Training-Free Activation Sparsity in Large Language Models

Anonymous authors
Paper under double-blind review

Abstract

Activation sparsity can enable practical inference speedups in large language models (LLMs) by reducing the compute and memory-movement required for matrix multiplications during the forward pass. However, existing methods face limitations that inhibit widespread adoption. Some approaches are tailored towards older models with ReLU-based sparsity, while others require extensive continued pre-training on up to hundreds of billions of tokens. This paper describes TEAL (**T**raining-Fre**e** **A**ctivation Sparsity in **L**LMs), a simple training-free method that applies magnitude-based activation sparsity to hidden states throughout the entire model. TEAL achieves 40-50% model-wide sparsity with minimal performance degradation across Llama-2, Llama-3, and Mistral families, with sizes varying from 7B to 70B. We improve existing sparse kernels and demonstrate wall-clock decoding speed-ups of up to 1.53× and 1.8× at 40% and 50% model-wide sparsity. TEAL is compatible with weight quantization, enabling further efficiency gains.

1 Introduction

Large language models (LLMs) demonstrate that scaling in both parameter count and training data leads to capabilities that are useful for addressing a variety of downstream tasks (Brown et al., 2020). However, the large number of parameters in modern LLMs can lead to substantial challenges during inference. In typical small-batch deployment settings, autoregressive inference is *memory-bound*, i.e., bottlenecked by the speed at which the parameters can be moved from off-chip to on-chip memory. This is in contrast to LLM training and prefill inference, which is generally *compute-bound*, i.e., bottlenecked by the speed at which computation can performed. A core strategy for overcoming this *memory wall* (Gholami et al., 2024) is through weight quantization (Frantar et al., 2022; Shao et al., 2023; Yuan et al., 2023; Lin et al., 2024; Dettmers et al., 2023c; Tseng et al., 2024; Egiazarian et al., 2024; Liu et al., 2024) and sparsification (Wang et al., 2019; Frantar & Alistarh, 2023; Xia et al., 2023; Ma et al., 2023), which can lead to practical speed-ups when coupled with specialized kernels that move the weights from off-chip to on-chip memory in quantized/sparse formats (Kim et al., 2023; Dettmers et al., 2023b; Frantar et al., 2024; Wang et al., 2024b; Xia et al., 2024; Guo et al., 2024).

The above methods directly compress a model's weights and apply the same (quantized/sparse) matrix to all inputs. Activation sparsity (Chen et al., 2023; Raihan & Aamodt, 2020; Kurtz et al., 2020) is an alternative method which enforces *input-dependent* structure on the weight matrices by leveraging (or inducing) sparsity in the hidden states. Since the weight channels corresponding to zero-valued activations are not used in computation, speed-up can be realized by selectively omitting these weights during memory transfer, which is possible due to the hardware-friendly channel-wise sparsity pattern. In older LLMs, activation sparsity is largely made possible by the high natural sparsity (around 95%) in the intermediate states of the MLP blocks in ReLU-based Transformer models (Li et al., 2023). Based on this, Liu et al. (2023) propose *DejaVu*, which learns a small auxiliary model that predicts the contextual activation sparsity patterns of future layers, and realize a 2× wall-clock speed-up on OPT-175B (Zhang et al., 2022). Because the hidden state is extremely sparse, the less expressive auxiliary model can afford to overestimate non-zero activations while maintaining accuracy and efficiency (e.g., 20% predicted vs. 5% actual non-zero entries).

However, modern LLMs have largely moved away from ReLU-based feedforward layers due to their worse performance compared to variants like SwiGLU (Shazeer, 2020). In these models the activations are no longer naturally sparse, making it difficult to apply methods like DejaVu. And while recent works have found that replacing SiLU with ReLU in the MLP blocks and performing continued pre-training can "recover" models that exhibit high activation sparsity (thus making older methods applicable) (Mirzadeh et al., 2023; Song et al., 2024a;b), such methods require training on up to hundreds of billions of tokens.

This work describes TEAL (**T**raining-Fre**e** **A**ctivation Sparsity in **L**LMs), a simple, training-free approach that applies activation sparsity based on magnitude pruning. TEAL is based on the observation that distributional shapes in LLaMA-architecture LLMs are zero-mean unimodal. By pruning low-magnitude, non-salient activations, we achieve 40-50% model-wide (input-dependent) sparsity, in contrast to prior work which only achieves sparsity in portions of the model (Lee et al., 2024b). We realize wall-clock speed-ups of up to $1.53\times$ and $1.8\times$ at 40% and 50% sparsity respectively through specialized kernels, and further demonstrate compatibility with weight quantization.

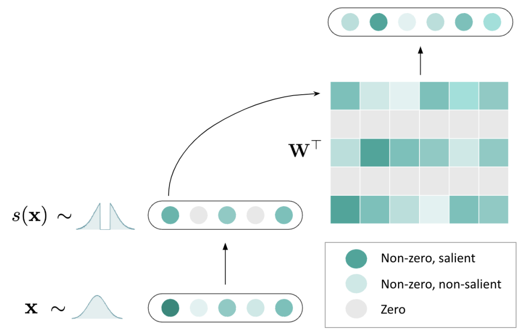

Figure 1: **Overview of TEAL**. During decoding, TEAL thresholds low-magnitude activation entries to zero, which obviates the need to move the associated weight channels onto the registers, thus enabling wall-clock speed-ups.

2 Related Work

Conditional computation (Bengio, 2013; Bengio et al., 2016) alleviates the burden of training and serving by selectively activating parts of a model. Shazeer et al. (2017) propose Mixture-of-Experts (MoE) in language models, applying conditional computation to the feed forward networks. Mixture-of-Experts models decouple parameter count with computational footprint (Fedus et al., 2022), and demonstrate superior scaling laws compared to dense baselines (Clark et al., 2022).

Activation sparsity occurs when a significant portion of a model's hidden states contain zero-valued entries, and can be seen as an instance of conditional computaton. Activation sparsity is known to naturally emerge in the intermediate states of ReLU-based MLPs (Li et al., 2023). Liu et al. (2023) leverage activation sparsity to accelerate LLM inference by avoiding the transfer of weight channels associated with zero-valued entries to GPU registers. Song et al. (2023) and Alizadeh et al. (2024) extend activation sparsity to CPU offloading, reducing weight transfer from CPU to GPU memory. However, newer architectures typically make use of non-ReLU-based MLPs (e.g., SwiGLU, Shazeer, 2020), making these off-the-shelf methods difficult to use in practice.

Recent work has thus focused on reintroducing activation sparsity in newer architectures. Mirzadeh et al. (2023) replace SiLU or GeLU activation functions with ReLU, followed by continued pretraining on hundreds of billions of tokens. Zhang et al. (2024b) experiment with different activations and find Squared ReLU (So et al., 2022) to be the most effective replacement. Song et al. (2024b) and Song et al. (2024a) introduce techniques such as activation regularization to push sparsity even higher in adapted models. Wang et al. (2024a) combine magnitude pruning with Squared ReLU and quantized activations, and establish scaling laws for sparsely activated LLMs during pretraining.

Lee et al. (2024a) propose *CATS*, and realize training-free activation sparsity on SwiGLU based LLMs by applying magnitude pruning on the output of $\mathbf{W}_{\text{gate}}$, with the intuition that in the training-free setting, ReLU-based methods suboptimally zero out nontrivial negative values but keep positive values with lower magnitude intact. They achieve up to 50% sparsity in $\mathbf{W}_{\text{up}}$ and $\mathbf{W}_{\text{down}}$ for Mistral and Llama-2-7B. However, other matrices including $\mathbf{W}_{\text{gate}}$ and $\mathbf{W}_{\text{q,k,v,o}}$ are computed densely, resulting in lower model-wide sparsity (roughly 25%), whereas we target every matrix in the model. We refer the reader to Appendix A.3 for formal definitions of the weight matrices and their interactions within each Transformer block.

3 Background: Activation Sparsity in Neural Networks

The activation sparsity of a hidden state $\mathbf{x}$ is defined as the proportion of zero-valued entries, which can interact with the model in two ways. The first is *input sparsity*: when computing $\mathbf{y} = \mathbf{x}\mathbf{W}^\top$ for $\mathbf{x} \in \mathbb{R}^m, \mathbf{W} \in \mathbb{R}^{n\times m}$, the columns $\mathbf{W}_{:,i}$ corresponding to zero-valued entries $\mathbf{x}_i$ are unused. The second is *output sparsity*: when computing $\mathbf{y} = \mathbf{s} \odot (\mathbf{x}\mathbf{W}^\top)$ for the aforementioned parameters and mask $\mathbf{s} \in \mathbb{R}^n$, the rows $\mathbf{W}_{i,:}$ corresponding to zero-valued entries $\mathbf{s}_i$ are unused. CATS makes use of output sparsity on GLU variants, treating $\mathbf{s} = \mathrm{sparsify}(\sigma(\mathbf{x}\mathbf{W}_{\text{gate}}^\top))$ as the mask and applying output sparsity on $\mathbf{x}\mathbf{W}_{\text{up}}^\top$, with the intuition that $\sigma(\cdot)$ serves as a gating mechanism. Interestingly, we find in Section 5.4.1 that input sparsity is still preferable in the training-free case for SwiGLU.

In LLMs, the computation $\mathbf{x}\mathbf{W}^\top$ is memory-bound in the decoding phase due to the high memory footprint of weights, and thus reducing the transfer of unnecessary entries (i.e., rows/columns corresponding to zero-valued activations) can enable speed-ups. However, GPUs are designed to fetch multiple consecutive memory entries in a single access to maximize memory bandwidth. When memory accesses are non-contiguous, as they are when unnecessary entries are scattered, this leads to inefficient use of memory bandwidth. To ensure memory coalescing and contiguous memory access, it is crucial to store weights associated with input sparsity in a *column-major* format, and weights associated with output sparsity in a *row-major* format.

4 TEAL: Training-Free Activation Sparsity in LLMs

4.1 Motivating Study: Distributional Properties of Activations in LLMs

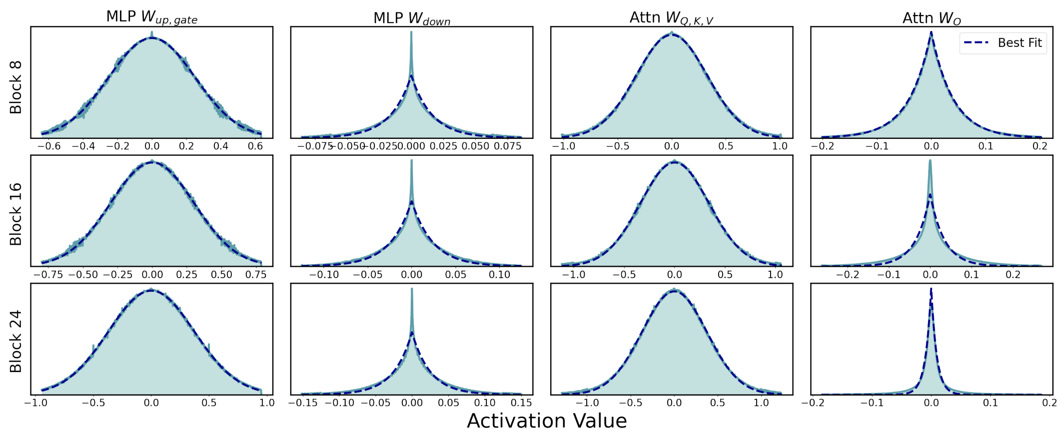

Figure 2: Activation distributions of Llama-3-8B's four hidden states at Blocks 8, 16, and 24. The activations preceding the Attention and MLP blocks typically exhibit Gaussian-like shapes, while intermediate states within these blocks exhibit Laplacian-like shapes. The best-fit Gaussian/Laplace distributions are overlayed in blue.

We perform a preliminary study of the distributional properties of activations of LLMs. We collect activations of Llama-3-8B (Dubey et al., 2024) sampled from C4 (Raffel et al., 2023) at the four hidden states in a Transformer block,[1] and visualize them in Figure 2. As indicated by prior work, some of the activations are heavy-tailed and contain outliers (Dettmers et al., 2022; Xiao et al., 2022; Wei et al., 2022; Nrusimha et al., 2024). The hidden states are moreover zero-mean unimodal, and qualitatively fall into two distinctly shaped distributions. The hidden states before the Attention and the MLP layers tend to be Gaussian-like, while the hidden states in the intermediate of such layers tend to be Laplacian-like. The concentration of the activations around zero motivates our magnitude-based activation pruning approach.

Remark. We do not attempt to explain why these distributions are shaped the way they are, nor do we give the theoretical underpinnings of why activation sparsity works. However, we make a few general observations. LLM weights are typically Gaussian (Dettmers et al., 2023a), and multiplying an independent isotropic Gaussian vector with an independent Gaussian matrix follows a multivariate generalized Laplace distribution Mattei (2017) (the weights and activations are clearly not independent in practice). Attention is a data-dependent linear operator (Poli et al., 2023) which may have similar properties. Distributions may be zero-mean due to layer normalization (Ba et al., 2016). We further derive the expected error induced by pruning low-magnitude activations in Appendix A.1, under a more restrictive assumption that weights and activations are independent Gaussians.

[1]Throughout, we use "block" to refer to an entire Transformer block consisting of the seven matrices and "layer" to refer to an individual layer (corresponding to a single matrix) within the Transformer block.

4.2 TEAL

The above analysis motivates our simple approach for activation sparsity based on magnitude pruning. While small-magnitude activations could still have a large effect on the output if the norms of corresponding channels of the weight matrix are large, we find that magnitude-based pruning is empirically effective. We first define a sparsification function for an activation vector as follows:

Definition 1. *For a random vector* $\tilde{\mathbf{x}} = (\tilde{x}_1, \dots, \tilde{x}_n)$ *and sparsity level* $p \in [0, 1]$*, define the threshold* t_p *as*

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{P}(|\tilde{x}_i| \leq t_p) = p.$$

The sparsification function $s_{t_p} : \mathbb{R}^n \to \mathbb{R}^n$ *is defined as:*

$$s_{t_p}(\mathbf{x}) = (s_{t_p}(x_1), \dots, s_{t_p}(x_n))$$

where $\mathbf{x}$ *is a realization of* $\tilde{\mathbf{x}}$*, and for each component:*

$$s_{t_p}(x_i) = \begin{cases} 0 & \text{if } |x_i| \leq t_p \\ x_i & \text{otherwise} \end{cases}$$

In practice we estimate t_p using an empirical distribution constructed offline using activations from generic text. The sparsity level p is characterized entirely by threshold t_{p_i}, which is useful in both implementation and kernel design (Section 4.4).

Let $\mathcal{W}$ be the set of matrices in the MLP and Attention blocks of a model, and further let $N = |\mathcal{W}|$. We define a model-level sparsification configuration as $\mathbf{p} = (p_1, ..., p_N)$, where each $p_i \in [0, 1]$ represents the sparsity level for the corresponding matrix $\mathbf{W}_i$. For each matrix $\mathbf{W}_i \in \mathcal{W}$, we define its layer-level sparsified forward pass as:

$$\hat{\mathbf{Y}} = s_{t_{p_i}}(\mathbf{x})\mathbf{W}_i^\top$$

for input $\mathbf{x}$ and magnitude-based sparsification function $s_{t_{p_i}}(\cdot)$ as defined in Definition 1. We apply this sparsified forward pass to all N matrices to obtain the model-level sparsified forward pass. For each Transformer block, this sparsifies all four of its hidden states: before $W_{\text{up, gate}}$, before W_{down}, before $W_{\text{Q,K,V}}$, and before W_{O}.

4.3 BLOCK-WISE GREEDY OPTIMIZATION

How should we find the optimal $\mathbf{p}$? We initially tried a gradient-based approach to learning the thresholds based on the straight through estimator (Bengio et al., 2013), but encountered optimization issues. We instead used a simple greedy approach illustrated in Algorithm 1, which was found to be effective.

For each Transformer block, we seek to minimize the block-wise ℓ_2 activation error subject to a block-level sparsity constraint. Each Transformer block consists of seven matrices: $\mathbf{W}_{\text{q}}, \mathbf{W}_{\text{k}}, \mathbf{W}_{\text{v}}, \mathbf{W}_{\text{o}}, \mathbf{W}_{\text{gate}}, \mathbf{W}_{\text{up}}, \mathbf{W}_{\text{down}}$. Algorithm 1 initializes the sparsity levels of all layers to zero, and attempts to increment the sparsity level of each layer by an amount inversely proportional to its memory footprint. The layer with the lowest ℓ_2 activation error is incremented, and the block-level sparsity plus associated layer-level sparsities are recorded. We assign the same block-level sparsity level across all Transformer blocks; therefore, all blocks have the same target sparsity level, but the individual layer-level sparsities could be different across different blocks.

Algorithm 1 Block-wise Greedy Optimization

```
Input: Block B, base step size α,
    input X ∈ ℝ^{B×seq×d}, n matrices
# Find size (memory footprint) of matrices
f_i ← size(W_i) for i = 1, ..., n
F ← Σ_{i=1}^n f_i # Find size of block
# Init block and all layer sparsities to zero
p ← 0_n, P ← 0
Y_gt ← B(X) # Forward pass through block B to find ground truth output
while P < 1 do
    for i = 1 to n do
        δ_i ← α · (F/f_i)
        # Error if we further sparsify this layer
        p_i += δ_i
        Ŷ_i ← L(X, p'_i)
        E_i ← ||Y_gt − Ŷ_i||_2
        p_i −= δ_i
    end for
    # Increment layer with lowest error
    j ← argmin_i E_i
    p_j += δ_j
    P ← Σ_{i=1}^n (p_i · f_i)/F
    Record p, P
end while
```

Cost. We describe the cost of our method. The time complexity is $\mathcal{O}(\frac{Mn^2}{\alpha})$ forward passes, where M is the number of samples, n is the number of matrices, and α is the average step size. In practice, we use length 2048 samples and $M, n, \alpha = 10, 7, 0.05$. The resulting cost over all blocks is therefore $10 \cdot 7^2 \cdot \frac{1}{0.05} = 9800$ forward passes, which is less than one GPU-hour on an A100 for Llama-3-8B. It consumes minimal device memory due to its being block-wise and requiring no backpropagation.

4.4 Hardware Aware Acceleration

We develop a specialized sparse GEMV kernel to achieve practical speed-ups, building on the Triton-based (Tillet et al., 2019) kernel introduced by DejaVu (Liu et al., 2023). This kernel takes in an input $\mathbf{x}$, boolean sparsity mask $\mathbf{s}$ and matrix $\mathbf{W}$, and returns $(\mathbf{x} \odot \mathbf{s})\mathbf{W}^\top$. Wall-clock speed-up is realized in three ways: (1) $\mathbf{W}$ is stored in column major format for optimal memory coalescing; (2) Columns $\mathbf{W}_{:,i}$ are selectively loaded based on the truth value of $\mathbf{s}_i$; (3) SplitK work decomposition is used, enabling finer-grained parallelism across thread blocks, combining partial results through atomic adds.

Our kernel makes the following improvements on top of the original kernel: (1) We fuse the mask creation process, as $\mathbf{s} = \mathbf{x}[|\mathbf{x}| > t_p]$ is entirely characterized by $\mathbf{x}$ and t_p in TEAL; (2) We accumulate along the outer SplitK dimension in `FP16` (keeping the inner in-register accumulation in `FP32`), as writing to global memory in `FP32` results in significant traffic; (3) We specify an eviction policy in NVIDIA PTX, prioritizing cache retention for activations which are reused across multiple thread blocks, and deprioritizing weights which are block-specific. This guarantees that activations are persistent in L2 cache.

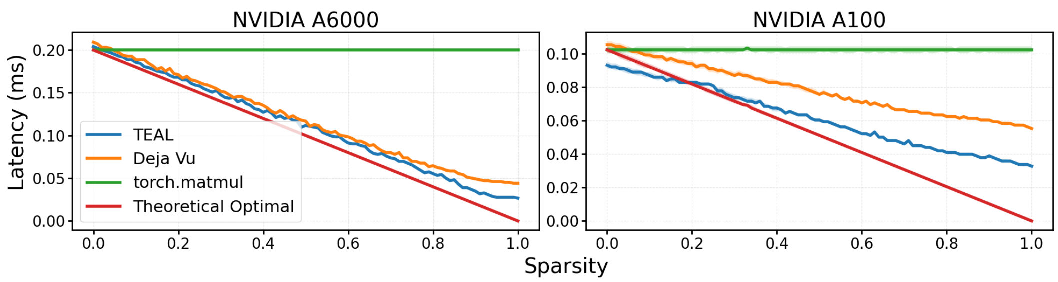

Figure 3: Latency vs. sparsity for matrix-vector multiplication (1x4096 × 4096x14336), comparing TEAL to Deja Vu. 'Theoretical Optimal' shows the latency reduction for `torch.matmul` assuming perfect linear scaling with sparsity.

Figure 3 shows a small speed-up on A6000, and a larger speed-up on A100 over the DejaVu kernel. Note that `torch.matmul` is not the strongest baseline in small batch settings (Hong et al., 2024), which is why our kernel is faster at 0% sparsity for A100. We use a stronger baseline for end-to-end evaluations (Section 5.2). The larger speed-up on A100 can be attributed to its higher memory bandwidth, which amplifies the impact of reducing other overhead factors. These overhead improvements become increasingly important as memory bandwidth across device tiers improves over time, particularly for quantized models and in latency-sensitive or resource-constrained applications.

5 Results

Models and Datasets. We evaluate TEAL on the Mistral (Jiang et al., 2023), Llama-2 (Touvron et al., 2023), and Llama-3 (Dubey et al., 2024) families. We measure the performance of sparsified models on language modeling using the WikiText (Merity et al., 2016) validation set, and on an aggregate of six downstream tasks using the EleutherAI LM Harness (Gao et al., 2023), including 5-shot MMLU, 25-shot ARC challenge, 10-shot HellaSwag, 5-shot GSM8K, zero-shot PiQA, and zero-shot Winogrande (Hendrycks et al., 2021; Clark et al., 2018; Zellers et al., 2019; Cobbe et al., 2021; Bisk et al., 2019; Sakaguchi et al., 2019). For language modeling, we evaluate all models on the same 128 random samples, using a 2048-token context and 512-token evaluation window.

Baselines. We use the block-wise greedily optimized sparsities from Section 4.3 for TEAL, and primarily compare to CATS (Lee et al., 2024a) in its training-free configuration with no finetuning. We report model-level sparsities for all methods.

CATS applies sparsity to MLP parameters, and does not apply sparsity to attention parameters. In particular, CATS sparsifies the output of $\mathbf{W}_{\text{gate}}$, replacing $\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}})$ with $s_{t_p}(\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}}))$

for sparsification function s_{t_p} associated with the distribution of $\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}})$. Overall, CATS sparsifies the intermediate state of the MLP by first performing dense computation on $\mathbf{W}_{\text{gate}}$, enforcing output sparsity on $\mathbf{W}_{\text{up}}$, and then enforcing input sparsity on $\mathbf{W}_{\text{down}}$. This is in contrast with TEAL, which enforces input sparsity on all matrices.

Table 1: Perplexity results. Results between Llama-3 and Llama-2/Mistral are not directly comparable due to differing vocabulary sizes.

	LLaMA-3		LLaMA-2			Mistral
Method / Model	**8B**	**70B**	**7B**	**13B**	**70B**	**7B**
Baseline (0%)	5.87	2.93	5.07	4.50	3.12	4.92
CATS 25%	6.78	3.64	5.52	4.99	3.42	5.87
TEAL 25%	**5.94**	**3.02**	**5.09**	**4.51**	**3.13**	**5.01**
CATS 40%	$7.6 \cdot 10^4$	96.97	43.8	53.9	171	$2.8 \cdot 10^4$
TEAL 40%	**6.21**	**3.52**	**5.22**	**4.60**	**3.25**	**5.13**
TEAL 50%	6.67	4.30	5.43	4.76	3.50	5.31
TEAL 65%	9.06	6.29	6.62	5.50	4.28	6.23

These methods are decoding solutions primarily, but some of the prefill needs to be sparsified for meaningful evaluation on log-likelihood based tasks (such as language modeling and MMLU). For such tasks we sparsify the second half of prefill along the sequence length dimension. See Section 5.4.3 for a more detailed analysis – most degradation in prefill is associated with the initial tokens, which is likely related to the attention sink phenomenon (Xiao et al., 2024), and we thus need to take care not to sparsify them. We do not sparsify prefill on generation tasks (such as GSM8K).

5.1 Accuracy

Main Results. TEAL is performant, as shown in Tables 1 and 2, showcasing near zero degradation at 25%, and minimal degradation at 40% sparsity. At 50% sparsity, Llama-3 variants show slightly more degradation compared to older Llama-2 and Mistral variants which are still fairly performant. This falls in line with prior work showing that quantization techniques are less effective on newer models trained on more tokens (Huang et al., 2024). Most models degrade significantly at 65% sparsity, with the exception of Llama-2-70B which is still reasonably performant. In terms of downstream task results, both of the 70B models are more sparsifiable than their smaller counterparts.

Table 2: Downstream task evaluation results. Reported results are averaged over six tasks. See Appendix A.2 for fine-grained results. We omit CATS 40% as it is degenerate.

	LLaMA-3		LLaMA-2			Mistral
Method / Model	**8B**	**70B**	**7B**	**13B**	**70B**	**7B**
Baseline (0%)	68.07	80.41	56.50	62.01	72.65	66.96
CATS 25%	64.15	79.25	54.60	60.48	71.93	64.25
TEAL 25%	**67.73**	**80.22**	**56.42**	**62.21**	**72.67**	**66.63**
TEAL 40%	66.21	79.29	55.45	61.27	72.57	65.46
TEAL 50%	63.42	78.26	54.26	60.41	72.02	64.16
TEAL 65%	52.59	73.07	48.16	55.71	69.30	58.93

ReLUfication is degenerate in the training-free setting. TEAL outperforms CATS at both 25% and 40% sparsity, which is mainly due to two factors. First and most importantly, TEAL sparsifies every matrix in the model, not just $\mathbf{W}_{\text{up}}$ and $\mathbf{W}_{\text{down}}$, allowing us to moderate sparsity levels across the model. When applied to Llama-2-7B, CATS sparsifies the intermediate state of MLPs to 56.2% at 25% overall sparsity, and to 89.7% at 40% overall sparsity. TEAL avoids such extreme sparsity in any single component. Second, our design choice to use input sparsity instead of output sparsity for $\mathbf{W}_{\text{up}}$ yields lower error, which we analyze in Section 5.4.1.

5.2 End-to-end Decoding Speed-up

We benchmark TEAL's end-to-end single-batch decoding latency by integrating it with GPT-Fast (PyTorch, 2024). We enable CUDA graphs and `torch.compile`. Tests use Llama-2 (7B, 13B) and

Table 3: Single-batch end-to-end inference speed results, measured in tokens per second. We exclude Mistral-7B and Llama-2-70B as they are architecturally similar to Llama-3-8B and 70B. We utilize tensor parallelism for Llama-3-70B: TP2 for A100, and TP4 for A6000.

		LLaMA-3		LLaMA-2	
GPU	Sparsity	**8B**	**70B**	**7B**	**13B**
A6000	Baseline	45.32 (1.00×)	15.93 (1.00×)	50.54 (1.00×)	26.43 (1.00×)
	0%	44.49 (0.98×)	15.57 (0.98×)	50.06 (0.99×)	26.25 (0.99×)
	25%	55.38 (1.22×)	18.93 (1.19×)	64.54 (1.28×)	33.67 (1.27×)
	40%	64.15 (1.42×)	20.86 (1.31×)	77.30 (**1.53**×)	40.20 (1.52×)
	50%	73.94 (1.63×)	23.77 (1.49×)	89.91 (1.78×)	47.60 (**1.80**×)
A100	Baseline	100.79 (1.00×)	21.85 (1.00×)	110.15 (1.00×)	61.01 (1.00×)
	0%	92.13 (0.91×)	20.32 (0.93×)	100.97 (0.92×)	56.33 (0.92×)
	25%	112.11 (1.11×)	25.18 (1.15×)	126.14 (1.15×)	70.66 (1.16×)
	40%	126.24 (1.25×)	28.78 (1.32×)	143.85 (1.31×)	81.90 (1.34×)
	50%	134.29 (1.33×)	29.99 (1.37×)	154.02 (1.40×)	88.38 (1.45×)

Llama-3 (8B, 70B) models at 0%, 25%, 40%, and 50% uniform sparsities. We use GPT-Fast's standard inference benchmarking setup, which passes in roughly 5 input tokens and generates at most 200 output tokens. Our GPU power limit settings are 500W and 300W for A100 and A6000 respectively. As shown in Table 3, TEAL achieves significant speed-ups of up to $1.53\times$ and $1.8\times$ at 40% and 50% sparsity respectively. TEAL is slower than the baseline at 0% sparsity on A100 due to `torch.compile` strengthening the `torch.matmul` baseline. This suggests further room for optimization of our kernel. We find lower speedups for Llama-3-8B compared to Llama-2-7B partially due to its larger LM Head, which we do not currently sparsify.

5.3 Compatibility with Quantization

We demonstrate compatibility with quantization, which is another promising direction for efficient LLM inference. We consider 8-bit channel-wise RTN, 4-bit AWQ (Lin et al., 2024), and 2/3-bit QuIP# (Tseng et al., 2024), and plot the perplexity of Llama-2-7B on WikiText in Figure 4. The point of sharp perplexity degradation is similar across bitwidths, suggesting that errors from activation sparsity and weight quantization compound somewhat independently. Combining activation sparsity with weight quantization unlocks new regimes with respect to memory transferred to GPU registers, allowing for higher inference speed-up. This requires developing specialized sparse + quantized kernels, which we leave to future work.

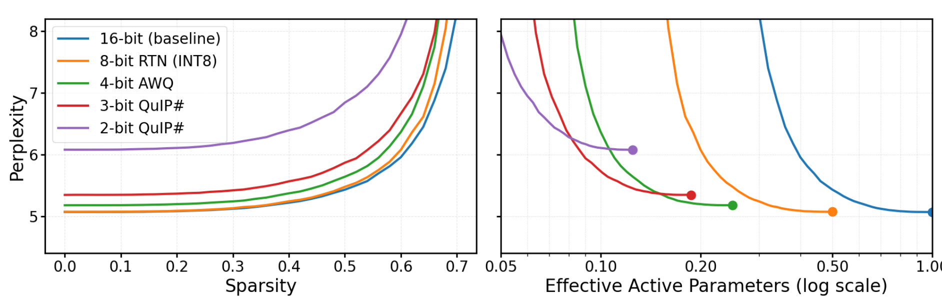

Figure 4: Perplexity vs. sparsity for Llama-2-7B quantized to various bitwidths on WikiText. Left: Performance over sparsity levels. Right: Performance normalized by bitwidth.

5.4 Analysis

5.4.1 Should $\mathbf{W}_{\text{up}}$ have Input or Output Sparsity?

TEAL naturally differs from CATS in its treatment of $\mathbf{W}_{\text{up}}$. TEAL uses input sparsity, whereas CATS uses output sparsity with output mask $\mathbf{s} = s_{t_p}(\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}}^\top))$, with the intuition that SiLU serves as a gating mechanism. We must choose one treatment over the other due to differing memory format constraints (see Section 3).

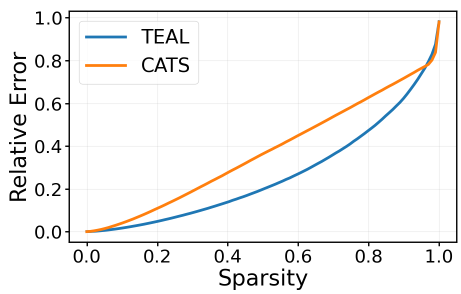

Figure 5: Layer-level activation error for $\mathbf{W}_{\text{up}}$ at Block 16 of Llama-3-8B: TEAL utilizing input sparsity, and CATS utilizing output sparsity.

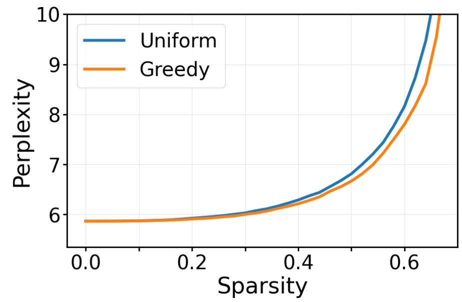

Figure 6: Perplexity of Llama-3-8B on Wiki-Text under uniform and block-wise greedy sparsity configurations.

We analyze the activation error in the intermediate state of MLPs, assuming $\mathbf{W}_{\text{gate}}$ is computed densely, as it is in CATS. The error associated with TEAL is $||(\mathbf{x} - s_{t_p}(\mathbf{x}))\mathbf{W}_{\text{up}}^\top \odot \text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}}^\top)||_2$, and the error associated with CATS is $||\mathbf{x}\mathbf{W}_{\text{up}}^\top \odot [\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}}^\top) - s'_{t_p}(\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}}^\top))]||_2$, where $s_{t_p}(\cdot)$ and $s'_{t_p}(\cdot)$ are sparsification functions associated with $\mathbf{x}$ and $\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}}^\top)$ respectively. We additionally normalize errors by the norm of the unsparsified product. Figure 5 shows that input sparsity outperforms across all levels. This is because output mask $\mathbf{s}$ has no information regarding the saliency of outputs with respect to $\mathbf{W}_{\text{up}}$, which is relevant since SiLU does not threshold exactly to zero. As a result, larger values of $\mathbf{x}\mathbf{W}_{\text{up}}$ may be unnecessarily pruned.

5.4.2 Block-wise Greedy Sparsities

We observe in Figure 6 that the block-level greedy method in Section 4.3 outperforms the uniform configuration across all sparsity levels. The resultant sparsities can be used to analyze the workings of modern LLMs. We make two interesting observations about Llama-3-70B, which tend to hold for the other models we analyze.

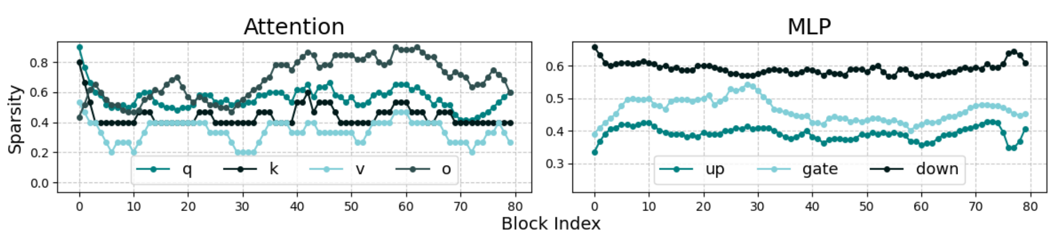

Figure 7: Greedy sparsities for Llama-3-70B at 50% model-level sparsity. Left: Attention parameters. Right: MLP parameters.

Attention: We plot sparsities of $\mathbf{W}_\text{q}, \mathbf{W}_\text{k}, \mathbf{W}_\text{v}, \mathbf{W}_\text{o}$ at 50% model-level sparsity. $\mathbf{W}_\text{q}, \mathbf{W}_\text{k}$ exhibit high sparsifiability in Block 0, followed by a sharp decline. $\mathbf{W}_\text{o}$'s sparsifiability varies dynamically: it starts at 50-60%, peaks at 80-90% mid-model, then returns to 50-60% in the final blocks. The blocks where $\mathbf{W}_\text{o}$ exhibits high sparsifiability seem to align with those of the Attention modules pruned in *FinerCut* (Zhang et al., 2024a), suggesting that the sparsifiability of $\mathbf{W}_\text{o}$ may have some correlation to saliency in Attention modules.

MLP: We plot sparsities of $\mathbf{W}_{\text{up}}, \mathbf{W}_{\text{gate}}, \mathbf{W}_{\text{down}}$ at 50% model-level sparsity. Across all blocks, $\mathbf{W}_{\text{down}}$ is more sparsifiable than $\mathbf{W}_{\text{gate}}$, which is more sparsifiable than $\mathbf{W}_{\text{up}}$. Intuitively, $\mathbf{W}_{\text{down}}$ is sparsifiable as it corresponds to a Laplacian shaped distribution, which is more densely concentrated around zero than a Gaussian shaped distribution. $\mathbf{W}_{\text{gate}}$ may be more sparsifiable than $\mathbf{W}_{\text{up}}$ due to SiLU decreasing the saliency of negative outputs.

5.4.3 Prefill Sparsification

We vary the proportion of prefill sparsified (along the sequence length dimension) in Figure 9. Sparsifying the second half of prefill is nearly identical to sparsifying 99% of prefill (all tokens besides the initial tokens). However, more severe degradation occurs when sparsifying the initial tokens. This is due to attention sinks (Xiao et al., 2024), a phenomenon in LLMs where initial tokens are allocated an outsized amount of attention due to the softmax operation. Degradation to keys and values of initial "attention sink" tokens results in more substantial model degradation due to their greater importance (Hooper et al., 2024).

TEAL is a decoding solution so this is typically not an issue, but care must be taken when sparsifying prefill for evaluation on log-likelihood based tasks.

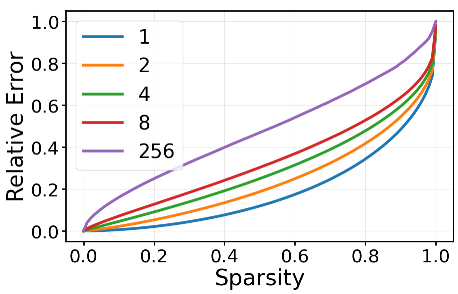

Figure 8: Layer-level activation error for $\mathbf{W}_{\text{down}}$ at Block 16 of Llama-2-7B, at varying batch sizes.

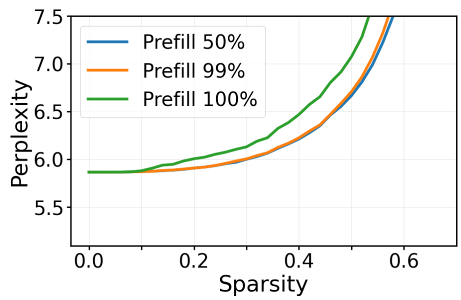

Figure 9: Perplexity of Llama-3-8B on WikiText, varying the proportion of prefill sparsified, using greedy sparsity configurations.

5.4.4 Batched Sparsification

We focus on the single-batch case, but it may be valuable to study activation sparsity in batched settings. The key challenge is that different inputs may prefer different sparsity patterns. We need to find a subset of weight columns associated with activations that are relatively low-magnitude for the entire batch.

We propose to sparsify based on the average magnitude of activations across the batch dimension, a natural extension from the single batch case. The resultant sparsification criterion is batch dependent, but is still entirely characterized by a threshold.

As a preliminary analysis, we find the layer-level activation error for $\mathbf{W}_{\text{down}}$ at Block 16 of Llama-2-7B, ablated across batch sizes, in Figure 8. At low batch sizes above 1, $\mathbf{W}_{\text{down}}$ still exhibits substantial sparsity. For example, in the single batch setting, $\mathbf{W}_{\text{down}}$ is assigned roughly 60% sparsity at 50% model-wide sparsity. To have the same error at batch size 4, $\mathbf{W}_{\text{down}}$ is assigned roughly 38% sparsity. As batch size tends to infinity, TEAL can be interpreted as a structured channel-wise pruning algorithm (Zhao et al., 2023), with a simple pruning metric based on activation magnitude.

6 Applications and Limitations

Applications. The most immediate application of TEAL is accelerating inference in resource constrained edge settings. These settings are typically single-batch, which is where TEAL realizes the most salient speed-up. Furthermore, TEAL is compatible with quantization (Section 5.3), which is another essential axis of efficiency in this setting.

Limitations. TEAL exhibits substantial sparsity in the low-batch setting (Section 5.4.4) but does not scale as well to higher batch sizes, which is a limitation of most activation sparsity work[2]. A way

[2]We note that Wang et al. (2024a) propose to enforce structured n:m sparsity on activations to address batched inference, but this is applicable only if inference is compute bound instead of memory bound, and is outside the scope of our work. A regime where inference is compute bound is with 1.58-bit models (Ma et al., 2024) in high-batch settings.

to alleviate this is to push sparsities higher through continued pretraining. While TEAL focuses on the training-free case, we provide many learnings that can aid future work in sparse aware adaptation.

A setting where batched inference is less difficult is in the low-batch setting of Mixture of Experts (Shazeer et al., 2017) based models, as the baseline itself does not scale well due to having to activate more experts and lowering the arithmetic intensity.

7 Conclusion

We propose TEAL, a simple method that applies magnitude-based activation sparsity to modern LLMs without training, achieving 40-50% model-wide sparsity with minimal degradation. We additionally optimize per-layer sparsity levels, improve existing sparse kernels, and demonstrate compatibility with quantization. We achieve wall-clock speed-ups in single-batch decoding, which is crucial in resource-constrained edge settings. We hope TEAL has impact in real-world applications and enhances our understanding of activation sparsity in LLMs.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. Llm in a flash: Efficient large language model inference with limited memory, 2024. URL https://arxiv.org/abs/2312.11514.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models, 2016. URL https://arxiv.org/abs/1511.06297.

Yoshua Bengio. Deep learning of representations: Looking forward, 2013. URL https://arxiv.org/abs/1305.0445.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

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, 2020. URL https://arxiv.org/abs/2005.14165.

Xuanyao Chen, Zhijian Liu, Haotian Tang, Li Yi, Hang Zhao, and Song Han. Sparsevit: Revisiting activation sparsity for efficient high-resolution vision transformer, 2023. URL https://arxiv.org/abs/2303.17605.

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models, 2022. URL https://arxiv.org/abs/2202.01169.

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.

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.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022. URL https://arxiv.org/abs/2208.07339.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023a. URL https://arxiv.org/abs/2305.14314.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems*, 36, 2023b.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. *arXiv preprint arXiv:2306.03078*, 2023c.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay

Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models, 2024. URL `https://arxiv.org/abs/2409.17481`.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. MARLIN: Mixed-precision auto-regressive parallel inference on large language models. *arXiv preprint arXiv:2408.11743*, 2024.

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

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. Ai and memory wall, 2024. URL https://arxiv.org/abs/2403.14123.

Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric P Xing, and Yoon Kim. Fast matrix multiplications for lookup table-quantized llms. *arXiv preprint arXiv:2407.10960*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference on gpus, 2024. URL https://arxiv.org/abs/2311.01282.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Wei Huang, Xingyu Zheng, Xudong Ma, Haotong Qin, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. An empirical study of llama3 quantization: From llms to mllms, 2024. URL https://arxiv.org/abs/2404.14047.

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

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5533–5543. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/kurtz20a.html.

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

Je-Yong Lee, Donghyun Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. Cats: Contextually-aware thresholding for sparsity in large language models, 2024b. URL https://arxiv.org/abs/2404.08763.

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, 2023. URL https://arxiv.org/abs/2210.06313.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL https://arxiv.org/abs/2306.00978.

James Liu, Guangxuan Xiao, Kai Li, Jason D. Lee, Song Han, Tri Dao, and Tianle Cai. Bitdelta: Your fine-tune may only be worth one bit, 2024. URL https://arxiv.org/abs/2402.10193.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. Deja vu: Contextual sparsity for efficient llms at inference time, 2023. URL https://arxiv.org/abs/2310.17157.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits, 2024. URL https://arxiv.org/abs/2402.17764.

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

Pierre-Alexandre Mattei. Multiplying a gaussian matrix by a gaussian vector, 2017. URL https://arxiv.org/abs/1702.02815.

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

Iman Mirzadeh, Keivan Alizadeh, 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, 2023. URL https://arxiv.org/abs/2310.04564.

Aniruddha Nrusimha, Mayank Mishra, Naigang Wang, Dan Alistarh, Rameswar Panda, and Yoon Kim. Mitigating the impact of outlier channels for language model quantization with activation regularization. *arXiv preprint arXiv:2404.03605*, 2024.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models, 2023. URL https://arxiv.org/abs/2302.10866.

Team PyTorch. Accelerating generative ai with pytorch ii: Gpt, fast, 2024. URL https://pytorch.org/blog/accelerating-generative-ai-2/.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023. URL https://arxiv.org/abs/1910.10683.

Md Aamir Raihan and Tor M. Aamodt. Sparse weight activation training, 2020. URL https://arxiv.org/abs/2001.01969.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*, 2023.

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

David R. So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. Primer: Searching for efficient transformers for language modeling, 2022. URL https://arxiv.org/abs/2109.08668.

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, 2024a. URL https://arxiv.org/abs/2402.13516.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu, 2023. URL https://arxiv.org/abs/2312.12456.

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, 2024b. URL https://arxiv.org/abs/2406.05955.

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, pp. 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.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks, 2024. URL https://arxiv.org/abs/2402.04396.

Hongyu Wang, Shuming Ma, Ruiping Wang, and Furu Wei. Q-sparse: All large language models can be fully sparsely-activated, 2024a. URL https://arxiv.org/abs/2407.10969.

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. 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)*, 2024b. URL https://www.usenix.org/conference/osdi24/presentation/wang-lei.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*, 2019.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems*, 35:17402–17414, 2022.

Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. Flash-LLM: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. In *Proceedings of VLDB*, 2024.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. *arXiv:2211.10438*, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL `https://arxiv.org/abs/2309.17453`.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL `https://arxiv.org/abs/1905.07830`.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022. URL `https://arxiv.org/abs/2205.01068`.

Yang Zhang, Yawei Li, Xinpeng Wang, Qianli Shen, Barbara Plank, Bernd Bischl, Mina Rezaei, and Kenji Kawaguchi. Finercut: Finer-grained interpretable layer pruning for large language models, 2024a. URL `https://arxiv.org/abs/2405.18218`.

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

Kaiqi Zhao, Animesh Jain, and Ming Zhao. Adaptive activation-based structured pruning, 2023. URL `https://arxiv.org/abs/2201.10520`.

A Appendix

A.1 Derivation of Sparsification Error

We derive the error of magnitude-based activation sparsity for the case where $\mathbf{W}$ and $\mathbf{X}$ are independent Gaussian in Theorem A.1. Our error metric is $\frac{\mathbb{E}_{\mathbf{X}}[\|\mathbf{Y}-\hat{\mathbf{Y}}\|_2]}{\mathbb{E}_{\mathbf{X}}[\|\mathbf{Y}\|_2]}$, where $\mathbf{X}$ is the input, $\hat{\mathbf{Y}}$ is the predicted output and $\mathbf{Y}$ is the ground truth output. We plot this error in Figure 10, along with empirical errors on $\mathbf{W}_{\text{up,down}}$ in Block 16 of Llama-3-8B, and the theoretical error obtained from random sparsification.

Definition A.1. *For a random vector $\mathbf{X} = (X_1, \ldots, X_n)$ and sparsity level $p \in [0, 1]$, define the threshold t_p as*

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{P}(|X_i| \leq t_p) = p.$$

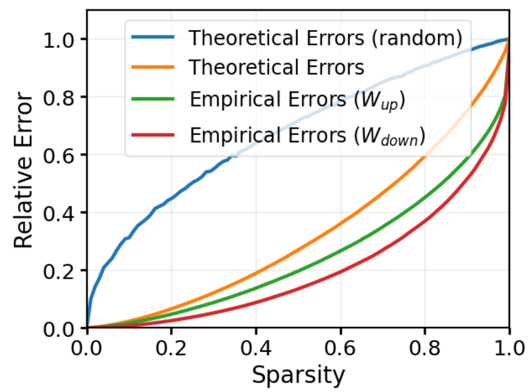

Figure 10: Errors at Block 16 of Llama-3-8B: Gaussian-based theoretical errors from random and magnitude based sparsification, empirical errors from $\mathbf{W}_{\text{up}}$ and $\mathbf{W}_{\text{down}}$.

The sparsification function $s_{t_p} : \mathbb{R}^n \to \mathbb{R}^n$ *is defined as:*

$$s_{t_p}(\mathbf{X}) = (s_{t_p}(X_1), \ldots, s_{t_p}(X_n))$$

where for each component:

$$s_{t_p}(X_i) = \begin{cases} 0 & \textit{if } |X_i| \leq t_p \\ X_i & \textit{otherwise} \end{cases}$$

Lemma A.1 (Variance of Scalar Sparsified Error)**.** *For independent random normal variables* $X \sim N(0, \sigma_X^2), W \sim N(0, \sigma_W^2)$ *and sparsification function* $s_{t_p}(\cdot)$*, the variance of* $(X - s_{t_p}(X))W$ *is given by:*

$$\text{Var}((X - s_{t_p}(X))W) = \sigma_X^2 \sigma_W^2 \left[p - \frac{2t_p}{\sigma_X} \varphi(\frac{t_p}{\sigma_X})\right]$$

where $\varphi(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2}$ *is the probability density function of the standard normal distribution.*

Proof. For $|x| \leq t_p$, $X - s_{t_p}(X)$ follows a truncated normal distribution with lower bound and upper bound given by $-t_p$ and t_p respectively. We thus have:

$$\text{Var}((X - s_{t_p}(X))W) = p\text{Var}(X - s_{t_p}(X) \,\big|\, |X| \leq t_p)\text{Var}(W)$$

$$= \sigma_X^2 p \left[1 - \frac{t\varphi(t) - (-t)\varphi(-t)}{\Phi(t) - \Phi(-t)} - \left(\frac{\varphi(t) - \varphi(-t)}{\Phi(t) - \Phi(-t)}\right)^2\right] \sigma_W^2$$

$$= \sigma_X^2 \sigma_W^2 p \left[1 - \frac{2t\varphi(t)}{2\Phi(t) - 1}\right]$$

$$= \sigma_X^2 \sigma_W^2 \left[p - \frac{2t_p}{\sigma_X} \varphi\left(\frac{t_p}{\sigma_X}\right)\right]$$

where $t = \frac{t_p}{\sigma_X}$, and $\Phi(t) = \frac{1}{2}(1 + \text{erf}(x/\sqrt{2}))$ is the cumulative density function of the standard normal distribution. □

Lemma A.2 (Expected ℓ_2 Norm of Sparsified Matrix-Vector Error)**.** *Let* $\mathbf{X} \in \mathbb{R}^m$ *be a vector where each* $X_i \sim N(0, \sigma_X^2)$*, and* $\mathbf{W} \in \mathbb{R}^{n \times m}$ *be a matrix where each* $W_{ji} \sim N(0, \sigma_W^2)$*, with all entries independent. For a sparsification function* $s_{t_p}(\cdot)$*, let* $\hat{\mathbf{Y}} = (\mathbf{X} - s_{t_p}(\mathbf{X}))\mathbf{W}^\top$*. Then:*

1) The variance of the j*-th entry of* $\hat{\mathbf{Y}}$ *is:*

$$Var(\hat{Y}_j) = n\sigma_X^2 \sigma_W^2 \left[p - \frac{2t_p}{\sigma_X} \varphi\left(\frac{t_p}{\sigma_X}\right)\right]$$

2) The expectation of the ℓ_2 norm of $\hat{\mathbf{Y}}$ is:

$$\mathbb{E}[\|\hat{\mathbf{Y}}\|_2] = \sigma_X \sigma_W \sqrt{mn\left[p - \frac{2t_p}{\sigma_X}\varphi\left(\frac{t_p}{\sigma_X}\right)\right]}$$

where t_p is the threshold value satisfying $F_{|X|}(t_p) = p$, and $\varphi(t) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}t^2}$ is the probability density function of the standard normal distribution.

Proof. For the variance of $\hat{Y}_j$: The j-th entry of $\hat{\mathbf{Y}}$ is the sum of n independent products $(X_i - s_{t_p}(X_i))W_{ji}$. Each product has variance $\sigma_X^2\sigma_W^2[p - \frac{2t_p}{\sigma_X p}\varphi(\frac{t_p}{\sigma_X})]$. Since variances of independent terms add, we multiply this by n to get the result.

For the expectation of $\|\hat{\mathbf{Y}}\|_2$: We first show $\mathrm{Cov}(Y_j, Y_k) = 0$ for $j \neq k$:

$$\mathbb{E}[\hat{Y}_j\hat{Y}_k] = \mathbb{E}\left[\sum_{i=1}^{n}(X_i - s_{t_p}(X_i))^2 W_{ji}W_{ki}\right] = 0$$

$\mathbb{E}[W_{ji}W_{ki}] = 0$ for $j \neq k$ due to independence and zero mean. $\hat{Y}_j$ and $\hat{Y}_k$ are uncorrelated for $j \neq k$ and are thus independent. Therefore:

$$\mathbb{E}[\|\hat{\mathbf{Y}}\|^2] = \mathbb{E}\left[\sum_{j=1}^{m}\hat{Y}_j^2\right] = \sum_{j=1}^{m}\mathbb{E}[\hat{Y}_j^2] = \sum_{j=1}^{m}\mathrm{Var}(\hat{Y}_j)$$

Substituting the variance from part 1, summing over m components, and taking the square-root completes the proof. □

Theorem A.1 (Distributional Relative Error)**.** *Let $\mathbf{X} \in \mathbb{R}^m$ and $\mathbf{W} \in \mathbb{R}^{n\times m}$ with elements independently drawn from $N(0, \sigma_X^2)$ and $N(0, \sigma_W^2)$ respectively. For a sparsification function $s_{t_p}(\cdot)$, define $\hat{\mathbf{Y}} = s_{t_p}(\mathbf{X})\mathbf{W}^\top$ and $\mathbf{Y} = \mathbf{X}\mathbf{W}^T$. The distributional relative error is given by:*

$$\frac{\mathbb{E}_{\mathbf{X}}[\|\mathbf{Y} - \hat{\mathbf{Y}}\|_2]}{\mathbb{E}_{\mathbf{X}}[\|\mathbf{Y}\|_2]} = \sqrt{p - \frac{2t_p}{\sigma_X}\varphi\left(\frac{t_p}{\sigma_X}\right)}$$

where $\varphi(t) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}t^2}$ is the standard normal probabilty density function.

Proof. From the previous theorem, we have $\mathbb{E}_{\mathbf{X}}[\|\mathbf{Y} - \hat{\mathbf{Y}}\|_2] = \sigma_X\sigma_W\sqrt{mn\left[p - \frac{2t_p}{\sigma_X}\varphi(\frac{t_p}{\sigma_X})\right]}$.

For the unsparsified case, we have $\mathbb{E}_{\mathbf{X}}[\|\mathbf{Y}\|_2] = \sigma_X\sigma_W\sqrt{mn}$. Dividing these expectations yields the result. □

A.2 Full Downstream Task Results

We provide the full downstream task results for all evaluated models. For Llama-3-8B in Table 4, we also provide results obtained from the uniform sparsity configuration, showing that the greedy sparsity configuration outperforms across the board. For CATS, we additionally provide the sparsity of the hidden state in the intermediate of the MLP blocks.

Table 4: Full downstream task results for Llama-3-8B.

Method	Sparsity	MMLU	ARC	HellaSwag	GSM8K	PiQA	WinoGrande	Average
Baseline	0	65.08	57.68	82.20	49.81	80.79	72.85	68.07
Uniform	25	64.48	57.34	81.63	48.52	79.92	73.88	67.63
	40	62.35	54.86	79.98	43.59	79.11	72.22	65.35
	50	59.07	53.50	77.60	36.47	79.22	70.17	62.67
	65	43.48	42.58	65.81	16.22	75.84	65.43	51.56
Greedy	25	64.61	57.17	81.77	48.90	80.47	73.48	67.73
	40	62.69	57.08	80.42	43.82	80.47	72.77	66.21
	50	59.68	53.84	78.44	38.06	78.94	71.59	63.42
	65	44.54	43.86	68.85	17.51	77.15	66.06	52.99
CATS	25 (46.45)	61.18	54.61	80.20	39.88	79.76	69.30	64.15

Table 5: Full downstream task results for Llama-3-70B.

Method	Sparsity	MMLU	ARC	HellaSwag	GSM8K	PiQA	WinoGrande	Average
Baseline	0	78.70	69.71	87.94	81.12	84.55	80.43	80.41
Greedy	25	78.44	69.37	87.83	80.36	84.55	80.74	80.22
	40	77.92	68.52	87.35	78.77	83.79	79.40	79.29
	50	76.48	67.75	86.73	78.17	83.08	77.35	78.26
	65	71.39	63.31	83.71	64.44	80.74	74.82	73.07
CATS	25 (45.54)	77.67	67.92	87.75	78.01	84.27	79.87	79.25

Table 6: Full downstream task results for Llama-2-7B.

Method	Sparsity	MMLU	ARC	HellaSwag	GSM8K	PiQA	WinoGrande	Average
Baseline	0	45.78	52.47	78.96	13.95	78.94	68.90	56.50
Greedy	25	45.34	52.56	78.66	14.25	78.78	68.90	56.42
	40	42.81	53.16	78.28	12.66	78.40	67.40	55.45
	50	40.52	52.47	76.54	10.84	77.86	67.32	54.26
	65	31.63	42.83	69.64	4.62	76.55	63.69	48.16
CATS	25 (56.2)	42.05	52.39	78.20	10.24	77.64	67.09	54.60

Table 7: Full downstream task results for Llama-2-13B.

Method	Sparsity	MMLU	ARC	HellaSwag	GSM8K	PiQA	WinoGrande	Average
Baseline	0	54.76	59.39	82.18	23.05	71.98	80.69	62.01
Greedy	25	54.96	59.47	82.31	23.58	72.14	80.79	62.21
	40	54.15	58.02	82.11	22.44	70.72	80.20	61.27
	50	52.13	57.85	81.38	19.71	71.11	80.25	60.41
	65	44.37	53.24	76.95	12.81	68.59	78.29	55.71
CATS	25 (56.02)	52.31	57.76	82.73	20.32	70.24	79.49	60.48

Table 8: Full downstream task results for Llama-2-70B.

Method	Sparsity	MMLU	ARC	HellaSwag	GSM8K	PiQA	WinoGrande	Average
Baseline	0	68.71	67.49	87.02	52.38	82.70	77.58	72.65
Greedy	25	68.80	67.24	86.93	52.38	82.70	77.98	72.67
	40	67.75	66.81	86.88	53.90	82.48	77.58	72.57
	50	66.79	66.72	86.38	52.69	82.59	76.95	72.02
	65	62.75	63.74	84.96	45.41	81.72	77.19	69.30
CATS	25 (45.54)	67.45	66.81	86.96	50.95	82.92	76.48	71.93

Table 9: Full downstream task results for Mistral 7B.

Method	Sparsity	MMLU	ARC	HellaSwag	GSM8K	PiQA	WinoGrande	Average
Baseline	0	62.46	61.43	83.47	38.21	82.10	74.11	66.96
Greedy	25	62.02	61.52	83.35	37.53	81.77	73.56	66.63
	40	61.00	60.84	82.65	33.89	81.28	73.09	65.46
	50	59.02	59.90	81.38	31.62	81.28	71.74	64.16
	65	51.92	54.01	76.79	21.01	80.41	69.46	58.93
CATS	25 (46.45)	60.10	59.81	82.08	29.72	80.41	73.40	64.25

A.3 Transformer Architecture Details

A Transformer block consists of an attention layer followed by a multilayer perceptron (MLP). Each block contains seven weight matrices that process the input $\mathbf{x} \in \mathbb{R}^d$ in sequence:

Attention: In Grouped Query Attention (GQA) (Ainslie et al., 2023), the matrices $\mathbf{W}_{\text{q}} \in \mathbb{R}^{d \times d}$ and $\mathbf{W}_{\text{k}}, \mathbf{W}_{\text{v}} \in \mathbb{R}^{d_h \times d}$ project the input into query, key, and value representations, which are fed into the attention operation. After the attention operation, $\mathbf{W}_{\text{o}} \in \mathbb{R}^{d \times d_h}$ projects the output back to the model dimension.

MLP: The SwiGLU variant uses three matrices: $\mathbf{W}_{\text{gate}}, \mathbf{W}_{\text{up}} \in \mathbb{R}^{d_m \times d}$ which project to a higher dimension, and $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d \times d_m}$ which projects back to the model dimension. The computation flow is:

$$\text{MLP}(\mathbf{x}) = (\text{SiLU}(\mathbf{x}\mathbf{W}_{\text{gate}}^\top) \odot \mathbf{x}\mathbf{W}_{\text{up}}^\top)\mathbf{W}_{\text{down}}^\top$$

where $\text{SiLU}(\mathbf{x}) = \mathbf{x} \odot \sigma(\mathbf{x})$, σ is the sigmoid function, and $\odot$ denotes element-wise multiplication.

A.4 Comparison to 2:4 Weight Sparsity

We compare TEAL to MaskLLM (Fang et al., 2024), a state-of-the-art approach to semi-structured 2:4 weight sparsity that learns weight masks through differentiable relaxation. The authors have not released checkpoints as of the time of writing, so we trained our own masks on Llama-3-8B using 2B tokens from C4. We use the greedily optimized sparsities described in Section 4.3 for TEAL, and evaluate both methods on WikiText:

Model	Perplexity
Llama-3-8B (baseline)	5.870
Llama-3-8B + TEAL (50%)	**6.673**
Llama-3-8B + MaskLLM (2:4)	8.532
Llama-3-8B + TEAL + MaskLLM	9.590

We observe that TEAL outperforms MaskLLM, while being training-free. In contrast, MaskLLM is computationally expensive—training on Llama-3-8B requires 2B tokens and approximately 8B frozen parameters plus 12B learnable parameters, which took us roughly 800 H100 hours. The combination of both methods works well, suggesting they are complementary rather than mutually exclusive.

We additionally compare single-batch decoding speed-up on Llama-2-7B using a single A6000 GPU (using MaskLLM's reported numbers from their Table 6):

Model	Speed-up
Llama-2-7B + TEAL (50%)	**1.78×**
Llama-2-7B + MaskLLM (2:4)	1.4×

This comparison may not be fully representative as 2:4 weight sparsity is more performant in high-batch settings and can additionally accelerate the prefill phase.

A.5 Compatibility with Fine-tuning

Table 10: Perplexity results with and without fine-tuning on Llama-3-8B.

Sparsity Level	PPL (No Fine-tuning)	PPL (With Fine-tuning)
0% (baseline)	5.870	—
50%	6.673	6.622
60%	7.827	7.515
70%	13.39	9.927
90%	$4.141 \cdot 10^5$	$4.589 \cdot 10^3$

While TEAL is primarily designed as a training-free method, it can be further enhanced with fine-tuning. We fine-tune Llama-3-8B using LoRA (Hu et al., 2021) with a rank of 32 (approximately 1% of parameters are trainable) and a learning rate of 0.0002. The model is fine-tuned on 30M tokens from C4. We evaluate on WikiText and use the greedily optimized sparsities described in Section 4.3.

We observe in Table 10 that fine-tuning provides marginal improvements at lower sparsity levels (50-60%). The benefits are more pronounced at higher sparsity levels (70-90%), where fine-tuning helps to recover some of the performance lost due to aggressive sparsification.

A.6 Calibration Set Sensitivity

We evaluate the sensitivity of TEAL with respect to calibration set. We measure the perplexity of Llama-3-8B on WikiText at 50% model-wide sparsity using our block-wise greedy optimized sparsity levels. We consider two distinct datasets for calibration: C4 (Raffel et al., 2023) and FineWeb (Penedo et al., 2024). These datasets are used both for determining thresholds and for the greedy optimization process described in Section 4.3.

Table 11: Perplexity of Llama-3-8B on WikiText using different calibration datasets. Results show minimal variation across calibration sets.

Dataset	Perplexity
Baseline (0% sparsity)	5.870
C4 (50% sparsity)	6.673
FineWeb (50% sparsity)	6.681

In Table 11 we observe little variation between different calibration sets. Each hidden state only has one degree of freedom (one threshold per hidden state), meaning it's difficult to overfit to a given calibration set.