PROX: TRAINING-FREE FFN ACTIVATION SPARSIFICATION IN LLMS VIA APPROXIMATE INTERMEDIATE-STATE SALIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Feed-forward networks (FFNs) dominate memory traffic and computation in large language model (LLM) inference, making them a primary target for activation sparsification. However, existing training-free methods suffer substantial model-quality degradation at high FFN sparsity due to limitations in their channel-selection strategies. We observe that the SwiGLU intermediate state provides a highly effective channel-selection signal, but obtaining it requires costly dense computation. To address this, we present *Prox*, a training-free framework for sparse SwiGLU FFNs based on the insight that sparse execution needs only the channel mask induced by the intermediate state, which can be constructed from the magnitude ranking of its entries (i.e., channel salience) rather than their precise values. Prox works in two stages: Stage 1 approximates the ranking via input sparsity and quantized proxy weights to construct the channel mask; Stage 2 computes the selected channels exactly using full-precision weights, enabling sparse execution of all three projections. Extensive evaluations across ten LLMs from six model families show that Prox consistently outperforms training-free baselines at various sparsity levels, achieving up to a $2.07\times$ end-to-end decoding speedup at 70% FFN sparsity, and is compatible with quantization and sparse attention.

1 INTRODUCTION

Efficient inference is critical for deploying large language models (LLMs), especially in resource-constrained and latency-sensitive settings. In small-batch autoregressive decoding, parameter transfers from off-chip high-bandwidth memory (HBM) to on-chip storage dominate latency. Modern LLMs commonly adopt SwiGLU feed-forward networks (FFNs), whose three projection matrices account for most model parameters, memory traffic, and computation (Figure 1), making them a high-leverage target for acceleration.

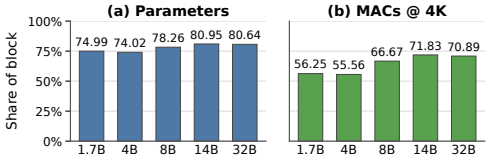


Figure 1: FFNs account for the majority of parameters and multiply-accumulate operations (MACs) under 4K context length in Qwen3 Transformer blocks across model scales.

Existing approaches accelerate LLM inference through weight quantization (Dettmers et al., 2022; Frantar et al., 2022; Lin et al., 2024) and model pruning (Ma et al., 2023; Frantar & Alistarh, 2023; Ashkboos et al., 2024; Men et al., 2025; Qiao et al., 2025; Sun et al., 2024), which lower memory footprint or computational cost by altering model precision or structure. Orthogonal to these static modifications, activation sparsity (Liu et al., 2023; Lee et al., 2024; Cheon & Kang, 2025; Liu et al., 2025; Chen et al., 2023; Raihan & Aamodt, 2020; Kurtz et al., 2020; Szatkowski et al., 2026) mitigates memory and compute bottlenecks by dynamically skipping low-salience channels and associated MACs. Prior works exploit input-dependent activation sparsity via either training-based predictors or training-free heuristics. However, these predictors must be trained separately for each model, while existing training-free schemes often suffer substantial accuracy degradation as sparsity increases due to limitations in their channel-selection strategies.

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.

Our analysis reveals two findings: (1) The SwiGLU intermediate state provides a highly effective channel-selection signal but requires costly dense computation. Our oracle experiment demonstrates that selecting via this exact state incurs less than a 3% relative perplexity increase at 70% sparsity across most models. (2) Sparse execution requires only the channel mask, which depends only on the magnitude ranking of the state entries rather than their exact values. Our experiment shows that a low-cost input-sparse proxy preserves this ranking and closely matches the oracle mask.

Based on these two key findings, we propose *Prox*, a two-stage training-free method for sparse SwiGLU inference. Stage 1 (*lightweight proxy construction*) combines magnitude-based input sparsity with quantized proxy weights to estimate both SwiGLU branches and construct a shared intermediate-channel mask. Stage 2 (*exact sparse computation*) computes the selected channels with the original up and gate weights, and applies the same mask to the down projection. Prox further allocates computation between the two stages under a target sparsity budget and uses dedicated kernels implemented in TileLang (Wang et al., 2026) to reduce weight accesses and computation across all three projections.

We evaluate Prox on ten representative LLMs from six model families. The results show that Prox consistently outpaces state-of-the-art training-free baselines, particularly at 60-70% FFN sparsity, yielding up to a $2.07\times$ end-to-end decoding speedup with minimal accuracy loss. Crucially, Prox is fully orthogonal to quantization and sparse attention, enabling synergistic efficiency gains.

Our main contributions are summarized as follows:

- **Insight**. We identify the FFN intermediate state as an effective channel-selection signal and show that a low-cost proxy preserves the magnitude ranking needed to recover the mask induced by the exact state. This motivates proxy-based channel selection followed by exact computation of the retained channels.
- **Prox Framework**. We propose Prox, a training-free two-stage framework that uses quantized proxies for channel selection and computes retained values exactly, sparsifying all three SwiGLU projections while preventing error accumulation.
- **Comprehensive Evaluation**. Our extensive experiments demonstrate that Prox delivers superior accuracy-efficiency trade-offs across various models and sparsity levels, while maintaining full compatibility with quantization and sparse attention.

2 Background and Related Work

2.1 SwiGLU Feed-Forward Networks

LLMs are typically built with stacked Transformer layers, each comprising an attention block and an FFN. While early Transformer FFNs employed ReLU or GELU activation functions, Gated Linear Units (GLUs) (Dauphin et al., 2017; Shazeer, 2020) have largely superseded them. Today, SwiGLU (Shazeer, 2020) remains the most widely adopted GLU variant in modern LLMs, with representative models including Qwen3 (Team, 2025b), Llama-3 (AI@Meta, 2024), and Mistral (Jiang et al., 2023).

A SwiGLU FFN comprises three projection matrices: the up projection $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, the gate projection $\boldsymbol{W}_{\text{gate}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, and the down projection $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$. Given an input representation $\boldsymbol{x} \in \mathbb{R}^{d_{\text{model}}}$, its forward computation is defined as

$$\text{FFN}(\boldsymbol{x}) = \big(\boldsymbol{x}\boldsymbol{W}_{\text{up}} \odot \text{SiLU}(\boldsymbol{x}\boldsymbol{W}_{\text{gate}})\big)\boldsymbol{W}_{\text{down}}.$$

For subsequent discussion, we define three variables:

$$\boldsymbol{u} = \boldsymbol{x}\boldsymbol{W}_{\text{up}}, \quad \boldsymbol{h} = \text{SiLU}(\boldsymbol{x}\boldsymbol{W}_{\text{gate}}), \quad \boldsymbol{s} = \boldsymbol{u} \odot \boldsymbol{h},$$

denoting the up-branch activation, the gated nonlinear activation, and the intermediate SwiGLU state, respectively.

2.2 Activation Sparsity

Activation sparsity mitigates both HBM-to-register memory bandwidth overhead and redundant computations on low-salience channels. For a linear projection $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W}$ with $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ and

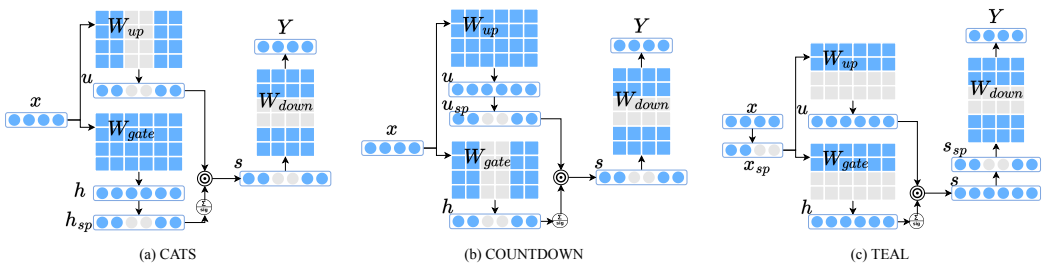


Figure 2: Computation patterns of representative sparsification methods for SwiGLU FFNs.

$\boldsymbol{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, sparsity can be enforced along either the input dimension or output dimension. Let $\mathcal{I}(\boldsymbol{m}) = \{i \mid m_i = 1\}$ denote the set of indices retained by a binary mask $\boldsymbol{m}$ along the target dimension.

Input sparsity applies a mask $\boldsymbol{m} \in \{0,1\}^{d_{\text{in}}}$ to the input dimension. Specifically, $m_i = 0$ masks the i-th input channel, eliminating the multiplication of x_i with the corresponding row of $\boldsymbol{W}$.

$$\Pi^{\text{in}}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{m}) = \boldsymbol{x}_{[\mathcal{I}(\boldsymbol{m})]} \boldsymbol{W}_{[\mathcal{I}(\boldsymbol{m}),:]}.$$

Output sparsity applies a mask $\boldsymbol{m} \in \{0,1\}^{d_{\text{out}}}$ to the output dimension and computes only the selected output coordinates:

$$\Pi^{\text{out}}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{m}) = \boldsymbol{x}\boldsymbol{W}_{[:,\mathcal{I}(\boldsymbol{m})]},$$

where the selected columns of $\boldsymbol{W}$ yield the retained output channels.

The evolution of activation sparsification closely tracks the architectural transition of Transformer FFNs from ReLU-based designs to modern SwiGLU variants.

Early activation sparsification methods primarily target ReLU-based FFNs, skipping input-dependent inactive or low-magnitude neurons to reduce computation and memory traffic (Li et al., 2022; Liu et al., 2023; Alizadeh et al., 2024). While these approaches excel with ReLU-induced exact or near-exact zeros, they cannot directly adapt to SiLU-based SwiGLU FFNs.

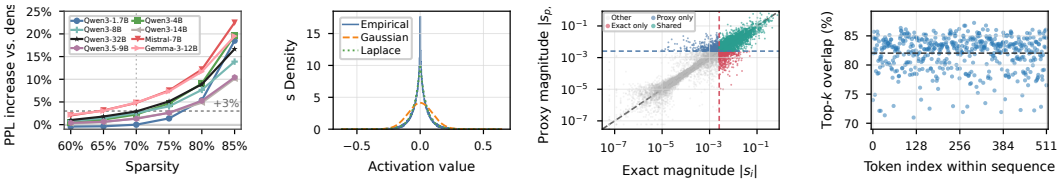
A second line of work, ReLUfication, converts non-ReLU FFNs to ReLU-style variants via activation replacement and post-training adaptation (Mirzadeh et al., 2024; Zhang et al., 2024; Song et al., 2025; 2024). However, these approaches typically require explicit architectural modifications or resource-intensive retraining, such as continued training or knowledge distillation.

For native SwiGLU FFNs, predictor-based methods estimate salient intermediate channels before executing the full FFN. For instance, Cheon & Kang (2025) require training model-specific, layer-wise low-rank predictors on FFN inputs to select channels from the intermediate state $\boldsymbol{s}$. However, limited prediction accuracy can degrade downstream model quality.

In contrast, training-free methods sparsify native SwiGLU FFNs using activation magnitudes. They differ mainly in two dimensions: which activation signals guide channel selection and how the resulting masks induce sparse execution. Figure 2 compares three representative approaches along these two dimensions. CATS (Lee et al., 2024) selects intermediate channels based on the magnitude of the densely computed gate activation $\boldsymbol{h}$, inducing output sparsity in the up projection and input sparsity in the down projection (Figure 2(a)). COUNTDOWN (Cheon & Kang, 2025) instead performs channel selection using the magnitude of the densely computed up-branch activation $\boldsymbol{u}$, yielding output sparsity in the gate projection and input sparsity in the down projection (Figure 2(b)).[1] Unlike these single-branch methods, TEAL (Liu et al., 2025) applies input sparsity to $\boldsymbol{x}$ in the up and gate projections and to $\boldsymbol{s}$ in the down projection (Figure 2(c)) simultaneously. R-Sparse follows a similar input-sparsity pattern while incorporating low-rank compensation across linear layers (Zhang et al., 2025).

Across the three representative methods in Figure 2, model quality deteriorates substantially as sparsity increases, exposing key limitations in their channel-selection strategies. CATS and COUNTDOWN estimate intermediate-channel salience solely from $\boldsymbol{h}$ and $\boldsymbol{u}$, respectively; each relies on a

[1] Throughout this paper, COUNTDOWN refers to the training-free M-COUNTDOWN variant unless stated otherwise.



(a) Oracle-$\boldsymbol{s}$ (b) Intermediate-state distribution (c) Proxy value alignment (d) Selection overlap

Figure 3: (a) Perplexity increase vs. sparsity under oracle-$\boldsymbol{s}$ channel selection. (b) Empirical distribution of $\boldsymbol{s}$ in Qwen3-8B Layer 8 and fitted Gaussian and Laplace densities; (c) Exact vs. proxy intermediate-state magnitudes for one representative token from Qwen3-8B Layer 3 at 70% effective sparsity; colors indicate whether each channel is retained by both top-ranking selections, only one selection, or neither. The two selections achieve 82.77% overlap. (d) Exact/proxy top-ranking overlap across 512 tokens; the dashed line marks the mean overlap of 82.04%.

single-branch signal that captures only partial information of the joint intermediate state $\boldsymbol{s} = \boldsymbol{u} \odot \boldsymbol{h}$, leading to misranked salient channels. TEAL instead applies input sparsity at two successive points: first to $\boldsymbol{x}$, the shared input to the up and gate projections, and then to the resulting intermediate state $\boldsymbol{s}$, the input to the down projection. The latter therefore operates on an already approximated intermediate state, allowing errors from the two sparsification steps to compound through the FFN.

These limitations motivate a more effective channel-selection mechanism tailored for training-free SwiGLU sparsification.

3 Motivation

3.1 Channel Selection with Intermediate State $\boldsymbol{s}$

Following the weighted-sum view of SwiGLU FFNs (Cheon & Kang, 2025), the FFN output is a weighted sum of down-projection rows:

$$\mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{s}\boldsymbol{W}_{\mathrm{down}} = \sum_{i=1}^{d_{\mathrm{ff}}} s_i \boldsymbol{W}_{\mathrm{down}}[i,:].$$

Analysis. The input-dependent coefficient s_i governs the output of each row, making $|s_i|$ a natural metric for channel selection. Beyond measuring output contribution, the intermediate state $\boldsymbol{s}$ is natively aligned with the shared intermediate dimension of all three FFN projections: each index corresponds to an output channel in both up- and gate-projections, as well as the matching input channel in the down-projection. Accordingly, if the exact intermediate state $\boldsymbol{s}$, denoted as $\boldsymbol{s}_{\mathrm{exact}}$, were available prior to FFN execution, a single $\boldsymbol{s}$-based mask would induce output sparsity in the up- and gate-projections alongside input sparsity in the down-projection, thereby sparsifying all three projections in a unified and consistent manner.

Assumption and oracle evaluation. To assess the performance upper bound of $\boldsymbol{s}$-based channel selection on modern SwiGLU models, we evaluate an ideal diagnostic baseline, *oracle-$\boldsymbol{s}$*, under the assumption that $\boldsymbol{s}_{\mathrm{exact}}$ is available at no cost prior to FFN execution. Given a target sparsity ρ, oracle-$\boldsymbol{s}$ retains the $(1-\rho)d_{\mathrm{ff}}$ coordinates of $\boldsymbol{s}$ with the largest magnitudes to construct a mask across all intermediate channels.

Figure 3(a) empirically validates the above analysis: *oracle-$\boldsymbol{s}$* preserves model quality across moderate-to-high sparsity levels. Even at 70% sparsity, most tested models experience less than a 3% relative perplexity increase over their dense counterparts. Figure 3(b) provides a complementary explanation for this robustness. The distribution of $\boldsymbol{s}$ is sharply peaked and Laplacian-like around zero, indicating that many channels carry negligible magnitudes and can be safely skipped with limited impact on FFN output.

Consequently, while the channel-selection signal provided by $\boldsymbol{s}_{\mathrm{exact}}$ serves as an ideal reference, the key challenge in practice is that the exact intermediate state is unavailable until after executing the dense computations for the up- and gate-projections.

3.2 Low-Cost Input-Sparse Proxy

In fact, sparse execution does not require the exact values of $\boldsymbol{s}$ in advance; it only requires the binary channel mask that $\boldsymbol{s}_{\text{exact}}$ would induce. Since this mask is determined by the magnitude ranking of entries in $\boldsymbol{s}$, our key insight is that an effective proxy needs only to preserve relative channel rankings rather than accurately construct the exact values of $\boldsymbol{s}$. We therefore design a low-cost proxy that bypasses full projection computations and serves exclusively to derive the channel mask.

Magnitude-based input sparsity is naturally suited for preserving this ranking, as it minimizes perturbations to the projected values that determine the channel ranking. Let $\boldsymbol{x}_{\text{sp}}$ denote the sparse vector retaining the largest-magnitude entries of $\boldsymbol{x}$, and let $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{x}_{\text{sp}}$ be the discarded residual. For a fixed density, this selection minimizes $\|\boldsymbol{r}\|_2$ across all coordinate-sparse approximations, thus discarding the minimum amount of input energy. For any projection matrix $\boldsymbol{W}$, this residual directly bounds the resulting output error:

$$\|\boldsymbol{x}\boldsymbol{W} - \boldsymbol{x}_{\text{sp}}\boldsymbol{W}\|_2 = \|\boldsymbol{r}\boldsymbol{W}\|_2 \leq \|\boldsymbol{r}\|_2\|\boldsymbol{W}\|_2.$$

Input sparsity thus reduces the effective input dimension of the projection, bounds the overall perturbation, and preserves all output channels.

Applying the same sparse input to the up and gate projections yields proxy activations $\tilde{\boldsymbol{u}}$ and $\tilde{\boldsymbol{h}}$, whose element-wise product $\tilde{\boldsymbol{s}} = \tilde{\boldsymbol{u}} \odot \tilde{\boldsymbol{h}}$ is used to rank intermediate channels by magnitude. Crucially, the proxy activations are used exclusively to construct the channel mask and are never propagated downstream through the FFN. Their approximation errors affect model outputs solely via channel-selection mismatches, which occur primarily when perturbations shift near-threshold channels across the selection boundary.

As Figure 3(c) illustrates, while exact and input-sparse proxy values exhibit numerical differences, they yield largely overlapping channel sets. Figure 3(d) further corroborates strong overlap among top-ranked channels across tokens. This ranking stability motivates our two-stage design: the input-sparse proxy is used exclusively to construct the channel mask, followed by exact computation of the retained channels.

Appendix A.2 provides additional quantitative analyses of intermediate-state compressibility and proxy ranking preservation across representative FFN layers.

4 Methodology

4.1 Two-Stage Sparse Inference

We propose Prox, a two-stage training-free framework for sparse SwiGLU FFN inference. As illustrated in Figure 4, given an FFN input $\boldsymbol{x}$, Prox proceeds in the following two stages.

Stage 1: lightweight proxy construction. The proxy activations are used exclusively to construct the channel mask $\boldsymbol{m}_{\boldsymbol{s}}$. Their approximation errors thus affect outputs only by altering channel selection, rather than propagating as activation values. This error isolation enables more aggressive approximation in the proxy path: Prox combines input sparsity with quantized proxy weights to reduce both the effective input dimension and weight memory overhead.

For FFN layer ℓ, let s_1 and s_2 denote the target sparsities of Stage 1 input entries and Stage 2 intermediate channels, respectively. To achieve the target input sparsity s_1, Prox constructs a binary input mask $\boldsymbol{m}_{\boldsymbol{x}} \in \{0,1\}^{d_{\text{model}}}$ by retaining entries with magnitudes exceeding a layer-wise threshold:

$$[\boldsymbol{m}_{\boldsymbol{x}}]_i = \mathbf{1}\left[|x_i| \geq \tau_{\boldsymbol{x},\ell}^{s_1}\right], \qquad i = 1, \ldots, d_{\text{model}},$$

where $\mathbf{1}[\cdot]$ denotes the indicator function. The threshold $\tau_{\boldsymbol{x},\ell}^{s_1}$ is calibrated offline on layer-ℓ inputs to approximately match the target sparsity s_1 over a calibration dataset. Given quantized proxy weights $\widetilde{\boldsymbol{W}}_{\text{up}}$ and $\widetilde{\boldsymbol{W}}_{\text{gate}}$, the proxy path leverages this input sparsity to compute only entries retained by $\boldsymbol{m}_{\boldsymbol{x}}$:

$$\tilde{\boldsymbol{u}} = \Pi^{\text{in}}(\boldsymbol{x}, \widetilde{\boldsymbol{W}}_{\text{up}}, \boldsymbol{m}_{\boldsymbol{x}}),\ \tilde{\boldsymbol{h}} = \text{SiLU}\Big(\Pi^{\text{in}}(\boldsymbol{x}, \widetilde{\boldsymbol{W}}_{\text{gate}}, \boldsymbol{m}_{\boldsymbol{x}})\Big).$$

It then constructs the proxy intermediate signal $\tilde{\boldsymbol{s}} = \tilde{\boldsymbol{u}} \odot \tilde{\boldsymbol{h}}$.

For the target intermediate-channel sparsity s_2, Prox applies a corresponding layer-wise magnitude threshold to the proxy intermediate state:

$$[\boldsymbol{m_s}]_j = \mathbf{1}\left[|\tilde{s}_j| \geq \tau^{s_2}_{\boldsymbol{s},\ell}\right], \qquad j = 1, \ldots, d_{\text{ff}},$$

where $\boldsymbol{m_s} \in \{0,1\}^{d_{\text{ff}}}$ denotes the active channels retained for the exact computation path. Analogous to $\tau^{s_1}_{\boldsymbol{x},\ell}$, the threshold $\tau^{s_2}_{\boldsymbol{s},\ell}$ is calibrated offline and remains fixed during inference.

Stage 2: exact sparse computation. Using the channel mask $\boldsymbol{m_s}$ from Stage 1, Prox performs exact SwiGLU computations exclusively over the selected channels using the original full-precision model weights.

$$\begin{aligned}
\boldsymbol{u}_{\text{sp}} &= \Pi^{\text{out}}(\boldsymbol{x}, \boldsymbol{W}_{\text{up}}, \boldsymbol{m_s}),\\
\boldsymbol{h}_{\text{sp}} &= \text{SiLU}(\Pi^{\text{out}}(\boldsymbol{x}, \boldsymbol{W}_{\text{gate}}, \boldsymbol{m_s})),\\
\boldsymbol{s}_{\text{sp}} &= \boldsymbol{u}_{\text{sp}} \odot \boldsymbol{h}_{\text{sp}},\\
\boldsymbol{y} &= \boldsymbol{s}_{\text{sp}} \boldsymbol{W}_{\text{down}}[\mathcal{I}(\boldsymbol{m_s}), :].
\end{aligned}$$

4.2 Sparsity Allocation between Two Stages

s_1 and s_2 jointly govern the overall computational cost of Prox. Reducing Stage 1 sparsity retains more input entries to yield a more informative proxy signal, while reducing Stage 2 sparsity preserves more intermediate channels for exact SwiGLU computation. Since both adjustments increase computation, s_1 and s_2 must be jointly optimized under a target FFN compute budget.

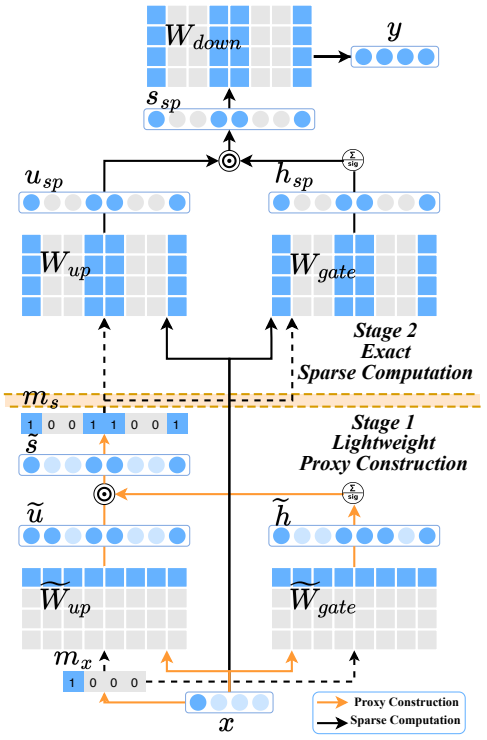


Figure 4: Overview of Prox. A quantized proxy derives a shared intermediate-channel mask; then the selected intermediate channels are computed with the original weights.

We model computational overhead with a normalized cost framework. Dense *up*-, *gate*-, and *down*-projections each carry a unit cost, yielding a total dense FFN computational cost $C_{\text{Dense}} = 3$. Let α denote the normalized cost of a quantized proxy projection relative to a full dense projection. Given sparsity s_1 and s_2, Stage 1 executes two proxy projections over a $(1 - s_1)$ fraction of the input dimension, while Stage 2 executes three full-precision projections over a $(1 - s_2)$ fraction of the intermediate channels. The total normalized cost is:

$$C_{\text{Prox}} = 2\alpha(1 - s_1) + 3(1 - s_2).$$

We define the resulting effective sparsity e relative to the dense FFN baseline as:

$$e = 1 - \frac{C_{\text{Prox}}}{C_{\text{Dense}}} = s_2 - \frac{2\alpha(1 - s_1)}{3}.$$

Our budget allocation policy is inspired by the findings of oracle-$\boldsymbol{s}$ in Figure 3(a). The results demonstrate that sparsifying approximately 70% of the exact intermediate channels incurs negligible performance degradation across the evaluated models, highlighting substantial channel redundancy in Stage 2 exact computation. We thus set $s_{\text{ref}} = 0.7$ as a quality-preserving anchor for Stage 2. Given a target effective sparsity e, we initially set $s_2 = s_{\text{ref}}$ and allocate the remaining compute budget to Stage 1 according to the effective-sparsity relation above.

However, Stage 1 sparsity cannot be increased indefinitely; removing excessive input coordinates diminishes the fidelity of the proxy signal, thereby compromising the reliability of channel selection. To mitigate this, we constrain s_1 to $[s_{1,\min}, s_{1,\max}] = [0, 0.7]$, ensuring at least 30% of the input entries are retained. Appendix D.3 further shows that this shared cap achieves the best average score across three Qwen3 model scales.

If the Stage 1 sparsity implied by $s_2 = s_{\text{ref}}$ falls outside this interval, we clamp s_1 to the nearest bound and dynamically adjust s_2 to satisfy the target effective sparsity e. Detailed budget allocations across operating points are provided in Appendix B.2.

Table 1: Downstream scores across model families and sparsity levels (higher is better).

		Qwen3				Qwen3.5		Ministral	Mistral	Llama3	Gemma3
Variant	**Sparsity**	**1.7B**	**4B**	**8B**	**14B**	**4B**	**9B**	**3.3B**	**7B**	**8B**	**12B**
Dense	0	61.9	72.1	76.3	77.4	69.7	70.0	69.4	69.6	69.6	77.3
CATS	40%	35.7	44.6	57.9	67.2	58.8	59.9	58.2	61.9	64.2	73.3
COUNTDOWN	40%	57.8	68.7	72.2	75.5	61.5	65.2	65.7	65.4	66.6	75.5
TEAL	40%	59.4	**70.5**	75.1	**76.6**	68.2	69.5	**68.9**	**67.1**	**69.1**	72.9
Prox	40%	**62.0**	70.1	**75.3**	76.4	**68.3**	**69.9**	68.2	66.8	68.7	**77.8**
CATS	50%	27.7	32.9	47.4	53.7	51.7	55.1	37.7	47.7	42.9	62.0
COUNTDOWN	50%	50.6	65.0	70.9	74.5	59.8	62.8	63.5	61.7	59.3	71.6
TEAL	50%	57.7	67.7	73.7	76.2	**67.9**	68.6	67.1	65.8	66.6	69.0
Prox	50%	**60.0**	**68.7**	**75.1**	**76.3**	67.0	**69.7**	**67.5**	**67.2**	**67.8**	**77.3**
CATS	60%	27.6	26.6	29.8	31.3	36.8	42.0	27.8	29.4	28.6	32.6
COUNTDOWN	60%	34.1	45.9	56.9	67.3	48.5	54.1	44.6	49.3	41.5	56.2
TEAL	60%	55.5	64.8	72.2	74.5	64.2	66.8	63.7	64.1	64.4	67.5
Prox	60%	**60.2**	**65.0**	**73.5**	**75.0**	**65.6**	**68.7**	**66.3**	**66.4**	**66.9**	**70.5**
TEAL	70%	41.0	52.9	64.4	69.9	53.9	59.2	50.9	57.7	54.1	64.5
Prox	70%	**56.6**	**59.6**	**67.2**	**74.6**	**59.6**	**64.2**	**61.1**	**64.5**	**61.4**	**70.7**

4.3 Hardware-Aware Implementation

We asymmetrically quantize FP16 $\boldsymbol{W}_{\text{up}}$ and $\boldsymbol{W}_{\text{gate}}$ to INT2 Stage 1 proxy weights, keeping them and their quantization metadata in GPU memory. We implement dedicated sparse kernels in TileLang (Wang et al., 2026), tailored for single-batch autoregressive decoding.

Stage 1 uses two kernels. The first jointly computes the up- and gate-proxy projections, sharing hidden-state reads and skipping threshold-rejected inputs. It decodes packed INT2 weights in registers, accumulates in FP32, and merges partials before memory writes to reduce traffic. The second reduces these partials and fuses SwiGLU with magnitude thresholding to produce the Stage 2 mask $\boldsymbol{m_s}$ without materializing separate FP16 up and gate proxy vectors. Launch geometries and reductions are autotuned across GPU architectures, matrix shapes, and sparsity levels.

In Stage 2, a fused output-sparse kernel computes exact up and gate projections with SwiGLU over channels selected by $\boldsymbol{m_s}$, followed by an input-sparse down-projection GEMV. Mask handling within the computation loops restricts weight accesses and MACs to active coordinates and channels.

5 Experiments

Models and datasets. We evaluate Prox on SwiGLU-based model families, including Qwen3 (Team, 2025b), Qwen3.5 (Qwen Team, 2026), Ministral, Mistral (Jiang et al., 2023), and Llama-3 (AI@Meta, 2024). To assess generalization beyond SwiGLU, we also evaluate Gemma-3 (Team, 2025a) that features a GeGLU-style FFN. Downstream accuracy is evaluated via the EleutherAI LM Harness (Gao et al., 2024) across six tasks: 8-shot GSM8K (Cobbe et al., 2021), 5-shot MMLU (Hendrycks et al., 2020), 10-shot HellaSwag (Zellers et al., 2019), zero-shot ARC-Easy and 25-shot ARC-Challenge (Clark et al., 2018), and zero-shot PIQA (Bisk et al., 2020).

Baselines. We compare Prox against three training-free activation sparsity baselines: CATS (Lee et al., 2024), COUNTDOWN (Cheon & Kang, 2025), and TEAL (Liu et al., 2025). We use M-COUNTDOWN as the COUNTDOWN baseline because it achieves higher average downstream scores than D-COUNTDOWN under practical inference.

To ensure a fair comparison focused on FFN sparsification, we apply all methods exclusively to FFN projections and report sparsity at the FFN-module level. Specifically, for CATS and COUNTDOWN, a projection-level sparsity of k yields an effective sparsity of $2k/3$; TEAL's effective sparsity is averaged across its three FFN projections, while Prox's effective sparsity follows the cost model in Sec. 4.2. Although the theoretical INT2-to-FP16 bit-width ratio implies $\alpha = 0.125$, it ignores quantization overhead. We therefore set $\alpha = 0.2$, a conservative rounded approximation based

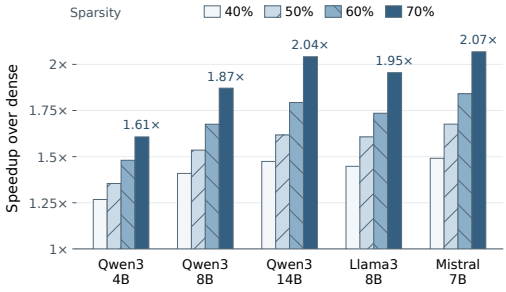


Figure 5: Single-batch decoding speedup of Prox over dense inference on an NVIDIA A6000 GPU across models and FFN sparsity levels.

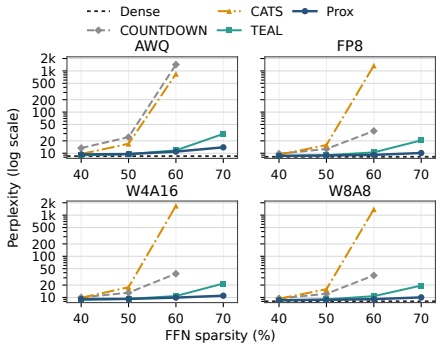


Figure 6: Perplexity of quantized Qwen3-8B across FFN sparsity levels (log scale; lower is better).

on the measured INT2-to-FP16 latency ratios across two GPUs and four Qwen3 scales (details in Appendix B.1).

These baselines primarily target decoding-phase sparsification. Nevertheless, for log-likelihood tasks, we sparsify the full sequence to ensure hidden states are computed under sparsity; for generation tasks such as GSM8K, we maintain a dense prompt prefill and sparsify decoding exclusively.

5.1 Results

Downstream accuracy. As shown in Table 1, Prox demonstrates its superiority at high sparsity levels while remaining competitive with TEAL at 40% sparsity. Specifically, Prox surpasses TEAL on 9 out of 10 evaluated models at 50% sparsity and consistently outperforms it across all 10 models at 60% and 70% sparsity. In contrast, CATS and COUNTDOWN exhibit premature degradation as sparsity increases; restricted to sparsifying only two FFN matrices, they fail to reach the 70% operating point sustained by TEAL and Prox. Detailed perplexity results are provided in Appendix D.1.

End-to-end decoding speedup. We benchmark Prox under single-batch end-to-end decoding on an NVIDIA A6000 GPU. As Figure 5 shows, its speedup increases consistently with FFN sparsity across all evaluated models, reaching 1.48–$2.07\times$ at 60–70% sparsity. These practical gains are enabled by our dedicated TileLang kernels, which fuse operations within each stage and restrict weight accesses and MACs to the selected coordinates and channels. A detailed latency breakdown and additional throughput measurements on an RTX 3090 GPU are provided in Appendix B.3.

Compatibility with model quantization. Model quantization and FFN sparsification are complementary techniques that can be combined for efficient inference. We evaluate Prox and three baselines on Qwen3-8B across four quantization configurations (AWQ, FP8, W4A16, W8A8) by measuring WikiText perplexity. As Figure 6 shows, Prox achieves the lowest perplexity across all four configurations at high sparsities, with its advantage becoming increasingly pronounced in high-sparsity regimes. Combining low-bit weights with activation sparsity can further reduce weight traffic and improve inference speed, but realizing these gains requires specialized kernels that jointly support sparse execution and low-bit arithmetic, which we leave to future work.

Compatibility with sparse attention. Prox operates solely on FFNs and is thus orthogonal to sparse-attention techniques, enabling combined computational savings across both attention and FFN modules. Table 2 presents results integrated with RocketKV (Behnam et al., 2025) and HShare (Wu et al., 2025) at 16K and 32K sequence lengths. Integrating Prox with either backend further improves decoding throughput while largely preserving long-context accuracy.

5.2 Ablation Studies

A1: Effect of proxy-weight precision. A1 tests whether higher proxy precision improves model quality under a fixed compute budget by replacing the INT2 proxy weights with FP16 weights.

Table 2: Compatibility of Prox (60% FFN sparsity) with different sparse-attention backends.

	Decoding Throughput (tok/s)		Accuracy
Variant	**16K**	**32K**	**LongBench / RULER**
Dense (Qwen3-8B)	33.1 (1.00×)	23.9 (1.00×)	100.0 / 100.0
RocketKV	40.7 (1.23×)	41.0 (1.72×)	100.0 / 100.0
RocketKV + Prox	63.6 (1.92×)	62.5 (2.62×)	96.6 / 98.0
HShare	34.6 (1.05×)	39.0 (1.63×)	93.5 / 99.0
HShare + Prox	40.0 (1.21×)	42.1 (1.76×)	92.5 / 100.0

Table 3: Ablation results on Qwen3-8B across target effective sparsity levels. Entries are aggregate downstream scores; higher is better.

	Target effective sparsity e			
Variant	**40%**	**50%**	**60%**	**70%**
Prox (full)	**75.3**	**75.1**	**73.5**	**67.2**
FP16 proxy (A1)	72.5	71.6	67.3	59.6
Fixed $s_1 = 0.7$ (A2)	74.2	72.8	71.8	67.2
Proxy-value propagation (A3)	68.4	67.2	66.2	40.9

Compute is reallocated between stages to account for the increased proxy cost while maintaining the same target budget. It underperforms Prox at all four sparsity levels (Table 3), indicating that higher proxy precision does not offset its increased cost under a fixed budget.

A2: Effectiveness of target-aware compute allocation. A2 tests the benefit of adapting compute allocation to the target budget by fixing Stage 1 sparsity at $s_1 = 0.7$ and adjusting only Stage 2 to meet each target budget. Prox outperforms this fixed allocation at three sparsity levels and matches it at 70% sparsity. The broader Qwen3-8B sweep in Appendix D.2 shows that no fixed split is uniformly best across targets. Together, these results support adapting the stage-wise allocation to the target budget.

A3: Proxy-value propagation. A3 tests whether proxy values can replace the exact values computed in Stage 2 by passing the selected proxy values directly to the down projection. To isolate the effect of replacing exact values with proxy values, A3 preserves Prox's Stage 1 computation and channel mask rather than reallocating the saved budget to improve the proxy. The degradation highlights the importance of recomputing the selected channels in Stage 2.

5.3 Discussion

Prox currently targets single-batch autoregressive decoding, a typical setting for latency-sensitive, on-device LLM applications. Consequently, its primary limitation is the lack of support for large-batch serving. Extending proxy-guided sparsification and its underlying sparse kernels to efficient batched inference remains a promising avenue for future exploration. Additionally, retaining Stage 1 INT2 proxy weights in GPU memory introduces an approximate 6% weight-storage overhead. Future work could reduce this overhead through a sliding-window scheme that dynamically loads and evicts proxy weights across Transformer layers.

6 Conclusion

We present Prox, a training-free FFN sparsification framework for SwiGLU LLMs. Prox uses a low-cost quantized proxy for input-dependent channel selection, followed by exact computation of the selected values. With target-aware stage-wise allocation and tailored sparse kernels, Prox achieves up to a 2.07× end-to-end decoding speedup at 70% FFN sparsity. Across ten LLMs from six model families, Prox consistently outperforms training-free baselines across sparsity levels and remains compatible with model quantization and sparse attention.

AI USE STATEMENT

In this work, we used ChatGPT 5.6 to assist with language editing, improve conciseness, and support the implementation of some experimental scripts. The authors conceived the research ideas, developed the methodology, designed and conducted the experiments, and interpreted the results. All AI-assisted text and code were reviewed and validated by the authors, who take full responsibility for the final content of this work, including its claims, results, and artifacts.

REPRODUCIBILITY STATEMENT

The Methodology section specifies the two-stage sparse inference procedure, sparsity-allocation rule, quantization scheme, and hardware-aware implementation, while the Experiments section documents the evaluated models, datasets, baselines, sparsity definitions, and evaluation protocols. The appendix provides the FFN parameter and MAC calculation (Appendix A.1), measured proxy-cost ratios and stage-wise configurations (Appendices B.1 and B.2), threshold calibration and runtime sparsity measurements (Appendix C.2), detailed latency and perplexity results (Appendices B.3 and D.1), and sensitivity analyses (Appendices D.2 and D.3).

The source code is provided in the supplementary material.

REFERENCES

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. Llm in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12562–12584, 2024.

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.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. RocketKV: Accelerating long-context LLM inference via two-stage KV cache compression. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=RyOpooIxDF.

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.

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

Jaewon Cheon and Pilsung Kang. Countdown: Contextually sparse activation filtering out unnecessary weights in down projection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 28381–28397, 2025.

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. *arXiv preprint arXiv:2110.14168*, 2021.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

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, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

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.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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`.

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, 2025.

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.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025.

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.

Kangyu Qiao, Shaolei Zhang, and Yang Feng. Ig-pruning: Input-guided block pruning for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10629–10640, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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

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

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, et al. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 2626–2644, 2025.

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

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.

Filip Szatkowski, Patryk Będkowski, Alessio Devoto, Jan Dubiński, Pasquale Minervini, Mikołaj Piórczyński, Simone Scardapane, and Bartosz Wójcik. Universal properties of activation sparsity in modern large language models, 2026. URL https://arxiv.org/abs/2509.00454.

Gemma Team. Gemma 3. 2025a. URL https://goo.gle/Gemma3Report.

Qwen Team. Qwen3 technical report, 2025b. URL https://arxiv.org/abs/2505.09388.

Lei Wang, Yu Cheng, Yining Shi, Zhiwen Mo, Zhengju Tang, Wenhao Xie, Tong Wu, Lingxiao Ma, Yuqing Xia, Jilong Xue, et al. Tilelang: Bridge programmability and performance in modern neural kernels. In *The Fourteenth International Conference on Learning Representations*, 2026.

Huaijin Wu, Lianqiang Li, Hantao Huang, Yi Tu, Jihang Zhang, Minghui Yu, and Junchi Yan. Hshare: Fast llm decoding by hierarchical key-value sharing. In *The Thirteenth International Conference on Learning Representations*, 2025.

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.

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

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.

APPENDIX

A Analytical Foundations **14**

A.1 FFN Parameter and MAC Calculation 14

A.2 Activation Distributions and Ranking Preservation 14

B System Configuration and Efficiency **16**

B.1 Empirical Proxy-Cost Ratios 16

B.2 Stage-Wise Compute Configurations 16

B.3 End-to-End Latency Analysis 16

C Thresholding Mechanism and Calibration **17**

C.1 Threshold-Based Selection and Token-Adaptive Computation 17

C.2 Threshold Calibration and Runtime Sparsity 19

D Additional Evaluation and Sensitivity Analyses **21**

D.1 Perplexity Results 21

D.2 Sensitivity to the Fixed Stage 1 Sparsity 21

D.3 Sensitivity to the Stage 1 Sparsity Cap 22

D.4 Detailed Analysis with TEAL 22

D.5 Compatibility with Fine-Tuning 23

A Analytical Foundations

A.1 FFN Parameter and MAC Calculation

We calculate the parameter and MAC shares in Figure 1 as follows. Let S denote the sequence length, d_{model} the hidden dimension, d_{ff} the FFN intermediate dimension, H_q and H_{kv} the numbers of query and key–value heads, and d_h the head dimension. The query and key–value projection dimensions are defined as:

$$d_q = H_q d_h, \qquad d_{\text{kv}} = H_{\text{kv}} d_h.$$

The parameter counts of the FFN and one Transformer block are

$$\begin{aligned} P_{\text{FFN}} &= 3 d_{\text{model}} d_{\text{ff}}, \\ P_{\text{Block}} &= 3 d_{\text{model}} d_{\text{ff}} + 2 d_{\text{model}} (d_q + d_{\text{kv}}) \\ &\quad + 2 d_{\text{model}} + 2 d_h, \\ R_{\text{param}} &= \frac{P_{\text{FFN}}}{P_{\text{Block}}}. \end{aligned}$$

At sequence length S, the corresponding MAC counts are

$$\begin{aligned} M_{\text{FFN}} &= 3 S d_{\text{model}} d_{\text{ff}}, \\ M_{\text{Block}} &= 3 S d_{\text{model}} d_{\text{ff}} + 2 S d_{\text{model}} (d_q + d_{\text{kv}}) \\ &\quad + 2 S^2 d_q, \\ R_{\text{MAC}} &= \frac{M_{\text{FFN}}}{M_{\text{Block}}}. \end{aligned}$$

We set $S = 4096$ and use the model configurations in Table 4. The calculation includes the Q/K/V/O projections and RMSNorm parameters, but excludes embeddings, the LM head, softmax, RoPE, activation functions, and other element-wise operations.

A.2 Activation Distributions and Ranking Preservation

Figure 7 contrasts the activation distributions at the two sides of a SwiGLU FFN. The FFN input $\boldsymbol{x}$, shared by the up and gate projections, is approximately Gaussian, whereas the intermediate state $\boldsymbol{s}$ supplied to the down projection is approximately Laplacian. These distinct shapes clarify why magnitude sparsification plays different roles in exact-$\boldsymbol{s}$ channel selection, TEAL's successive input-sparsity scheme, and the selection-oriented proxy used by Prox.

Intermediate-state compressibility. The Laplacian-like shape of $\boldsymbol{s}$ consists of a sharp central peak and comparatively heavy tails, so many channels have negligible magnitudes while a small fraction carries most of the activation energy. This concentration can be quantified under an ideal Laplace model. Let $S \sim \text{Laplace}(0, b)$, and prune the fraction p of entries with the smallest magnitudes. With $a_p = -\log(1-p)$, the retained fraction of squared activation energy is

$$q_{\text{Lap}}(p) = \frac{\mathbb{E}\left[S^2 \mathbf{1}(|S| > b a_p)\right]}{\mathbb{E}[S^2]} = \frac{1-p}{2}\left(a_p^2 + 2a_p + 2\right).$$

At $p = 0.7$, retaining only the largest-magnitude 30% of entries preserves approximately 87.9% of the squared activation energy. Although activation energy alone does not determine the final FFN error because each coefficient s_i scales a different row of $\boldsymbol{W}_{\text{down}}$, this calculation provides a distributional explanation consistent with the oracle-$\boldsymbol{s}$ result in Figure 3(a): most low-magnitude intermediate channels can be removed with little quality degradation.

Value distortion under input sparsity. The Gaussian-like FFN input is less concentrated than the Laplacian-like intermediate state, so magnitude pruning at the same sparsity discards a larger fraction of its energy. Following the idealized analysis of magnitude-based activation sparsity in TEAL (Liu et al., 2025), let $X \sim \mathcal{N}(0, \sigma_x^2)$, assume independent isotropic Gaussian weights, and prune the fraction p of input entries satisfying $|X| \le \sigma_x z_p$, where $z_p = \Phi^{-1}((1+p)/2)$, and Φ and ϕ denote the standard Gaussian cumulative distribution and density functions. The retained input-energy fraction is

$$q_{\text{G}}(p) = 1 - p + 2 z_p \phi(z_p),$$

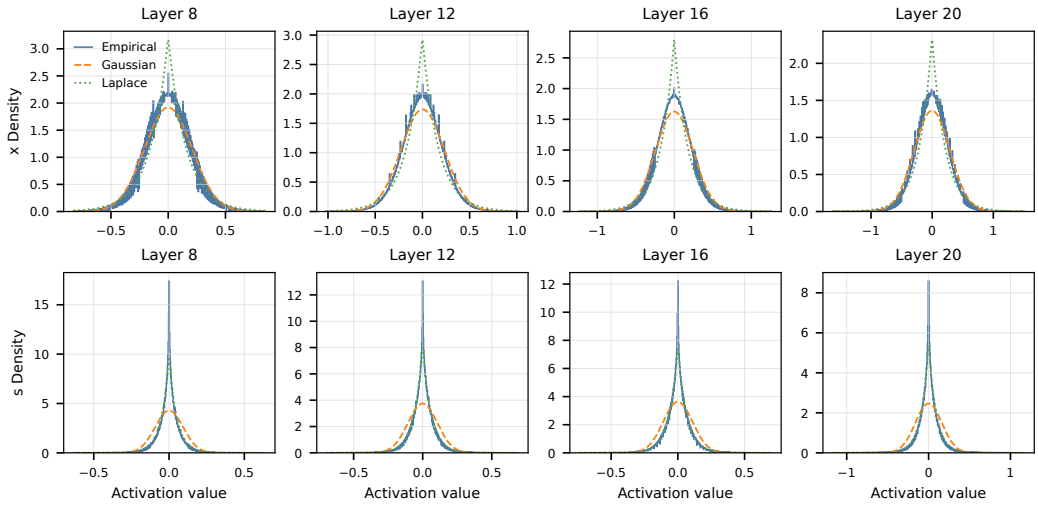


Figure 7: Activation-value distributions at representative SwiGLU FFN layers of Qwen3-8B, collected using the same calibration set as threshold calibration: the first 10 contiguous sequences of 2,048 tokens from the Alpaca training split. The top row shows the shared input $\boldsymbol{x}$ to the up and gate projections, together with fitted Gaussian and Laplace densities. The bottom row shows the intermediate state $\boldsymbol{s}$ supplied to the down projection. The FFN input is Gaussian-like, whereas the intermediate state is Laplacian-like and more sharply concentrated around zero.

Table 4: Qwen3 configurations used to calculate the FFN parameter and MAC shares in Figure 1.

Model	d_{model}	d_{ff}	H_q	H_{kv}	d_h
Qwen3-1.7B	2048	6144	16	8	128
Qwen3-4B	2560	9728	32	8	128
Qwen3-8B	4096	12288	32	8	128
Qwen3-14B	5120	17408	40	8	128
Qwen3-32B	5120	25600	64	8	128

and the corresponding relative root-mean-square projection error is

$$\eta(p) = \sqrt{1 - q_{\text{G}}(p)} = \sqrt{p - 2z_p\phi(z_p)}.$$

For example, at 70% input sparsity, $q_{\text{G}}(0.7) \approx 0.783$ and $\eta(0.7) \approx 0.465$. Thus, magnitude sparsification is substantially better than discarding inputs indiscriminately, but its projected values can still differ considerably from their dense counterparts. In methods such as TEAL with successive input-sparsity stages, approximation errors accumulate as they propagate through the up and gate branches, their element-wise product, and the down projection, making aggressive input sparsity increasingly fragile.

Ranking preservation under input sparsity. Large value error does not imply that the proxy has lost its ranking information. For one linear-projection output channel with weight vector $\boldsymbol{w}_j$, decompose $\boldsymbol{x} = \boldsymbol{x}_{\text{sp}} + \boldsymbol{r}$ into the retained input and discarded residual:

$$y_j = \langle \boldsymbol{x}, \boldsymbol{w}_j \rangle = \underbrace{\langle \boldsymbol{x}_{\text{sp}}, \boldsymbol{w}_j \rangle}_{\tilde{y}_j} + \underbrace{\langle \boldsymbol{r}, \boldsymbol{w}_j \rangle}_{e_j}.$$

Under the same independence and isotropy assumptions, $\tilde{y}_j$ and e_j are uncorrelated. Consequently,

$$\text{Corr}(y_j, \tilde{y}_j) = \sqrt{\frac{\text{Var}(\tilde{y}_j)}{\text{Var}(y_j)}} \approx \sqrt{q_{\text{G}}(p)}.$$

At 70% input sparsity, this idealized correlation is approximately 0.885, despite the relative projection error of 0.465. Applying the same sparse input to both SwiGLU branches therefore preserves

Table 5: Measured INT2-to-FP16 proxy-cost ratio α on NVIDIA A6000 and GeForce RTX 3090 GPUs.

Model	**A6000**	**RTX 3090**
Qwen3-4B	0.1720	0.1870
Qwen3-8B	0.1594	0.1660
Qwen3-14B	0.1645	0.1710
Qwen3-32B	0.1573	0.1730

Table 6: Stage-wise allocations used for each target sparsity. The configurations follow the proposed allocation rule with $s_{\text{ref}} = 0.7$, $s_1 \in [0, 0.7]$, and $\alpha = 0.2$.

Target e	**Stage 1 sparsity** s_1	**Stage 2 sparsity** s_2
40%	0%	53.33%
50%	0%	63.33%
60%	25%	70%
70%	70%	74%

a substantial shared signal from which Prox constructs $\tilde{\boldsymbol{s}}$. If $\tilde{s}_i = s_i + \delta_i$, the magnitude order of channels i and j is guaranteed to remain unchanged whenever

$$\big||s_i| - |s_j|\big| > |\delta_i| + |\delta_j|.$$

The large-magnitude tail channels of the Laplacian-like $\boldsymbol{s}$ can remain well separated from the selection boundary, while ranking disagreements concentrate among channels near that boundary. The Gaussian calculation does not model SiLU, the product of the two branches, trained-weight dependencies, or proxy-weight quantization, so it should be interpreted as a mechanism rather than a guarantee. The observed 82.04% exact–proxy selection overlap in Figure 3(d) verifies that this mechanism persists in the complete proxy. Prox exploits this weaker ranking requirement and computes the retained values exactly, whereas TEAL propagates the approximate values themselves.

B System Configuration and Efficiency

B.1 Empirical Proxy-Cost Ratios

To estimate α in Sec. 4.2, we measure the INT2-to-FP16 projection latency ratio on NVIDIA A6000 and GeForce RTX 3090 GPUs across four Qwen3 scales (Table 5).

The measured ratios range from 0.1573 to 0.1870, with a pooled mean of 0.1688. We use $\alpha = 0.2$ as a conservative rounded approximation in the allocation model.

B.2 Stage-Wise Compute Configurations

Table 6 lists the Stage 1 and Stage 2 sparsities used to realize each target effective sparsity e in our experiments. Applying the allocation rule with $s_{\text{ref}} = 0.7$ and $\alpha = 0.2$ yields $s_1 = 25\%$ at the 60% operating point while keeping s_2 at 70%. At the 40% and 50% operating points, the Stage 1 allocation implied by $s_2 = s_{\text{ref}}$ falls below its lower bound, so s_1 is clamped to 0% and s_2 is adjusted to 53.33% and 63.33%, respectively. At the 70% operating point, the implied s_1 exceeds its upper bound; we therefore clamp it to 70% and adjust s_2 to 74%.

B.3 End-to-End Latency Analysis

Figure 8 shows the latency breakdown of Prox on Qwen3-8B at 70% effective FFN sparsity. Even at this sparsity level, the FFN takes 4.490 ms/token, comparable to the 5.230 ms/token spent on attention, RMSNorm, and residual operations combined (panel (a)). This substantial FFN latency reinforces our motivation for accelerating FFN computation.

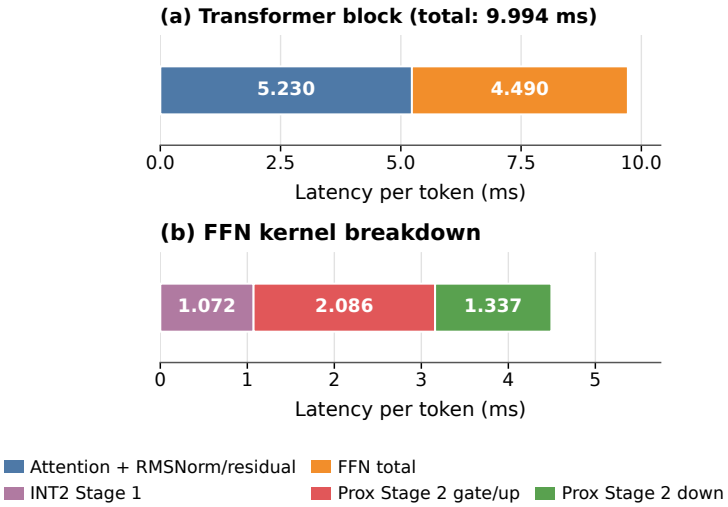


Figure 8: Detailed latency breakdown of Prox on Qwen3-8B with 70% effective FFN sparsity on an NVIDIA A6000 GPU. Values are normalized per generated token over 36 Transformer layers. (a) The Transformer-block profile totals 9.994 ms/token. Attention, RMSNorm, and residual are reported jointly because Inductor fuses them into one kernel. (b) Latencies of the three FFN kernels. All regions and kernels are profiled independently.

Figure 8(b) further decomposes the FFN: Stage 2 gate/up is the largest component at 2.086 ms/token (46.5% of FFN time), while INT2 Stage 1 and Stage 2 down take 1.072 and 1.337 ms/token, respectively.

We evaluate end-to-end decoding throughput on NVIDIA A6000 and GeForce RTX 3090 GPUs using five representative models, with unmeasured model–device configurations marked by dashes. Table 7 reports the dense baseline and the resulting throughput for Prox and TEAL at each target FFN sparsity.

Across both GPUs, Prox and TEAL deliver broadly comparable throughput over the evaluated models and sparsity levels, with neither method consistently dominating.

C Thresholding Mechanism and Calibration

C.1 Threshold-Based Selection and Token-Adaptive Computation

Prox converts two continuous magnitude signals into binary masks: the FFN input $\boldsymbol{x}$ determines the proxy-path input mask $\boldsymbol{m}_{\boldsymbol{x}}$, while the proxy intermediate state $\tilde{\boldsymbol{s}}$ determines the final intermediate-channel mask $\boldsymbol{m}_{\boldsymbol{s}}$. We use layer-wise magnitude thresholds for both signals instead of selecting a fixed number of coordinates independently for each token.

Per-token top-k selection. Given a salience signal $\boldsymbol{z}$ for one token, top-k selection retains the indices of its k largest-magnitude entries:

$$[\boldsymbol{m}^{\text{top-}k}(\boldsymbol{z})]_i = \mathbf{1}[i \in \text{TopKIndices}(|\boldsymbol{z}|, k)] \,.$$

This rule strictly controls the number of retained coordinates for every token, but uses only the magnitude ranking within that token. Consequently, two tokens receive the same compute budget even when their overall activation scales or importance distributions differ substantially. Top-k therefore enforces a per-token budget rather than a shared absolute importance criterion.

Layer-wise threshold selection. For FFN layer ℓ, a threshold rule instead retains every coordinate whose magnitude exceeds a fixed layer-specific value:

$$[\boldsymbol{m}^{\tau}(\boldsymbol{z})]_i = \mathbf{1}[|z_i| \geq \tau_\ell] \,.$$

Unlike top-k, this rule does not force all tokens to retain the same number of coordinates. Tokens with fewer entries above the threshold use less computation, whereas tokens with more salient entries

Table 7: Steady-state single-batch end-to-end decoding throughput (tokens/s) on two GPUs. Dense throughput is shared by Prox and TEAL within each model–device configuration; dashes indicate unmeasured configurations.

NVIDIA A6000						
Model	**Method**	**Dense**	**40%**	**50%**	**60%**	**70%**
Qwen3-4B	Prox	75.4	95.6	102.2	111.6	121.2
	TEAL		101.0	109.3	114.2	131.5
Qwen3-8B	Prox	43.0	60.6	66.0	72.0	80.4
	TEAL		60.2	65.8	73.8	84.0
Qwen3-14B	Prox	23.7	35.0	38.4	42.6	48.4
	TEAL		33.8	37.7	43.2	49.3
Llama3-8B	Prox	43.5	63.0	69.9	75.5	85.1
	TEAL		61.8	67.6	76.0	87.8
Mistral-7B	Prox	45.8	68.4	76.8	84.4	94.8
	TEAL		66.9	75.0	86.5	98.0
NVIDIA GeForce RTX 3090						
Model	**Method**	**Dense**	**40%**	**50%**	**60%**	**70%**
Qwen3-4B	Prox	95.5	121.8	130.8	140.4	150.1
	TEAL		119.1	125.8	137.1	149.1
Qwen3-8B	Prox	54.7	74.1	80.6	86.3	95.1
	TEAL		73.5	77.4	86.1	95.7
Qwen3-14B	Prox	–	–	–	–	–
	TEAL		–	–	–	–
Llama3-8B	Prox	55.3	77.0	84.1	90.6	101.5
	TEAL		72.7	78.1	87.2	100.0
Mistral-7B	Prox	58.6	83.1	91.6	100.6	112.1
	TEAL		78.4	85.8	98.9	116.7

retain more coordinates. Thresholding therefore controls the average compute budget while allowing that budget to vary with the token-specific activation distribution. It also avoids a runtime per-token ranking operation.

Measuring token-adaptive computation. For token t at layer ℓ, Prox applies this rule separately to the input and proxy intermediate signals:

$$[\boldsymbol{m}_{\boldsymbol{x},\ell,t}]_i = \mathbf{1}\left[|x_{\ell,t,i}| \geq \tau^{s_1}_{\boldsymbol{x},\ell}\right],$$

$$[\boldsymbol{m}_{\boldsymbol{s},\ell,t}]_j = \mathbf{1}\left[|\tilde{s}_{\ell,t,j}| \geq \tau^{s_2}_{\boldsymbol{s},\ell}\right].$$

The thresholds $\tau^{s_1}_{\boldsymbol{x},\ell}$ and $\tau^{s_2}_{\boldsymbol{s},\ell}$ are calibrated to meet the target Stage 1 and Stage 2 sparsities on average, rather than enforcing those sparsities independently for every token. We define the token-wise sparsities induced by the two masks as

$$r^{\boldsymbol{x}}_{\ell,t} = 1 - \frac{\|\boldsymbol{m}_{\boldsymbol{x},\ell,t}\|_0}{d_{\text{model}}}, \qquad r^{\tilde{\boldsymbol{s}}}_{\ell,t} = 1 - \frac{\|\boldsymbol{m}_{\boldsymbol{s},\ell,t}\|_0}{d_{\text{ff}}}.$$

We measure their variation at two scales. For the within-layer view, we compute the standard deviation and the 10th and 90th percentiles across tokens in each FFN layer, then average each statistic across layers. For the across-layer view, we first average each token's sparsity over the L FFN layers,

$$\bar{r}^{\boldsymbol{x}}_t = \frac{1}{L}\sum_{\ell=1}^{L} r^{\boldsymbol{x}}_{\ell,t}, \qquad \bar{r}^{\tilde{\boldsymbol{s}}}_t = \frac{1}{L}\sum_{\ell=1}^{L} r^{\tilde{\boldsymbol{s}}}_{\ell,t},$$

and then compute the corresponding statistics across tokens.

Table 8 shows substantial variation in the compute allocated to different tokens. Within individual layers, the P10–P90 ranges span 17.33 percentage points for the input mask and 17.51 points for the proxy-intermediate mask. Even after averaging over all FFN layers, the corresponding ranges remain 12.07 and 11.96 points. In contrast, a fixed per-token top-k rule has zero token-wise sparsity

Table 8: Token-wise sparsity variation under layer-wise threshold selection. Within-layer statistics are computed across tokens per layer and then averaged across layers; across-layer statistics are computed after averaging each token's sparsity over all FFN layers.

Aggregation	Signal	Std. (pts)	P10	P90
Within-layer	$\boldsymbol{x}$ mask	6.90	60.49%	77.82%
	$\tilde{\boldsymbol{s}}$ mask	6.80	63.83%	81.34%
Across-layer	$\boldsymbol{x}$ mask	4.78	63.16%	75.23%
	$\tilde{\boldsymbol{s}}$ mask	4.60	67.05%	79.01%

Table 9: Aggregate downstream scores using layer-wise threshold selection and per-token top-k at matched target effective sparsity. Higher is better; bold marks the better selection rule in each setting.

Model	Eff. sparsity	Threshold	Top-k
Qwen3-8B	40%	**75.30**	73.04
	50%	**75.10**	71.62
	60%	**73.50**	70.49
	70%	67.20	**67.57**
Qwen3-14B	40%	76.40	**77.00**
	50%	**76.30**	74.72
	60%	**75.00**	74.81
	70%	**74.60**	73.74

variation by construction. These results confirm that threshold selection gives Prox a natural token-adaptive computation mechanism instead of assigning every token the same retained-channel count.

Thresholding versus top-k. Having verified that thresholding enables token-adaptive computation, we directly compare the two selection rules on Qwen3-8B and Qwen3-14B at matched target effective sparsities. Table 9 reports the aggregate downstream scores.

Thresholding achieves a higher score in six of the eight settings and improves the average score by 1.30 points over top-k. On Qwen3-8B, it improves scores by 2.26–3.48 points at 40–60% sparsity, but trails top-k by 0.37 points at 70% sparsity. On Qwen3-14B, it improves scores by 0.19–1.58 points at 50–70% sparsity, but trails top-k by 0.60 points at 40% sparsity. Thus, preserving an absolute layer-wise importance criterion improves downstream quality on average across the evaluated settings at a matched target effective sparsity. Having established the motivation and empirical benefits of thresholding, Appendix C.2 describes how the layer-wise thresholds are calibrated and evaluates their runtime sparsity stability.

C.2 Threshold Calibration and Runtime Sparsity

Calibration data and scale. Prox calibrates its fixed layer-specific thresholds using generic instruction text from the Alpaca training split. For each model, we tokenize the calibration corpus with its native tokenizer, partition the resulting tokens into contiguous sequences of length 2,048, and use the first 10 sequences for each target-sparsity configuration. Thus, each configuration is calibrated on 20,480 token positions. We independently calibrate the ten models in our main evaluation—Qwen3-1.7B, Qwen3-4B, Qwen3-8B, Qwen3-14B, Qwen3.5-4B, Qwen3.5-9B, Ministral-3.3B, Mistral-7B, Llama-3-8B, and Gemma-3-12B—at target sparsities of 40%, 50%, 60%, and 70%, producing 40 threshold configurations in total.

For each layer and each calibration signal, we maintain a reservoir of at most 200,000 activation samples and obtain the threshold directly from the corresponding empirical quantile. We additionally save a 4,096-bin activation histogram for each layer to support analysis and reproducibility. Threshold values are determined separately for each layer, and no additional layer-wise sparsity profile is imposed: within a configuration, all layers share the same target s_1 and s_2, while their

Algorithm 1: Sequential threshold calibration

Require: Calibration set $\mathcal{D}$ and a model with L FFN layers
1: **for** $\ell = 1, \ldots, L$ **do**
2: Run $\mathcal{D}$ through layers $1, \ldots, \ell - 1$ using their finalized sparse rules
3: Collect the current-layer inputs $\boldsymbol{x}_\ell$
4: Choose $\tau^{s_1}_{\boldsymbol{x},\ell}$ and construct $\boldsymbol{m}_{\boldsymbol{x},\ell}$
5: Compute $\tilde{\boldsymbol{s}}_\ell$ using $\boldsymbol{m}_{\boldsymbol{x},\ell}$ and the quantized proxy weights
6: Choose $\tau^{s_2}_{\boldsymbol{s},\ell}$ and finalize the sparse rule for layer ℓ
7: **end for**

Table 10: Target and runtime FFN sparsities for Prox. Runtime values aggregate five task datasets and are averaged over ten models, with 100 examples evaluated per dataset.

Target sparsity e	**Mean measured sparsity** $\widehat{e}$
40%	38.71%
50%	48.74%
60%	58.44%
70%	68.33%

numerical thresholds are determined by their own activation distributions. All evaluations use the saved calibration artifacts; thresholds are neither re-estimated nor updated during inference.

Sequential layer-wise calibration. A direct approach would calibrate every layer independently using activations collected from the dense model, but this creates a mismatch with sparse inference: sparsification in earlier layers changes the input distribution observed by later layers. Prox therefore calibrates layers in network-depth order so that each layer observes the effects of all previously finalized sparse layers. Algorithm 1 summarizes the procedure.

Specifically, when calibrating layer ℓ, Prox first collects its FFN input $\boldsymbol{x}_\ell$ and determines $\tau^{s_1}_{\boldsymbol{x},\ell}$. It then applies the resulting input mask and the model-specific asymmetric-INT2 proxy weights to obtain

$$\tilde{\boldsymbol{s}}_\ell = \left((\boldsymbol{m}_{\boldsymbol{x},\ell} \odot \boldsymbol{x}_\ell)\widetilde{\boldsymbol{W}}_{\mathrm{up},\ell}\right) \odot \mathrm{SiLU}\Big((\boldsymbol{m}_{\boldsymbol{x},\ell} \odot \boldsymbol{x}_\ell)\widetilde{\boldsymbol{W}}_{\mathrm{gate},\ell}\Big),$$

from which it determines $\tau^{s_2}_{\boldsymbol{s},\ell}$. All preceding FFNs use their finalized Stage 1 and Stage 2 thresholds, whereas the current layer remains dense while its input activations are collected. Consequently, each layer's calibration distribution incorporates the error propagation and distribution shift induced by earlier sparse layers and more closely matches runtime sparse inference.

Runtime sparsity. Prox uses fixed magnitude thresholds rather than per-token top-k selection; consequently, the realized mask sparsities can differ from their calibration targets under a new input distribution. Let $\widehat{s}_1$ and $\widehat{s}_2$ denote the Stage 1 and Stage 2 sparsities measured by accumulating runtime mask counts. Consistent with the cost model in Sec. 4.2, the measured runtime FFN sparsity is

$$\widehat{e} = \widehat{s}_2 - \frac{2\alpha(1 - \widehat{s}_1)}{3}, \qquad \alpha = 0.2.$$

This quantity includes both the sparse exact FFN computation and the additional INT2 proxy cost.

We measure $\widehat{e}$ using 100 examples from each of five task datasets: MMLU, HellaSwag, ARC-Easy, ARC-Challenge, and PIQA. For every model and target configuration, we aggregate the Stage 1 and Stage 2 mask counts across all five datasets before applying the cost model. Table 10 reports the resulting mean over the ten evaluated models.

Across all four operating points, the mean measured sparsity is slightly below the target, with gaps of 1.26–1.67 percentage points and no sharp increase at higher target sparsity. These results indicate that static thresholds calibrated on generic instruction text preserve the intended compute level reasonably well across the evaluated models and task datasets.

Table 11: WikiText perplexity across model families and sparsity levels. Lower scores are better.

		Qwen3				Qwen3.5		Ministral	Mistral	Llama3	Gemma3
Variant	**Sparsity**	**1.7B**	**4B**	**8B**	**14B**	**4B**	**9B**	**3.3B**	**7B**	**8B**	**12B**
Dense	0	14.69	11.32	8.19	7.19	8.04	7.42	8.52	4.88	7.10	7.44
CATS	40%	29.78	15.14	9.14	7.83	10.36	8.75	14.78	6.83	8.61	34.00
COUNTDOWN	40%	22.03	14.61	9.75	8.22	9.46	8.34	11.41	5.29	8.86	31.14
TEAL	40%	**14.26**	11.74	**8.33**	7.31	**8.27**	**7.48**	**8.87**	5.09	**7.35**	806.98
Prox	40%	14.55	**11.72**	8.44	**7.29**	8.35	7.67	9.07	**5.05**	7.64	**24.60**
CATS	50%	98.26	61.64	15.12	11.59	15.83	12.23	95.28	35.51	17.44	221.83
COUNTDOWN	50%	47.67	20.83	12.71	10.43	12.54	10.64	16.93	6.39	12.58	254.65
TEAL	50%	**14.42**	12.36	8.65	7.49	8.62	**7.66**	9.43	5.23	**7.78**	898.75
Prox	50%	14.66	**11.89**	**8.59**	**7.34**	**8.50**	7.75	**9.36**	**5.15**	7.92	**96.54**
CATS	60%	1.72e3	3.57e4	1.30e3	521.46	121.73	56.69	9.73e4	7.50e4	2.28e5	1.58e4
COUNTDOWN	60%	1.62e3	56.49	34.06	21.85	36.60	24.39	64.58	13.51	40.16	**712.18**
TEAL	60%	17.57	14.34	9.56	8.00	9.62	8.16	10.98	5.92	8.86	1.20e3
Prox	60%	**15.06**	**12.45**	**8.84**	**7.49**	**8.78**	**7.95**	**10.01**	**5.33**	**8.38**	1.10e3
TEAL	70%	48.84	24.50	16.14	9.95	13.20	10.30	16.82	8.60	13.46	2.94e3
Prox	70%	**16.09**	**14.32**	**9.61**	**7.88**	**9.56**	**8.48**	**12.02**	**5.91**	**9.63**	**1.28e3**

D Additional Evaluation and Sensitivity Analyses

D.1 Perplexity Results

For this evaluation, we use the same 128 randomly sampled sequences from the WikiText validation set (Merity et al., 2016) for all models, with a 2,048-token context and a 512-token evaluation window. We sparsify the full sequence so that the hidden states scored for perplexity are computed under sparsity. Because perplexity is computed over each model's native token space, Gemma-3-12B's substantially larger vocabulary and different tokenization granularity make its absolute perplexity values not directly comparable with those of the other model families.

Table 11 reports WikiText perplexity across model families and sparsity levels. At 40% sparsity, Prox remains competitive with TEAL. From 50% sparsity onward, Prox matches or outperforms TEAL on most model families; at 60% and 70%, it achieves lower perplexity on at least nine of the ten evaluated models, with the gap widening markedly at 70%. CATS and COUNTDOWN degrade earlier, frequently producing large perplexity increases at 50% and 60% sparsity.

Despite its large perplexity increase under sparsification, Gemma-3-12B retains strong downstream accuracy. Consistent with our observation, Szatkowski et al. (2026) report that Gemma-3-12B retains approximately 99% of its dense average downstream accuracy at 78.77% activation sparsity.

D.2 Sensitivity to the Fixed Stage 1 Sparsity

Table 12 compares Prox with fixed Stage 1 sparsities from 0.5 to 0.9 on Qwen3-8B to test the sensitivity of A2's $s_1 = 0.7$ setting. For each fixed s_1 and target effective sparsity e, we adjust s_2 using the cost model in Sec. 4.2 to match the same compute budget and report the aggregate downstream score under the main six-task protocol.

The best fixed allocation changes with the target, indicating that no single fixed s_1 is uniformly optimal across operating points. In contrast, Prox uses target-aware allocation and maintains consistently strong performance without per-target downstream sweeps. At 70%, overly aggressive Stage 1 sparsity reduces the score from 67.2 at $s_1 = 0.7$ to 56.0 at $s_1 = 0.9$.

Table 12: Aggregate downstream scores on Qwen3-8B for Prox and fixed Stage 1 sparsities. Higher is better; bold marks the best fixed allocation for each target effective sparsity.

	Target effective sparsity e			
Setting	**40%**	**50%**	**60%**	**70%**
Prox	75.3	75.1	73.5	67.2
$s_1 = 0.5$	**74.8**	72.2	71.5	**68.0**
$s_1 = 0.6$	74.8	**73.7**	71.5	66.9
$s_1 = 0.7$	74.2	72.8	71.8	67.2
$s_1 = 0.8$	71.2	70.6	**72.0**	63.5
$s_1 = 0.9$	70.6	68.4	63.3	56.0

Table 13: Sensitivity to the maximum Stage 1 sparsity $s_{1,\max}$ across Qwen3 models. Entries are aggregate downstream scores (Higher is better); each setting adjusts Stage 2 sparsity s_2 to match the listed effective sparsity.

			Qwen3		
Stage 1 cap $s_{1,\max}$	**Effective sparsity**	**Stage 2 sparsity** s_2	**4B**	**8B**	**14B**
50%	70%	76.7%	59.3	**68.0**	73.1
60%	70%	75.3%	56.8	66.9	73.4
70%	70%	74.0%	**59.6**	67.2	**74.6**
80%	70%	72.7%	54.1	63.5	71.9
90%	70%	71.3%	46.0	56.0	69.9

D.3 Sensitivity to the Stage 1 Sparsity Cap

Table 13 evaluates the sensitivity to $s_{1,\max}$ across Qwen3-4B, Qwen3-8B, and Qwen3-14B at 70% effective sparsity. For each cap, we adjust s_2 according to the cost model in Sec. 4.2 to preserve the same effective-sparsity budget.

Overall, $s_{1,\max} = 70\%$ provides the strongest cross-model performance, while larger caps consistently degrade performance. We therefore use it as a shared cap without per-model downstream tuning.

D.4 Detailed Analysis with TEAL

TEAL's block-wise greedy calibration assigns sparsity separately to the up, gate, and down projections while preserving the target effective sparsity. Starting from dense computation, it repeatedly increases the sparsity of the projection whose candidate update incurs the smallest block-output ℓ_2 error on calibration activations, until the target block-level budget is reached. Figure 9 shows the resulting layer-wise allocation for Qwen3-8B at 60% effective sparsity.

The allocation is both projection- and depth-dependent. In the first four layers, the down-projection sparsity reaches 75–90%, compared with 50–52.5% for the up projection and 40–52.5% for the gate projection. Beyond these early layers, the three rates become closer, but the down projection generally remains the most sparse and the up projection the least sparse. Thus, TEAL's calibrated 60% operating point does not correspond to applying a uniform sparsity rate to every projection and layer; it instead reveals that the down projection tolerates more approximation, whereas the up projection is generally more sensitive.

This fine-grained search makes TEAL a stronger reference than a uniformly allocated baseline. In contrast, Prox uses a simple target-aware heuristic motivated by the oracle-$\boldsymbol{s}$ results: it treats 70% Stage 2 channel sparsity as a quality-preserving anchor and assigns the remaining budget to Stage 1, using the same allocation across layers. Despite TEAL's greater allocation flexibility, Prox scores 73.5 versus TEAL's 72.2 on Qwen3-8B at the matched 60% operating point and outperforms TEAL

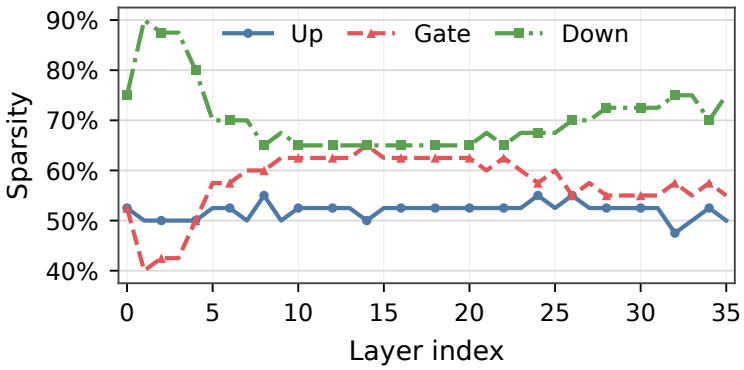


Figure 9: Layer-wise sparsity assigned by TEAL's greedy calibration on Qwen3-8B at 60% effective sparsity. At every layer, the up-, gate-, and down-projection sparsities average to 60%.

on all ten evaluated models at both 60% and 70% sparsity in Table 1. Together with the exact-computation ablation in Table 3, this result supports the central two-stage design: the proxy is used only to select channels, while the retained values are computed with the original weights, preventing proxy-value errors from propagating directly through the final FFN computation.

The branch-wise allocation provides a further insight into proxy construction. The lower sparsity generally assigned to the up projection suggests that preserving its computation is more important under TEAL's block-error objective. This trend is consistent with the comparison between CATS and COUNTDOWN: CATS densely computes the gate-branch signal $\boldsymbol{h}$ for channel selection, whereas COUNTDOWN uses the densely computed up-branch signal $\boldsymbol{u}$; across the 30 matched model–sparsity settings at 40–60% in Table 1, COUNTDOWN always achieves a higher aggregate score than CATS. Although projection sensitivity and selection-signal quality are not identical quantities, these results jointly suggest that $\boldsymbol{u}$ provides a more informative approximation to the joint intermediate signal $\boldsymbol{s}$ than $\boldsymbol{h}$ alone.

Prox currently uses the same sparsity allocation across all layers. A promising extension is to follow TEAL's layer-wise allocation and use different Stage 1 sparsities when constructing the $\boldsymbol{u}$- and $\boldsymbol{h}$-branch proxies, which may improve the accuracy of proxy-based channel selection.

D.5 Compatibility with Fine-Tuning

Although Prox is training-free, optional fine-tuning can recover quality lost to sparsification. We fine-tune the pretrained Qwen3-8B checkpoint (without instruction tuning) using low-rank adaptation (LoRA) (Hu et al., 2021) with rank 32 and a learning rate of 2×10^{-4} on 15M tokens from C4, and evaluate perplexity on WikiText. Table 14 compares Prox before and after fine-tuning at target FFN sparsities of 40–90%.

Table 14: WikiText perplexity with and without LoRA fine-tuning on Qwen3-8B (without instruction tuning). Sparsity levels denote target FFN sparsity; lower perplexity is better.

Target sparsity	PPL (No fine-tuning)	PPL (With fine-tuning)
40%	6.087	6.048
50%	6.214	6.138
60%	6.334	6.234
70%	6.892	6.703
80%	8.403	7.435
90%	146.755	10.128

Fine-tuning reduces perplexity at every target, with modest improvements at 40–70% sparsity and substantially larger gains at 80–90%. At 90%, perplexity falls from 146.755 to 10.128, showing that a small fine-tuning corpus can recover much of the quality lost under aggressive sparsification.