COUNTDOWN: Contextually Sparse Activation Filtering Out Unnecessary Weights in Down Projection

Jaewon Cheon
Industrial and Management Engineering
Korea University
jaewon_cheon@korea.ac.kr

Pilsung Kang*
Industrial Engineering
Seoul National University
pilsung_kang@snu.ac.kr

Abstract

The growing size of large language models has created significant computational inefficiencies. To address this challenge, sparse activation methods selectively deactivate non-essential parameters during inference, reducing computational costs in Feed-Forward Networks (FFN) layers. While existing methods focus on non-linear gating mechanisms, we hypothesize that the sparsity lies globally in the form of a linear combination over its internal down projection matrix. Based on this insight, we propose two methods: M-CountDown, leveraging indirect coefficients, and D-CountDown, utilizing direct coefficients of the linear combination. Experimental results demonstrate that D-CountDown can omit 90% of computations with performance loss as low as 5.5% ideally, while M-CountDown provides a predictor-free solution with up to 29.4% better performance preservation compared to existing methods. Our specialized kernel implementations effectively realize these theoretical gains into substantial real-world acceleration.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse applications, from handling specific tasks to orchestrating agent-based operations (OpenAI et al., 2024; DeepSeek-AI et al., 2024; Gemma Team et al., 2025). However, these advancements came at the cost of dramatically increased model sizes, creating enormous computational and resource demands.

The inference process has emerged as a particularly acute efficiency constraint, forming a critical bottleneck for deploying LLMs in practical applications. This inefficiency is further amplified by recent trends in test-time scaling, where models generate extensive reasoning, significantly increasing computational demands during inference (Jang et al., 2024; Deng et al., 2024). Consequently, research on LLM inference efficiency has intensified, aiming to reduce latency and memory consumption while preserving generation quality (Liu et al., 2024; Kwon et al., 2023; Cai et al., 2024).

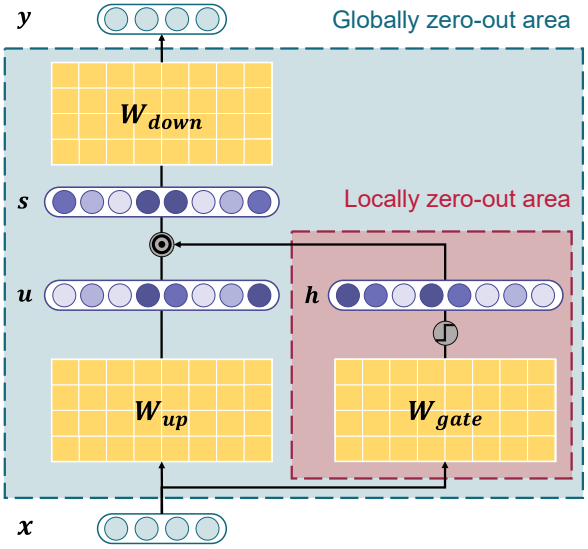

Figure 1: Comparison of sparsity determinations: our approach determines sparsity from the full FFN computation (turquoise box), whereas conventional methods like CATS (Lee et al., 2024) rely solely on non-linear activations (red box).

In this context, sparse activation has emerged as a prominent strategy to improve the efficiency of the Feed-Forward Networks (FFN) in LLM (Liu et al., 2023; Lee et al., 2024; Akhauri et al., 2024; Alizadeh et al., 2024). Sparse activation methods dynamically identify and deactivate parameters unnecessary for a given input, thereby reducing computational load and accelerating inference. These methods are particularly beneficial since FFN layers incur significant computational overhead in modern LLM architectures (Awasthi et al., 2024).

The zero-out gating property of ReLU (Agarap, 2019) creates extensive sparsity in FFN layers by forcing a large portion of neurons to output zero (Mirzadeh et al., 2024). This natural sparsity makes computations associated with these zero-valued neurons completely redundant. Existing sparse

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 28393–28409
November 4-9, 2025 ©2025 Association for Computational Linguistics

activation methods leverage this property to identify and skip these unnecessary computations (Sun et al., 2024; Zhang et al., 2024). However, recent LLMs largely employ activations such as GeLU or SiLU (Hendrycks and Gimpel, 2016; Elfwing et al., 2018) with far less prevalent zero-out behavior (Mirzadeh et al., 2024), limiting these methods' applicability. Further, Gated-MLP structures, now widely adopted as FFN layers (Shazeer, 2020; Dauphin et al., 2017), introduce more complex parameter interactions than standard architectures. This invalidates the assumption that sparsity occurs only around non-linear activations.

To overcome these limitations, we propose an approach that defines sparsity from a global view, extending beyond the non-linear activations by reformulating the FFN layer's output as a weighted sum, as illustrated in Figure 1. Based on this approach, we derive two sparse activation methodologies: MONO-COUNTDOWN (M-COUNTDOWN) and DUAL-COUNTDOWN (D-COUNTDOWN). M-COUNTDOWN identifies sparsity based on the output of a single weight matrix in Gated-MLP, while D-COUNTDOWN leverages two weight matrices. In evaluations, M-COUNTDOWN consistently outperforms the baseline method CATS (Lee et al., 2024), achieving up to 29.4% better performance preservation with comparable inference speed. D-COUNTDOWN attains greater efficiency gains, reducing computations by up to 90% in FFN layers with performance loss as low as 5.5% under optimal conditions.

The contributions of this paper are as follows.

- We introduce a novel theoretical framework that redefines sparsity through a weighted-sum perspective over down projection matrices, extending beyond the conventional focus on activation functions.
- We demonstrate that analyzing coefficient vectors in the weighted sum enables superior sparsity decisions, resulting in two distinct approaches with complementary strengths.
- We provide practical acceleration through optimized kernel implementations, enabling both methods to achieve substantial throughput improvements across multiple state-of-the-art LLM architectures.

2 Related Works

ReLU-based Sparse Activation Early works on sparse activation primarily leveraged the property of ReLU to enhance computational efficiency. These approaches identified that ReLU activation functions naturally create substantial built-in sparsity by producing zeros for negative values (Li et al., 2023b). Several approaches have tried to detect these zero-valued activations to preemptively skip associated computations, as these neurons would have no impact on subsequent layers (Mirzadeh et al., 2024). Deja Vu (Liu et al., 2023) extended this concept by training lightweight predictors to anticipate which neurons would be zeroed out, further improving efficiency. While these methods showed impressive speed gains with minimal performance loss, their application faced significant constraints. Notably, these approaches were practical only on architectures explicitly designed with ReLU activations, limiting their applicability as LLMs increasingly adopted alternative activation functions (Akhauri et al., 2024).

Non-ReLU Sparse Activation As LLMs evolved to favor non-ReLU activation functions such as GeLU and SiLU, which rarely produce exact zeros, new methods emerged to extend sparsity benefits to these architectures. One direction involved ReLUfication techniques that replace non-ReLU functions with ReLU, enabling the reuse of existing sparsity strategies (Song et al., 2024b, 2025; Zhang et al., 2024; Alizadeh et al., 2024). Another approach, such as by CATS (Lee et al., 2024), redesigned sparsity criteria to identify and skip computations associated with near-zero activations rather than exact zeros. While these adaptations improved compatibility with modern LLM architectures, they remain fundamentally constrained by their narrow focus on local patterns around non-linear transformations, overlooking potential sparsity from a global perspective of the FFN layer. This localized perspective may fail to fully capitalize on the potential sparsity distributed throughout modern Gated-MLP architectures, particularly considering the complex interactions among multiple weight matrices that define these structures.

3 Generalization of Sparse Activation

Problem Formulation A Gated-MLP block consists of three weight matrices: $W_{up}, W_{gate}, W_{down} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{inter}}}$. For this block, the input vector x and the output vector y are in $\mathbb{R}^{d_{\text{model}}}$. The computation involves intermediate states defined as $u = x \cdot W_{up}$, $h = \sigma(x \cdot W_{gate})$, $s = u \odot h$ in $\mathbb{R}^{d_{\text{inter}}}$.

When no sparsification is applied, which we refer to as the `Dense` scenario, all parameters are activated, and the operation proceeds as follows:

$$y = \big((x \cdot W_{up}) \odot \sigma(x \cdot W_{gate})\big) \cdot W_{down}^{\intercal} \quad (1)$$

where σ denotes a non-linear activation function, typically GeLU or SiLU.

We now introduce our sparsity propagation framework, establishing sparse activation from a global perspective. We can activate only a valuable subset of weight vectors, with a marginal performance loss. Specifically, sparse activation under our framework follows:

$$y = \big((x \cdot W_{up}^{I}) \odot \sigma(x \cdot W_{gate}^{I})\big) \cdot \big(W_{down}^{I}\big)^{\intercal} \quad (2)$$

where I denotes the column of indices of the weights selected for computation:

$$W^{I} = W[:, IDX], \quad IDX = THLD(\cdot) \quad (3)$$

where $THLD$ is any function filtering effective I.

Notably, when individual threshold functions are defined separately for each matrix, identical output can be achieved through the unified intersection IDX:

$$IDX = IDX_{up} \cap IDX_{gate} \cap IDX_{down} \quad (4)$$

Consequently, even when sparsifying just one matrix and keeping others dense, the computation remains equivalent to applying this unified IDX across all matrices, which we denotes as shared-index property. Thus, if valuable sparsity patterns are identified in one matrix, they can propagate throughout the entire Gated-MLP.

A critical challenge, therefore, is defining the optimal filtering function $THLD$ to identify the most effective index set IDX to preserve globally essential computations while significantly reduce computational overhead.

Limitation of Comparative Methodology CATS (Lee et al., 2024) partially satisfies our sparsity propagation framework. It identifies sparsity by examining the activation magnitude $h = \sigma(x \cdot W_{gate})$, assuming activations squashed near zero indicate parameters to omit. Specifically, given a sparsity ratio $k \in (0, 1)$, CATS computes a threshold τ_{C}^{k} via the Quantile$(k, |h|)$ operation, selecting a cutoff below which the lowest k fraction of activations is excluded. Based on this threshold, CATS defines a sparse activation index as shown in Equation 6a.

CATS leverages the shared-index property. However, since the optimal $THLD$ might depend on factors beyond non-linear activation region, CATS is theoretically limited in propagating an optimal IDX throughout the Gated-MLP. Additionally, although $h[i]$ is large, if the corresponding $u[i] = x \cdot W_{up}[i]$ is near zero, the final contributions become minimal, which ideally should be filtered out due to their elementwise product.

Threshold Variants To overcome these limitations, we reformulate the Gated-MLP computation as a linear combination of the W_{down} weight vectors, thereby exploring additional possibilities for defining $THLD$ as follows:

$$\begin{aligned} y &= \big((x \cdot W_{up}) \odot \sigma(x \cdot W_{gate})\big) \cdot W_{down}^{\intercal} \\ &= \sum_{i} s[i] \cdot W_{down}^{\intercal}[i] \end{aligned} \quad (5)$$

This reformulation allows us to interpret output y as a weighted sum over $W_{down}^{\intercal}$ row vectors, where coefficient $s[i] = \big((x \cdot W_{up}) \odot \sigma(x \cdot W_{gate})\big)[i]$ reflects the i-th row vector's contribution to computation. The magnitude of these coefficients provides a natural metric for determining which parameters to activate, as they quantify each vector's significance to the output.

Furthermore, since s is calculated as the elementwise multiplication of $u = x \cdot W_{up}$ and $h = \sigma(x \cdot W_{gate})$, these intermediate vectors can also serve as indirect coefficient signals. This generalized view reveals that each computation stage in the Gated-MLP can provide a distinct sparsity indicator, with selecting h as the basis being equivalent to CATS's approach.

$$THLD_{\text{C}}^{k}(h, \tau_{\text{C}}^{k}) = \{i \mid |h[i]| > \tau_{\text{C}}^{k}\} \quad (6a)$$

$$THLD_{\text{M}}^{k}(u, \tau_{\text{M}}^{k}) = \{i \mid |u[i]| > \tau_{\text{M}}^{k}\} \quad (6b)$$

$$THLD_{\text{D}}^{k}(s, \tau_{\text{D}}^{k}) = \{i \mid |s[i]| > \tau_{\text{D}}^{k}\} \quad (6c)$$

Based on this view, we propose two variants of sparse activation that extend beyond prior approaches relying solely on the magnitude of h. As shown in Equation 6, where subscripts C, M, and D denote CATS, M-CountDown, and D-CountDown methods respectively, the first method, M-CountDown, applies thresholding directly to vector u, while the second method, D-CountDown, applies thresholding to s. For each method, thresholds τ_{M}^{k} and τ_{D}^{k} are calculated via Quantile$(k, |u|)$ and Quantile$(k, |s|)$ respectively.

These methods offer complementary strengths: M-CountDown provides practical implementation with minimal overhead by examining only one matrix multiplication, while D-CountDown can offer more precise sparsity identification through direct coefficients of the weighted sum. A detailed discussion of these methods follows in section 4

4 Implementation of Sparse Activation

$\mathsf{SP}^{\mathtt{Ideal}}$ and $\mathsf{SP}^{\mathtt{Prac}}$ In the previous section, we focused on establishing $THLD$ and the corresponding indicator that theoretically guarantee the safe omission of parameters. Ideally, if these indicators are tractable in real-time inference, we can achieve the upper-bound performance defined by the method. However, accessing the indicator and deriving IDX from it is not trivial.

Given this constraint, we distinguish between two distinctive perspectives: $\mathsf{SP}^{\mathtt{Ideal}}$ examines the theoretical upperbound performance achievable by each method, assuming that filtering based on sparsity indicators incurs no computational overhead. $\mathsf{SP}^{\mathtt{Prac}}$ accounts for real-world deployment constraints, particularly the latency of identifying sparse activation patterns. It evaluates whether methods can deliver actual inference speedups when all practical overheads are considered.

The distinction is critical because methods with strong $\mathsf{SP}^{\mathtt{Ideal}}$ performance may not translate to $\mathsf{SP}^{\mathtt{Prac}}$ benefits if their practical implementation is computationally expensive. Conversely, focusing solely on $\mathsf{SP}^{\mathtt{Prac}}$ without understanding the theoretical $\mathsf{SP}^{\mathtt{Ideal}}$ limits can lead to suboptimal solutions that fail to approach the best possible performance. Effective sparse activation requires both identifying truly essential computations via $\mathsf{SP}^{\mathtt{Ideal}}$ and creating an efficient implementation to realize total computational savings through $\mathsf{SP}^{\mathtt{Prac}}$.

Constructing $\mathsf{SP}^{\mathtt{Prac}}$ for CountDown We now describe how to transform the theoretical $\mathsf{SP}^{\mathtt{Ideal}}$ formulations of M-CountDown and D-CountDown into efficient, practical $\mathsf{SP}^{\mathtt{Prac}}$ implementations.

For M-CountDown, the implementation is straightforward because its indicator u depends only on the matrix W_{up}. Therefore, its index set IDX^k_{M} defined in its $\mathsf{SP}^{\mathtt{Ideal}}$ perspective can be obtained independently of other matrices in the Gated-MLP. This allows M-CountDown to operate without additional inference-time components, as computation over the remaining matrices can be selectively skipped based on u.

To further reduce overhead, we avoid computing τ^k_{M} dynamically for each input. Instead, we approximate it with a layerwise constant $\hat{\tau}^k_{\mathrm{M}} = \frac{1}{T}\sum_{t=1}^{T} \mathrm{Quantile}(k, |u^{(t)}|)$ estimated during a calibration phase with T sampled inputs.

In contrast, implementing D-CountDown poses greater challenges because its indicator s requires nearly the entire Gated-MLP computation, negating the advantages of sparse activation. To tackle this challenge, we train a lightweight predictor that estimates the optimal index set IDX^k_{D} directly from input x, avoiding the need to compute s during inference. For each layer, the predictor outputs a score vector $\hat{s}$ where:

$$\hat{s}[i] = \begin{cases} +\infty & \text{if } |s[i]| > \mathrm{Quantile}(k, |s|) \\ -\infty & \text{otherwise} \end{cases}$$

Using this output, we define the predicted index set as $\widehat{IDX}^k_{\mathrm{D}} = \{i \mid \hat{s}[i] > 0\}$ and activate only the corresponding weight columns during inference.

For efficiency, the predictor must be highly accurate and computationally inexpensive during inference. Following (Liu et al., 2023; Alizadeh et al., 2024), we employ a low-rank approximator consisting of two matrices: $\theta_{\mathrm{A}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{rank}}}$ and $\theta_{\mathrm{B}} \in \mathbb{R}^{d_{\text{rank}} \times d_{\text{inter}}}$, minimizing computational overhead while preserving prediction accuracy. algorithm 1 details the complete training procedure.

Hardware Aware Kernel Design Once the sparse activation index set IDX is determined, computation can be restricted to only the corresponding subset of weights, reducing the actual floating-point operation count (FLOPs). However, reducing FLOPs does not necessarily translate to improved inference latency. For instance, materializing an indexed weight matrix and performing standard vector-matrix multiplication may still reduce FLOPs, but at the cost of increased memory access (Song et al., 2024a; Xue et al., 2024). Therefore, sparse computation should avoid incurring excessive memory traffic solely for the sake of reducing arithmetic operations.

To address this, we implement custom kernels for both M-CountDown and D-CountDown using Triton (Tillet et al., 2019). The M-CountDown kernel builds upon CATS's structure (Lee et al., 2024), but optimizes it by fusing the non-linear activation to reduce additional memory access. For D-CountDown, we design a kernel

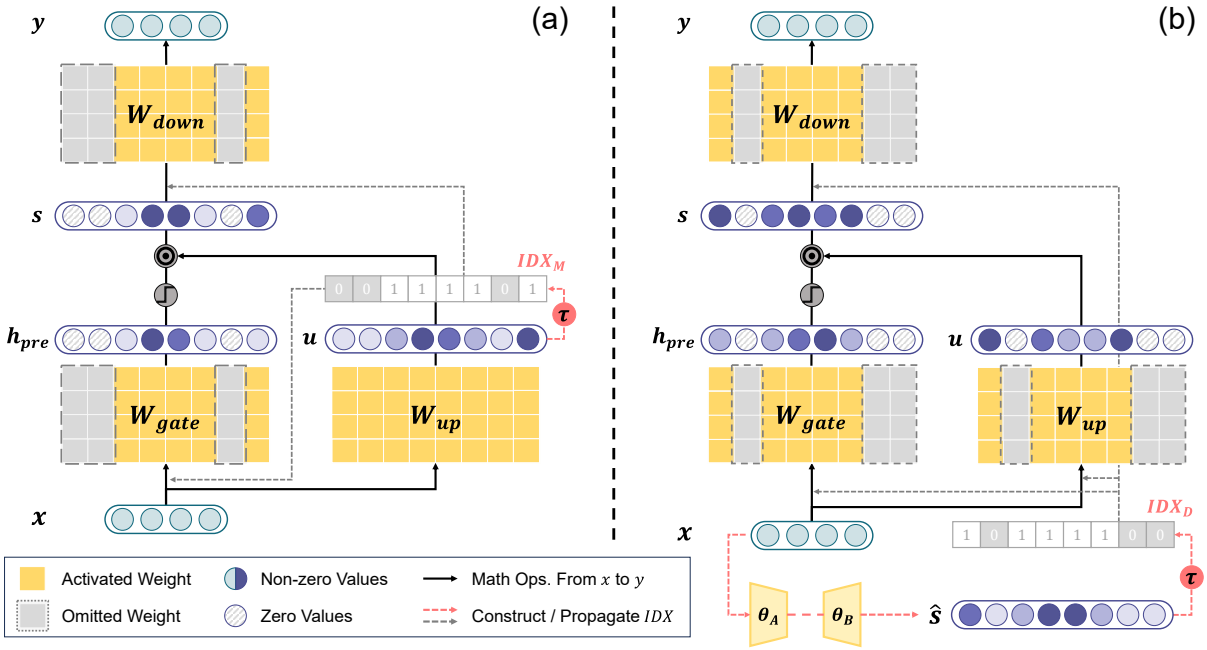

Figure 2: CountDown Pipeline. Note that $h_{pre} = x \cdot W_{gate}$. *Left (a):* In M-CountDown, we determine which parameters to activate by binarizing densely computed u with pre-calculated $\hat{\tau}^{k}_{M}$. *Right (b):* In D-CountDown, low-rank predictors (θ_A, θ_B) determine which parameters to activate.

that efficiently supports predictor-based activation patterns. A naive implementation would require eight separate kernel launches for sparse computation: indexing and GEMV for each of the three matrices, plus non-linear activation and elementwise multiplication. Our implementation compresses this workload into just two kernels. This design ensures that FLOPs reductions directly translate into improved token throughput. Full implementation details and pseudocode are in algorithm 2 and algorithm 3.

5 Experiments

Experimental Setup We evaluate the proposed methods against other sparse activation baselines, primarily CATS (Lee et al., 2024), CHESS () and Deja Vu (Liu et al., 2023). We also include a `Dense` variant without any sparse activation for comparison. Experiments are conducted using four diverse state-of-the-art LLMs ranging from 8B to 14B parameters: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), gemma-2-9b-it (Gemma Team et al., 2024), Qwen2.5-14B-Instruct (Qwen et al., 2024), and phi-4 (Abdin et al., 2024). We test multiple sparsity ratios by varying k from 0.7 to 0.9, representing the fraction of parameters excluded from computation. Implementation details are provided in Appendix A.2.

We examine both model performance preservation and computational efficiency. For model performance, we use the lm-eval-harness (Gao et al., 2024) framework to assess downstream tasks including ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), OpenbookQA (Mihaylov et al., 2018), TruthfulQA (Lin et al., 2022), WinoGrande (Sakaguchi et al., 2020), and GSM8K (Cobbe et al., 2021). Unlike prior sparse activation studies, we also evaluate conversational ability using LLM-as-a-Judge framework AlpacaEval 2.0 (Li et al., 2023a).

To assess computational efficiency and inference speed, we benchmark kernel-level latency to quantify Gated-MLP speedups from sparse activation. We also measure end-to-end token throughput and analyze theoretical reductions in floating-point operations (FLOPs) and memory traffic.

Downstream Task Performance As shown in Table 1, in the $\texttt{SP}^{\texttt{Ideal}}$ setting, D-CountDown consistently outperforms all methods across all models and sparsity ratios, exhibiting negligible degradation even when compared to the dense baseline. This demonstrates the effectiveness of D-CountDown's sparsity criterion: the indicator s accurately reflects each parameter's importance to the final output, serving as the coefficient in our linear combination formulation. This provides more informed filtering than methods like CATS which rely solely on gating magnitude. Even at

InferenceMode	Llama-3.1-8B-Instruct			gemma-2-9b-it			Qwen2.5-14B-Instruct			phi-4		
	$k=0.7$	$k=0.8$	$k=0.9$	$k=0.7$	$k=0.8$	$k=0.9$	$k=0.7$	$k=0.8$	$k=0.9$	$k=0.7$	$k=0.8$	$k=0.9$
Dense												
Full		**0.616**			**0.645**			**0.674**			**0.655**	
SP$^{\text{Ideal}}$												
DEJAVU	0.314	0.315	0.322	0.360	0.360	0.360	0.379	0.382	0.385	0.398	0.405	0.396
CATS	0.471	0.412	0.337	0.592	0.483	0.367	0.502	0.428	0.389	0.615	0.535	0.427
M-CountDown	0.570	0.513	0.421	0.624	0.607	0.549	0.644	0.610	0.479	0.636	0.608	0.512
D-CountDown	**0.603**	**0.587**	**0.525**	**0.635**	**0.625**	**0.590**	**0.660**	**0.647**	**0.555**	**0.651**	**0.649**	**0.620**
SP$^{\text{Prac}}$												
CATS	0.504	0.450	0.350	0.605	0.502	0.360	0.556	0.478	0.390	0.633	0.591	0.448
M-CountDown	**0.571**	**0.528**	**0.447**	**0.632**	**0.617**	**0.588**	**0.651**	**0.624**	**0.535**	**0.639**	**0.620**	**0.555**
D-CountDown	0.442	0.419	0.387	0.555	0.563	0.520	0.526	0.457	0.437	0.499	0.445	0.417

Table 1: Average SP$^{\text{Ideal}}$ and SP$^{\text{Prac}}$ scores compared to Dense across all downstream tasks. Full task-wise results are provided in Appendix C.

90% sparsity, D-CountDown retains only the most impactful neurons, limiting performance drop to 5.5% in the best case among evaluated models.

M-CountDown, although less effective than D-CountDown, consistently outperforms CATS. The gap between the two widens as the sparsity ratio increases, reaching over 29.4%. This demonstrates that M-CountDown 's indicator u is more predictive of useful computation than CATS' indicator h. This may seem counterintuitive since u and h contribute symmetrically via their element-wise product and thus should be equally informative. We revisit this comparison in section 6.

Deja Vu, which assumes ReLU-style zero-out behavior, suffers severe degradation in the SP$^{\text{Ideal}}$ setting. Given its reliance on predictors, which would further degrade under the SP$^{\text{Prac}}$ setting, we excluded it from subsequent experiments.

In the SP$^{\text{Prac}}$ setting, D-CountDown experiences performance loss relative to the SP$^{\text{Ideal}}$ due to predictor sub-optimality, suggesting better prediction strategies are needed to fully realize its potential in deployment. In contrast, M-CountDown, thanks to its predictor-free design, exhibits nearly identical performance to its SP$^{\text{Ideal}}$ counterpart. Notably, M-CountDown continues to outperform CATS across all sparsity settings, reinforcing the effectiveness of its signal even under realistic constraints.

LLM Chat Performance While prior studies rely on downstream task accuracy or perplexity, these metrics often fail to capture conversational performance. To address this, we evaluate each method using an LLM-as-a-Judge framework that directly assesses chat-level performance.

As shown in Table 2, M-CountDown maintains nearly identical performance between the SP$^{\text{Ideal}}$ and SP$^{\text{Prac}}$ settings, while also outperforming CATS in both. D-CountDown exhibits noticeable degradation in SP$^{\text{Prac}}$ due to predictor limitations, but retains a dominant lead under SP$^{\text{Ideal}}$. This trend aligns with the results observed in the downstream task evaluations.

InferenceMode	AlpacaEval 2.0		
	$k = 0.7$	$k = 0.8$	$k = 0.9$
SP$^{\text{Ideal}}$			
CATS	25.10	1.72	0.19
M-CountDown	45.84	29.22	3.90
D-CountDown	**48.86**	**45.85**	**29.95**
SP$^{\text{Prac}}$			
CATS	31.63	10.47	0.25
M-CountDown	**38.31**	**33.80**	**15.88**
D-CountDown	3.40	2.81	1.16

Table 2: Average SP$^{\text{Ideal}}$ and SP$^{\text{Prac}}$ win rates against Dense across all models. Full model-wise results are provided in Table 7.

Efficiency and Speed To confirm that reductions in computation indeed translate into inference speedups, we measured kernel-level execution latency under various sparsity ratios. Each kernel's execution time was recorded from the start of the Gated-MLP computation, explicitly excluding other operations like token embedding or attention mechanisms. This allowed us to isolate the precise efficiency gains attributable to sparse activation.

As shown in Figure 3, D-CountDown achieves the fastest kernel execution time overall, despite the presence of a predictor, by skipping all three weight matrix computations. Although both M-CountDown and CATS are predictor-free, M-CountDown slightly outperforms CATS in kernel speed. Given that the only architectural difference between their kernels is whether the non-

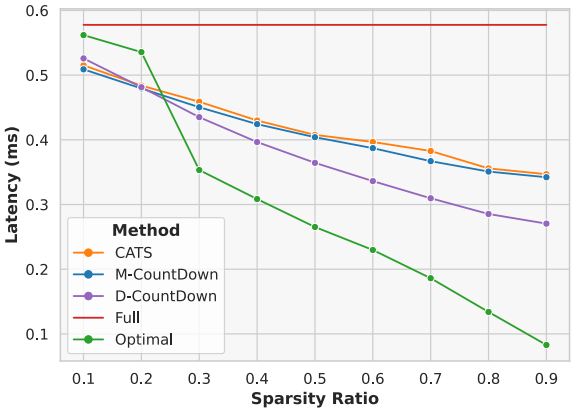

Figure 3: Kernel Speed for Llama-3.1-8B-Instruct. CATS, M-CountDown and D-CountDown show their respective SP$^{\text{Prac}}$ kernel speeds, Full and Optimal show Dense while $int(d_{inter} \times k)$ instead of d_{inter} for the Optimal. Results for other models are in Figure 5.

k	Method	FLOPs(M)	Mem.(MB)	Throughput	
				512	1024
0.0	Dense	352.41	168.121	24.64	22.63
0.7	CATS	188.00	89.746	32.62	29.40
	MC	187.95	89.719	33.61	30.32
	DC	**124.59**	**59.480**	**30.69**	**27.80**
0.8	CATS	164.52	78.550	32.72	29.60
	MC	164.46	78.522	33.80	30.61
	DC	**89.37**	**42.684**	**30.70**	**27.57**
0.9	CATS	141.02	67.345	32.98	29.81
	MC	140.96	67.318	33.51	30.78
	DC	**54.11**	**25.877**	**30.73**	**27.55**

Table 3: Theoretical FLOPs and Memory Traffic of Gated-MLP and actual throughput per second at sequence lengths 512 and 1024 for Llama-3.1-8B-Instruct ($d_{model} = 4096$, $d_{inter} = 14336$). MC and DC refer to M-CountDown and D-CountDown respectively.

linear activation function is fused, this suggests that M-CountDown gains a minor but consistent speed advantage by fusing the activation computation, thereby reducing memory traffic and avoiding additional overhead.

Furthermore, we measured average tokens generated per second for generation lengths of 512 and 1024, providing a model-level speedup assessment in typical generation scenarios. As shown in Table 3, M-CountDown achieves the highest end-to-end token throughput. Meanwhile, D-CountDown demonstrates the best performance at the kernel level, and with further optimization, its overall throughput may be further enhanced.

6 Analysis

M-CountDown vs CATS While CATS and M-CountDown share similar core ideas for sparse activation, our experimental results show that M-CountDown consistently achieves better performance. To understand the performance gap between the indirect coefficient vectors u and h, we conduct a comparative analysis of how each influences and aligns with the oracle-like reference signal s, the direct coefficient used in D-CountDown.

To enable direct comparison, we define binary masks S^k, U^k, and H^k based on the top-k magnitude entries of each vector. Each mask marks components as "alive" (1) if they survive quantile thresholding, and "dead" (0) otherwise. These binary masks are equivalent to the index sets IDX^k used for sparse activation, as each represents the support of the corresponding IDX^k in vector form.

We first define a metric called **Comparative Influential Factor (CIF)** to measure how much influence u (or h) has on the final decision of s, especially in cases where it overrides the other component. Analogously, for instance, $\text{CIF}^k(u, \text{alive})$ measures how often u "rescues" a component that would otherwise have been pruned by h, allowing it to survive in s due to its strong contribution. Formally, this is computed as:

$$\text{CIF}^k(u, \text{alive}) = \frac{|S^k \wedge \neg H^k|}{|S^k|} \tag{7}$$

This formulation follows from the definition of s as the elementwise product of u and h. When $s[i]$ is alive but $h[i]$ is small enough to be pruned, it implies that $u[i]$ must have been large enough to compensate, effectively "saving" that entry.

Next, we define the **Comparative Agreement Factor (CAF)** to evaluate how often one signal aligns with s while the other disagrees. For instance, $\text{CAF}^k(u, \text{alive})$ measures how frequently u agrees with s on keeping a component, specifically when h disagrees. This is given by:

$$\text{CAF}^k(u, \text{alive}) = \frac{|S^k \wedge \neg H^k \wedge U^k|}{|S^k|} \tag{8}$$

Both CIF and CAF can also be defined symmetrically for the "dead" case by inverting the roles of activation and pruning.

As shown in Figure 4, u outperforms h across all sparsity levels in both CIF and CAF. These results suggest that u more closely reflects the true activation behavior captured by s and exerts a greater direct impact on sparsity decisions than h. In other

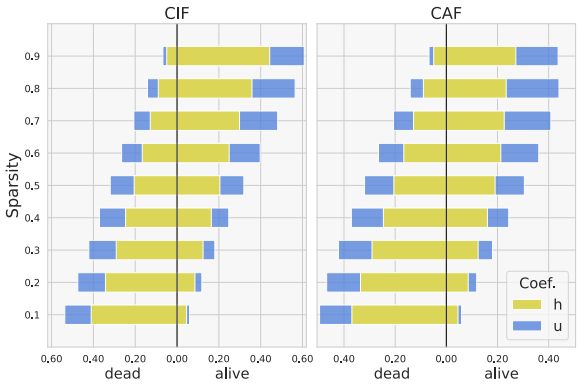

Figure 4: Tornado plots of CIF and CAF across whitening ratios. Bars to the right indicate the proportion of $\text{CIF}^k(\cdot, alive)$, while those to the left indicate $\text{CIF}^k(\cdot, dead)$.

words, u is more effective at preserving important activations and filtering out unimportant ones, explaining M-CountDown's stable and reliable performance under sparsity.

Nevertheless, since M-CountDown still relies on an indirect coefficient u, it cannot fully match the upper-bound performance of D-CountDown, which uses the full signal s directly. Despite u's strong CIF and CAF scores, substantial mismatches with respect to the oracle mask S^k remain, with peak CIF values reaching only about 0.6 and CAF values about 0.4, underscoring the need for future work to translate D-CountDown's upper-bound potential into SP^{Prac} deployments.

Possible Predictor Candidate: TernaryLinear D-CountDown demonstrates a theoretically sound and effective sparse activation strategy, achieving strong performance in the SP^{Ideal} setting. However, in the SP^{Prac} scenario, performance degradation occurs due to the predictor's limited accuracy in recovering optimal sparsity patterns. This reflects the difficulty of the prediction task rather than a flaw in the sparsity criterion itself. The task simultaneously demands precision and computational efficiency, presenting a significant challenge with considerable room for improvement.

To empirically explore this potential, we evaluate an alternative predictor architecture, **TernaryLinear**, whose weights are quantized as $\theta_{\text{ternary}} \in \{-1, 0, +1\}^{d_{\text{model}} \times d_{\text{inter}}}$. We compare its performance with the previously utilized low-rank approximator. TernaryLinear achieves significant parameter compression by sacrificing numerical precision while preserving the matrix rank structure. Motivated by recent studies demonstrating successful LLM pretraining with ternary quantization while retaining strong model performance (Ma et al., 2024), we regard TernaryLinear as a promising candidate due to its demonstrated expressiveness even under aggressive quantization.

Metric	TernaryLinear	Low-Rank
Latency (ms)	0.082	**0.030**
Theoretical footprint (MiB)	**112**	144
F1-score (%)	**0.435**	0.403

Table 4: Comparison between TernaryLinear and the Low-Rank Approximator. Latency for TernaryLinear was measured using the BitBLAS library (Wang et al., 2024). F1 score is reported as the average binary classification performance on $S^{0.7}$ across all evaluated models.

As shown in Table 4, TernaryLinear outperforms the low-rank baseline in F1 score, while also being more compact in terms of memory footprint. This suggests that preserving rank information, even at the cost of numerical precision, is more effective for sparse mask recovery than the reverse approach.

However, TernaryLinear has not yet been adopted due to its relatively slower runtime despite its small size. This limitation stems not from algorithmic complexity, but rather the lack of optimized GPU kernel support for ultra-low-precision operations. Prior work (Ma et al., 2025) suggests that future advances in kernel optimization and ultra-low-bit quantization are needed to fully leverage such architectures. With these improvements, techniques like TernaryLinear could become viable candidates for enabling D-CountDown to achieve its full SP^{Ideal} performance in SP^{Prac} scenarios.

7 Conclusion

We introduce CountDown, a novel sparse activation framework for improving inference efficiency of large language models. To overcome the limitations of traditional non-linear activation-based sparsity, we reformulate the computation as a weighted sum over the FFN's down projection matrix, effectively capturing inherent sparsity in modern Gated-MLP architectures. From this perspective, we present two complementary strategies: M-CountDown, which uses u derived from a single matrix W_{up} as its activation indicator, achieves faster inference and better performance preservation than prior state-of-the-art methods while remaining predictor-free. D-CountDown directly leverages s, the coefficient vector of the weighted sum, for fine-grained sparsity selection, demonstrating robust performance even when skipping 90% of computations under ideal conditions.

Limitations

Like most prior work on sparse activation, our study assumes a single-batch greedy decoding setting in on-device environments. While this scenario is realistic for latency-sensitive edge inference, it may be less applicable in multi-batch or server-based deployments. In such cases, strategies such as computing the union of predicted index sets IDX across multiple samples could be explored. However, such an approach would require further investigation into how much parameter activation can be shared across inputs, a direction we leave for future work.

Additionally, our sparsity criteria rely exclusively on activation magnitude. This choice offers clear interpretability and aligns well with the weighted-sum perspective we adopt. Nevertheless, alternative sparsity metrics, such as those explored by (Akhauri et al., 2024), remain an open research avenue. Expanding beyond simple magnitude-based thresholding could further enhance the performance of sparse activation methods.

Ethical Considerations

We affirm adherence to the ACL Rolling Review (ARR) ethical guidelines, explicitly addressing potential risks and responsible research practices. This research focuses on optimizing computational efficiency in large language models (LLMs), aimed at reducing resource usage and consequently lowering environmental impact. We foresee no direct risks or potential harms to individuals or communities resulting from this work.

Comprehensive details regarding the ethical use of scientific artifacts, reproducibility of computational experiments, and related considerations are thoroughly documented in Appendix A.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(No. 2340012238). This work was also supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-02214591, Development of an Innovative AI Agent for Worker-Friendly Autonomous Manufacturing), (RS-2024-00460011, Climate and Environmental Data Platform for Enhancing Climate Technology Capabilities in the Anthropocene (CEDP)), and (RS2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)). This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2024-00407803, RS-2025-23523657). This work was also supported by the BK21 FOUR Program (Education and Research Center for Industrial Innovation Analytics) funded by the Ministry of Education, Korea (No. 4120240214912).

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, and 8 others. 2024. Phi-4 technical report. *Preprint*, arXiv:2412.08905.

Abien Fred Agarap. 2019. Deep learning using rectified linear units (relu). *Preprint*, arXiv:1803.08375.

Yash Akhauri, Ahmed F AbouElhamayed, Jordan Dotzel, Zhiru Zhang, Alexander M Rush, Safeen Huda, and Mohamed S Abdelfattah. 2024. ShadowLLM: Predictor-based contextual sparsity for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 19154–19167, Miami, Florida, USA. Association for Computational Linguistics.

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, S Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. LLM in a flash: Efficient large language model inference with limited memory. In *ACL*.

Pranjal Awasthi, Nishanth Dikkala, Pritish Kamath, and Raghu Meka. 2024. Learning neural networks with sparse activations. In *The Thirty Seventh Annual Conference on Learning Theory, June 30 - July 3, 2023, Edmonton, Canada*, volume 247 of *Proceedings of Machine Learning Research*, pages 406–425. PMLR.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Forty-first International Conference on Machine Learning, ICML*

2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803. 05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and Others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110. 14168*.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 933–941. PMLR.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2024. DeepSeek-V3 technical report. *arXiv [cs.CL]*.

Yuntian Deng, Yejin Choi, and Stuart M Shieber. 2024. From explicit CoT to implicit CoT: Learning to internalize CoT step by step. *CoRR*, abs/2405.14838.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. A framework for few-shot language model evaluation.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *arXiv [cs.CL]*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, and 179 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv [cs.CL]*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *arXiv [cs.AI]*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELUs). *arXiv [cs.LG]*.

Joonwon Jang, Jaehee Kim, Wonbin Kweon, and Hwanjo Yu. 2024. Verbosity-aware rationale reduction: Effective reduction of redundant rationale via principled criteria. *arXiv preprint arXiv:2412. 21006*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

Donghyun Lee, Jaeyong Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. 2024. CATS: Context-aware thresholding for sparsity in large language models. In *First Conference on Language Modeling*.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023a. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J. Reddi, Ke Ye, Felix Chern, Felix X. Yu, Ruiqi Guo, and Sanjiv Kumar. 2023b. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3214–3252. Association for Computational Linguistics.

Jiahao Liu, Qifan Wang, Jingang Wang, and Xunliang Cai. 2024. Speculative decoding via early-exiting for faster LLM inference with thompson sampling control mechanism. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 3027–3043. Association for Computational Linguistics.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. 2023. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR.

Shuming Ma, Hongyu Wang, Shaohan Huang, Xingxing Zhang, Ying Hu, Ting Song, Yan Xia, and Furu Wei. 2025. Bitnet b1.58 2b4t technical report. *Preprint*, arXiv:2504.12285.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. 2024. The era of 1-bit LLMs: All large language models are in 1.58 bits.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2381–2391. Association for Computational Linguistics.

Seyed-Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. ReLU strikes back: Exploiting activation sparsity in large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

OpenAI, Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, A J Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, and 400 others. 2024. GPT-4o system card. *arXiv [cs.CL]*.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, and 24 others. 2024. Qwen2.5 technical report. *arXiv [cs.CL]*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. WinoGrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press.

Noam Shazeer. 2020. GLU variants improve transformer. *arXiv [cs.LG]*.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. 2025. ProSparse: Introducing and enhancing intrinsic activation sparsity within large language models. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 2626–2644. Association for Computational Linguistics.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024a. Powerinfer: Fast large language model serving with a consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP 2024, Austin, TX, USA, November 4-6, 2024*, pages 590–606. ACM.

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

Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. 2024. Massive activations in large language models. In *First Conference on Language Modeling*.

Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. 2019. 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@PLDI 2019, Phoenix, AZ, USA, June 22, 2019*, pages 10–19. ACM.

Lei Wang, Lingxiao Ma, Shijie Cao, Quanlu Zhang, Jilong Xue, Yining Shi, Ningxin Zheng, Ziming Miao, Fan Yang, Ting Cao, Yuqing Yang, and Mao Yang. 2024. Ladder: Enabling efficient low-precision deep learning computing through hardware-aware tensor transformation. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 307–323, Santa Clara, CA. USENIX Association.

Zhenliang Xue, Yixin Song, Zeyu Mi, Xinrui Zheng, Yubin Xia, and Haibo Chen. 2024. Powerinfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

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

A Experimental Details

A.1 Hyperparameters for Predictors

Name	Low-Rank$(\theta_{\text{A}}, \theta_{\text{B}})$	BitLinear$(\theta_{\text{ternary}})$
Loss	Binary Cross Entropy	
Sparsity ratio (k)	$\{0.7, 0.8, 0.9\}$	
Learning rate	{**1e-3**, 5e-4}	
Training batch size	$\{\mathbf{16}, 32\}$	
Optimizer	AdamW	
Target	Binary mask s^k_{alive}	
Epochs	$\{10, 20, 40, \mathbf{80}\}$	
Seed	42	
Predictor shape	Low-Rank Approximator	TernaryLinear
d_{rank}	$\{128, 256, \mathbf{512}, 1024\}$	–
Hardware	1 × NVIDIA A100 80GB	

Table 5: Hyperparameter settings and additional reproducibility details for training predictors used in D-CountDown. All experiments were conducted using a single run without multiple random seeds.

A.2 Environments

All experiments were performed on an NVIDIA A100 80GB GPU. We used Triton v3.1.0 for custom kernel development, while the rest of the experimental pipeline was built on HuggingFace Transformers v4.51.3, PyTorch v2.5.1, and CUDA v12.1.

A.3 Dataset Description

Table 6 summarizes the licenses and dataset statistics used for evaluation.

We evaluate seven Natural Language Understanding(NLU) tasks and one Natural Language Generation(NLG) task focused on mathematical reasoning (GSM8K). All datasets primarily contain English text.

Dataset	License	Train	Test
ARC-Easy	cc-by-sa-4.0	2251 (500)	2376
ARC-Challenge	cc-by-sa-4.0	1119 (500)	1172
HellaSwag	MIT	39905 (500)	10042
PIQA	AFL 3.0	16113 (500)	1838
WinoGrande	apache-2.0	40398 (500)	1267
OpenBookQA	apache-2.0	4957 (500)	500
TruthfulQA	apache-2.0	0	817
GSM8K	MIT	0	1319

Table 6: Summary statistics and licenses for datasets used in evaluation. Following previous research (Akhauri et al., 2024), we used subsets of each downstream task's training set, each containing 500 examples.

B Pseudo Codes

B.1 Training Procedures

D-CountDown's predictor training only requires 2 hours for smaller models (Llama-3.1-8B, Gemma-2-9B) and 4 hours for larger models (Phi-4, Qwen2.5-14B) on a single NVIDIA A100 GPU. The resulting predictors are as compact as 576-900MB, representing merely 6-7% of model parameters.

Algorithm 1: Training the predictor for a Gated-MLP in D-CountDown

Input: Training samples $\{x_j\}_{j=1}^{N}$, Target module $GatedMLP$, Target sparsity ratio k
Output: Trained predictor parameters θ

foreach *training sample* x_j **do**
  $s_j \leftarrow$ Compute $GatedMLP(x_j)$;
  $s_j \leftarrow$ Binarize using Quantile$(k, |s_j|)$;
if *Predictor is Low-Rank* **then**
  Initialize parameters $\theta_{\text{A}}, \theta_{\text{B}}$;
else if *Predictor is TernaryLinear* **then**
  Initialize parameters θ_{ternary};
foreach *training iteration* **do**
  Sample mini-batch $\{x_b, s_b\}$;
  if *Predictor is Low-Rank* **then**
    $\hat{s}_b = x_b \cdot \theta_{\text{A}} \cdot \theta_{\text{B}}$;
  else if *Predictor is TernaryLinear* **then**
    $\hat{s}_b = x_b \cdot \theta_{\text{ternary}}$;
  Compute BCE loss between $\hat{s}_b$ and s_b;
  Update predictor parameters θ;

B.2 Kernel in Detail: M-CountDown

Algorithm 2: M-CountDown Inference Kernel (Triton-style)

Input: X, W_{up}, $\hat{\tau}_{\text{M}}$
Output: U, Mask
 # PyTorch ;
$U \leftarrow X \mathbin{@} W_{\text{up}}$;
Mask $\leftarrow (|U| \geq \hat{\tau}_{\text{M}})$;

Input: X, U, W_{gate}, Mask, $\text{BLK}_M, \text{BLK}_N$
Output: S
 # Triton 1 ;
start_m $\leftarrow$ tl.program_id(0) ;
rm $\leftarrow$ start_m $\times \text{BLK}_M +$ tl.arange$(0, \text{BLK}_M)$;
rn $\leftarrow$ tl.arange$(0, \text{BLK}_N)$;
Mask $\leftarrow$ Mask + rm ;
flag $\leftarrow$ tl.load(Mask) > 0 ;
$W_{\text{gate}} \leftarrow W_{\text{gate}} + (\text{rm}[:,\text{None}] \times d_{model} + \text{rn}[\text{None},:])$;
$X \leftarrow X +$ rn ;
acc $\leftarrow$ tl.zeros$((\text{BLK}_M))$;
i_mask $\leftarrow$ flag$[:, \text{None}]$;
foreach *block in rn* **do**
  $w \leftarrow$ tl.load$(W_{\text{gate}}, \text{mask} = \text{i_mask}, \text{other} = 0)$;
  $x \leftarrow$ tl.load(X) ;
  acc $\leftarrow$ acc + tl.sum$(w \times x[\text{None}, :], 1)$;
  $W_{\text{gate}} \leftarrow W_{\text{gate}} + \text{BLK}_N$;
  $X \leftarrow X + \text{BLK}_N$;
$U \leftarrow U +$ rm ;
$u \leftarrow$ tl.load$(U, \text{mask} = \text{flag}, \text{other} = 0)$;
acc $\leftarrow$ silu(acc) $\times u$;
$S \leftarrow S +$ rm ;
tl.store$(S, \text{acc}, \text{mask} = rm < d_{inter})$;

Input: S, W_{down}, Mask, $\text{BLK}_M, \text{BLK}_N$
Output: Y
 # Triton 2 ;
start_m $\leftarrow$ tl.program_id(0) ;
start_n $\leftarrow$ tl.program_id(1) ;
rm $\leftarrow$ start_m $\times \text{BLK}_M +$ tl.arange$(0, \text{BLK}_M)$;
rn $\leftarrow$ start_n $\times \text{BLK}_N +$ tl.arange$(0, \text{BLK}_N)$;
Mask $\leftarrow$ Mask + rm ;
flag $\leftarrow$ tl.load(Mask) > 0 ;
$W_{\text{down}} \leftarrow W_{\text{down}} + (\text{rm}[:,\text{None}] \times d_{model} + \text{rn}[\text{None},:])$;
$S \leftarrow S +$ rm ;
$w \leftarrow$ tl.load$(W_{\text{down}}, \text{mask} = \text{flag}[:,\text{None}], \text{other} = 0)$;
$x \leftarrow$ tl.load(S) ;
acc $\leftarrow$ tl.sum$(w \times x[:, \text{None}], 0)$;
$Y \leftarrow Y +$ rn ;
tl.atomic_add$(Y,$ acc$)$;

B.3 Kernel in Detail: D-CountDown

Algorithm 3: D-CountDown Inference Kernel (Triton-style)

Input: $X, \theta_{\text{A}}, \theta_{\text{B}}, \tau_D$
Output: Mask
 # PyTorch ;
$\hat{s} \leftarrow X \mathbin{@} \theta_{\text{A}} \mathbin{@} \theta_{\text{B}}$;
Mask $\leftarrow (\hat{s} \geq \tau_D)$;

Input: $X, W_{\text{gate}}, W_{\text{up}}$, Mask, $\text{BLK}_M, \text{BLK}_N$
Output: S
 # Triton 1 ;
start_m $\leftarrow$ tl.program_id(0) ;
rm $\leftarrow$ start_m $\times \text{BLK}_M +$ tl.arange$(0, \text{BLK}_M)$;
rn $\leftarrow$ tl.arange$(0, \text{BLK}_N)$;
Mask $\leftarrow$ Mask + rm ;
flag $\leftarrow$ tl.load(Mask) > 0 ;
$W_{\text{gate}} \leftarrow W_{\text{gate}} + (\text{rm}[:,\text{None}] \times d_{model} + \text{rn}[\text{None},:])$;
$W_{\text{up}} \leftarrow W_{\text{up}} + (\text{rm}[:,\text{None}] \times d_{model} + \text{rn}[\text{None},:])$;
$X \leftarrow X +$ rn ;
gate $\leftarrow$ tl.zeros$([\text{BLK}_M])$;
up $\leftarrow$ tl.zeros$([\text{BLK}_M])$;
i_mask $\leftarrow$ flag$[:, \text{None}]$;
foreach *block in rn* **do**
  $w_{\text{gate}} \leftarrow$ tl.load$(W_{\text{gate}}, \text{mask} = \text{i_mask}, \text{other} = 0)$;
  $w_{\text{up}} \leftarrow$ tl.load$(W_{\text{up}}, \text{mask} = \text{i_mask}, \text{other} = 0)$;
  $x \leftarrow$ tl.load(X) ;
  gate $\leftarrow$ gate + tl.sum$(w_{\text{gate}} \times x[\text{None}, :], \text{axis} = 1)$;
  up $\leftarrow$ up + tl.sum$(w_{\text{up}} \times x[\text{None}, :], \text{axis} = 1)$;
  X $\leftarrow$ X + BLK_N ;
  $W_{\text{gate}} \leftarrow W_{\text{gate}} + \text{BLK}_N$;
  $W_{\text{up}} \leftarrow W_{\text{up}} + \text{BLK}_N$;
up $\leftarrow$ up $\times$ SiLU(gate) ;
tl.store$(S,$ up, mask $=$ rm $< M)$;

Input: S, W_{down}, Mask, $\text{BLK}_M, \text{BLK}_N$
Output: Y
 # Triton 2 ;
start_m $\leftarrow$ tl.program_id(0) ;
start_n $\leftarrow$ tl.program_id(1) ;
rm $\leftarrow$ start_m $\times \text{BLK}_M +$ tl.arange$(0, \text{BLK}_M)$;
rn $\leftarrow$ start_n $\times \text{BLK}_N +$ tl.arange$(0, \text{BLK}_N)$;
Mask $\leftarrow$ Mask + rm ;
flag $\leftarrow$ tl.load(Mask) > 0 ;
$W_{\text{down}} \leftarrow W_{\text{down}} + (\text{rm}[:,\text{None}] \times d_{model} + \text{rn}[\text{None},:])$;
$S \leftarrow S +$ rm ;
$w \leftarrow$ tl.load$(W_{\text{down}}, \text{mask} = \text{flag}[:,\text{None}], \text{other} = 0)$;
$x \leftarrow$ tl.load(S) ;
acc $\leftarrow$ tl.sum$(w \times x[:, \text{None}], 0)$;
$Y \leftarrow Y +$ rn ;
tl.atomic_add$(Y,$ acc$)$;

C Full Results

All downstream task results are in Table 11 and Table 12. Chat performance results are shown in Table 7. Kernel Speed results are shown in Figure 5. Sparsity$^{\text{Real}}$ indicates the actual proportion of indicator elements filtered out during SP^{Prac} inferences.

Method	Scenario	Llama-3.1-8B			Gemma-2-9B		
		Target Sparsity			Target Sparsity		
		0.70	0.80	0.90	0.70	0.80	0.90
CATS	SP^{Ideal}	1.02	0.48	0.50	35.41	2.55	0.00
	SP^{Prac} (Win)	3.26	0.55	0.72	40.76	6.72	0.00
	SP^{Prac} ($Sparsity^{Real}$)	*70.8*	*80.0*	*89.7*	*68.8*	*79.3*	*88.1*
DC	SP^{Ideal}	45.79	39.33	11.85	50.44	48.90	37.79
	SP^{Prac} (Win)	1.35	1.57	0.77	6.72	7.99	3.57
	SP^{Prac} ($Sparsity^{Real}$)	*68.8*	*71.5*	*80.8*	*67.0*	*72.9*	*83.2*
MC	SP^{Ideal}	46.59	2.74	0.60	47.81	41.83	6.91
	SP^{Prac} (Win)	9.68	3.19	0.74	48.78	47.88	28.08
	SP^{Prac} ($Sparsity^{Real}$)	*72.7*	*82.3*	*91.0*	*68.0*	*77.8*	*87.6*

Method	Scenario	Qwen2.5-14B			Phi-4		
		Target Sparsity			Target Sparsity		
		0.70	0.80	0.90	0.70	0.80	0.90
CATS	SP^{Ideal}	21.94	0.38	0.00	42.05	3.45	0.25
	SP^{Prac} (Win)	33.62	7.80	0.00	48.87	26.81	0.31
	SP^{Prac} ($Sparsity^{Real}$)	*70.0*	*80.0*	*89.1*	*67.6*	*78.3*	*89.7*
DC	SP^{Ideal}	50.10	48.71	32.73	49.10	46.46	37.45
	SP^{Prac} (Win)	4.40	0.77	0.20	1.11	0.90	0.12
	SP^{Prac} ($Sparsity^{Real}$)	*66.6*	*82.8*	*87.6*	*65.4*	*78.5*	*86.8*
MC	SP^{Ideal}	45.01	36.90	2.83	43.96	35.43	5.28
	SP^{Prac} (Win)	48.57	42.51	15.89	46.24	41.62	18.82
	SP^{Prac} ($Sparsity^{Real}$)	*70.0*	*80.0*	*90.0*	*70.1*	*79.6*	*89.6*

Table 7: Summary of Win Rate on AlpacaEval 2.0

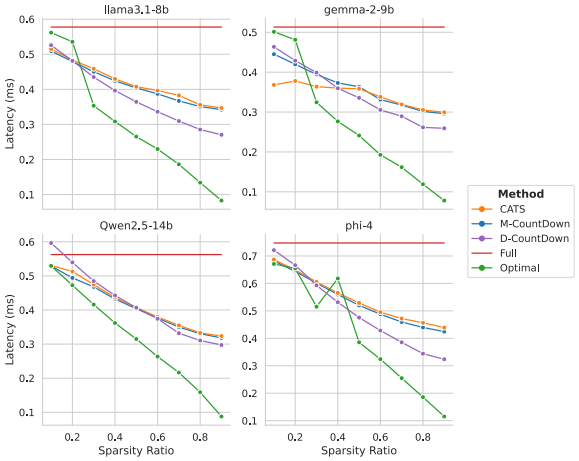

Figure 5: All results for kernel speed.

D Theoretical Analysis Details

D.1 Notation

Notation	Explanation
d_m	d_{model}
d_i	d_{inter}
d_r	d_{rank}
s	$int(d_{inter} \times k)$
c_{act}	act FLOPs (e.g. SiLU ≈ 5)

Table 8: Notation Used in Theoretical Analysis

D.2 heoretical FLOPs Analysis

Method	Compute	Explanation
Dense	$6\,d_m\,d_i$	Full GEMV $\times 3$
	$+\,c_{\text{act}}\,d_i$	Full σ
	$+\,d_i$	Full $\odot$
CATS	$2\,d_m\,d_i$	Full GEMV W_{gate}
	$+\,c_{\text{act}}\,d_i$	Full σ
	$+\,2\,d_i$	Apply abs and $THLD$
	$+\,2\,d_m\,s$	Sparse GEMV W_{up}
	$+\,s$	Sparse $\odot$
	$+\,2\,d_m\,s$	Sparse GEMV W_{down}
M-CountDown	$2\,d_m\,d_i$	Full GEMV W_{up}
	$+\,2\,d_i$	Apply abs and $THLD$
	$+\,2\,d_m\,s$	Sparse GEMV W_{gate}
	$+\,c_{\text{act}}\,s$	Sparse σ
	$+\,s$	Sparse $\odot$
	$+\,2\,d_m\,s$	Sparse GEMV W_{down}
D-CountDown	$2\,d_m\,d_r$	Low-rank GEMV θ_A
	$+\,2\,d_r\,d_i$	Low-rank GEMV θ_B
	$+\,d_i$	Apply $THLD$
	$+\,4\,d_m\,s$	Sparse GEMV W_{gate}, W_{up}
	$+\,c_{\text{act}}\,s$	Sparse σ
	$+\,s$	Sparse $\odot$
	$+\,2\,d_m\,s$	Sparse GEMV W_{down}

Table 9: Comparison of Theoretical FLOPs Across Methods

D.3 Theoretical Memory Traffic Analysis

Method	Mem. R/W	Explanation
Dense	$2\,d_m\,d_i$	Read Full W_{up}, W_{gate}
	$+\,2\,d_m$	Read $x \times 2$
	$+\,2\,d_i$	Write gate, up
	$+\,d_i$	Read gate
	$+\,d_i$	Write act_gate
	$+\,2\,d_i$	Read act_gate, up
	$+\,d_i$	Write inter
	$+\,d_m\,d_i$	Read Full W_{down}
	$+\,d_i$	Read inter
	$+\,d_m$	Write y
CATS	$d_m\,d_i$	Read Full W_{gate}
	$+\,d_m$	Read x
	$+\,d_i$	Write gate
	$+\,d_i$	Read gate
	$+\,d_i$	Write act_gate
	$+\,d_i$	Read act_gate
	$+\,d_i$	Write abs_act_gate
	$+\,d_i$	Read abs_act_gate
	$+\,d_i$	Write mask
	$+\,d_m\,s$	Read Sparse W_{up}
	$+\,d_m$	Read x
	$+\,s$	Read Sparse act_gate
	$+\,d_i$	Read mask
	$+\,d_i$	Write inter
	$+\,d_m\,s$	Read Sparse W_{down}
	$+\,d_i$	Read inter
	$+\,d_m$	Write y

Method	Mem. R/W	Explanation
M-Countdown	$d_m\, d_i$	Read Full W_{up}
	$+\, d_m$	Read x
	$+\, d_i$	Write up
	$+\, d_i$	Read up
	$+\, d_i$	Write abs_up
	$+\, d_i$	Read abs_up
	$+\, d_i$	Write mask
	$+\, d_m\, s$	Read Sparse W_{gate}
	$+\, d_m$	Read x
	$+\, s$	Read Sparse up
	$+\, d_i$	Read mask
	$+\, d_i$	Write inter
	$+\, d_m\, s$	Read Sparse W_{down}
	$+\, d_i$	Read inter
	$+\, d_m$	Write y
D-Countdown	$d_m\, d_r$	Read θ_A
	$+\, d_m$	Read x
	$+\, d_r$	Write latent
	$+\, d_r\, d_i$	Read θ_B
	$+\, d_r$	Read latent
	$+\, d_i$	Write $\hat{s}$
	$+\, d_i$	Read $\hat{s}$
	$+\, d_i$	Write mask
	$+\, 2\, d_m\, s$	Read Sparse W_{up}, W_{gate}
	$+\, d_m$	Read x
	$+\, d_i$	Read mask
	$+\, d_i$	Write inter
	$+\, d_m\, s$	Read Sparse W_{down}
	$+\, d_i$	Read inter
	$+\, d_m$	Write y

Table 10: Comparison of Theoretical Memory Traffic Across Methods

Sparsity	Method	ARC-C	TFQA	HS	ARC-E	PIQA	WG	OBQA	GSM8K
Llama-3.1-8B-Instruct									
0.0	DENSE	**0.520**	**0.367**	**0.590**	**0.819**	**0.800**	**0.737**	**0.336**	**0.760**
0.7	DEJAVU	0.292	0.229	0.272	0.445	0.553	0.503	0.218	0.000
	CATS	0.453	0.343	0.523	0.754	0.739	0.653	0.298	0.003
	M-COUNTDOWN	0.493	**0.372**	0.568	0.784	0.776	0.695	0.330	0.544
	D-COUNTDOWN	**0.509**	0.370	**0.592**	**0.812**	**0.795**	**0.727**	**0.332**	**0.688**
0.8	DEJAVU	0.282	0.231	0.273	0.440	0.557	0.511	0.228	0.000
	CATS	0.358	0.326	0.428	0.651	0.676	0.582	0.278	0.000
	M-COUNTDOWN	0.458	0.343	0.534	0.759	0.748	0.661	0.314	0.288
	D-COUNTDOWN	**0.502**	**0.356**	**0.585**	**0.809**	**0.789**	**0.713**	**0.334**	**0.605**
0.9	DEJAVU	0.296	0.230	0.273	0.455	0.557	0.530	0.236	0.000
	CATS	0.293	0.252	0.303	0.495	0.574	0.534	0.242	0.000
	M-COUNTDOWN	0.411	0.304	0.430	0.649	0.676	0.613	0.286	0.001
	D-COUNTDOWN	**0.484**	**0.330**	**0.548**	**0.776**	**0.755**	**0.680**	**0.312**	**0.313**
Qwen2.5-14B-Instruct									
0.0	DENSE	**0.608**	**0.517**	**0.657**	**0.861**	**0.817**	**0.758**	**0.364**	**0.807**
0.7	DEJAVU	0.336	0.318	0.365	0.612	0.616	0.533	0.254	0.000
	CATS	0.488	0.443	0.585	0.777	0.729	0.629	0.318	0.043
	M-COUNTDOWN	0.573	0.488	0.638	0.829	0.792	0.704	0.352	**0.776**
	D-COUNTDOWN	**0.588**	**0.518**	**0.654**	**0.850**	**0.801**	**0.736**	**0.364**	0.770
0.8	DEJAVU	0.340	0.322	0.357	0.609	0.619	0.554	0.258	0.000
	CATS	0.410	0.371	0.472	0.683	0.632	0.568	0.284	0.000
	M-COUNTDOWN	0.532	0.476	0.614	0.813	0.743	0.670	0.352	0.681
	D-COUNTDOWN	**0.579**	**0.488**	**0.644**	**0.837**	**0.799**	**0.716**	**0.360**	**0.751**
0.9	DEJAVU	0.358	0.333	0.369	0.612	0.621	0.531	0.256	0.000
	CATS	0.356	0.327	0.385	0.619	0.621	0.547	0.260	0.000
	M-COUNTDOWN	0.468	0.421	0.525	0.736	0.686	0.589	0.304	0.100
	D-COUNTDOWN	**0.512**	**0.436**	**0.607**	**0.801**	**0.756**	**0.648**	**0.312**	**0.371**
gemma-2-9b-it									
0.0	DENSE	**0.632**	**0.433**	**0.597**	**0.856**	**0.812**	**0.761**	**0.404**	**0.663**
0.7	DEJAVU	0.339	0.246	0.300	0.596	0.590	0.532	0.276	0.000
	CATS	0.575	0.412	0.559	0.840	0.755	0.680	0.348	0.565
	M-COUNTDOWN	0.605	**0.421**	0.592	0.849	0.793	0.726	0.374	0.632
	D-COUNTDOWN	**0.626**	0.417	**0.600**	**0.854**	**0.800**	**0.750**	**0.384**	**0.649**
0.8	DEJAVU	0.346	0.246	0.296	0.599	0.581	0.548	0.262	0.000
	CATS	0.490	0.366	0.486	0.788	0.696	0.604	0.328	0.105
	M-COUNTDOWN	0.583	0.408	0.582	0.842	0.767	0.707	0.360	0.610
	D-COUNTDOWN	**0.604**	**0.421**	**0.599**	**0.851**	**0.796**	**0.728**	**0.374**	**0.624**
0.9	DEJAVU	0.356	0.246	0.303	0.616	0.573	0.523	0.264	0.000
	CATS	0.364	0.242	0.310	0.617	0.589	0.537	0.278	0.000
	M-COUNTDOWN	0.534	0.383	0.517	0.799	0.727	0.648	0.344	0.438
	D-COUNTDOWN	**0.578**	**0.410**	**0.572**	**0.833**	**0.777**	**0.676**	**0.352**	**0.524**
phi-4									
0.0	DENSE	**0.558**	**0.404**	**0.632**	**0.814**	**0.808**	**0.766**	**0.338**	**0.923**
0.7	DEJAVU	0.387	0.311	0.348	0.655	0.626	0.587	0.266	0.000
	CATS	0.536	0.400	0.595	0.794	0.791	0.696	0.304	0.807
	M-COUNTDOWN	0.533	0.384	0.616	0.800	0.796	0.733	**0.334**	0.888
	D-COUNTDOWN	**0.554**	**0.411**	**0.630**	**0.809**	**0.807**	**0.752**	0.332	**0.916**
0.8	DEJAVU	0.409	0.333	0.354	0.655	0.632	0.585	0.270	0.000
	CATS	0.516	0.397	0.539	0.771	0.760	0.644	0.298	0.351
	M-COUNTDOWN	0.503	0.386	0.594	0.792	0.778	0.715	0.330	0.767
	D-COUNTDOWN	**0.552**	**0.408**	**0.622**	**0.807**	**0.810**	**0.755**	**0.340**	**0.898**
0.9	DEJAVU	0.392	0.317	0.357	0.640	0.630	0.566	0.266	0.000
	CATS	0.426	0.356	0.414	0.676	0.672	0.591	0.280	0.000
	M-COUNTDOWN	0.479	0.370	0.524	0.759	0.728	0.654	0.296	0.287
	D-COUNTDOWN	**0.529**	**0.399**	**0.601**	**0.798**	**0.789**	**0.695**	**0.318**	**0.827**

Table 11: $\texttt{SP}^{\texttt{Ideal}}$ scores compared to `Dense` across all downstream tasks. `Dense` scores are in bold, as well as the highest score for each task within each sparsity level.

Sparsity	Method	Sparsity$^{\text{Real}}$	ARC-C	TFQA	HS	ARC-E	PIQA	WG	OBQA	GSM8K
Llama-3.1-8B-Instruct										
0.7	CATS	0.684	0.461	0.355	0.549	0.778	0.764	0.683	**0.316**	0.127
	M-COUNTDOWN	0.709	**0.484**	**0.375**	**0.574**	**0.788**	**0.778**	**0.708**	0.310	**0.547**
	D-COUNTDOWN	0.705	0.422	0.318	0.373	0.748	0.714	0.663	0.298	0.002
0.8	CATS	0.784	0.420	0.322	0.495	0.718	0.721	0.624	0.296	0.000
	M-COUNTDOWN	0.806	**0.460**	**0.361**	**0.549**	**0.770**	**0.757**	**0.680**	**0.322**	**0.322**
	D-COUNTDOWN	0.739	0.382	0.306	0.388	0.688	0.673	0.621	0.292	0.003
0.9	CATS	0.902	0.299	0.273	0.323	0.521	0.607	0.537	0.238	0.000
	M-COUNTDOWN	0.895	**0.416**	**0.321**	**0.471**	**0.711**	**0.721**	**0.620**	**0.304**	**0.009**
	D-COUNTDOWN	0.843	0.349	0.285	0.345	0.628	0.635	0.593	0.260	0.000
Qwen2.5-14B-Instruct										
0.7	CATS	0.698	0.518	0.460	0.612	0.805	0.761	0.660	0.336	0.293
	M-COUNTDOWN	0.719	**0.590**	**0.509**	**0.640**	**0.838**	**0.792**	**0.712**	**0.358**	**0.767**
	D-COUNTDOWN	0.678	0.513	0.426	0.536	0.798	0.748	0.668	0.322	0.197
0.8	CATS	0.802	0.472	0.421	0.551	0.754	0.712	0.627	0.284	0.000
	M-COUNTDOWN	0.804	**0.553**	**0.492**	**0.625**	**0.826**	**0.769**	**0.669**	**0.354**	**0.704**
	D-COUNTDOWN	0.827	0.454	0.394	0.468	0.740	0.693	0.615	0.292	0.000
0.9	CATS	0.906	0.347	0.350	0.393	0.631	0.616	0.527	0.258	0.000
	M-COUNTDOWN	0.889	**0.492**	**0.450**	**0.580**	**0.794**	**0.727**	**0.632**	**0.320**	**0.287**
	D-COUNTDOWN	0.893	0.434	0.384	0.429	0.689	0.669	0.605	0.282	0.000
gemma-2-9b-it										
0.7	CATS	0.695	0.580	0.427	0.567	0.843	0.770	0.693	0.368	0.593
	M-COUNTDOWN	0.685	**0.608**	**0.431**	**0.598**	**0.854**	**0.801**	**0.745**	**0.386**	**0.633**
	D-COUNTDOWN	0.689	0.567	0.403	0.493	0.821	0.751	0.702	0.364	0.340
0.8	CATS	0.806	0.542	0.392	0.501	0.811	0.729	0.615	0.346	0.083
	M-COUNTDOWN	0.779	**0.596**	**0.412**	**0.589**	**0.847**	**0.788**	**0.712**	0.370	**0.618**
	D-COUNTDOWN	0.755	0.564	0.401	0.506	0.819	0.758	0.702	**0.374**	0.381
0.9	CATS	0.911	0.340	0.258	0.306	0.617	0.586	0.514	0.262	0.000
	M-COUNTDOWN	0.875	**0.573**	**0.395**	**0.554**	**0.829**	**0.761**	**0.686**	**0.360**	**0.544**
	D-COUNTDOWN	0.853	0.529	0.383	0.492	0.806	0.747	0.665	0.354	0.187
phi-4										
0.7	CATS	0.675	0.539	**0.417**	0.613	0.801	0.795	0.724	0.322	0.856
	M-COUNTDOWN	0.707	**0.540**	0.393	**0.620**	**0.804**	**0.796**	**0.736**	**0.332**	**0.894**
	D-COUNTDOWN	0.687	0.471	0.368	0.485	0.750	0.733	0.685	0.294	0.208
0.8	CATS	0.771	0.525	**0.390**	0.587	**0.795**	**0.786**	0.673	0.300	0.675
	M-COUNTDOWN	0.799	**0.527**	0.381	**0.607**	0.793	0.784	**0.715**	**0.334**	**0.817**
	D-COUNTDOWN	0.815	0.418	0.343	0.438	0.707	0.692	0.657	0.268	0.036
0.9	CATS	0.889	0.458	0.360	0.460	0.713	0.692	0.605	0.294	0.000
	M-COUNTDOWN	0.894	**0.498**	**0.386**	**0.563**	**0.777**	**0.750**	**0.698**	**0.316**	**0.450**
	D-COUNTDOWN	0.895	0.408	0.294	0.404	0.674	0.668	0.616	0.270	0.000

Table 12: SP^{Prac} scores compared across all downstream tasks. Bold indicates the highest score at each sparsity level for each task.