SVDP: Training-Free Contextual Sparsity Predictors for Fast LLM Inference

Georgii Serbin[♠] **Kirill Koshkin**[♢] **Zhongao Sun**[♠]
Anastasiya Bistrigova[♢] **C.C. Korikov**[♠]

[♠]Huawei Technologies Ltd. [♢] Moscow State University

constantine.korikov@gmail.com

Code: https://github.com/ckorikov/noah-research/tree/SVDP/SVDP

Abstract

Contextual sparsity is one of the approaches used to reduce computational complexity in the inference process of large language models (LLMs). Existing techniques for efficient LLM inference acceleration based on contextual sparsity with minimal accuracy degradation require training sparse pattern predictors. This paper presents a framework for accelerating the inference of ReGLU-based feed-forward networks (FFNs) within LLMs. The proposed framework provides a fast, training-free method for building sparse pattern predictors using truncation-aware singular value decomposition (SVD) of the gate projection matrix, along with a threshold calibration algorithm, and inference executors supporting conditional computation on CUDA and CANN devices. Experiments on three sparse LLMs with an average activation sparsity level of 90% in the FFNs demonstrate up to a 1.8× end-to-end decoding time speedup with less than 1% degradation in benchmark scores on tasks involving math and code generation. This work advances the deployment of LLMs on edge devices.

1 Introduction

The remarkable capabilities of LLMs have led to their widespread integration across diverse domains [5, 11]. While on-device AI deployment offers advantages in privacy, latency, and cost compared to server-based solutions [28], the substantial computational requirements of LLM inference present significant challenges for edge deployment.

A key characteristic of ReGLU-based FFNs within LLMs is dynamic (or contextual) sparsity – the phenomenon where only a small subset of neurons are activated for any given input [19]. This inherent property enables inference acceleration by reducing both computational overhead and memory bandwidth requirements during generation [22].

A prevalent approach for leveraging contextual sparsity involves employing lightweight, learnable sparsity predictors [22]. These predictors generate input-specific sparse activation patterns prior to FFN forward passes. The sparse pattern identifies neuron indices that will produce positive dot products with the input vector. In ReGLU-based FFNs, computations for the remaining neurons can be safely skipped without affecting output correctness, as the activation function nullifies negative values. Nevertheless, prediction errors may arise from false negative classifications, which compromise model accuracy. Effective predictors must balance high recall rates with substantial sparsity ratios while maintaining sufficient speed to reduce end-to-end (E2E) text generation latency.

In this study, we present a simple yet effective technique leveraging low-rank singular value decomposition (SVD) to build sparse pattern predictors, thereby enabling significant inference acceleration in ReGLU-based LLMs. Our methodology employs low-rank approximations of gate projection weight

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

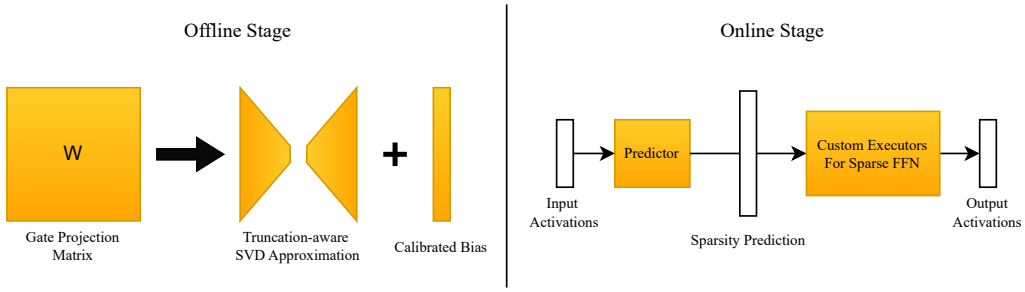


Figure 1: Overview of our framework. During the offline stage, sparsity predictors are constructed using truncation-aware SVD approximation with calibrated per-neuron thresholds. During the online stage, SVD-based predictors precede each FFN to exploit contextual sparsity and accelerate generation.

matrices in each layer as lightweight predictors of FFN sparsity during generation. Our method provides theoretical guarantees on prediction error.

To enhance the accuracy of SVD-based predictors, we employ a combination of techniques:

- **Calibrated bias**. Direct application of SVD approximations as predictors results in notable accuracy degradation. However, we observe that these predictors effectively separate active and inactive subsets of neurons. Introducing neuron-specific thresholds – implemented as bias terms – effectively mitigates this issue. These biases are calibrated offline using a custom greedy algorithm that does not require gradient computation.
- **Data whitening technique**. Given the predictor speed constraints, the rank must be significantly smaller than the hidden dimension. To improve prediction accuracy under these conditions, we apply truncation-aware data whitening [36, 37, 40]. This technique manages activation outliers by incorporating input activation statistics, requiring only a small calibration dataset.

Implementation-level optimizations further enhance inference speedup:

- **Sequential computation of gate and up projections**. Gate and up projections are computed sequentially, enabling further acceleration through early elimination of false positive predictions. After computing gate projections, falsely activated neurons are identified and excluded from subsequent up and down projections, thereby increasing sparsity levels in the final FFN stages.
- **Inference executors**. Our implementations support conditional computation on CUDA [26] and CANN [13] devices. For CUDA, we employ custom kernels rather than Triton-based abstractions [35], prioritizing enhanced low-level control. On Huawei Ascend hardware [20], comparable performance gains are achieved exclusively through high-level PyTorch APIs, demonstrating effective utilization of the platform's native software stack.

Figure 1 illustrates our framework architecture. Our key contributions are summarized as follows:

1. We propose training-free SVD-based contextual sparsity predictors for ReGLU-based LLM inference acceleration;
2. We introduce calibrated bias terms that significantly improve prediction accuracy, with biases tuned on small datasets using a custom gradient-free algorithm;
3. We develop executors supporting sparse FFN inference on CUDA and CANN devices;
4. We achieve up to 1.8× end-to-end inference speedup with minimal accuracy degradation.

2 Related work

Sparsification is one of the most effective paradigms for accelerating LLM inference, typically categorized into **static** and **dynamic sparsity** methods. **Static sparsity** removes redundant parameters

before deployment [3], reducing computation and memory costs but often sacrificing accuracy at high sparsity levels. In contrast, **dynamic sparsity** – also known as contextual sparsity – adapts sparsity patterns at runtime based on input context, enabling flexible efficiency-accuracy trade-offs.

Predictor-based dynamic sparsity methods, such as Deja Vu [22] and ShadowLLM [1], learn lightweight neural predictors to estimate layer-wise sparsity. Shrestha et al. [30] focuses on batched inference, delivering up to 2.2× E2E speedups by exploiting head sparsity in the Attention. In contrast, our work focuses on single-query execution, which is common for edge devices and where E2E latency is dominated by FFN. **Training-free dynamic sparsity** eliminates learned predictors and instead derives sparsity from activation statistics or calibrated thresholds. Dong et al. [10] selects active "expert" neurons using prefill activations, while Ma et al. [23] introduces offline-calibrated thresholds allowing adaptive activation per layer. Both maintain fixed sparsity patterns within a prompt. Yang et al. [39] updates sparsity during generation but requires recomputation of previous states, hindering KV-cache reuse. Liu et al. [21] and Chen et al. [7] exploit input sparsity, which differs from activation sparsity in that it arises from redundancy in the input representations rather than neuron activations, using calibrated thresholds. Shin et al. [29] proposes a calibrated solution for ReLU-fied LLMs using sign-bit comparisons, which lacks theoretical justifications.

Dynamic sparsity can also underpin efficient on-device inference frameworks. Song et al. [32] accelerates LLMs on consumer GPUs by preloading frequently activated neurons into GPU memory, minimizing CPU transfers. PowerInfer achieves up to 11.69× speedup, particularly in CPU/GPU hybrid settings, whereas our work targets homogeneous GPU inference.

The effectiveness of dynamic sparsity depends heavily on activation functions. ReLU activations naturally induce sparse activations, unlike smoother functions such as Swish or SiLU used in modern LLMs [41]. Recent studies show that replacing them with ReLU variants plus fine-tuning recovers substantial sparsity [25]. Song et al. [31] achieved ∼90% sparsity in LLaMA2-7B through progressive regularization and activation threshold shifting, while Song et al. [33] used double ReLU (dReLU) activation function to sparsify models like Qwen2-7B and Mistral-7B.

Beyond pruning, **SVD** offers complementary compression capabilities. Yuan et al. [40] leverages activation-aware transformations to attain 10–30% model compression, and Wang et al. [37] provides theoretical connections between discarded components and the compression loss.

Khaki et al. [16] integrates SVD-based estimators into parameter-efficient fine-tuning, dynamically slicing weights by input relevance. However, sparse computations are applied only to *context tokens* (the prefix tokens provided as input), while *output tokens* (the target tokens used for loss computation) remain densely processed to avoid performance degradation. Achieving accelerated and high-quality decoding of LLMs through contextual sparsity remains an open challenge not addressed by SparseLoRA.

3 Preliminaries

Consider a ReGLU-activated FFN during decoding, defined as:

$$\text{FFN}(x) = W^{\text{down}} \left(\text{ReLU}(W^{\text{gate}} x) \odot (W^{\text{up}} x) \right), \tag{1}$$

where $x \in \mathbb{R}^d$ is the input activation vector, and $W^{\text{gate}}, W^{\text{up}} \in \mathbb{R}^{D \times d}$, and $W^{\text{down}} \in \mathbb{R}^{d \times D}$ are the gate, up-projection, and down-projection matrices, respectively. Here, d denotes the hidden size and D the intermediate size. Let $X = [x_1, \ldots, x_N] \in \mathbb{R}^{d \times N}$ denote the hidden states of N calibration tokens, and let r denote the predictor rank. Finally, let $n(x) \in \mathbb{R}^D$ denote the neuron values

$$n(x) = \text{ReLU}(W^{\text{gate}} x) \odot (W^{\text{up}} x). \tag{2}$$

For brevity, we omit layer indices.

Our objective is to obtain a predictor P that outputs a binary mask $m = P(x) \in \{0, 1\}^D$ over the D intermediate neurons. This mask identifies active neurons by selecting the corresponding rows of W^{gate} and W^{up}, together with their aligned columns of W^{down}, yielding sparse computation algebraically equivalent to the dense FFN due to ReLU sparsity.

4 SVD-based sparsity predictors

This section introduces *training-free SVD-based sparsity predictors* (SVDP), designed for efficient inference in ReGLU-based LLMs. Our approach consists of two stages: an offline phase for predictor construction and an online phase for accelerated sparse inference.

4.1 Training-free predictor building

The predictors are constructed by approximating the gate projection matrix W^{gate} using a low-rank SVD. This approach offers a theoretical upper bound on local error (see Section A.1).

We propose predictors of the affine form

$$P(x) = H(ABx + b), \tag{3}$$

where $H : \mathbb{R} \to \{0, 1\}$ is the Heaviside step function applied element-wise. $H(z) = 1$ if $z > 0$ and 0 otherwise. The product AB approximates W^{gate}, while $b \in \mathbb{R}^D$ adjusts per-neuron decision thresholds.

4.1.1 Data-aware SVD

To improve the accuracy of the SVD approximation, we adopt a data whitening technique (Algorithm 4). This minimizes the Frobenius norm error $\|WX - ABX\|_F$ on calibration data X, by incorporating input statistics through a whitening matrix S:

$$W = (WS)S^{-1} = (U\Sigma V^\top)S^{-1} \approx U_r\Sigma_r V_r^\top S^{-1} = AB. \tag{4}$$

We follow Wang et al. [37] in choosing S as the Cholesky decomposition of $XX^\top$, which outperforms ASVD in terms of perplexity at low ranks, as shown in Table 2 of Yuan et al. [40]. The predictor construction pipeline is performed independently for each layer. Throughout, we fix the predictor rank r across layers (see Section I for a discussion on rank selection).

4.1.2 Bias calibration

Empirically we observe that the SVD-based sparsity predictors exhibit good separation between active and inactive neurons; however, naive thresholding (e.g., at zero) leads to substantial accuracy degradation. This effect arises from a shift in the predictor output distribution induced by truncation of singular value components, as illustrated in Figure 2.

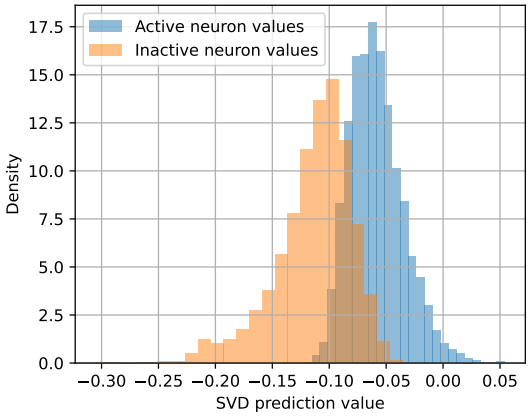


Figure 2: Distribution of SVD-based predictor outputs ($r = 256$) for a sample neuron obtained from approximately 25,000 tokens. Although active and inactive states are well separated, zero thresholding causes significant false negatives for active states.

To compensate for this distribution shift, we introduce a bias term in the second layer of the SVD-based predictor. The central problem is to calibrate this bias so as to achieve a prescribed sparsity

level while limiting the associated accuracy loss. Simple schemes that tune thresholds to match a target sparsity or recall on a calibration set were empirically found to yield suboptimal trade-offs.

Instead, we employ a fast greedy calibration algorithm, which explicitly accounts for the non-uniform impact of different neurons on model accuracy. More precisely, from a computational standpoint, all neurons are equally valuable for acceleration, since pruning any neuron reduces the cost by the same amount, whereas their contributions to output accuracy differ substantially. Consequently, some neurons require high recall, whereas others can tolerate higher false-negative rates without noticeably affecting overall performance.

Formally, we minimize a cost function $C : \mathbb{R}^D \to \mathbb{R}$

$$C(b) = \sum_{i=1}^{D} \alpha_i \left[\sum_{t=1}^{N} \left(1 - P\left(x_t\right)\right) \cdot n^2\left(x_t\right) \right]_i \tag{5}$$

over bias b on calibration data X with a sparsity constraint

$$\frac{1}{DN} \sum_{i=1}^{D} \sum_{t=1}^{N} \left[1 - P\left(x_t\right)\right]_i \geq s, \tag{6}$$

where

- $s \in (0, 1]$ — the desired sparsity ratio hyperparameter that controls the speed-accuracy trade-off;
- $\alpha_i = \left\| W^{\text{down}}_{:,i} \right\|_2^2$ — the weight coefficient that determines the importance of i-th neuron.

The cost function aggregates values of neurons falsely predicted to be inactive over the calibration data, reflecting FFN local error.

We solve the problem with a greedy algorithm, which can be briefly described as follows:

1. Initialize the thresholds ($\tau = -b$) in such a way that the prediction ensures high recall (likely due to low predicted sparsity ratio).
2. Iteratively increase the predicted sparsity ratio via greedy threshold updates.
   At each iteration, the threshold is increased for the neuron, for which the growth of the cost function is minimal.
3. Stop when the predicted sparsity on the calibration data reaches s.

The details are provided in Algorithm 1. Under certain conditions, our greedy algorithm is guaranteed to provide an optimal solution (see Section A.2).

To properly handle significantly different sparsity ratios across layers, we set s layer-wise

$$s^{(l)} = s \cdot \hat{s}^{(l)}, \quad l = 0, 1, \ldots, \tag{7}$$

where l is the layer index and $\hat{s}^{(l)}$ is the estimated sparsity on calibration data X.

Trainable predictors require collecting hidden activations of hundreds of thousands of token samples [30, 31] for supervised learning. In contrast, the proposed predictor construction is fast and data-efficient. Empirically we observe that only a few thousand calibration tokens are sufficient for 7B models (see Section G).

Overall, to construct SVDP for each layer we collect the calibration hidden states, obtain A and B from the data-whitened SVD of Algorithm 4, and calibrate the bias with Algorithm 1.

4.2 Runtime execution

Only the FFNs are modified, while the rest of the model remains untouched, enabling seamless integration with existing attention optimizations. During inference, the prefill stage remains unchanged. At the decoding stage, we leverage CUDA and CANN executors for sparse FFN computation. Our predictor comprises two linear layers, requiring no specialized implementation.

Sparsity prediction pipeline To partially hide the latency of the predictor, methods, such as Deja Vu [22] and Polar Sparsity [30], overlap predictor execution with the Attention layer via asynchronous computation. However, such pipelining may be infeasible on certain hardware platforms (see Section B). Given our focus on local inference for edge deployment scenarios, similar to PowerInfer engine, we do not employ this technique. Instead, our predictors are inserted at the beginning of each FFN and sequentially generate sparsity patterns for their respective gate projections.

Gate/up execution order A prevalent optimization for efficient deployment of LLMs with gated FFNs involves parallelized execution of the gate and up projections. While effective in dense computation scenarios, this approach may not be optimal for local inference systems leveraging sparse activations. We delineate two distinct computational methodologies to address this:

- *Parallel pipeline* e.g., PowerInfer. A predictor first generates a sparsity mask, which is concurrently applied to independently compute both gate and up projections. This maintains identical sparsity levels across both projections during execution.
- *Sequential pipeline* e.g., SparseInfer and our approach. The gate and up projections are computed sequentially, eliminating false positives: neurons predicted as active are revalidated after gate projection computation, further increasing sparsity in subsequent projections.

Algorithmic complexity In our case, corrected sparsity equals or exceeds the predicted sparsity s by filtering erroneous predictions. Thanks to the sequential pipeline, we can use more lightweight predictors that provide relatively low sparsity at the gate projection stage without compromising the FFN's performance. A dense FFN requires $3dD$ multiplications (excluding element-wise product), whereas our sparse approach reduces this to:

$$\underbrace{r(d+D)}_{\text{predictor}} + \underbrace{d(1-s)D}_{\text{gate}} + \underbrace{2d(1-s')D}_{\text{up/down}}, \tag{8}$$

where s is the predicted sparsity and s' the corrected sparsity. For example, in ProSparse-LLaMA2-7B [31] ($d = 4096$, $D = 11008$, $s = 0.5$, $s' = 0.9$, $r = 256$), operations decrease by $3.8\times$. Note that hardware-level dynamic sparsity challenges prevent this from directly translating to wall-clock speedup. See Section J for sparsity broken down by source and the decode latency breakdown.

Hardware platforms CUDA and CANN devices are supported for sparse FFN inference. See implementation details in Section B.

5 Experimental results

5.1 Experimental setup

Hardware A single consumer-grade NVIDIA RTX 3090 GPU with memory size of 24 GB is used for both quality and performance evaluation. The inference is performed entirely on GPU. For evaluation on NPU, a single Huawei Ascend 310P3 [14] device is used. The process of constructing predictors takes approximately 10 minutes for 7B models.

Software We use the vLLM framework [17], preferring it to the Transformers framework [38] for the experiments, as it better reflects real-world deployment scenarios. Notably, we do not utilize any server-specific features of vLLM, such as tensor parallelism. Inference is performed in half precision, whereas predictor construction in the offline stage uses double precision. Calibration data and the whitening/bias split are described in Section I.

Models We evaluate our solution across three open-source sparse 7B language models: **ProSparse-LLaMA2-7B** [31] and two instruction-tuned models – **SparseQwen2-7B** and **TurboSparse-Mistral-Instruct** [33]. The instruction-tuned models incorporate double ReLU (dReLU) activation in their FFN layers:

$$\text{FFN}(x) = W^{\text{down}}\left(\text{ReLU}\left(W^{\text{gate}}x\right) \odot \text{ReLU}(W^{\text{up}}x)\right). \tag{9}$$

Table 1: Accuracy when varying hyperparameter s for different models. s implicitly sets sparsity, which scales with end-to-end speedup. The accuracy is averaged across all benchmarks. See Section C for complete evaluation results.

		ProSparse-LLaMA2-7B		TurboSparse-Mistral-Instruct		SparseQwen2-7B	
Method	s	Avg. Score, %	E2E Speedup	Avg. Score, %	E2E Speedup	Avg. Score, %	E2E Speedup
Dense	–	38.92	1.00×	54.30	1.00×	65.84	1.00×
SVDP	0.4	38.66 (-0.26)	1.61×	53.72 (-0.58)	1.67×	65.41 (-0.43)	1.78×
	0.5	38.78 (-0.14)	1.64×	53.56 (-0.74)	1.73×	65.20 (-0.64)	1.81×
	0.6	38.77 (-0.15)	1.67×	53.18 (-1.12)	1.79×	65.13 (-0.71)	1.85×
	0.7	38.57 (-0.35)	1.71×	53.44 (-0.86)	1.84×	64.49 (-1.35)	1.86×

Table 2: Comparison with other methods of sparsity prediction on ProSparse-LLaMA2-7B, evaluated within our framework: asynchronous overlap is disabled for all methods, including SVDP. For Polar Sparsity, we use threshold-based selection instead of top-k. SVDP outperforms PowerInfer's predictors in most benchmarks. A more lightweight SVDP ($r = 256$) enables additional speedups without compromising quality due to sequential pipeline.

Predictor	TriviaQA acc	ARC-E acc	ARC-C acc	HumanEval pass@1	MBPP pass@1	GSM8K acc	BBH acc	CMMLU acc	Avg. Score	E2E Speedup
Dense	62.94	75.42	53.58	17.68	22.59	11.45	35.85	31.84	38.92	1.00×
GRIFFIN, 20% sparsity	59.77	74.75	53.33	<u>15.85</u>	**22.18**	8.95	35.49	<u>32.01</u>	37.79	1.14×
GRIFFIN, 50% sparsity	35.74	72.77	51.96	12.20	18.17	4.47	35.32	31.88	32.81	1.46×
Deja Vu	60.78	75.21	53.50	2.44	1.54	7.43	35.44	30.67	33.38	1.64×
Polar Sparsity	**62.88**	75.25	53.50	6.10	8.11	10.24	<u>35.61</u>	**32.18**	35.48	1.64×
PowerInfer	61.10	74.54	52.99	<u>15.85</u>	21.15	<u>10.69</u>	35.29	31.80	37.93	1.63×
SVDP ($r = 1024$)	62.66	**75.38**	<u>53.58</u>	**17.68**	**22.18**	10.54	**35.86**	31.85	<u>38.72</u>	1.61×
SVDP ($r = 256$)	<u>62.74</u>	<u>75.34</u>	**53.67**	**17.68**	<u>22.07</u>	**11.22**	35.58	31.94	**38.78**	1.64×

Quality evaluation We evaluate model performance using a diverse set of open-source benchmarks spanning reasoning, coding, knowledge, and multilingual understanding: **GSM8K** [9] (math reasoning), **HumanEval** [6] (code generation), **ARC-E/ARC-C** [8] (scientific and commonsense reasoning), **TriviaQA** [15] (open-domain QA), **MBPP** [4] (small Python tasks), **BBH** [34] (complex reasoning), and **CMMLU** [18] (Chinese multi-task understanding). We adopt the **UltraEval** framework [12], aligning with prior ProSparse evaluations. All benchmarks use text generation rather than perplexity scoring to better reflect real-world LLM usage. For all benchmarks we use greedy sampling. Versions and licenses for used assets are listed in Section K.

Performance evaluation Model performance is measured using the **PyTorch profiler** [2]. The results are averaged across multiple samples from different language domains. E2E latency includes the prefill stage and 200-token generation. Speedup results are stable across independent runs under identical experimental conditions, with a sample standard deviation of $\sigma < 0.001\times$ computed using the closed-form formula. Given this low run-to-run variability, error bars are omitted for brevity.

5.2 Performance under different sparsity ratios

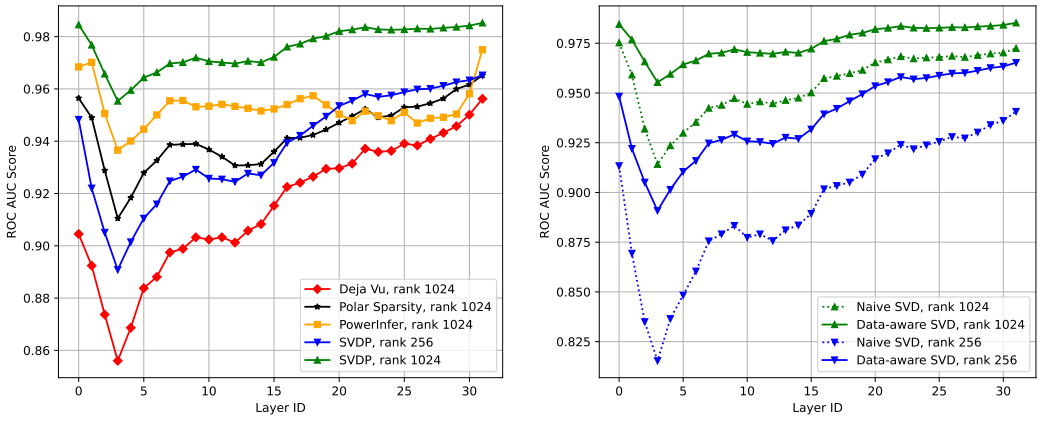
The desired sparsity hyperparameter s from Algorithm 1 influences the calibrated thresholds and can be used to trade off accuracy and acceleration. As Table 1 shows, lower thresholds lead to lower sparsity and lower acceleration. In contrast, higher thresholds provide more speedup but may degrade quality. Long generation tasks are the most sensitive, while for short answer tasks the sparsity can be significantly increased. With our approach we can achieve up to $1.8\times$ E2E generation speedup with less than one percentage point decrease in the average score across the evaluated benchmarks. See Section C for detailed evaluation results.

5.3 Comparison with other methods

We compare SVDP against training-free sparsity methods (GRIFFIN [10]) and learnable predictors (Deja Vu [22], PowerInfer [32], Polar Sparsity [30]). GRIFFIN relies on prompt-based statistical sparsity patterns, while others use trained neural networks. For Deja Vu and Polar Sparsity, we trained predictors following their official repository instructions (see details in Section D). For PowerInfer, we used their recommended pretrained predictors. For SVDP, we tested configurations with $r = 1024$ and $r = 256$, where in the latter configuration the predictor yields a lower level of sparsity but operates faster. In this set of experiments everything runs in one synchronous sequential pipeline, so the tables compare the predictors themselves under identical conditions, not the methods as systems.

Benchmarks Table 2 shows that our SVDP matches PowerInfer's accuracy and performance while requiring less data and compute for construction. GRIFFIN struggles with stable performance at high sparsity ratios, particularly for long generations. The sequential pipeline implementation enables reducing SVDP size by $4\times$ without compromising either accuracy or inference speed.

ROC-AUC analysis We also evaluate predictor separation ability via layer-wise ROC-AUC scores on held-out test sequences (Figure 3a). SVDP outperforms learnable baselines on unseen data. Moreover, SVD-based predictors with $4\times$ smaller size are on par with trainable predictors. Trained neural networks may suffer from out-of-distribution sensitivity due to their data-dependent nature, while SVD predictors directly leverage the LLM's internal weight structure.



(a) (b)

Figure 3: Layer-wise ROC-AUC score of predictors averaged across several sequence samples for ProSparse-LLaMA2-7B. Score of 1 indicates perfect separation and AUROC score of 0.5 corresponds to random guessing. (a) Comparison with trained neural networks. SVDP outperforms PowerInfer on all layers. (b) Evaluation of the impact of data whitening on the predictive ability. Data whitening significantly improves the prediction quality, especially at small rank.

5.4 Ablation study

Table 3 shows ablation of SVD sparsity estimators. Starting from SparseLoRA's baseline [16], referred to as Naive SVD, we sequentially employ techniques: (1) *data whitening* (Sec. 4.1.1), (2) *bias calibration* (Sec. 4.1.2). Bias calibration significantly improves accuracy by accounting for distribution shift after low-rank approximation. Data whitening substantially enhances ROC-AUC separation on held-out test sequences (Figure 3b), independent of bias calibration. The improvement is most noticeable for small rank. For the ablation study against standard threshold selection algorithms see Section H.

5.5 Inference executors

Table 4 presents FFN performance of our CUDA and CANN executors across varying sparsity ratios. Notably, we achieve comparable performance gains on the Ascend hardware using only high-level

Table 3: SVD sparsity estimator ablation study. †Averaged across all benchmarks (see Appendix C for detailed results). Data whitening substantially enhances model quality. Bias calibration allows near lossless inference.

Configuration	ProSparse-LLaMA2-7B			TurboSparse-Mistral-Instruct			SparseQwen2-7B		
	GSM8K	HumanEval	Avg.†	GSM8K	HumanEval	Avg.†	GSM8K	HumanEval	Avg.†
Dense baseline	11.45	17.68	38.92	61.71	34.15	54.30	77.1	64.02	65.84
Parallel pipeline									
Naive SVD	6.60	10.98	34.08	60.20	24.39	51.52	74.00	53.66	62.05
+ Data whitening	10.16	16.46	37.10	60.88	37.20	53.02	75.89	61.59	63.99
+ Bias calibration	10.54	17.68	38.72	62.24	36.59	54.34	76.65	62.20	65.37
Sequential pipeline									
Naive SVD	1.90	0.61	9.69	9.17	1.83	30.87	63.00	12.20	51.60
+ Data whitening	6.90	8.54	31.00	48.60	29.27	46.49	72.93	57.32	60.80
+ Bias calibration	11.22	17.68	38.78	61.41	33.54	53.56	76.80	62.80	65.22

Table 4: FFN performance of our CUDA and CANN executors at different sparsity ratios. The speedups are compared to the corresponding dense implementations. Predictor latency is not included. For this experiment, sparsity ratio is fixed across all linear layers within FFN. As sparsity increases, computation becomes more efficient. A remarkable acceleration up to $30\times$ can be achieved with our CANN executor.

	ProSparse-LLaMA2-7B		TurboSparse-Mistral-Instruct		SparseQwen2-7B	
Sparsity	CUDA	CANN	CUDA	CANN	CUDA	CANN
20%	1.30×	0.76×	1.29×	0.76×	1.33×	0.81×
50%	1.90×	1.23×	1.90×	1.18×	1.95×	1.32×
80%	3.34×	3.38×	3.44×	3.10×	3.47×	3.41×
95%	4.67×	30.67×	4.92×	22.85×	4.91×	28.46×

PyTorch APIs, eliminating the need for manual low-level instruction optimization. This result facilitates deeper exploration of contextual sparsity optimizations within the DaVinci architecture. It is worth noting that when the sparsity value approaches 90%, the L1 [20] cache hit rate reaches almost 100%, resulting in a significant speedup increase.

6 Limitations

- Our method targets sparse models where non-linear activation functions naturally produce significant zero-valued outputs. Extension to non-ReGLU-based architectures remains an important direction for future research. See Section E for detailed discussion.
- Batched processing inherently reduces sparsity levels in FFNs [30], limiting achievable speedups and rendering the direct use of SVDPs ineffective for prefill and multi-token prediction. See Section F for details and possible solutions.
- The algorithm modifies only FFNs, leaving Attention blocks unchanged. Therefore, the E2E speedup is limited by FFN contribution to the total inference time. SVD-based contextual sparsity predictors for Attention layer is a future research direction.
- The calibration data should be drawn from the same distribution as the target application. See details in Section G

Conclusion

This paper introduces a training-free framework for accelerating the inference of ReGLU-based large language models through contextual sparsity exploitation. The proposed solution comprises two stages: (1) an offline calibration phase where sparsity predictors are constructed using data-aware truncated singular value decomposition of gate projection matrices with calibrated bias parameters, and (2) an online inference phase enabling efficient sparse LLM execution on CUDA and CANN

devices. Evaluation across three sparse language models demonstrates up to 1.8× end-to-end decoding acceleration with minimal accuracy degradation across diverse benchmarks, including code and mathematical reasoning tasks.

Acknowledgements

We thank the MSU Laboratory of Computer Engineering for conducting all GPU-based experiments. These were performed exclusively by authors affiliated with Moscow State University; Huawei-affiliated co-authors, who declare no access to GPU-dependent aspects of this work due to U.S. export control regulations, contributed solely to theoretical analysis, algorithm design, and CANN performance evaluation.

Impact statement

This paper presents a training-free method for accelerating LLM inference through contextual sparsity, with particular relevance to edge device deployment. The main potential societal consequence of our work is AI's democratization. By eliminating the need to train sparsity predictors, our approach significantly lowers the barrier to applying contextual sparsity techniques.

References

[1] Yash Akhauri, Ahmed F. AbouElhamayed, Jordan Dotzel, Zhiru Zhang, Alexander M. Rush, Safeen Huda, and Mohamed S. Abdelfattah. ShadowLLM: Predictor-based contextual sparsity for large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 19154–19167, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1068. URL https://aclanthology.org/2024.emnlp-main.1068/.

[2] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL https://docs.pytorch.org/assets/pytorch2-2.pdf.

[3] Saleh Ashkboos, Maximilian Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Representation Learning*, volume 2024, pages 11682–11701, 2024.

[4] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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

[8] 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, 2018. URL https://arxiv.org/abs/1803.05457.

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

[10] Harry Dong, Beidi Chen, and Yuejie Chi. Prompt-prompted adaptive structured pruning for efficient LLM generation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=4aqq9xTtih.

[11] Xiaofei Dong, Xueqiang Zhang, Weixin Bu, Dan Zhang, and Feng Cao. A survey of llm-based agents: Theories, technologies, applications and suggestions. In *2024 3rd International Conference on Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIoTC)*, pages 407–413, 2024. doi: 10.1109/AIoTC63215.2024.10748304.

[12] Chaoqun He, Renjie Luo, Shengding Hu, Ranchi Zhao, Jie Zhou, Hanghao Wu, Jiajie Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. UltraEval: A lightweight platform for flexible and comprehensive evaluation for LLMs. In Yixin Cao, Yang Feng, and Deyi Xiong, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 247–257, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-demos.23. URL https://aclanthology.org/2024.acl-demos.23/.

[13] Huawei Technologies Co., Ltd. Ascend CANN. https://www.hiascend.com/zh/cann. Official product page of CANN for Ascend AI chip series.

[14] Huawei Technologies Co., Ltd. Atlas 200 AI Accelerator Module User Guide. https://support.huawei.com/enterprise/en/doc/EDOC1100223189/968ec576/basic-specifications, January 2023. Official product page of the Ascend 310P3 chip.

[15] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

[16] Samir Khaki, Xiuyu Li, Junxian Guo, Ligeng Zhu, Konstantinos N. Plataniotis, Amir Yazdanbakhsh, Kurt Keutzer, Song Han, and Zhijian Liu. SparseloRA: Accelerating LLM fine-tuning with contextual sparsity. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=z83rodY0Pw.

[17] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[18] Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. CMMLU: Measuring massive multitask language understanding in Chinese. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11260–11285, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.671. URL https://aclanthology.org/2024.findings-acl.671/.

[19] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J. Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, and Sanjiv Kumar. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=TJ2nxciYCk-.

[20] Heng Liao, Jiajin Tu, Jing Xia, and Xiping Zhou. Davinci: A scalable architecture for neural network computing. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–44, 2019. doi: 10.1109/HOTCHIPS.2019.8875654.

[21] James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. Training-free activation sparsity in large language models, 2024. URL https://arxiv.org/abs/2408.14690.

[22] 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*, pages 22137–22176. PMLR, 2023.

[23] Chi Ma, Mincong Huang, Ying Zhang, Chao Wang, Yujie Wang, Lei Yu, Chuan Liu, and Wei Lin. First activations matter: Training-free methods for dynamic activation in large language models, 2024. URL https://arxiv.org/abs/2408.11393.

[24] Meghana Madhyastha, Daniel Haziza, Jesse Cai, Newsha Ardalani, Zhiqi Bu, and Carole-Jean Wu. To 2:4 sparsity and beyond: Neuron-level activation function to accelerate llm pre-training, 2026. URL https://arxiv.org/abs/2602.06183.

[25] Seyed 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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=osoWxY8q2E.

[26] NVIDIA Corporation. *CUDA C Programming Guide*, 12.4 edition, July 2024. URL https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Version 12.4 of the CUDA Toolkit.

[27] Uliana Parkina and Maxim Rakhuba. COALA: Numerically stable and efficient framework for context-aware low-rank approximation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[28] Vasuki Shankar. Edge ai: A comprehensive survey of technologies, applications, and challenges. In *2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET)*, pages 1–6, 2024. doi: 10.1109/ACET61898.2024.10730112.

[29] Jiho Shin, Hoeseok Yang, and Youngmin Yi. Sparseinfer: Training-free prediction of activation sparsity for fast llm inference. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7, 2025. doi: 10.23919/DATE64628.2025.10992997.

[30] Susav Shrestha, Bradley Settlemyer, Nikoli Dryden, and A. L. Narasimha Reddy. Polar sparsity: High throughput batched LLM inferencing with scalable contextual sparsity. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=cg2S1qqNSq.

[31] Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. ProSparse: Introducing and enhancing intrinsic activation sparsity within large language models. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert, editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 2626–2644, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.180/.

[32] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, page 590–606, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400712517. doi: 10.1145/3694715.3695964. URL https://doi.org/10.1145/3694715.3695964.

[33] 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, 2024. URL https://arxiv.org/abs/2406.05955.

[34] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022. URL https://arxiv.org/abs/2210.09261.

[35] Philippe Tillet, H. T. 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*, MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

[36] Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. SVD-LLM v2: Optimizing singular value truncation for large language model compression. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4287–4296, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.217. URL https://aclanthology.org/2025.naacl-long.217/.

[37] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=LNYIUouhdt.

[38] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[39] Yiheng Yang, Yujie Wang, Chi Ma, Lei Yu, Emmanuele Chersoni, and Chu-Ren Huang. Sparse brains are also adaptive brains: Cognitive-load-aware dynamic activation for LLMs. In Vera Demberg, Kentaro Inui, and Lluís Marquez, editors, *Findings of the Association for Computational Linguistics: EACL 2026*, pages 5124–5138, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-386-9. doi: 10.18653/v1/2026.findings-eacl.270. URL https://aclanthology.org/2026.findings-eacl.270/.

[40] Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models, 2025. URL https://arxiv.org/abs/2312.05821.

[41] Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu2 wins: Discovering efficient activation functions for sparse llms. *ArXiv*, abs/2402.03804, 2024. URL https://api.semanticscholar.org/CorpusID:267499856.

[42] Zhengyan Zhang, Chaojun Xiao, Qiujieli Qin, Yankai Lin, Zhiyuan Zeng, Xu Han, Zhiyuan Liu, Ruobing Xie, Maosong Sun, and Jie Zhou. Exploring the benefit of activation sparsity in pre-training. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 60040–60056. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhang24bq.html.

A Theoretical analysis

A.1 Predictor error bounds

Let $m, d > 0$ and $0 < r \leq \min(m, d)$. Consider a weight matrix $W \in \mathbb{R}^{m \times d}$ and its rank-r approximation $W_r \in \mathbb{R}^{m \times d}$ (e.g., truncated SVD), with bias vector $b \in \mathbb{R}^m$. For any input $x \in \mathbb{R}^d$, $x \neq 0$, we have:

$$A(x) := \mathrm{ReLU}(Wx), \tag{10}$$
$$g_b(x) := \mathbf{1}[W_r x + b > 0], \tag{11}$$
$$B_b(x) := A(x) \odot g_b(x). \tag{12}$$

Throughout, we use the following notation:

$$\delta_i \in \mathbb{R}^m, \quad i \in [m], \quad \text{denotes the } i\text{-th standard basis vector } (0, \ldots, \underbrace{1}_{i}, \ldots, 0)^\top, \tag{13}$$

$$t_+ := \max(t, 0), \quad \text{the ReLU function (i.e., } t_+ = 0 \text{ if } t < 0 \text{ and } t_+ = t \text{ if } t \geq 0). \tag{14}$$

We aim to derive deterministic bounds on the relative gating error

$$\mathcal{E}_b(x) := \frac{\|A(x) - B_b(x)\|_2}{\|x\|_2} \tag{15}$$

in terms of the spectral error of W_r and the bias b.

Exact error representation and monotonicity

Definition A.1 (Residual operator)**.** Define the residual matrix and residual vector

$$\Delta W := W - W_r, \qquad e(x) := \Delta W x. \tag{16}$$

Proposition A.2 (Exact representation of the gating error)**.** *For every $x \in \mathbb{R}^d$,*

$$A(x) - B_b(x) = \mathrm{ReLU}(Wx) \odot \mathbf{1}[W_r x + b \leq 0], \tag{17}$$

and therefore

$$\|A(x) - B_b(x)\|_2^2 = \sum_{i=1}^{m} ((Wx)_i)_+^2 \; \mathbf{1}[(W_r x + b)_i \leq 0]. \tag{18}$$

Proof. The claim follows immediately from the componentwise definition of $B_b(x)$. □

Proposition A.3 (Monotonicity w.r.t. biases)**.** *Let $b, b' \in \mathbb{R}^m$ and suppose $b' \leq b$ componentwise. Then for all $x \in \mathbb{R}^d$,*

$$\|A(x) - B_{b'}(x)\|_2 \;\geq\; \|A(x) - B_b(x)\|_2 \,. \tag{19}$$

Proof. If $b' \leq b$, then $\mathbf{1}[W_r x + b \leq 0] \;\leq\; \mathbf{1}[W_r x + b' \leq 0]$ componentwise. Using the identity (17) and the fact that $\mathrm{ReLU}(Wx) \geq 0$ componentwise, the ℓ_2 norm can only increase when we replace b by the smaller bias b', which activates the mask on a superset of coordinates. □

Remark A.4. Using thresholds $\tau = -b$ clarifies the trade-off: increasing τ increases sparsity (more gates are off) but never decreases the error; equivalently, increasing b reduces sparsity and never increases the error.

A deterministic bound via the shifted residual

Lemma A.5 (Residual-based upper bound)**.** *For every $x \in \mathbb{R}^d$ and $b \in \mathbb{R}^m$,*

$$\|A(x) - B_b(x)\|_2 \;\leq\; \left\|(\Delta W x - b)_+\right\|_2 \;\leq\; \|\Delta W x - b\|_2 \,. \tag{20}$$

Proof. Let $z = Wx$, $z_r = W_r x$, and $e := z - z_r = \Delta W x$. Define the gated index set

$$I := \{i \in [m] : (z_r + b)_i \le 0\}. \tag{21}$$

By definition,

$$\|A(x) - B_b(x)\|_2^2 = \sum_{i \in I} (z_i)_+^2 . \tag{22}$$

For $i \in I$ we have $z_{r,i} \le -b_i$, hence $z_i = z_{r,i} + e_i \le e_i - b_i$. By monotonicity of $t \mapsto (t)_+$, $(z_i)_+ \le (e_i - b_i)_+$. Therefore

$$\sum_{i \in I} (z_i)_+^2 \le \sum_{i=1}^{m} (e_i - b_i)_+^2 = \left\|(e-b)_+\right\|_2^2 , \tag{23}$$

which gives the first inequality. The second inequality follows from $\left\|(u)_+\right\|_2 \le \|u\|_2$. □

Worst-case bounds Due to the previous result, the analysis of the nonlinear gating error reduces to a deterministic quantity depending only on the residual $\Delta W x$ and the bias b. We therefore can study the worst-case value

$$\sup_{\|e\|_2 \le R} \left\|(e-b)_+\right\|_2 , \tag{24}$$

which characterizes the maximal gating error at residual energy R.

Uniform bias

Theorem A.6 (Worst-case bound for uniform bias)**.** *Let $b = \beta \mathbf{1}$ with $\beta \in \mathbb{R}$. For all $R \ge 0$,*

$$\sup_{\|e\|_2 \le R} \left\|(e - \beta\mathbf{1})_+\right\|_2 = \begin{cases} (R-\beta)_+ , & \beta \ge 0, \\ R - \beta\sqrt{m}, & \beta < 0. \end{cases} \tag{25}$$

Proof.
Case 1: $\beta \ge 0$**.** Write $y := (e - \beta\mathbf{1})_+ \in \mathbb{R}^m_{\ge 0}$ and $t := \|y\|_2$.

Let $S := \{i : y_i > 0\}$ (if $t = 0$ there is nothing to prove, otherwise $R > \beta$).

For $i \in S$ we have $e_i = \beta + y_i$, hence

$$\|e\|_2^2 \;\ge\; \sum_{i \in S} (\beta + y_i)^2 \;=\; \beta^2 |S| + 2\beta \sum_{i \in S} y_i + \sum_{i \in S} y_i^2 . \tag{26}$$

Since $y \ge 0$, we have $\sum_{i \in S} y_i = \|y\|_1 \ge \|y\|_2 = t$, and also $|S| \ge 1$. Therefore

$$\|e\|_2^2 \;\ge\; \beta^2 + 2\beta t + t^2 \;=\; (t + \beta)^2, \tag{27}$$

so $\|e\|_2 \ge t + \beta$. Under $\|e\|_2 \le R$ this yields $t \le R - \beta$, i.e., $\left\|(e - \beta\mathbf{1})_+\right\|_2 \le (R-\beta)_+$.

Equality holds by taking

$$R \le \beta : \quad e = 0, \tag{28}$$
$$R > \beta : \quad e = R\delta_1 . \tag{29}$$

Case 2: $\beta < 0$**.** Write $\beta = -c$ with $c > 0$. Then for any e,

$$\left\|(e - \beta\mathbf{1})_+\right\|_2 = \left\|(e + c\mathbf{1})_+\right\|_2 \le \|e + c\mathbf{1}\|_2 \le \|e\|_2 + c\,\|\mathbf{1}\|_2 \le R + c\sqrt{m} = R - \beta\sqrt{m}. \tag{30}$$

Equality holds for $e = \frac{R}{\sqrt{m}}\mathbf{1}$. □

General bias vector

Theorem A.7 (Worst-case bound for general bias)**.** *Let $b \in \mathbb{R}^m$ and $R \geq 0$. Then*

$$\sup_{\|e\|_2 \leq R} \left\|(e-b)_+\right\|_2 = \begin{cases} \max_{i\in[m]}(R-b_i)_+, & b \geq 0 \ \textit{componentwise}, \\ R + \|b\|_2, & b < 0 \ \textit{componentwise}. \end{cases} \tag{31}$$

Proof.
Case 1: $b < 0$**.**

$$\|(e-b)_+\|_2 \leq \|e-b\|_2 \leq R + \|b\|_2 . \tag{32}$$

Tightness: $e = \frac{R}{\|b\|_2}(-b)$ gives exactly $R + \|b\|_2$.

Case 2: $b \geq 0$**.** For $i^\star = \arg\min_i b_i$, let $b_c = b_{i^\star}\mathbf{1}$. Then $(e-b)_+ \leq (e-b_c)_+$ componentwise, so

$$\|(e-b)_+\|_2 \leq \|(e-b_c)_+\|_2 \overset{(A.6)}{\leq} (R-b_{i^\star})_+ = \max_i (R-b_i)_+. \tag{33}$$

Tightness: $e = R\delta_{i^\star}$ for $i^\star = \arg\max_i (R-b_i)_+$ achieves equality. □

Remark A.8. Theorem A.7 identifies two regimes in which the worst-case residual can be characterized exactly: a fully negative bias, where the bound grows linearly with $\|b_-\|_2$, and a fully nonnegative bias, where the optimum concentrates on a single coordinate.

Corollary A.9 (Main deterministic bound)**.** *Let $b \in \mathbb{R}^m$ and $R \geq 0$ be such that $\|\Delta W x\|_2 \leq R$. Then for all $x \in \mathbb{R}^d$,*

$$\mathcal{E}_b(x) = \frac{\|A(x) - B_b(x)\|_2}{\|x\|_2} \leq \begin{cases} \dfrac{\max_{i\in[m]}(R-b_i)_+}{\|x\|_2}, & b \geq 0 \ \textit{componentwise}, \\ \dfrac{R + \|b\|_2}{\|x\|_2}, & b < 0 \ \textit{componentwise}. \end{cases} \tag{34}$$

Corollary A.10 (Truncated SVD)**.** *By Eckart–Young–Mirsky, if W_r is the rank-r truncated SVD of W^{gate}, then $\|\Delta W\|_2 = \sigma_{r+1}(W^{\text{gate}}) = \sigma_{r+1}$, so for all $x \in \mathbb{R}^d$,*

$$\mathcal{E}_b(x) \leq \begin{cases} \max\limits_{i\in[m]} \left(\sigma_{r+1} - \dfrac{b_i}{\|x\|_2}\right)_+, & b \geq 0 \ \textit{componentwise}, \\ \sigma_{r+1} + \dfrac{\|b\|_2}{\|x\|_2}, & b < 0 \ \textit{componentwise}. \end{cases} \tag{35}$$

Remark A.11. While rather loose in practice due to data distribution, this bound cleanly separates approximation error σ_{r+1} from gating bias b, revealing bias addition's influence on residuals.

A.2 Greedy threshold selection

Greedy threshold calibration under a fixed sparsity budget We study a data-dependent calibration rule for componentwise thresholds $\tau \in \mathbb{R}^m$ under a fixed *drop budget* K (equivalently, fixed sparsity). Throughout, the gate is

$$g_\tau(x) = \mathbf{1}[ABx > \tau], \tag{36}$$

where AB denotes any rank-r approximation of $W^{\text{gate}} \in \mathbb{R}^{m\times d}$; the specific choice used in our experiments is given in Section 4.1.1.

Proxy scores and damage weights Let $\mathcal{D} = \{x^{(t)}\}_{t=1}^T$ be a dataset and write $[m] = \{1, \ldots, m\}$, $[T] = \{1, \ldots, T\}$. For each coordinate $i \in [m]$ and sample $t \in [T]$ define the proxy score

$$s_i^{(t)} := (ABx^{(t)})_i, \tag{37}$$

and define the *marginal damage weight*

$$w_i^{(t)} := \left|\text{ReLU}\left(W^{\text{gate}}_{i,:}x^{(t)}\right) \cdot W^{\text{up}}_{i,:}x^{(t)}\right|^2 \left\|W^{\text{down}}_{:,i}\right\|_2^2 \qquad (\geq 0). \tag{38}$$

Algorithm 1 Greedy Bias Calibration (Offline Phase)

Input:
$W^{\text{gate}}, W^{\text{up}} \in \mathbb{R}^{D\times d}$, $W^{\text{down}} \in \mathbb{R}^{d\times D}$ – weight matrices,
$\mathcal{D} = \{x^{(t)} \in \mathbb{R}^d\}_{t=1}^T$ – calibration dataset of $T \in \mathbb{N}$ tokens,
A, B – low-rank approximation from Algorithm 4,
$s \in [0;1]$ – desired sparsity level, $\eta \in \mathbb{N}$ – discrete step size.
Output: $b \in \mathbb{R}^D$

$w_i^{(t)} \leftarrow \left|\text{ReLU}\left(W_{i,:}^{\text{gate}} x^{(t)}\right) \cdot W_{i,:}^{\text{up}} x^{(t)}\right|^2 \left\|W_{:,i}^{\text{down}}\right\|_2^2$ *(Compute neurons' importance)*

$s_i^{(t)} \leftarrow (ABx^{(t)})_i$ *(Compute predictor's outputs)*

for $i = 1$ **to** D **do**

$s_{i,(1)} \le \cdots \le s_{i,(T)}$ *(Sort samples in nondecreasing order)*

$$C_i(k) = \begin{cases} \sum_{j=1}^k w_{i,(j)}, & k = 1,\dots,T \\ C_i(T), & T < k < T+\eta \\ +\infty, & k = T+\eta \end{cases}$$ *(Compute cumulative costs with padding)*

$k_i \leftarrow \min\{k : w_{i,(k)} > 0\}, \quad \tau_i \leftarrow s_{i,(k_i)}$ *(Initialize pointers and thresholds)*

end for

while $\frac{1}{DT}\sum_{i=1}^D \sum_{t=1}^T \mathbf{1}[s_i^{(t)} \le \tau_i] < s$ **do**

$i^\star \leftarrow \arg\min_i \left[C_i(k_i+\eta) - C_i(k_i)\right]$ *(Greedy step)*

$k_{i^\star} \leftarrow \min(k_{i^\star} + \eta,\, T), \quad \tau_{i^\star} \leftarrow s_{i^\star,(k_{i^\star})}$ *(Update pointers and thresholds)*

end while

$b \leftarrow -\tau$

return b

Given thresholds τ, the empirical output error incurred by dropping (i,t) whenever $s_i^{(t)} \le \tau_i$ is

$$\mathcal{L}(\tau) = \sum_{t=1}^T \sum_{i=1}^m w_i^{(t)}\, \mathbf{1}[s_i^{(t)} \le \tau_i]. \tag{39}$$

Remark A.12 (Origin of the weight (38)). Consider the standard gated FFN decomposition

$$z(x) = \text{ReLU}(W^{\text{gate}}x) \odot (W^{\text{up}}x), \qquad h(x) = W^{\text{down}} z(x). \tag{40}$$

Then $h(x)$ decomposes into rank-one terms:

$$h(x) = \sum_{i=1}^m z_i(x)\, W_{:,i}^{\text{down}}, \quad z_i(x) = \text{ReLU}(W_{i,:}^{\text{gate}}x) \cdot W_{i,:}^{\text{up}} x. \tag{41}$$

Dropping the i-th hidden coordinate removes exactly the contribution $v_i(x) = z_i(x) W_{:,i}^{\text{down}}$, hence

$$\|v_i(x)\|_2^2 = \left|\text{ReLU}\left(W_{i,:}^{\text{gate}}x\right) \cdot W_{i,:}^{\text{up}} x\right|^2 \left\|W_{:,i}^{\text{down}}\right\|_2^2, \tag{42}$$

which matches (38) for $x = x^{(t)}$.

Prefix formulation and fixed drop budget Fix $i \in [m]$. Sort samples in nondecreasing order of proxy scores:

$$s_{i,(1)} \le s_{i,(2)} \le \cdots \le s_{i,(T)}, \tag{43}$$

and let $w_{i,(j)}$ denote the corresponding permuted weights. Define

$$C_i(k) := \sum_{j=1}^k w_{i,(j)}, \quad C_i(0) = 0, \qquad \Delta_i(k) := C_i(k) - C_i(k-1) = w_{i,(k)}. \tag{44}$$

For each coordinate i, setting τ_i so exactly k_i samples satisfy $s_i^{(t)} \le \tau_i$ incurs cost $C_i(k_i)$. The total loss is thus

$$\mathcal{L}(\tau) = \sum_{i=1}^m C_i(k_i), \qquad k_i := \#\{t :\ s_i^{(t)} \le \tau_i\} \in \{0,\dots,T\}. \tag{45}$$

With fixed drop budget $K \in \{0, 1, \ldots, mT\}$, threshold selection reduces to

$$\min_{\substack{k_i \in \{0,\ldots,T\} \\ \sum_{i=1}^m k_i = K}} \sum_{i=1}^m C_i(k_i). \tag{46}$$

Greedy optimality

Definition A.13 (Global optimality)**.** A feasible solution $k^\star = (k_1^\star, \ldots, k_m^\star)$ with $\sum_i k_i^\star = K$ is *globally optimal* for (46) if

$$\sum_{i=1}^m C_i(k_i^\star) \le \sum_{i=1}^m C_i(k_i) \quad \text{for all feasible } k = (k_1, \ldots, k_m). \tag{47}$$

Assumption A.14 (Monotone marginals within each coordinate)**.** For every $i \in [m]$, the marginal increments are nondecreasing:

$$\Delta_i(1) \le \Delta_i(2) \le \cdots \le \Delta_i(T). \tag{48}$$

Theorem A.15 (Greedy is globally optimal)**.** *Initialize $k_i = 0$ for all i. For $\ell = 1, 2, \ldots, K$ (drop budget) choose*

$$i_\ell \in \arg\min_{i:\, k_i < T} \Delta_i(k_i + 1), \qquad \textit{and update} \qquad k_{i_\ell} \leftarrow k_{i_\ell} + 1. \tag{49}$$

Under Theorem A.14 the resulting k is globally optimal for (46)*.*

Moreover, it can be realized by thresholds τ via

$$\tau_i \in [\, s_{i,(k_i)},\ s_{i,(k_i+1)} \,), \tag{50}$$

with conventions $s_{i,(0)} := -\infty$ and $s_{i,(T+1)} := +\infty$.

Proof. The problem is equivalent to selecting exactly K increments $\Delta_i(j)$ while respecting prefix constraints: selecting $\Delta_i(j)$ requires all $\Delta_i(1), \ldots, \Delta_i(j-1)$.

The greedy algorithm selects the smallest available "next increment" $\Delta_i(k_i + 1)$ at each step. Let G be greedy's sequence of K increments, and O any optimal sequence.

Suppose $G \neq O$. Let $a = \Delta_p(k_p + 1)$ be the first (chronologically) greedy increment not in O. Since $|G| = |O| = K$, there exists $b = \Delta_q(j) \in O \setminus G$.

Choose b as the *last* increment of coordinate q in O (maximal j). Removing b preserves q's prefix constraint. Adding a extends p's prefix, also preserving feasibility.

By greedy choice, $a \le \Delta_q(k_q + 1)$ where k_q counts greedy's selections in q up to that point. Since $b \notin G$ but O includes it, $j \ge k_q + 1$. By Theorem A.14 (nondecreasing increments), $\Delta_q(k_q + 1) \le \Delta_q(j) = b$.

Thus $a \le b$, so swapping $b \leftrightarrow a$ preserves feasibility and does not increase cost. Repeating such exchanges transforms O into G, proving optimality.

The threshold realization follows from the definition of k_i. □

Remark A.16 (Role of rank-r approximation and empirical validation). Theorem A.15 requires monotone marginal increments after sorting by proxy scores (Theorem A.14). This is *not guaranteed*, since proxy scores $s_i^{(t)} = (ABx^{(t)})_i$ depend only on rank-r approximation $AB \approx W^{\mathrm{gate}}$, while damage, defined above, also involves W^{up} and W^{down}.

If AB approximates W^{gate} well *on the input data distribution*, then $(ABx)_i \approx (W^{\mathrm{gate}}x)_i$, so proxy sorting should align with true damage ordering. In practice, we perform greedy selection over non-overlapping η-step sums to mitigate data noise:

$$S_i^{(m)} := \sum_{j=(m-1)\eta+1}^{m\eta} w_{i,(j)}, \quad m = 1, \ldots, \lfloor T/\eta \rfloor. \tag{51}$$

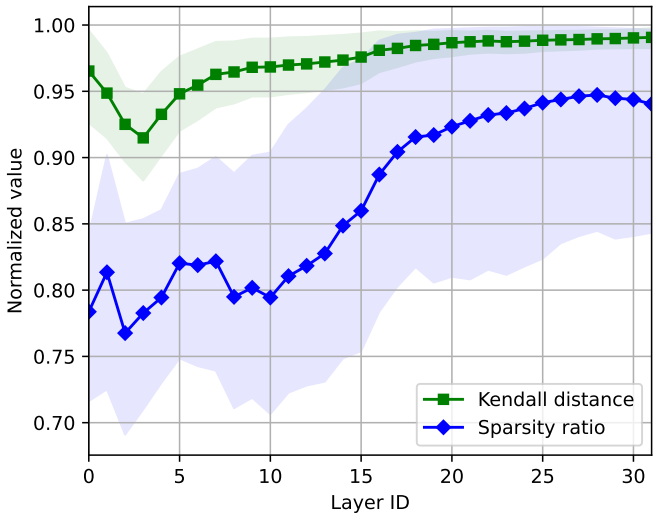


Figure 4: Mean Kendall τ_i^{K} across neurons for proxy-sorted η-step sums from ProSparse-LLaMA2-7B. The x-axis shows layer number. For clarity, layer-wise sparsity ratios are depicted. The shaded regions denote the 10th to 90th percentile interval, illustrating the central 80% of the data distribution.

We validate their monotonicity using normalized Kendall tau distance between group sums and their position indices:

$$\tau_i^{\mathrm{K}} := 1 - \frac{2K_d\big(\{S_i^{(m)}\}_{m=1}^{M},\ \{m\}_{m=1}^{M}\big)}{M(M-1)}, \tag{52}$$

where $K_d(\tau_1, \tau_2) = \#\big\{(m_1, m_2) :\ m_1 < m_2,\ [\tau_1(m_1) \lesseqgtr \tau_1(m_2) \wedge \tau_2(m_1) \gtreqless \tau_2(m_2)]\big\}$ counts discordant pairs.

The use of Kendall tau distance is justified by the fact that it directly quantifies *ordinal consistency* required by Theorem A.14: for all $m_1 < m_2$, we expect $S_i^{(m_1)} \leq S_i^{(m_2)}$. Formally, τ_i^{K} measures the fraction of concordant pairs while properly accounting for ties – the precise condition for greedy optimality in Theorem A.15. On sample data used to calibrate biases, Figure 4 shows stable Kendall τ_i^{K} in $[0.9, 1]$, confirming proxy sorting $ABx \approx W^{\mathrm{gate}}x$ and the suitability of our greedy approach.

Remark A.17 (Knapsack reformulation and practical optimality). Problem (46) is a 0–1 knapsack problem with mT items $\{\Delta_i(j)\}_{i=1}^{m}{}_{j=1}^{T}$, capacity K, and prefix constraints that selection of $\Delta_i(j)$ requires prior selection of $\Delta_i(1), \ldots, \Delta_i(j-1)$. While standard dynamic programming requires $O(mTK)$ time, our greedy algorithm runs in $O((K+m)\log m)$ via priority queue and is *exactly optimal* under Theorem A.14, yielding superior efficiency.

Time and space complexity For one FFN block with D neurons and T calibration tokens of hidden size d, computing and storing the neuron importances is a single FFN forward pass, $O(dDT)$, which is normally already available from the calibration pass that the SVD stage requires. Sorting the activations per neuron is $O(DT \log T)$. The calibration itself is the knapsack problem of Theorem A.17, with DT items and a drop budget $K < DT$ set by the target sparsity, solved greedily in $O((K+D)\log D)$ with a priority queue. In total $O(dDT + DT\log T + (K+D)\log D)$ time and $O(DT)$ additional memory.

B Sparse FFN implementation

Figure 5 illustrates the sparse inference pipeline, where speedup is determined by the token-dependent sparsity ratio.

Prediction latency can be hidden via asynchronous computation, where predictor execution time is overlapped with data transfer from global memory or inter-process communication under tensor parallelism. Unlike Deja Vu's [22] asynchronous pipeline, which leverages Ampere's

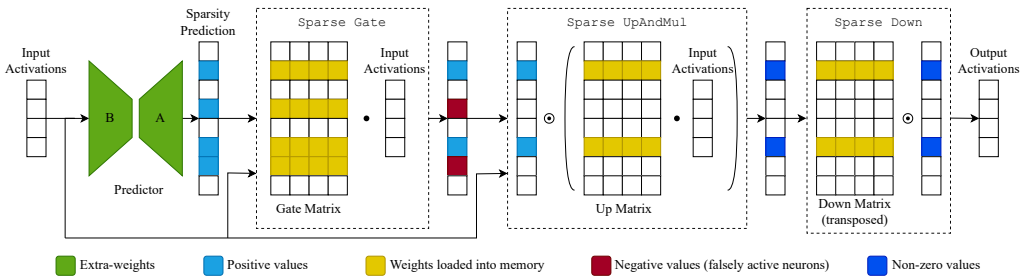


Figure 5: Sparse inference using SVD-based sparsity predictors. Neurons with positive predictor values are considered active during gate projection. Gate and up projections are executed sequentially to increase sparsity.

Algorithm 2 Sparse FFN Forward Pass on CUDA (Decoding Phase)

Input: $W^{\text{gate}}, W^{\text{up}} \in \mathbb{R}^{D \times d}$, $W^{\text{down}} \in \mathbb{R}^{d \times D}$, $x \in \mathbb{R}^d$, $A \in \mathbb{R}^{D \times r}$, $B \in \mathbb{R}^{r \times d}$, $b \in \mathbb{R}^D$
Output: $x \in \mathbb{R}^d$
$m \leftarrow ABx + b$ *(Predict sparsity pattern)*
$m \leftarrow \textsc{SparseGateKernel}(W^{\text{gate}}, x, m)$
$x \leftarrow \textsc{SparseUpAndMulKernel}(W^{\text{up}}, x, m)$
$x \leftarrow \textsc{SparseDownKernel}(W^{\text{down}}, x)$
return x

`cuda::memcpy_async` operations [26], our approach targets consumer-grade GPUs with compute capability 7.0 for broader applicability. Predictors are inserted at the beginning of each FFN and sequentially generate sparsity patterns for their respective gate projections. This design ensures minimal integration overhead with existing frameworks.

Custom CUDA kernels for sparse FFN We implement three CUDA kernels (Algorithm 2). The predictor's output $m \in \mathbb{R}^D$ highlights active rows of W^{gate}. Throughout gate and up projections each CUDA block processes one neuron, first verifying its activity via m. Gate outputs are stored back into m for reuse. Key optimizations:

- **Memory**: Column-major storage of W^{down} for coalesced memory access.
- **Precision**: Full precision register operations; half precision for memory reads and writes.
- **Determinism**: Avoid atomic additions via shared-memory partial sums in down projection.

Our kernel design diverges from existing ProSparse [31] and Deja Vu [22] implementations. Both prior approaches rely on atomic addition operations within the down projection phase, introducing non-determinism and potential precision instability during floating-point accumulation. To address these limitations, our algorithm employs partial sum reduction implemented via shared memory. This technique enforces a predefined summation order and utilizes a hierarchical reduction strategy. Consequently, our approach guarantees deterministic outputs while simultaneously achieving enhanced numerical stability through reduced computational error propagation. We postpone kernel fusion due to diminishing returns, prioritizing maintainability.

NPU solution for sparse FFN We implement our NPU solution for sparse FFN, detailed in Algorithm 3, leveraging high-level PyTorch instructions. The predictor's output $m \in \mathbb{R}^D$ is used to construct smaller dense submatrices by retrieving the corresponding rows or columns. This approach enables highly efficient execution by utilizing the dedicated matrix multiplication accelerator unit (cube) inherent to the CANN NPU architecture [20]. W^{down} matrix is stored in column-major format. Algorithm 3 is written for the ReGLU case for clarity; the dReLU variant differs only in the additional ReLU on m_2.

Algorithm 3 Sparse FFN Forward Pass on CANN (Decoding Phase)

Input: $W^{\text{gate}}, W^{\text{up}} \in \mathbb{R}^{D\times d}$, $W^{\text{down}} \in \mathbb{R}^{d\times D}$, $x \in \mathbb{R}^d$, $A \in \mathbb{R}^{D\times r}$, $B \in \mathbb{R}^{r\times d}$, $b \in \mathbb{R}^D$
Output: $x \in \mathbb{R}^d$
$m \leftarrow ABx + b > 0$ *(Predict sparsity pattern)*
$p \leftarrow \text{NONZERO}(m)$ *(Obtain indices)*
$g \leftarrow \text{SELROWS}(W^{\text{gate}}, p)$ *(Select rows of W^{gate} indexed by p)*
$m_1 \leftarrow \text{MATMUL}(g, x)$
$u \leftarrow \text{SELROWS}(W^{\text{up}}, p)$ *(Select rows of W^{up} indexed by p)*
$m_2 \leftarrow \text{MATMUL}(u, x)$
$m_3 \leftarrow \text{RELU}(m_1) \cdot m_2$
$d \leftarrow \text{SELCOLS}(W^{\text{down}}, p)$ *(Select columns of W^{down} indexed by p)*
$x \leftarrow \text{MATMUL}(d, m_3)$
return x

Table 5: ProSparse-LLaMA2-7B evaluation results, $r = 256$.

Method	s	TriviaQA acc, %	ARC-E acc, %	ARC-C acc, %	HumanEval pass@1, %	MBPP pass@1, %	GSM8K acc, %	BBH acc, %	CMMLU acc, %	Avg. Score	E2E Speedup
Dense	-	62.94	75.42	53.58	17.68	22.59	11.45	35.85	31.84	38.92	1.00×
SVDP	0.4	62.38	75.29	53.67	17.68	22.38	10.61	35.46	31.81	38.66	1.61×
	0.5	62.74	75.34	53.67	17.68	22.07	11.22	35.58	31.94	38.78	1.64×
	0.6	62.90	75.29	53.75	18.29	21.97	10.54	35.54	31.91	38.77	1.67×
	0.7	62.04	75.42	53.58	17.07	21.87	11.22	35.43	31.90	38.57	1.71×

C Detailed accuracy results

Table 5, Table 6, Table 7 provide detailed evaluation results for three sparse LLMs equipped with our SVD-predictors. Table 8, Table 10, Table 9 provide a detailed ablation study of SVD sparsity estimators for three sparse LLMs.

D Baseline training configurations

Both Deja Vu and Polar Sparsity predictors were trained with the pipelines and configs from the official repositories. Training configurations are provided in Table 11. Training took about 5 minutes per layer, i.e. about 3 hours per model.

E Extension to non-ReLU activations

The solution can in principle be applied to LLMs with non-ReLU activation functions, such as SiLU. However, the straightforward utilization may lead to substantial accuracy degradation on long generation tasks. The quality evaluation results for Qwen2-7B-Instruct model with SiLU activation function are presented in Table 12. Potential modifications to the current solution may include transformations to the intermediate activations to mitigate errors that arise.

Table 6: TurboSparse-Mistral-Instruct evaluation results, $r = 352$.

Method	s	TriviaQA acc, %	ARC-E acc, %	ARC-C acc, %	HumanEval pass@1, %	MBPP pass@1, %	GSM8K acc, %	BBH acc, %	CMMLU acc, %	Avg. Score	E2E Speedup
Dense	-	48.58	87.21	71.67	34.15	42.20	61.71	42.35	46.52	54.30	1.00×
SVDP	0.4	45.46	87.08	71.67	32.93	42.09	61.49	42.46	46.55	53.72	1.67×
	0.5	43.26	87.04	71.76	33.54	42.71	61.41	42.32	46.47	53.56	1.73×
	0.6	40.63	87.08	72.10	33.54	42.09	61.41	42.19	46.43	53.18	1.79×
	0.7	39.21	87.21	72.10	37.20	42.09	60.96	42.33	46.45	53.44	1.84×

Table 7: SparseQwen2-7B evaluation results, $r = 512$.

Method	s	TriviaQA acc, %	ARC-E acc, %	ARC-C acc, %	HumanEval pass@1, %	MBPP pass@1, %	GSM8K acc, %	BBH acc, %	CMMLU acc, %	Avg. Score	E2E Speedup
Dense	-	50.76	91.92	83.87	64.02	46.3	77.1	48.42	64.35	65.84	1.00×
SVDP	0.4	50.87	91.92	83.87	62.80	44.56	76.50	48.40	64.35	65.41	1.78×
	0.5	49.96	91.92	83.87	62.80	43.53	76.80	48.40	64.35	65.20	1.81×
	0.6	48.82	91.92	83.87	63.41	43.53	76.80	48.34	64.35	65.13	1.85×
	0.7	47.42	91.92	83.87	62.20	41.99	75.82	48.36	64.35	64.49	1.86×

Table 8: SVD sparsity estimator ablation study for ProSparse-LLaMA2-7B.

	TriviaQA	ARC-E	ARC-C	HumanEval	MBPP	GSM8K	BBH	CMMLU	Avg.
Dense baseline	62.94	75.42	53.58	17.68	22.59	11.45	35.85	31.84	38.92
Parallel pipeline									
Naive SVD	58.19	69.91	49.32	10.98	13.35	6.60	33.98	30.28	34.08
+ Data whitening	53.85	75.34	53.75	16.46	19.92	10.16	35.46	31.83	37.10
+ Bias calibration	62.66	75.38	53.58	17.68	22.18	10.54	35.86	31.85	38.72
Sequential pipeline									
Naive SVD	34.53	3.58	4.35	0.61	0.21	1.90	11.12	21.23	9.69
+ Data whitening	27.58	75.21	53.33	8.54	11.40	6.90	33.71	31.34	31.00
+ Bias calibration	62.74	75.34	53.67	17.68	22.07	11.22	35.58	31.94	38.78

F Dynamic sparsity for batched processing

A power-law distribution exists in neuron activation within FFNs of LLMs, suggesting that the number of activated FFN neurons does not grow linearly with the batch size [22, 32]. Nevertheless, Shrestha et al. [30] shows that activation sparsity in the FFNs degrades with batching. In line with this, many contextual sparsity methods, such as TEAL [21], do not optimize the prefill stage. As FFNs become compute-bound under batching, MoE-based sparsity approach outperforms neuron-based sparsity approach [42]. In this scenario, training-free contextual sparsity predictors, such as SVDP, can be utilized as a replacement for training-free routers. For instance, Zhang et al. [42] and Madhyastha et al. [24] use k-means clustering to MoE-fy the dense model, accelerating the LLM pre-training through sparsity. While the learnable predictors are not employed because of high training costs, SVDP can be adapted for more efficient sparsity utilization.

G Sensitivity analysis to calibration data

Data size We analyzed the sensitivity of the predictor construction algorithm to the size of the calibration dataset (Figure 6). Since both data whitening and bias calibration require calibration tokens, we analyzed them independently. For data whitening (Figure 6a), we measure ROC AUC scores on held-out test sequences. For bias calibration, we measure FFN relative errors. As few as 10,000 tokens are enough for reliable construction, however small gains can be obtained by increasing

Table 9: SVD sparsity estimator ablation study for SparseQwen2-7B.

	TriviaQA	ARC-E	ARC-C	HumanEval	MBPP	GSM8K	BBH	CMMLU	Avg.
Dense baseline	50.76	91.92	83.87	64.02	46.30	77.10	48.42	64.35	65.84
Parallel pipeline									
Naive SVD	46.24	91.92	83.87	53.66	34.39	74.00	48.00	64.35	62.05
+ Data whitening	42.92	91.92	83.87	61.59	43.02	75.89	48.37	64.35	63.99
+ Bias calibration	50.20	91.92	83.87	62.20	45.28	76.65	48.45	64.35	65.37
Sequential pipeline									
Naive SVD	40.03	91.92	83.87	12.20	9.03	63.00	48.43	64.35	51.60
+ Data whitening	30.00	91.92	83.87	57.32	37.78	72.93	48.22	64.35	60.80
+ Bias calibration	49.96	91.92	83.87	62.80	43.63	76.80	48.40	64.35	65.22

Table 10: SVD sparsity estimator ablation study for TurboSparse-Mistral-Instruct.

	TriviaQA	ARC-E	ARC-C	HumanEval	MBPP	GSM8K	BBH	CMMLU	Avg.
Dense baseline	48.58	87.21	71.67	34.15	42.20	61.71	42.35	46.52	54.30
Parallel pipeline									
Naive SVD	41.40	87.04	71.76	24.39	38.91	60.20	42.13	46.34	51.52
+ Data whitening	37.98	86.57	71.25	37.20	41.79	60.88	42.27	46.18	53.02
+ Bias calibration	46.21	87.12	71.76	36.59	41.89	62.24	42.44	46.45	54.34
Sequential pipeline									
Naive SVD	35.29	71.09	57.85	1.83	8.73	9.17	37.72	25.24	30.87
+ Data whitening	18.09	87.12	72.27	29.27	28.44	48.60	41.87	46.22	46.49
+ Bias calibration	43.26	87.04	71.76	33.54	42.71	61.41	42.32	46.47	53.56

Table 11: Training configurations for Deja Vu and Polar Sparsity predictors.

Hyperparameter	Deja Vu	Polar Sparsity
Dataset	C4	WikiText
Max Samples	500	5000
Tokens	240k	400k
Train/Test	95/5	95/5
Batch Size	1024	64
Epochs	20	20
Loss	BCE	BCE
Learning Rate	1e-4	1e-4
Architecture	Linear($r = 1024$) + ReLU + Linear, no bias	Linear($r = 1024$) + Linear, no bias
Early Stopping	patience 5	patience 5
Optimizer	AdamW	AdamW
Weight Decay	0.01	0.01

the data size. This contrasts with the trainable approach, where hundreds of thousands of token samples are used. For instance, [31] and [30] use 400,000 tokens. Importantly, when performing SVD compression with a small number of tokens, $XX^\top$ is likely to be non-positive-definite, making Cholesky decomposition impossible. Instead, SVD-LLM V2 [36] can be used. Here, we also rely on weighted low-rank approximation *with regularization* [27], which is effective when data is scarce:

$$\min_{\text{rank}(AB)\leq r} \|WX - ABX\|_F^2 + \mu\|W - AB\|_F^2, \tag{53}$$

and we set $\mu = 1$.

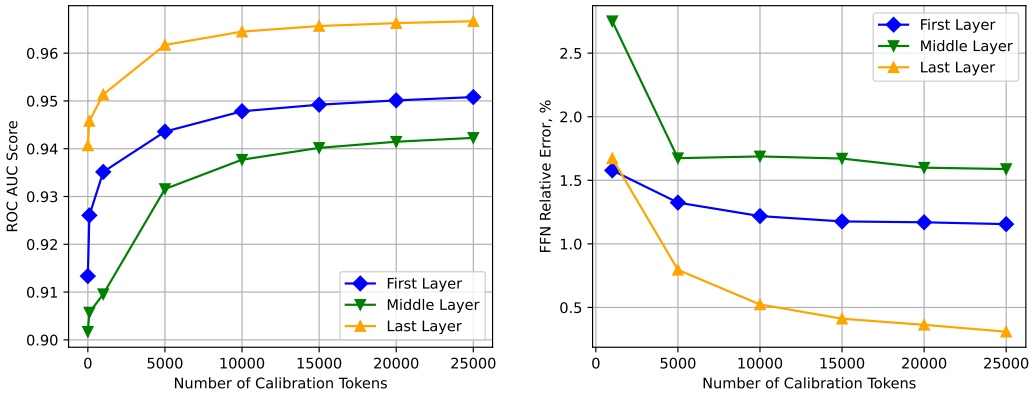
Data distribution To quantify what happens when calibration data is not drawn from the target application, we calibrate the whitening and bias of ProSparse-LLaMA2-7B on the dev split of CMMLU. We then evaluated on the remaining 7 benchmarks, all of which are English code, math, QA and reasoning tasks. CMMLU itself is excluded, since it is the calibration domain here. A degradation is present, though small (Table 13). In practice it could be avoided, since calibration needs only ≈20,000 unlabeled tokens and about 10 minutes per 7B model.

H Comparison with standard threshold selection baselines

To justify the necessity of the proposed greedy bias assignment we provide the ablation against uniform scaling and quantile matching. Algorithm 1 dominates both on the Pareto front. At matched quality it is faster, and at matched speedup it is more accurate (Figure 7).

Table 12: Qwen2-7B-Instruct accuracy results at 25% average predicted sparsity.

Method	TriviaQA acc, %	ARC-C acc, %	ARC-E acc, %	HumanEval pass@1, %	MBPP pass@1, %	GSM8K acc, %	BBH acc, %	CMMLU acc, %	Avg. Score
Dense	60.96	86.43	93.81	60.98	44.46	36.85	47.81	79.58	63.86
SVDP	51.48	86.43	93.81	49.39	36.55	37.83	47.61	79.58	60.34



(a) Sensitivity of data whitening

(b) Sensitivity of bias calibration

Figure 6: Sensitivity analysis to calibration data size for first, middle and last layers of ProSparse-LLaMA2-7B. As few as 10,000 tokens are enough for reliable construction. Here, $r = 256$. (a) For data whitening, sensitivity is measured with ROC AUC scores. Here, zero calibration tokens corresponds to naive SVD without data whitening. (b) For bias calibration, sensitivity is measured with FFN relative error, averaged across test tokens. Here, $s = 0.5$ and we keep the number of calibration tokens for data whitening fixed.

Table 13: Sensitivity of the algorithm to calibration data distribution for ProSparse-LLaMA2-7B ($r = 256$, $s = 0.5$). Here, ✓ — the stage was calibrated on data close to the downstream tasks (a mix of train splits); × — the stage was calibrated on out-of-distribution data (CMMLU dev, Chinese).

Whitening	Bias	TriviaQA	ARC-E	ARC-C	HumanEval	MBPP	GSM8K	BBH	Average
×	×	61.76	75.21	53.67	15.85	21.05	10.61	35.46	39.09
✓	×	62.90	75.21	53.50	17.07	21.87	10.54	35.53	39.52
✓	✓	62.74	75.34	53.67	17.68	22.07	11.22	35.58	39.76

I Technical details

LLMs hyperparameter values We provide hyperparameter values for the tested models in Table 14, as they influence E2E latency. Specifically, all evaluated optimization techniques modify only the FFN portion of inference, leaving the Attention block unchanged. Thus, a higher intermediate size

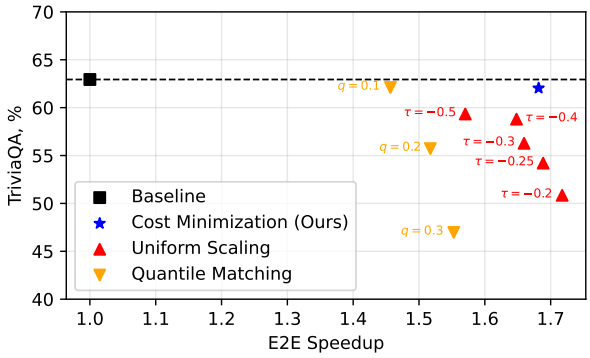


Figure 7: The ablation study against uniform scaling and quantile matching on TriviaQA benchmark for ProSparse-LLaMA2-7B. Uniform scaling applies one τ to every neuron, quantile matching sets each neuron's threshold at the q-quantile of its own calibration values, and Baseline is dense inference without prediction. Cost minimization is our Algorithm 1.

Algorithm 4 Building SVD-predictors (Offline Phase)

Input: $W^{\text{gate}} \in \mathbb{R}^{D \times d}$, $X \in \mathbb{R}^{d \times N}$, $r \in \mathbb{N}$
Output: $A \in \mathbb{R}^{D \times r}$, $B \in \mathbb{R}^{r \times d}$
$S \in \mathbb{R}^{d \times d} \leftarrow \text{Cholesky}\left(XX^\top\right)$
$U, \Sigma, V \leftarrow \text{SVD}\left(W^{\text{gate}} S\right)$
$A \in \mathbb{R}^{D \times r} \leftarrow U_{:,:r} \Sigma_{:r,:r}$
$B \in \mathbb{R}^{r \times d} \leftarrow V_{:r,:} S^{-1}$
return A, B

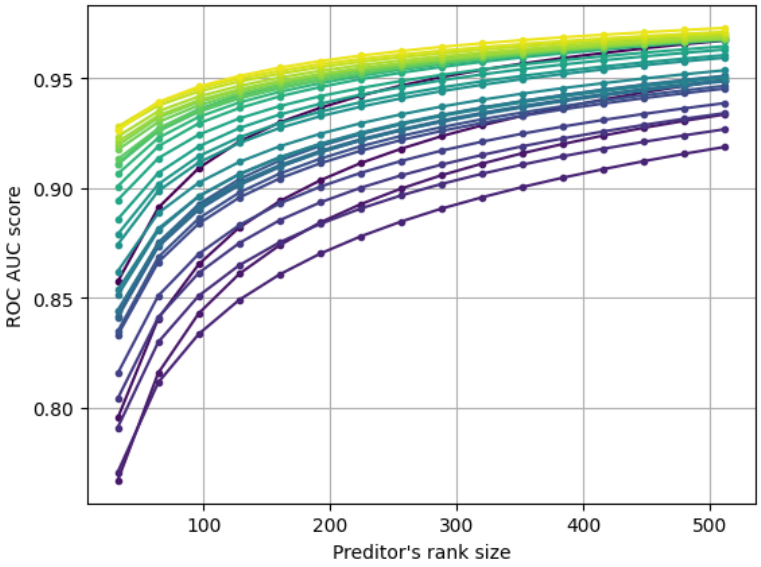


Figure 8: ROC AUC scores of SVD-predictors at different layers averaged across several sequence samples for ProSparse-LLaMA2-7B. The brighter curves represent deeper layers. For most of the layers a quality metric reaches a plateau at relatively small rank.

increases the FFN's contribution to E2E latency. Similarly, the use of Grouped Query Attention reduces the attention contribution to total inference time.

Table 14: LLMs hyperparameter values.

	hyperparameters			
Model	Hidden size	Intermediate size	N attention heads	N KV heads
ProSparse-LLaMA2-7B	4096	11008	32	32
TurboSparse-Mistral-Instruct	4096	14336	32	8
SparseQwen2-7B	3584	18944	28	4

Calibration data For calibration in total we use 112 few-shot samples (≈20,000 tokens) from the train splits of the GSM8K, ARC-E, ARC-C, OpenBookQA, QAsper, CodeXGLUE datasets. We split these tokens in half for independent calibration of data whitening and bias. Accuracies are measured on the benchmark test splits, the ROC-AUC curves on separately held-out sequences.

Predictor's rank One can trade off the predicted sparsity ratio and predictor size. While a larger predictor allows for more sparse computations within gate projection, it requires more compute for prediction. In turn, the s hyperparameter can be used to calibrate the predicted sparsity ratio. Figure 8 shows how the separation capability of SVD predictors depends on their size. Due to this insight, higher acceleration can be achieved with relatively small predictors. Moreover, small predictors

Table 15: Sparsity decomposition by source for the dReLU models on HumanEval (≈20,000 generated tokens, mean ± std). †Averaged across all layers.

Sparsity Source	TurboSparse-Mistral-Instruct				SparseQwen2-7B			
	First Layer	Middle Layer	Last Layer	Avg.†	First Layer	Middle Layer	Last Layer	Avg.†
gate-predicted	50 ± 5	51 ± 4	26 ± 6	55 ± 4	35 ± 7	43 ± 4	59 ± 11	59 ± 3
gate-side-correction	86 ± 4	80 ± 2	83 ± 4	85 ± 1	79 ± 3	70 ± 2	90 ± 2	83 ± 1
up-side	74 ± 4	55 ± 2	53 ± 2	60 ± 1	48 ± 1	35 ± 2	37 ± 2	37 ± 1
gate-up-union	94 ± 2	87 ± 2	90 ± 2	91 ± 1	88 ± 1	78 ± 2	93 ± 2	88 ± 1

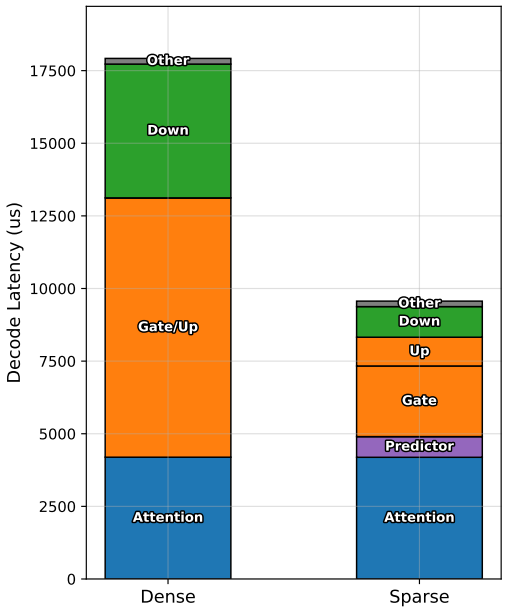


Figure 9: Decode latency breakdown obtained with the PyTorch profiler on the first decoded token of a sample sequence for TurboSparse-Mistral-Instruct. The "Other" block comprises normalizations, logits processing, and sampling. FFNs dominate end-to-end latency during single-batch decoding on the GPU. Even without asynchronous prediction, SVDP combined with the sequential pipeline achieves over $2.5\times$ FFN speedup. Note that the speedup depends on the token-dependent sparsity ratio.

consume less additional memory. Similarly to r, we set s to be constant across all layers. Empirically, we found that setting the rank to approximately 2% of the intermediate size D works well.

J Translating sparsity into end-to-end speedup

The executors do use the up-side zeros: the down-projection kernel selects active columns by the actual non-zeros of its input, not by the predicted gate mask (this applies to all baselines equally). The sparsity decomposition by source for the dReLU models is provided in Table 15. Overall, sparsity grows from the predictor output through the gate ReLU to the down projection, with the up-side ReLU adding the final 5–6%. The sparsity varies considerably across layers and differs between models. Table 4 benchmarks a single FFN block taken from the model, on synthetic inputs with the same sparsity at every stage. In real decoding, sparsity is non-uniform and rarely reaches 95% on all layers, so the $30\times$ point is not reached on the full decode path. Figure 9 presents the decode latency breakdown during inference for both the dense baseline and the SVDP-accelerated version. The E2E gain is bounded by the FFN share of the decode step.

K Licenses for existing assets

The licences and versions for used assets are provided in Table 16.

Table 16: Assets, licences, versions, and URLs used in this work.

Asset	Licence	Version	URL
UltraEval	Apache-2.0	from commit 5d967b4	github.com/OpenBMB/UltraEval
vLLM	Apache-2.0	0.8.5.post1	github.com/vllm-project/vllm
GSM8K	MIT	accessed 2026-01-15	huggingface.co/datasets/openai/gsm8k
HumanEval	MIT	accessed 2026-01-15	huggingface.co/datasets/openai/openai_humaneval
ARC-E/ARC-C	CC BY-SA 4.0	accessed 2026-01-15	huggingface.co/datasets/allenai/ai2_arc
TriviaQA	Apache-2.0	accessed 2026-01-15	huggingface.co/datasets/mandarjoshi/trivia_qa
MBPP	CC BY 4.0	accessed 2026-01-15	huggingface.co/datasets/Muennighoff/mbpp
BBH	MIT	accessed 2026-01-15	huggingface.co/datasets/lukaemon/bbh
CMMLU	CC BY-NC 4.0	accessed 2026-01-15	huggingface.co/datasets/lmlmcat/cmmlu

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We describe our scope and contributions in the abstract and Section 1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our method in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The theoretical results and a complete proofs are provided in Section A

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We describe the pipeline of our method in Section 4 and experiment details are presented in Section 5. We provide the complete codes as supplementary materials.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The complete codes are included.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Experiment details are given in Section 5, Section C, Section I. The complete codes are included.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report standard deviation of speedup values to capture low run-to-run variability across independent runs given experimental conditions. To ensure robustness, we evaluate our method across a range of LLM architectures, and downstream tasks, as detailed in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The information of the employed compute resources is described in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss potential positive societal impacts of the work in Section 6.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The released code has no risks of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We correctly cite all assets and adhere to the usage restrictions. Licenses and versions for used assets are listed in Section K.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The code and other supplementary materials are followed with readme and instructions.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We do not involve any crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We do not involve any crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.