ell^0$ already satisfies the budget and lies in these intervals. Thus, the final table meets the residual budget and remains within $B_{g(p)}$ of the initial sparsity for every projection. It is fixed after calibration and used during inference.

D COMPUTE-EQUIVALENT SPARSITY ACCOUNTING

PRISM introduces a principal path in addition to the residual sparse path. This design improves the approximation quality, but it also introduces extra computation. Therefore, comparing PRISM with other activation sparsity methods at the same residual sparsity would not be fair, because PRISM would use additional computation from the principal path. To make the comparison fair, all sparsity levels reported for PRISM are defined as compute-equivalent global sparsity.

Let $\mathcal{P}$ denote the set of linear projections where activation sparsity is applied, and let $C_{\ell,p}$ be the dense computation cost of projection p in layer ℓ. The total dense computation over the sparsified scope is

$$C_{\text{dense}} = \sum_{\ell,p\in\mathcal{P}} C_{\ell,p}. \tag{D.1}$$

For a target global sparsity s, the corresponding computation budget is defined as

$$C_{\text{target}}(s) = (1-s)C_{\text{dense}}, \tag{D.2}$$

where C_{dense} is the dense computation cost over all sparsified projections, and $C_{\text{target}}(s)$ is the target computation budget under sparsity s. TEAL, R-Sparse, and PRISM are matched to the same target computation budget. For PRISM, this accounting includes both the principal and residual paths. For LaRoSA, the matched sparse-projection budget excludes online rotation overhead.

For standard Top-K activation sparsity, the computation is determined by the fraction of retained activation channels. In contrast, PRISM contains two computation paths:

$$\begin{aligned} C_{\text{PRISM}} &= C_{\text{pri}} + C_{\text{res}} \\ &= \sum_{\ell,p\in\mathcal{P}} C_{\ell,p}^{\text{pri}} + \sum_{\ell,p\in\mathcal{P}} \rho_{\ell,p} C_{\ell,p}, \end{aligned} \tag{D.3}$$

where C_{pri} is the total principal-path cost, C_{res} is the total residual sparse-path cost, $C_{\ell,p}^{\text{pri}}$ denotes the principal-path cost for projection p in layer ℓ, and $\rho_{\ell,p}$ denotes the residual keep ratio. Given a target sparsity s, PRISM first accounts for the principal-path computation and then assigns the remaining budget to the residual sparse path:

$$C_{\text{res}} = \sum_{\ell,p\in\mathcal{P}} \rho_{\ell,p} C_{\ell,p} \approx C_{\text{target}}(s) - C_{\text{pri}}. \tag{D.4}$$

Equivalently, PRISM's computation over the sparsified projections satisfies

$$C_{\text{PRISM}} \approx C_{\text{target}}(s) = (1-s)C_{\text{dense}}. \tag{D.5}$$

This accounting means that the residual path in PRISM is generally assigned a higher sparsity than the reported global sparsity, because part of the compute budget has already been used by the principal path. Thus, a reported sparsity level such as 60% or 75% refers to the total compute-equivalent sparsity of the complete PRISM computation, rather than the sparsity of the residual path alone.

The structure-aware allocation strategy is also applied under the same constraint. It redistributes the residual computation budget across layers and projections according to their structural sensitivity, but it does not change the global budget:

$$\sum_{\ell,p\in\mathcal{P}} \rho_{\ell,p} C_{\ell,p} \approx C_{\text{target}}(s) - C_{\text{pri}}. \tag{D.6}$$

After structure-aware corrections, the residual sparsity table is projected back to this budget constraint. Therefore, the improvement of PRISM does not come from using more computation, but from using the same compute-equivalent budget more effectively.

All principal-path configurations and residual sparsity tables are determined offline using calibration data. They are fixed before evaluation, and no evaluation data are used for calibration or configuration selection.

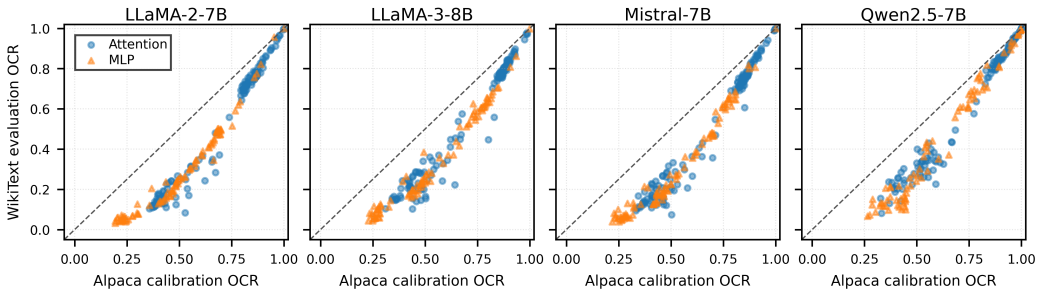

Figure 8: Cross-sample consistency of projection-wise OCR. PCA bases are computed offline from Alpaca calibration activations and then evaluated on both Alpaca calibration samples and WikiText samples. Each point corresponds to a layer-projection pair. The dashed line denotes $y = x$.

E Cross-Sample Consistency of Calibration Subspaces

PRISM constructs principal subspaces offline from calibration activations. A natural concern is whether these subspaces only fit the calibration samples, or whether they capture stable activation structures that transfer to unseen text. Prior work such as TEAL observes that activation distributions in modern LLMs exhibit relatively stable statistical patterns, which makes offline calibration feasible for training-free activation sparsity (Liu et al., 2025a). Here, we further examine this property from the perspective of projection-wise PCA output retention.

For a given layer and projection, the dense linear output is

$$Y_{\text{dense}} = XW^\top, \tag{E.1}$$

where X denotes the input activation matrix and W is the corresponding projection weight matrix. Given a PCA basis P computed from Alpaca calibration activations, the principal-path output is

$$Y_{\text{PCA}} = (XP)(WP)^\top, \tag{E.2}$$

where P is fixed after offline calibration. We define the output contribution retention (OCR) as

$$\text{OCR} = 1 - \frac{\|Y_{\text{dense}} - Y_{\text{PCA}}\|_F^2}{\|Y_{\text{dense}}\|_F^2}. \tag{E.3}$$

A higher OCR indicates that the PCA subspace preserves a larger fraction of the dense projection output.

Figure 8 compares the OCR measured on Alpaca calibration samples and WikiText evaluation samples. Each point corresponds to one layer-projection pair, with attention projections and MLP projections shown separately. Although many points lie below the diagonal, indicating that absolute OCR can decrease on WikiText, the projection-wise trends remain highly consistent across datasets.

To quantify this consistency, we compute both Pearson correlation and Spearman rank correlation between Alpaca OCR and WikiText OCR across all projections. The Pearson correlations are 0.976, 0.980, 0.980, and 0.986 for LLaMA-2-7B, LLaMA-3-8B, Mistral-7B, and Qwen2.5-7B, respectively. The corresponding Spearman correlations are 0.986, 0.976, 0.974, and 0.978. These consistently high correlations indicate that projections with high PCA output retention on calibration samples tend to remain highly retained on unseen WikiText samples, while projections with lower retention also remain relatively weaker.

This observation supports the use of offline calibration activations for constructing the principal path. Importantly, the result does not require Alpaca and WikiText activations to have identical distributions. Instead, it shows that the relative projection-wise retention pattern is stable across samples. Therefore, the calibration PCA basis captures transferable projection-level structure rather than merely overfitting the calibration text. This property is important for PRISM, since the principal path and the residual sparsity allocation are both determined offline and then reused directly during evaluation.

Table 3: Sensitivity to calibration dataset and sample size on LLaMA-2-7B at 60% compute-equivalent sparsity.

Calibration data	Samples	PIQA	ARC-E	ARC-C	HellaSwag	WinoGrande	Avg_5
Alpaca	32	73.59	66.17	37.85	67.21	63.64	61.69
Alpaca	64	74.27	66.75	38.05	67.20	63.14	61.88
Alpaca	128	74.27	67.51	38.65	67.40	63.22	62.21
WikiText-2	128	74.05	64.31	36.95	67.49	61.96	60.95

Table 4: Accuracy on additional models under different sparsity levels. AVG is the mean accuracy across the seven benchmarks. For each model and sparsity level, the best sparse-method result in each column is shown in bold.

Model	Sparsity	Method	PIQA	HellaSwag	ARC-C	ARC-E	WinoGrande	GSM8K	MMLU	AVG
Gemma-2-9B	0%	Dense	80.14	79.95	65.27	84.39	76.09	79.68	72.28	76.83
	60%	R-Sparse	78.84	72.85	57.68	81.06	**71.06**	72.02	64.73	71.18
		TEAL	78.03	74.25	58.96	81.31	68.74	71.00	66.61	71.27
		LaRoSA	**79.87**	**75.95**	60.92	82.41	70.64	68.23	63.34	71.62
		PRISM	**79.87**	75.28	**61.77**	**82.95**	70.17	**74.64**	**67.58**	**73.18**
	75%	R-Sparse	69.53	46.62	34.64	60.48	57.22	7.13	34.31	44.28
		TEAL	**72.69**	58.00	41.30	67.93	61.56	25.52	46.48	53.35
		LaRoSA	58.60	43.88	32.54	56.53	50.51	16.00	23.17	40.18
		PRISM	72.47	**59.89**	**45.65**	**72.01**	**61.64**	**31.08**	**46.74**	**55.64**
LLaMA-2-13B	0%	Dense	80.58	79.38	49.15	77.48	72.14	23.05	55.17	62.42
	60%	R-Sparse	75.41	69.61	42.49	70.75	66.14	6.37	44.85	53.66
		TEAL	77.91	75.15	45.48	74.61	67.48	13.50	45.34	57.07
		LaRoSA	78.02	75.22	45.22	73.02	66.46	13.95	45.71	56.80
		PRISM	**78.88**	**76.25**	**46.54**	**75.86**	**69.59**	**15.59**	**46.45**	**58.45**
	75%	R-Sparse	61.53	36.94	23.63	44.40	54.54	0.83	26.58	35.49
		TEAL	65.02	48.27	32.42	59.11	57.33	2.43	27.05	41.66
		LaRoSA	58.81	33.09	27.13	42.59	55.93	0.91	26.05	34.93
		PRISM	**68.66**	**50.44**	**34.94**	**63.55**	**58.33**	**4.05**	**29.55**	**44.22**

Sensitivity to calibration dataset and size. The preceding analysis shows that projection-wise PCA output retention follows similar trends on Alpaca and WikiText samples. We further assess the effect of calibration data on downstream accuracy by varying the dataset and sample size. Table 3 reports results for LLaMA-2-7B at 60% compute-equivalent sparsity, using 128 Alpaca samples as the default. All variants use unlabeled calibration text, without access to downstream evaluation labels. Avg_5 is the mean accuracy over PIQA, ARC-E, ARC-C, HellaSwag, and WinoGrande.

With only 32 Alpaca samples, PRISM obtains an Avg_5 of 61.69, within 0.52 points of the 128-sample default. Replacing Alpaca with WikiText-2 at the same sample size yields an Avg_5 of 60.95, a decrease of 1.26 points. Together with the cross-sample OCR consistency shown in Figure 8, these results suggest that PRISM is relatively insensitive to the tested calibration choices on this model and sparsity setting, without requiring a large calibration set.

F Results on Additional Models

We further evaluate PRISM on Gemma-2-9B and LLaMA-2-13B using the seven reasoning benchmarks considered in the main experiments. Table 4 reports the results at 60% and 75% sparsity.

PRISM achieves the highest seven-task average on both additional models at both sparsity levels. At 75% sparsity, it exceeds the strongest sparse baseline by 2.29 points on Gemma-2-9B and 2.56 points on LLaMA-2-13B.

G Detailed Experimental Settings

G.1 Hardware and Software Environment

All experiments are carried out on a local server equipped with 8 NVIDIA GeForce RTX 4090 GPUs, each with 24 GB of GPU memory. The server contains two AMD EPYC 9754 128-Core processors, providing 256 physical CPU cores and 512 hardware threads in total. It is also equipped with 1.5 TiB of system memory. The NVIDIA driver version is 580.105.08.

The software environment is based on Python 3.10.19. We use PyTorch 2.9.1 with the CUDA 12.8 runtime, Transformers 4.57.3, Hugging Face Datasets 3.6.0, lm-evaluation-harness 0.4.9.1, Accelerate 1.12.0, NumPy 2.2.6, SciPy 1.15.3, and Triton 3.5.1. Unless otherwise specified, model inference, calibration, and evaluation are conducted on GPU in half precision. All experiments are performed without fine-tuning, distillation, or numerical updates to the original model weights.

G.2 Models Used

We use six decoder-only LLMs in our experiments: four in the main experiments and two additional models in Appendix F. The selected models cover multiple model families and parameter scales from 7B to 13B.

LLaMA-2-7B. LLaMA-2 (Touvron et al., 2023) is a family of open-weight foundation and chat models released by Meta, with model sizes ranging from 7B to 70B parameters. LLaMA-2-7B is the smallest model in this family and is commonly used as a standard 7B-scale dense Transformer model in LLM efficiency studies. In our experiments, we use the 7B checkpoint.

LLaMA-2-13B. LLaMA-2-13B (Touvron et al., 2023) is a larger model from the same LLaMA-2 family. We include the 13B checkpoint to evaluate PRISM at a larger model scale.

LLaMA-3-8B. LLaMA-3 (Grattafiori et al., 2024) is a later generation of the LLaMA model family. The LLaMA-3 family includes pretrained and instruction-tuned models and is designed to support a broad range of capabilities, including language understanding, reasoning, coding, and tool-use settings. We use the 8B checkpoint as a representative recent LLaMA-family model.

Mistral-7B. Mistral-7B (Jiang et al., 2023) is a 7B-scale open-weight language model developed by Mistral AI. The model is known for its efficient architecture, including grouped-query attention and sliding-window attention, which are designed to improve inference efficiency and support longer context handling. We use the 7B checkpoint in our experiments.

Qwen2.5-7B. Qwen2.5 (Qwen et al., 2025) is a family of dense decoder-only language models released by the Qwen team. The Qwen2.5 series covers multiple model sizes and is designed for diverse language understanding, reasoning, mathematics, and coding tasks. We use the 7B checkpoint as a representative Qwen-family model.

Gemma-2-9B. Gemma-2-9B (Team et al., 2024) is a 9B-scale decoder-only language model from the Gemma 2 family. We include it to evaluate PRISM on an additional model family.

G.3 Baselines

We compare sparse methods with the original dense model and three representative training-free activation sparsity baselines.

Dense model. The dense model refers to the original full-computation checkpoint without activation sparsification. It is used as the reference for both accuracy and throughput comparisons.

TEAL. TEAL (Liu et al., 2025a) is a training-free activation sparsity method that applies magnitude-based input activation sparsification to Transformer linear layers. In our experiments, TEAL is evaluated under the same model checkpoints and task settings as other methods.

R-Sparse. R-Sparse (Zhang et al., 2025) is a training-free activation sparsity method that combines sparse computation with a low-rank compensation path. It serves as a strong sparse baseline for high-sparsity LLM inference.

LaRoSA. LaRoSA (Liu et al., 2025b) applies full-dimensional layerwise rotations to activations before selecting sparse channels in the rotated coordinates. It serves as a transformed-coordinate activation sparsity baseline for comparison with PRISM.

G.4 Calibration Dataset

Alpaca. Alpaca (Taori et al., 2023) is an instruction-following dataset constructed for studying instruction tuning of language models. The commonly used Alpaca release contains about 52K instruction-following examples, where each example consists of an instruction, an optional input, and a response. In our experiments, the unlabeled text from Alpaca is used only for offline calibration, including activation statistics collection and method-specific configuration generation. We do not use Alpaca responses as supervised training targets, and PRISM does not perform fine-tuning. No downstream evaluation examples are used for calibration.

G.5 Evaluation Datasets

We evaluate model performance on commonsense reasoning, mathematical reasoning, general knowledge, and language modeling benchmarks. The evaluation tasks are implemented with LM Evaluation Harness (Gao et al., 2021). Unless otherwise specified, we use the standard benchmark splits and evaluation settings provided by the evaluation framework, and we do not create new train, development, or test splits for any evaluation dataset.

PIQA. PIQA (Bisk et al., 2020) is a physical commonsense reasoning benchmark. The commonly used release contains 16,113 training examples, 1,838 validation examples, and 3,084 test examples. Each example presents a goal or physical situation together with two candidate solutions, and the model is required to select the more plausible one. This benchmark focuses on everyday physical interactions and commonsense knowledge.

ARC-Easy and ARC-Challenge. The AI2 Reasoning Challenge (ARC) (Clark et al., 2018) is a multiple-choice science question answering benchmark. It is divided into ARC-Easy and ARC-Challenge. ARC-Easy contains 2,251 training examples, 570 validation examples, and 2,376 test examples. ARC-Challenge contains 1,119 training examples, 299 validation examples, and 1,172 test examples. ARC-Easy contains relatively easier science questions, while ARC-Challenge contains more difficult questions that require stronger reasoning and knowledge.

HellaSwag. HellaSwag (Zellers et al., 2019) is a commonsense natural language inference benchmark. The commonly used release contains 39,905 training examples, 10,042 validation examples, and 10,003 test examples. Given a context, the model must choose the most plausible continuation from several candidate endings. The benchmark is designed to be easy for humans but challenging for language models.

WinoGrande. WinoGrande (Sakaguchi et al., 2021) is a large-scale commonsense reasoning benchmark based on Winograd-style coreference resolution. The commonly used WinoGrande-XL configuration contains 40,398 training examples, 1,267 validation examples, and 1,767 test examples. Each example requires the model to resolve an ambiguous reference using contextual and commonsense cues.

GSM8K. GSM8K (Cobbe et al., 2021) is a grade-school mathematics benchmark. The commonly used main split contains 7,473 training examples and 1,319 evaluation examples. It consists of natural language math word problems that typically require multi-step arithmetic reasoning. We use GSM8K to evaluate mathematical reasoning after sparsification.

MMLU. MMLU (Hendrycks et al., 2020) is a multitask benchmark covering a wide range of subjects, including STEM, humanities, social sciences, and professional knowledge. It covers 57

tasks and contains 285 development examples, 1,531 validation examples, and 14,042 test examples, together with an auxiliary training set of 99,842 examples. It is used to evaluate broad knowledge and multitask understanding.

WikiText2. WikiText2 (Merity et al., 2016) is a language modeling dataset derived from high-quality Wikipedia articles. The standard word-level split contains 2,088,628 training tokens, 217,646 validation tokens, and 245,569 test tokens. In our experiments, WikiText2 samples are used to measure structure-related drift in attention probabilities, attention context outputs, and model output distributions.

G.6 Evaluation Protocol

We use LM Evaluation Harness (Gao et al., 2021) for benchmark evaluation. For PIQA, ARC-Easy, ARC-Challenge, HellaSwag, and WinoGrande, we use zero-shot multiple-choice evaluation. For GSM8K and MMLU, we use 5-shot evaluation with the same prompting settings across all evaluated methods. All methods are evaluated with the same local model checkpoints, cached datasets, task definitions, and evaluation scripts.

G.7 Sparsification Settings and Compute Budget

For sparse methods, we use Top-K selection to ensure that each forward pass retains a fixed number of activation channels. This avoids token-level sparsity drift caused by fixed thresholds and makes the realized computation budget more stable across inputs.

For PRISM, the reported sparsity denotes compute-equivalent global sparsity. Both the principal path and the residual sparse path are counted in the computation budget. Given a target sparsity level, the residual keep ratio is adjusted so that the complete PRISM computation is aligned with the target budget. In the benchmark comparisons, PRISM and all three baselines are evaluated at the same target sparsity. For LaRoSA, the matched sparse-projection budget excludes online rotation overhead.

G.8 Hyperparameter Search and Selected Configurations

This subsection supplements the sparsification settings and compute-equivalent budget described in Section 4.1. All hyperparameters are selected using the same unlabeled calibration data used for offline statistics. No downstream evaluation labels or test examples are used for rank selection, residual sparsity allocation, or structure-aware correction. After calibration, the selected ranks and residual sparsity tables are fixed and reused during evaluation.

Rank-selection hyperparameters. PRISM selects the principal rank using the compute-constrained rule in Section 3.3. For each candidate rank, we first account for the principal-path cost and remap the residual Top-K budget so that the total computation matches the target compute-equivalent sparsity. We then evaluate the normalized projection approximation error on calibration activations:

$$D_{\mathrm{proj}}(r) = \frac{\|Y_{\mathrm{dense}} - Y_{\mathrm{PRISM}}(r)\|_F^2}{\|Y_{\mathrm{dense}}\|_F^2}. \tag{G.1}$$

The candidate set is

$$\mathcal{R} = \{4, 8, 16, 24, 32, 48, 64, 80, 96, 128\}. \tag{G.2}$$

Let $D_{\min} = \min_{r\in\mathcal{R}} D_{\mathrm{proj}}(r)$. We first filter candidate ranks by the following near-optimal condition:

$$D_{\mathrm{proj}}(r) \le (1+\epsilon) D_{\min}, \tag{G.3}$$

where $\epsilon = 0.05$. After checking model-level output drift on calibration sequences, we select the lowest-cost rank from the remaining candidates. This rule avoids increasing the principal-path rank when the additional reduction in calibration error is marginal, thereby preserving more computation for the residual Top-K path. The PCA basis is computed up to the maximum candidate rank once during calibration, and lower-rank configurations reuse the first r principal directions.

Table 5: Calibration-selected principal ranks. For Qwen2.5-7B, the rank is selected from $\{16, 24\}$ by the same compute-constrained calibration rule under different target budgets.

Model	Calibration-selected rank
LLaMA-2-7B	64
LLaMA-3-8B	64
Mistral-7B	24
Qwen2.5-7B	16 or 24
Gemma-2-9B	64
LLaMA-2-13B	16

Table 6: Effect of principal rank on Mistral-7B at 60% compute-equivalent sparsity.

Rank r	16	24	32	64
Avg_5	68.71	**69.65**	69.57	69.19

Effect of principal rank. Table 6 evaluates different principal ranks on Mistral-7B at 60% compute-equivalent sparsity. Avg_5 denotes the mean accuracy over PIQA, ARC-E, ARC-C, HellaSwag, and WinoGrande.

Avg_5 increases from 68.71 at $r = 16$ to 69.65 at $r = 24$, but does not improve further at larger ranks. Under a fixed compute-equivalent budget, increasing the principal rank also reduces the computation available to the residual path. This result illustrates the trade-off considered by our compute-constrained rank-selection rule. The rank is selected using unlabeled calibration data rather than downstream task accuracy.

Structure-aware allocation hyperparameters. All evaluated models use the same structure-aware allocation pipeline in Algorithm 1. The structural signals and bounded-shift rule are defined in Appendix C. Here, we specify the allocation settings and the selection of correction bounds.

We use $\delta = 0.02$ for both projection-level and structure-level probes. For the projection-level ALLOCATE procedure, the update step is $\eta = 0.005$, the stopping tolerance is $\tau = 0.001$, and the maximum number of iterations is $T_{\max} = 30$. During this procedure, projection sparsities are restricted to $[0, 0.96]$ to leave room for the $+\delta$ probes. After structure-level correction, the valid sparsity range is $[0, 0.98]$. These settings are fixed across models and target sparsity levels.

For each structural group, the correction bound is chosen from

$$B_g \in \{0, 0.005, 0.010, \ldots, 0.050\}. \tag{G.4}$$

For each model, we start with $B_g = 0.01$ for all groups and perform two coordinate search passes in the fixed order QK, VO, GU, and D. At each step, we test the candidate bounds for one group while holding the other bounds fixed. Each candidate is evaluated after bounded correction and budget rebalancing, so that candidates at a given target sparsity use the same residual compute budget.

We select the bound vector using the average KL divergence between dense and sparse next-token distributions on unlabeled calibration data, averaged over the reported target sparsity levels. Ties are resolved in favor of the smaller bound. This common model-output criterion is used only to select the bounds; the allocation itself uses the group-specific structural signals defined above. The selected bound vector is fixed for each model and reused across downstream tasks.

All models compute projection-level sensitivities and structure-level relative damage scores using the same procedure. A positive bound permits a correction but does not imply that the realized shift is nonzero: equal group scores produce no structure-level correction. The smaller bounds reported for the LLaMA-family models limit the maximum permitted shifts, while the bounds for Mistral-7B and Qwen2.5-7B permit larger shifts where the measured scores warrant them. In every case, feasibility projection restores the target residual budget while respecting the correction bounds.

Table 7: Calibration-selected structure-aware configurations. B_g limits the sparsity correction for group g; the realized correction also depends on its relative damage score.

Model	Rank	Correction bounds
LLaMA-2-7B	64	$B_g = 0.01$ for all groups
LLaMA-3-8B	64	$B_g = 0.01$ for all groups
Mistral-7B	24	$B_g = 0.035$ for all groups
Qwen2.5-7B	16 or 24	$B_{\mathrm{QK}} = B_{\mathrm{VO}} = 0.03$, $B_{\mathrm{GU}} = 0.04$, $B_{\mathrm{D}} = 0.05$
Gemma-2-9B	64	$B_g = 0.02$ for all groups
LLaMA-2-13B	16	$B_g = 0.01$ for all groups

The ranks in Table 5 and the correction bounds in Table 7 are fixed before downstream evaluation. No task-specific tuning is performed, and configuration selection uses no downstream labels or test examples.

G.9 End-to-End Inference Speed Evaluation

We evaluate inference speed using a modified GPT-Fast-based generation implementation. All speed measurements use the same local model checkpoints and tokenizer files as those used in the accuracy evaluation. Dense models are evaluated with the original dense generation path, while PRISM is evaluated with the principal-residual inference path and our Triton-based sparse linear kernels.

Before generation, PRISM loads the offline PCA bases, compact projected weights, and sparsity configuration tables. These offline artifacts are prepared before timing and remain fixed during generation. The measured runtime includes the online computation of the principal path, residual construction, Top-K selection, and sparse residual linear operations. Thus, the reported throughput reflects the actual end-to-end generation cost of the PRISM inference path rather than an idealized sparse-only computation.

For a fair comparison, we keep the decoding configuration identical across dense and sparse runs. This includes the input prompt length, maximum generation length, sampling configuration, precision setting, and tokenizer. When supported, CUDA graph compilation and kernel warm-up are performed before timing. We then measure throughput after warm-up and average the results over repeated generation runs.

Throughput is reported as generated tokens per second:

$$\text{Throughput} = \frac{\#\text{generated tokens}}{\text{generation time}}. \tag{G.5}$$

For sparse runs, we report the target sparsity together with the corresponding effective compute setting when applicable. This ensures that the speed comparison is made under consistent model checkpoints, decoding settings, and sparsity configurations.

Principal-path overhead. We further profile LLaMA-2-7B on an RTX 4090 with batch size 1. At 60% global compute-equivalent sparsity, the principal path accounts for approximately 1.98% of the original dense computation over the sparsified projections. The compute-weighted residual sparsity is therefore adjusted to approximately 61.98%, keeping the two paths within the target 40% compute budget described in Appendix D.

As shown in Table 8, the principal path occupies approximately 11.5% of PRISM's end-to-end time at 60% sparsity, despite its much smaller share of dense computation. Narrow principal-path operations at batch size 1 can have limited GPU utilization. Nevertheless, the savings from sparse residual computation produce a net $1.46\times$ speedup in this setting.

Caching the PCA bases and projected weights increases peak GPU memory from 14.57 GB for dense inference to 15.25 GB for PRISM, an increase of 0.68 GB (approximately 4.7%). The bases are computed offline and shared within a layer among projections receiving the same input activation, as detailed in Appendix A.

Table 8: End-to-end inference breakdown for LLaMA-2-7B on an RTX 4090 with batch size 1. Time percentages are relative to PRISM's total execution time at each sparsity level.

Sparsity	Principal	Top-K	Sparse residual	Other ops	Throughput (tokens/s)
Dense	–	–	–	–	62.72 (1.00×)
60%	11.5%	11.9%	54.2%	22.4%	91.38 (1.46×)
75%	12.1%	12.5%	51.8%	23.6%	111.83 (1.78×)

H Artifact Licenses and Intended Use

This work uses publicly available artifacts, including model checkpoints, calibration datasets, evaluation benchmarks, and evaluation tools. The model checkpoints used in our experiments include LLaMA-2, LLaMA-3, Mistral, Qwen2.5, and Gemma 2. The datasets and benchmarks include Alpaca for calibration, and PIQA, ARC-Easy, ARC-Challenge, HellaSwag, WinoGrande, GSM8K, MMLU, and WikiText2 for evaluation and diagnostic analysis. We also use standard open-source evaluation tools such as LM Evaluation Harness.

All third-party artifacts are used in accordance with their original licenses, access conditions, and intended-use policies. We cite the original creators of the models, datasets, benchmarks, and evaluation tools used in this paper. Our use of these artifacts is limited to academic research on training-free activation sparsity and efficient LLM inference. We do not redistribute third-party model weights, datasets, or benchmark contents beyond what is permitted by their respective licenses. Users should obtain these artifacts from their official sources and follow the corresponding license terms.

The calibration data are used only for collecting activation statistics and generating offline sparsity configurations. No evaluation samples are used for calibration or configuration selection. Our released code includes the implementation of PRISM and experiment scripts, but does not redistribute third-party model weights or benchmark data.