CATS: Contextually-Aware Thresholding for Sparsity in Large Language Models

Je-Yong Lee[*,1] **Donghyun Lee**[*,2] **Genghan Zhang**[3] **Mo Tiwari**[3]

Azalia Mirhoseini[3]

Abstract

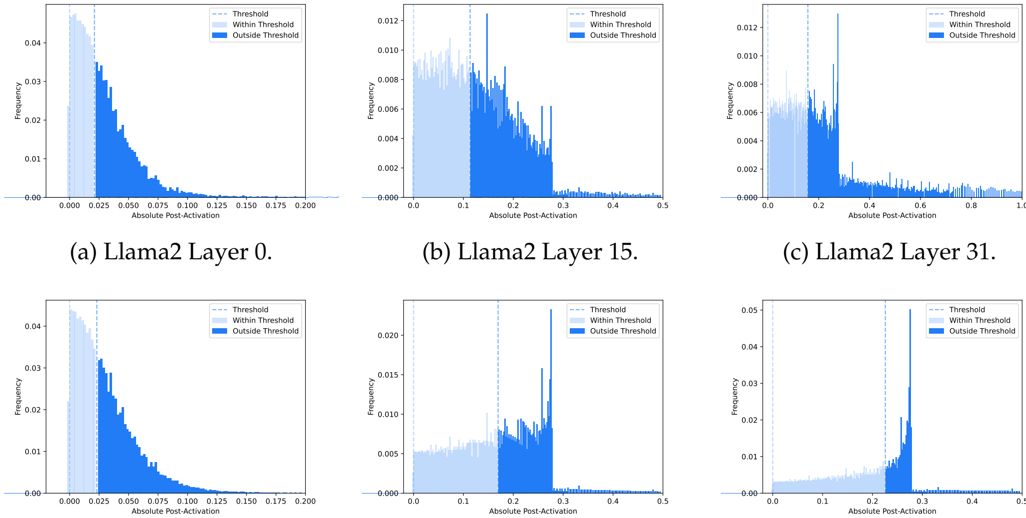
Large Language Models (LLMs) have dramatically advanced AI applications, yet their deployment remains challenging due to their immense inference costs. Recent studies ameliorate the computational costs of LLMs by increasing their activation sparsity but suffer from significant performance degradation on downstream tasks. In this work, we introduce a new framework for sparsifying the activations of base LLMs and reducing inference costs, dubbed Contextually Aware Thresholding for Sparsity (CATS). CATS is relatively simple, easy to implement, and highly effective. At the heart of our framework is a new non-linear activation function. We demonstrate that CATS can be applied to various base models, including Mistral-7B and Llama2-7B, and outperforms existing sparsification techniques in downstream task performance. More precisely, CATS-based models often achieve downstream task performance within 1-2% of their base models without any fine-tuning and even at activation sparsity levels of 50%. Furthermore, CATS-based models converge faster and display better task performance than competing techniques when fine-tuning is applied. Finally, we develop a custom GPU kernel for efficient implementation of CATS that translates the activation of sparsity of CATS to real wall-clock time speedups. Our custom kernel implementation of CATS results in a ∼15% improvement in wall-clock inference latency of token generation on both Llama-7B and Mistral-7B.

1 Introduction

LLMs have demonstrated remarkable success across a variety of fields (Devlin et al., 2018; Brown et al., 2020; Achiam et al., 2023; Brohan et al., 2023). However, the scientific progress achieved by these models comes with significant costs. The training of GPT-3 is estimated to have consumed over 3,000,000 GPU-hours and emitted three thousand times the CO_2 equivalent of a round-trip flight from San Francisco to New York (Patterson et al., 2021). Furthermore, inference costs often eclipse training costs for models that serve trillions of queries. As such, there is significant interest in reducing the inference costs of LLMs while preserving task performance.

Various techniques have been proposed to mitigate LLM inference costs. These approaches are often based on quantization (Frantar et al., 2022; Dettmers et al., 2022), pruning (Ma et al., 2023; Sun et al., 2023), and other forms of weight sparsification Frantar & Alistarh (2023). Mixture of Experts (MoE) techniques have emerged as particularly promising and are employed by current state-of-the-art LLMs (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022c; Jiang et al., 2024).

∗ denotes Equal contribution
[1]Mathematical Institute, Oxford University, je-yong.lee@worc.ox.ac.uk
[2]Department of Computer Science, University College London, lukeleeai@gmail.com
[3]Department of Computer Science, Stanford University, zgh23,motiwari,azalia@stanford.edu



(a) Llama2 Layer 0. (b) Llama2 Layer 15. (c) Llama2 Layer 31.

(d) Mistral7B Layer 0. (e) Mistral7B Layer 15. (f) Mistral7B Layer 31.

Figure 1: Histogram of post-MLP activations different layers in different models. Subfigures (a), (b), and (c) correspond to Layers 0, 15, and 31 in Llama2-7B, respectively. Subfigures (d), (e), and (f) correspond to Layers 0, 15, and 31 in Mistral-7B, respectively. The absolute threshold indicates 50% sparsity, where values smaller than the threshold are considered negligible in our technique and thus zeroed out.

MoE techniques activate only a subset of parameters at each inference stage, thereby reducing memory and computational requirements compared to using the entire model. Prevailing implementations of MoE techniques introduce many multi-layer perceptrons (MLPs; the "experts") and dynamically select which experts to multiply with the hidden vector. This selection is performed by a "router"–a small neural network trained to determine the appropriate experts to activate based on the input (Lewis et al., 2021; Rajbhandari et al., 2020).

Concurrently, recent work has observed that activations in the MLP blocks of LLMs are sparse Liu et al. (2023b); Mirzadeh et al. (2023). This implies that only a few rows (or columns) of the corresponding weight matrices are required for the forward pass. Intuitively, if we could predict *a priori* which elements of the weight matrices were unnecessary via an oracle, we could obviate their respective computations. This is thematically similar to MoE approaches: the activated neurons of the weight matrices can be viewed as activated "experts" and the oracle can be seen as the "router."

We observe that the activation patterns of common LLMs suggest a path to such an oracle. Figure 1 shows a histogram of the post-MLP activations for Layers 0, 15, and 31 for Llama-7B and Mistral-7B on a sample of 500 datapoints from the RefinedWeb dataset (Penedo et al., 2023). Many of the activations are concentrated about 0; if these activations could be made exactly 0, the corresponding weights of the MLP blocks could be made unnecessary during inference. It is this observation that motivates our study.

In this work, we make the following **contributions**:

1. We draw a connection between the MoE framework and multiplication performed by dense matrices in the MLP blocks of LLMs.

2. We motivate the development of a new sparsification procedure based on a novel activation function, dubbed CATS (for Contextually Aware Thresholding for Sparsity), motivated by an empirical evaluation of activation distributions (Figure 1). Crucially, CATS allows for a controllable level of sparsity.

3. We demonstrate that, without any fine-tuning, CATS can be applied to various base models, including Mistral-7B and Llama2-7B, and achieves comparable downstream task performance even at sparsity levels as high as 50%.
4. We demonstrate that, with fine-tuning, CATS outperforms existing state-of-the-art sparsification techniques in downstream task performance at the same sparsity level and number of fine-tuning steps.
5. We provide a custom GPU kernel implementation that exploits the sparsity of CATS and achieves a $\sim$15% improvement in wall-clock inference latency of token generation over the dense base models.

2 Related Work

Significant recent work focuses on reducing the inference costs of LLMs. Approaches that utilize mixture-of-experts or activation sparsity are most similar to our work.

Mixture-of-Experts (MoE) techniques induce effective sparsity in LLMs by determining which subset of subnetworks (the "experts") to activate during the inference pass, often via a trained "router" subnetwork. This is a popular line of work with significant research interest (Shazeer et al., 2017; Hazimeh et al., 2021; Zhou et al., 2022; Lewis et al., 2021; Roller et al., 2021; Zuo et al., 2021; Komatsuzaki et al., 2022; Lou et al., 2021; Mustafa et al., 2022; Rajbhandari et al., 2022; Zhang et al., 2022a;b; Fedus et al., 2022a; Zoph et al., 2022; Kudugunta et al., 2021; Fedus et al., 2022c; Lepikhin et al., 2020; Du et al., 2022; Fedus et al., 2022b; Jiang et al., 2024). For a review of MoE models, we refer the reader to (Fedus et al., 2022a).

Activation Sparsity: Activations are known to be sparse in LLMs that utilize ReLU non-linearities in their MLP blocks (Li et al., 2022); however, the reasons for this are not well-understood Hoefler et al. (2021). Nonetheless, activation sparsity induced by ReLU non-linearities has been explored to reduce memory usage and inference time (Jaszczur et al., 2021; Liu et al., 2023b; Szatkowski et al., 2023). Recent work in this area has framed the rows of weight matrices in MLP layers as experts, similar to our work, and/or deploys a small neural network to predict which activations will be non-zero to reduce inference costs (Zhang et al., 2024; Liu et al., 2023b) in these ReLU-based models.

Crucially, however, recent state-of-the-art LLMs such as Mistral-7B (Jiang et al., 2023), Llama2-7B (Touvron et al., 2023), and Gemma (Team et al., 2024)) employ MLP blocks based on more complex nonlinearities that do not inherently induce sparsity Mirzadeh et al. (2023). As such, most of the work on ReLU-based activation sparsity is inapplicable to these models. To the best of our knowledge, ReLUfication is the only work that attempts to bridge this gap (Mirzadeh et al., 2023). ReLUfication replaces the SiLU and GeLU activation functions in LLMs with ReLU to induce sparsity. ReLUfication is the primary baseline against which we compare CATS. In contrast with ReLUfication, CATS contains a controllable level of sparsity. Furthermore, in Section 5, we demonstrate that CATS demonstrates significantly better downstream task performance and fine-tuning efficiency than ReLUfication.

We note that Zhang et al. (2024) is concurrent to our work. In contrast with their work, however, our work is not an empirical evaluation of existing activation functions. Rather, we propose a new framework for sparsifying LLMs. Our framework utilizes a novel activation function and enables controllable sparsity. We validate the performance of CATS in extensive evaluations and provide a custom GPU kernel that translates CATS' sparsity to real wall-clock time gains in Section 5.

We discuss additional research areas on LLM efficiency, such as quantization, structure pruning, knowledge distillation, and hardware-aware optimization in Appendix A.

3 Background

Motivation: As described in Section 1, MoE models selectively activate expert subnetworks via a trained router. Crucially, we may view the rows (or columns) of MLP layers as experts in an MoE model. To identify the layers most likely to benefit from this MoE view (where many activations can be zeroed), we examine the activations of different layers in LLMs. Figure 1 demonstrates that activations of the Gated-MLP layers tend to concentrate around zero across different LLMs. This behavior suggests that many neurons of MLP layers minimally affect the output in future operations.

Gated-MLP Blocks: We now describe the components of LLMs that our work aims to accelerate: the Gated-MLP blocks. Gated-MLP blocks are commonly used in LLMs, including in the Llama2 family of models, Mistral-7B, and Gemma. A Gated-MLP block consists of several fully-connected layers and performs the following computation:

$$\text{Gated-MLP}(x) := (\text{SiLU}(xW_{\text{gate}}) * (xW_{\text{up}}))W_{\text{down}} \tag{1}$$

where $x \in \mathbb{R}^{b\times d}$, $W_{\text{up}} \in \mathbb{R}^{m\times d}$, $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d\times m}$, $*$ indicates elementwise multiplication, and

$$\text{SiLU}(x) := x * \text{sigmoid}(x) = \frac{x}{1+e^{-x}} \tag{2}$$

Crucially, the operation $\text{SiLU}(xW_{\text{gate}})$ can be viewed as the router in an MoE model. Under this lens, the columns of W_{up} and the rows of W_{down} are the experts. If $\text{SiLU}(x)$ is always binary, i.e., 1 or 0, it would turn on/off elements of the remaining computation (multiplication by $W_{\text{up}}W_{\text{down}}$). When $\text{SiLU}(x)$ is not binary, it can be viewed as a "soft" router that weighs the experts by different amounts.

4 Method: Contextually-Aware Thresholding for Sparsification (CATS)

We now describe CATS, a framework to accelerate the Gated-MLP blocks of LLMs. The CATS framework proposes a new activation function and exploits the sparsity induced by this activation. In Section 5, we apply CATS to Mistral-7B and Llama2-7B and show that CATS-based models still exhibit significant activation sparsity, even when fine-tuned.

4.1 Stage 1: Determining Cutoff Threshold

We assume we are given a desired sparsity level k (e.g., 70%) as input. For each Gated-MLP block in the LLM, we compute the activations over a random subset of the training data. We then compute the *cutoff threshold* as the kth percentile of the resulting values.

More formally, the cutoff threshold t is

$$t := \min\{t' : F(t') \geq k\} \tag{3}$$

where F is the empirical CDF of activations' absolute values for the given MLP block.

Figure 1 shows histograms of the absolute values of activations of the different MLP block in different models over the RefinedWeb dataset (Penedo et al., 2023). A sparsity level of 70% corresponds to a threshold of approximately 0.15; different sparsity levels correspond to different thresholds. We note that these thresholds are chosen and fixed before any further fine-tuning.

4.2 Stage 2: Sparsifying Gate-MLP Blocks

Given the cutoff threshold $t \geq 0$ corresponding to the input sparsity level k, we wrap the $\text{SiLU}(x)$ activations in each MLP block with the CATS activation. The CATS operation, denoted as $\text{CATS}_t(\cdot)$, is defined as:

$$\text{CATS}_t(\mathbf{x})_j := \begin{cases} x_j, & \text{if } |x_j| \geq t \\ 0, & \text{if } |x_j| < t \end{cases} \tag{4}$$

Here, t is the sparsification threshold and x_j is the j-th element of vector $\mathbf{x}$, respectively.

This results in a new activation $\text{CATS}_t(\text{SiLU}(\cdot))$:

$$\text{CATS}_t(\text{SiLU}(xW_{\text{gate}})) = \begin{cases} \text{SiLU}(xW_{\text{gate}}) & |\text{SiLU}(xW_{\text{gate}})| \geq t \\ 0 & |\text{SiLU}(xW_{\text{gate}})| < t \end{cases} \tag{5}$$

Intuitively, the resulting model zeros out activations which were likely to be close to 0 because their corresponding inputs were small. This procedure results in a trained model whose activations are sparse and whose performance may then be evaluated. We empirically validate that this procedure results in a model with sparsity level approximate k, even after fine-tuning, in Appendix C.

4.3 Custom Kernel Design

The previous subsections describe the procedure for sparsifying LLM's activations, obviating computations, and reducing the required number of floating point operations (FLOPs) in each MLP block. We now translate the reduction in FLOPs to a reduction in actual wall-clock latency and increase in generation throughput via a custom GPU kernel.

Custom GPU Kernel 1 MLP using CATS

1: **Input:** threshold $t > 0$, hidden layer x, weights W_{gate}, W_{down}, and W_{up}
2: $v \leftarrow \text{SiLU}(xW_{gate})$
3: $\texttt{Mask} \leftarrow 1$ if $|v| \geq t$ else 0
4: $x_1 \leftarrow (xW_{\text{up}}[\texttt{Mask}] * v[\texttt{Mask}])$
5: $y \leftarrow x_1 W_{\text{down}}[\texttt{Mask}]$

We focus on reducing the latency of the MLP blocks by reducing memory accesses because the MLP blocks are known to be memory-bound during inference (Kim et al., 2023). As shown in Line 5 of the Custom GPU Kernel 1, we first fuse the element-wise multiplication of $v[\texttt{Mask}]$ into each tiling of $xW_{\text{up}}[\texttt{Mask}]$ where v is the hidden vector after the SiLU activations and `Mask` is a binary mask that labels the elements of v with large absolute value. This fusion saves memory operations that would be necessary for storing and loading x_1 several times. We then directly use `Mask` to control which parts of the weight matrices W_{up} and W_{down} to load, instead of using the compressed indices directly as in Zhang et al. (2023) This further improves the kernel speed because it avoids expensive synchronization operations. We demonstrate the success of this custom GPU kernel at reducing the inference latency of CATS-based models as the sparsity increases in Section 5.2.

5 Experiments

In this section, we describe the experiments with which we assess the performance of CATS. We first describe the experimental details that are common to all experimental settings. We then describe experiments on downstream task performance. Finally, we measure CATS' effect on wall-clock time inference when implemented with the custom GPU kernel from Section 4. We find that CATS-based models outperform the baseline models and their ReLUfication versions in downstream task performance, with or without fine-tuning, and can exploit their sparsity for wall-clock inference time speedups over the base models.

We first describe the experimental setup, including base models, CATS-based models, metrics, datasets, and computational environment.

Base Models: We apply CATS to both Mistral-7B and Llama2-7B as base models to verify it is generally applicable to different LLMs. We measure the performance of each CATS model against the original base model. We also compare the performance to of the CATS-based models to the base model transformed by ReLUfication from Mirzadeh et al. (2023).

CATS-based Models: For a given base model, we train three CATS-based variants that exhibit different sparsity levels in the MLP blocks: 50%, 70%, and 90% activation sparsity. We call these models CATS 50%, CATS 70%, and CATS 90%, respectively, where the base models are clear from context.

Model ╲ Dataset	WG acc↑	PIQA acc↑	SciQ acc↑	QA acc↑	HS acc↑	BoolQ acc↑	Arc-E acc↑	Arc-C acc↑	Avg acc↑
Mistral-7B	0.7419	0.8069	0.959	0.3260	0.6128	0.8370	0.8085	0.5034	0.6994
CATS 50%	0.7245	0.8009	0.948	0.3200	0.6097	0.8193	0.7849	0.5043	0.6890
CATS 70%	0.7190	0.8003	0.929	0.292	0.6057	0.8028	0.7492	0.4693	0.6709
CATS 90%	0.5627	0.6001	0.422	0.212	0.3359	0.7086	0.3754	0.2773	0.4368
ReLUfication	0.5043	0.5092	0.236	0.142	0.2580	0.4208	0.2723	0.2415	0.3230
Llama2-7B	0.6906	0.7807	0.94	0.314	0.5715	0.7774	0.7630	0.4343	0.6589
CATS 50%	0.6748	0.7693	0.927	0.322	0.5711	0.7263	0.7441	0.4121	0.6433
CATS 70%	0.6693	0.7584	0.902	0.294	0.5500	0.6590	0.7008	0.3805	0.6143
CATS 90%	0.5738	0.6627	0.611	0.212	0.3848	0.6284	0.4566	0.2816	0.4764
ReLUfication	0.4893	0.5408	0.2570	0.154	0.2586	0.6003	0.2795	0.2406	0.3525

Table 1: Zero-shot downstream task performance of base models, CATS-based models, and ReLUfication across benchmarks. CATS and ReLUfication are applied to base models without any further fine-tuning. CATS maintains base-level performance at 50% sparsity in terms of average accuracy and outperforms ReLUfication at higher sparsity levels.

Metrics: We compare models using several metrics. In the first set of experiments, we compare each model's accuracy on downstream tasks. In the second set of experiments, we compare each model's wall-clock time inference latency.

Datasets: For the downstream task performance experiments, we use the OpenBookQA, ARC_Easy, Winogrande, HellaSwag, ARC_Challenge, PIQA, BoolQ, and SCI-Q datasets from the Eleuther AI Evaluation Harness (Gao et al., 2023) as in Mirzadeh et al. (2023) for ease of comparison; these tasks were originally chosen to measure various abilities of the models across various domains, such as reading comprehension and reasoning. For the latency experiments, we assess the wall-clock inference time on the RefinedWeb test dataset (Penedo et al., 2023).

Computational Environment: All experiments were run on a single machine with 8 L40S GPUs. Latency experiments were run on a single L40S GPU as each 7B base model was able to fit in a single GPU RAM when performing inference in brain float 16 (BF16) or floating point 16/32 (FP16/32) precision. We used DeepSpeed with BF16 precision to manage the high memory overhead during training. We also employed LoRA and targeted 1% of the parameters (Query and Key in attention modules, W_{gate}, and W_{down}) in the fine-tuning experiments. During inference, we used the `transformers v4.36.2` HuggingFace library, `PyTorch v2.1.2`, and `CUDA v12.1`. We used `Triton v2.1.0` for our GPU kernels. All experiments were run in FP32 precision; changing this to FP16 did not materially affect results. All of our code, including a one-line script to set up an environment and reproduce all of our results, is available in the supplementary material.

5.1 Downstream Task Performance

We now compare the downstream task performance of the CATS-based models to the baseline models in several settings and draw several conclusions.

CATS-based models perform comparably to the base models and outperform ReLUfication in zero-shot accuracy without any fine-tuning: We first compare the performance of CATS-based models to the baseline models without any fine-tuning. In this setting, the CATS prescription is applied directly to the base models, i.e., the activation functions are simply replaced in the MLP blocks and no fine-tuning is performed. Table 1 shows our results across 8 different benchmark tasks. CATS-based models demonstrate performance comparable to the unchanged, out-of-the-box base models, even at high sparsity levels. In particular, at CATS 50% demonstrates performance comparable to the base model. CATS significantly outperforms ReLUficiation in downstream task performance at the same sparsity level (90%).

CATS-based models perform comparably to the base models and outperform ReLUfication in zero-shot accuracy with "general" fine-tuning: In this setting, CATS is applied the base models Llama-7B and Mistral-7B. All models are then fine-tuned on the RefinedWeb dataset Penedo et al. (2023); their downstream performance is then measured on the 8

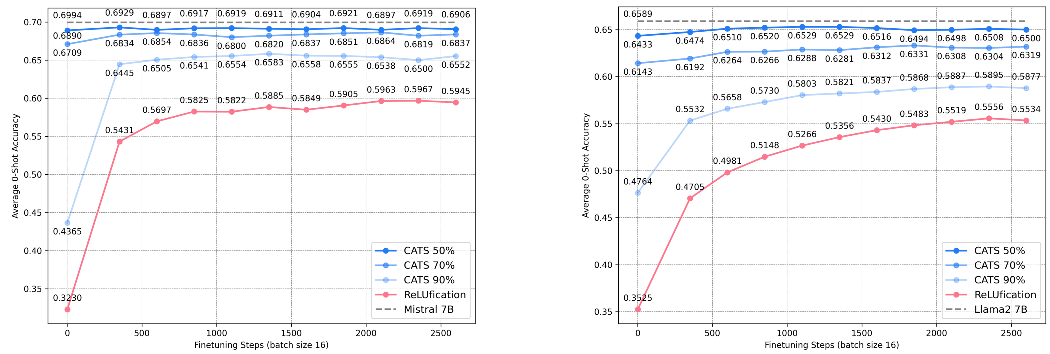


Figure 2: Downstream task performance of the base model, CATS models with different sparsity levels, and ReLUfication versus number of fine-tuning steps on the RefinedWeb dataset applied to Mistral-7B (left) and Llama2-7B (right). The CATS models demonstrate faster convergence and better fine-tuning efficiency than the ReLUfication variants. Furthermore, CATS-50% and CATS-70% demonstrate comparable performance to the base models without any fine-tuning (0 fine-tuning steps).

evaluation datasets. We emphasize that the dataset upon which the models are fine-tuned is different from the evaluation datasets in this setting. Figure 2 demonstrates our results. We note several key observations:

1. CATS-based models still exhibit sparsity after fine-tuning (see Appendix C).
2. CATS-50% demonstrates performance comparable to the base models, even without any fine-tuning (0 fine-tuning steps). This is in contrast with ReLUficiation, which demonstrates poor performance without fine-tuning.
3. CATS-50%, CATS-70%, and CATS-90% all display better task performance than ReLUfication when controlling for the number of fine-tuning steps. In particular, even with very few fine-tuning steps, the CATS-based models achieve comparable performance to the base models.
4. CATS-based models, even with sparsity levels as high as 70%, achieve performance comparable to the base models within 500 steps of fine-tuning, whereas ReLUfication does not.

CATS-based models perform comparably to the base models and outperform ReLUfication in zero-shot accuracy with task-specific fine-tuning: In this setting, the CATS prescription is applied to Mistral-7B. All variants are then fine-tuned for 10 epochs on the training data and evaluated on test dataset for the Cola, SST2, and BoolQ datasets. Table 2 demonstrates our results. Our observations are similar to those for "general" fine-tuning:

1. CATS-based models still exhibit sparsity after fine-tuning (see Appendix C).
2. CATS-50% demonstrates performance comparable to the base models. This is in contrast with ReLUficiation, which demonstrates a significant performance degradation.
3. CATS-50%, CATS-70%, and CATS-90% all display better task performance than ReLUfication.

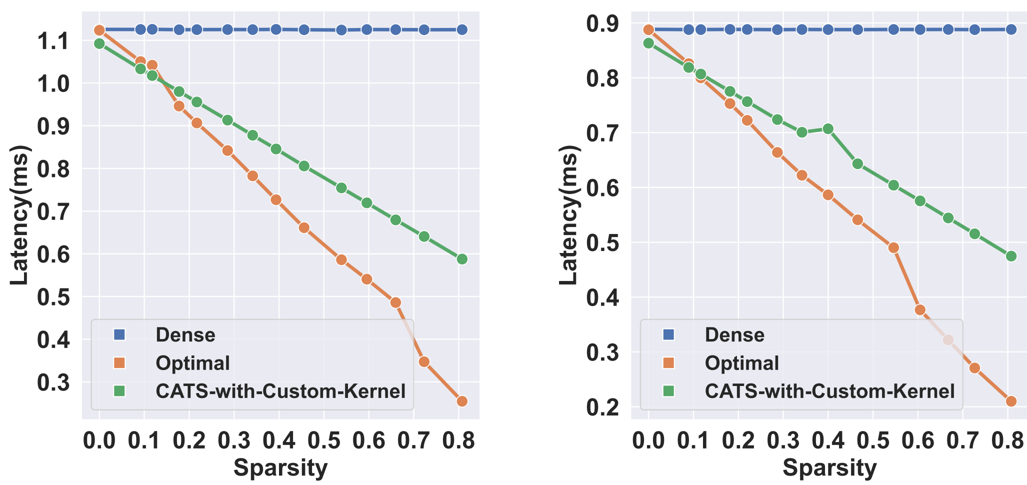
5.2 Wall-clock Time Speedups for Inference

Activation sparsity of a model is not sufficient to directly enable wall-clock time inference speedups (Frantar & Alistarh, 2023). In this subsection, we demonstrate that our custom GPU kernel translates the activation sparsity induced by CATS to real wall-clock time gains.

CATS-based models can translate their activation sparsity to wall-clock time speedups:

Dataset/Sparsity	Base Model	0.5	0.7	0.9	ReLUfication
Cola	**0.8667**	0.8658 (-0.10%)	0.8552 (-1.32%)	0.8303 (-4.21%)	0.6922 (-20.13%)
SST2	0.9644	0.9656 (+0.12%)	**0.9702** (+0.60%)	0.9427 (-2.25%)	0.7856 (-18.55%)
BoolQ	**0.8905**	0.8862 (-0.48%)	0.8807 (-1.10%)	0.7920 (-11.06%)	0.6624 (-25.61%)
Average	**0.9072**	0.9059 (-0.13%)	0.9020 (-0.52%)	0.8550 (-5.22%)	0.7134 (-19.38%)

Table 2: Downstream task performance of Mistral-7B and its CATS-based and ReLUfication variants across three different benchmark datasets. Top accuracies are marked in bold and second-highest in underline. Relative performance degradation is given in parentheses. CATS-50% demonstrates performance within 0.5% of the base model, whereas ReLUfication demonstrates a significant performance drop.



(a) Mistral-7B (b) Llama2-7B

Figure 3: Latency of the original Mistral-7B MLP block (left, "Dense"), Llama-7B MLP block (right, "Dense"), and their CATS-based variants at different sparsity levels, compared to "Optimal." Our custom GPU kernel improves the latency of the CATS-based variants and achieves performance close to "Optimal" for most reasonable sparsity levels.

Figure 3 shows the wall-clock inference time of of the dense model compared to CATS implemented via the custom GPU kernel descripted in Section 4.3, for various sparsity levels of CATS. We evaluate the latency of a single MLP block and the throughput of the generation stage of the end-to-end inference. Mistral-7B contains 32 MLPs with $m = 14336$ and $d = 4096$, and Llama2-7B contains 32 MLPs with $m = 11008$ and $d = 4096$ (m and d are defined after Equation 1).

In Figures 3a and 3b, we compare our method ("CATS-with-Custom-Kernel") with the dense MLP with $m_{dense} = m$ ("Dense") and the dense MLP with $m_{optimal} = m * Sparsity$ ("Optimal"), the latter of which is a proxy for the best wall-clock time we could hope to achieve. At 50% (respectively, 70%) sparsity, the sparse kernel achieves a ~40% (respectively, ~70%) speedup over the original dense MLP. Latency measurements are obtained by doing 20 rounds of warmups, repeating the kernel 80 times, and computing the geometric mean of the latency of each round. The comparison with Dense shows that our sparse kernel can consistently outperform the original MLP. The comparison with Optimal shows that our sparse kernel is close to the Optimal when the sparsity level is low. As the sparsity level increases, the gap between our performance and Optimal increases, as is expected. We note that our sparse kernel performs the same number of memory access as the Optimal but, due to difference in access patterns, the different methods result in different wall-clock time measurements. We note that Optimal can be worse than our sparse kernels when $m_{optimal}$ is not the shape for which GPU libraries have optimized (Tillet & Cox, 2017). Our

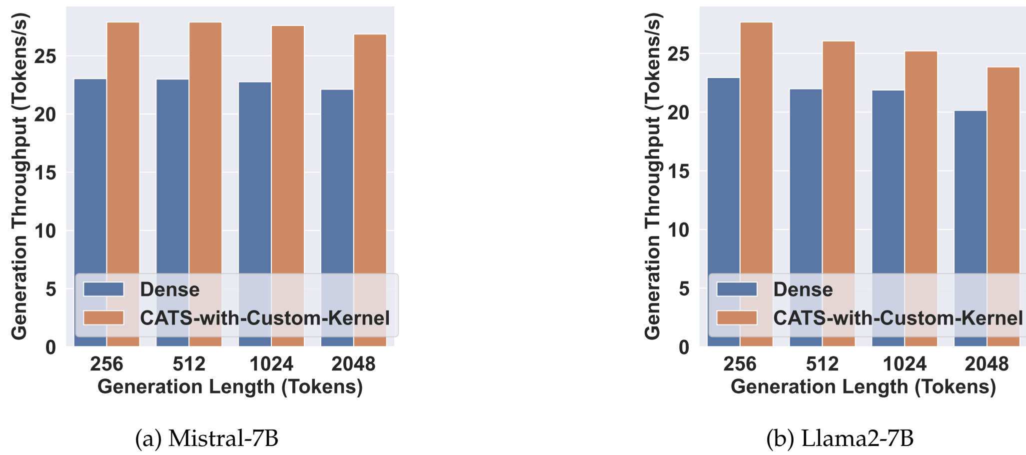


Figure 4: Throughput of Mistral-7B (left, "Dense") and Llama2-7B (right, "Dense") and CATS-50% with the custom GPU kernel. CATS-50% demonstrates significantly higher throughput.

sparse kernel can be worse than Optimal when the overhead of operations on zero values outweighs the benefit of reduced memory access.

In Figures 4a and 4b, we compare dense models with CATS-with-Custom-Kernel (50% sparsity) on the throughput of the generation stage. The generation stage (or "decoding" stage) is known to be memory-bound (Kim et al., 2023), which suggests CATS can improve inference througput. We test the generation throughput at batch size 1 and beam width 1, and record the latency from the first generated token to the last token. The throughput is calculated by the generated length over latency. The final throughput is averaged (geometric mean) over 50 samples from the RefinedWeb test dataset. CATS can accelerate the generation stage by $\sim$18% for Llama2-7B and $\sim$21% for Mistral-7B at 50% sparsity.

Though we only test on Huggingface (Wolf et al., 2020), our methodology is orthogonal to the framework and thus can be used in other LLM serving systems such as DeepSpeed (Rajbhandari et al., 2022) and TensorRT-LLM (Nvidia, 2024).

6 Discussion and Conclusion

We presented CATS, a novel framework for inducing and exploiting activation sparsity in LLMs. At the heart of our framework is the CATS activation, given in Equation 5, that induces a controllable level of activation sparsity in LLMs. We also provide a custom GPU kernel implementation that exploits CATS's sparsity to achieve real wall-clock time gains in inference latency.

CATS-based models demonstrate downstream task performance comparable to unmodified base models and better than baseline models with no fine-tuning, even at sparsity levels as high as 50%. CATS-based models also exhibit better behavior than ReLUfication at similar levels of fine-tuning, and often achieve performance comparable to the base model at high levels of sparsity, both with general and task-specific fine-tuning.

Limitations and Future Work: Our work leaves several opportunities for future work. Most importantly, our empirical evaluations of CATS were restricted to the Mistral-7B and Llama2-7B base models. While we suspect CATS would also apply to other, larger models, we leave a precise empirical study to future studies. Future work may also investigate how to apply techniques similar to CATS to other MLP architectures beyond Gated-MLP, or to attention layers but without a task performance degradation. It may be possible, for example, to use recent techniques to accelerate attention layers (such as those from Zhang et al. (2022a) and Voita et al. (2019)) in conjunction with CATS.

Ethics Statement

We do not foresee any ethical problems with our work as our framework only targets memory and wall-time efficiency of language models.

Reproducibility Statement

We have submitted code to reproduce all of our experiments via a one-line script in our supplementary material; please see the `README.md`.

Acknowledgements

We thank the anonymous reviewers, area chairs, and program chairs for their reviews and comments of our paper. We will add other acknowledgements for the camera-ready version of this paper.

References

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

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701*, 2020.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022a.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022b.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022c. ISSN 1533-7928. URL `http://jmlr.org/papers/v23/21-0998.html`.

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Daniel Y. Fu, Hermann Kumbong, Eric Nguyen, and Christopher Ré. Flashfftconv: Efficient convolutions for long sequences with tensor cores, 2023a.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. Specializing smaller language models towards multi-step reasoning, 2023b.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5, 2023.

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`.

Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347, 2021.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Lukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. Sparse is enough in scaling transformers. *Advances in Neural Information Processing Systems*, 34:9895–9907, 2021.

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

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017*, 2023.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. *arXiv preprint arXiv:2212.05055*, 2022.

Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. *arXiv preprint arXiv:2110.03742*, 2021.

Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*, 2022.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

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

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023a.

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

Yuxuan Lou, Fuzhao Xue, Zangwei Zheng, and Yang You. Cross-token modeling with conditional computation. *arXiv preprint arXiv:2109.02008*, 2021.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

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. *arXiv preprint arXiv:2310.04564*, 2023.

Basil Mustafa, Carlos Riquelme, Joan Puigcerver, Rodolphe Jenatton, and Neil Houlsby. Multimodal contrastive learning with limoe: the language-image mixture of experts. *Advances in Neural Information Processing Systems*, 35:9564–9576, 2022.

Nvidia. Tensorrt-llm, 2024. URL `https://nvidia.github.io/TensorRT-LLM/`.

Haojie Pan, Chengyu Wang, Minghui Qiu, Yichang Zhang, Yaliang Li, and Jun Huang. Meta-kd: A meta knowledge distillation framework for language model compression across domains. *arXiv preprint arXiv:2012.01266*, 2020.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

Markus N. Rabe and Charles Staats. Self-attention does not need $o(n^2)$ memory, 2022.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International Conference on Machine Learning*, pp. 18332–18346. PMLR, 2022.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash layers for large sparse models. *Advances in Neural Information Processing Systems*, 34:17555–17566, 2021.

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. *arXiv preprint arXiv:1701.06538*, 2017.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

Siqi Sun, Zhe Gan, Yu Cheng, Yuwei Fang, Shuohang Wang, and Jingjing Liu. Contrastive distillation on intermediate representations for language model compression. *arXiv preprint arXiv:2009.14167*, 2020.

Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Kamil Adamczewski. Sadmoe: Exploiting activation sparsity with dynamic-k gating. *arXiv e-prints*, pp. arXiv–2310, 2023.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Philippe Tillet and David Cox. Input-aware auto-tuning of compute-bound hpc kernels. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–12, 2017.

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.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

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

Peter West, Chandra Bhagavatula, Jack Hessel, Jena D. Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. Symbolic knowledge distillation: from general language models to commonsense models, 2022.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*, 2022.

Zhongming Yu, Guohao Dai, Shang Yang, Genghan Zhang, Hengrui Zhang, Feiwen Zhu, June Yang, Jishen Zhao, and Yu Wang. Hypergef: A framework enabling efficient fusion for hypergraph neural network on gpus. *Proceedings of Machine Learning and Systems*, 5, 2023.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. Prune once for all: Sparse pre-trained language models. *arXiv preprint arXiv:2111.05754*, 2021.

Genghan Zhang, Yuetong Zhao, Yanting Tao, Zhongming Yu, Guohao Dai, Sitao Huang, Yuan Wen, Pavlos Petoumenos, and Yu Wang. Sgap: towards efficient sparse tensor algebra compilation for gpu. *CCF Transactions on High Performance Computing*, 5(2):210–227, 2023.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*, 2022a.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer Feed-forward Layers are Mixtures of Experts. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 877–890. Association for Computational Linguistics, 2022b. doi: 10.18653/v1/2022.findings-acl.71. URL `https://aclanthology.org/2022.findings-acl.71`.

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

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Tuo Zhao, and Jianfeng Gao. Taming sparsely activated transformer with stochastic experts. *arXiv preprint arXiv:2110.04260*, 2021.

A Additional Related Work

In this appendix, we discuss additional veins of related work.

Hardware-Aware Optimization that relies on customizing the algorithm implementation for the underlying hardware can result in significant performance speedup Dao et al. (2022); Fu et al. (2023a), especially for sparse kernels Gale et al. (2023); Yu et al. (2023). Recent hardware-aware methods in LLMs have shown to be highly effective in lowering the cost of attention operation Rabe & Staats (2022); Dao (2023); Liu et al. (2023a). Similar to attention operation, MLP is also memory-bounded on highly parallel machines like GPU Kim et al. (2023). The sparsity has the potential to expedite MLP because it can increase the arithmetic intensity. Based on the Roofline analysis Williams et al. (2009), higher arithmetic intensity means shorter wall-clock time for memory-bounded operations. In this work, we focus on leveraging the sparsity to reduce the memory transfers associated with the MLP weights. We do so by designing algorithmic optimizations that adaptively induce sparsity and implementing hardware-aware optimizations that translate the achieved nominal sparsity into actual wall-clock time speedup.

Structural pruning techniques induce sparsity by setting certain weights to zero so their corresponding activations need not be computed Wang et al. (2019); Kurtic et al. (2022); Xia et al. (2022); Zafrir et al. (2021); Ma et al. (2023). However, applying such techniques naïvely may not result in actual wall-clock time speedups if the resulting sparsity pattern does not lower the number of General Matrix Multiplication (GEMM) calls. Furthermore, the pruning pattern is determined at the model level and is not adaptive to the inputs, which may result in a degradation in task performance.

Quantization and Knowledge Distillation from larger models to smaller models are other popular forms of LLM inference optimization Bai et al. (2020); Frantar et al. (2022); Dettmers et al. (2023); Sun et al. (2019; 2020); Pan et al. (2020); West et al. (2022); Fu et al. (2023b). These methods often reduce the memory and computational complexity at the cost of performance degradation or require extensive finetuning. Our work can be applied to quantized or distilled models as well, although the achieved sparsity level on these models may differ.

B Accelerating Attention Layers

B.1 Method

In this section, we discuss how we can apply CATS to reduce the inference costs of attention layers inside Transformer blocks. The basic operations of a Transformer block can be written as:

$$\text{MLP}_i(\text{Attention}_i(\mathbf{x})) \tag{6}$$

where $\mathbf{x}$ is the hidden vector right before the i-th layer and where we have excluded operations like batch normalization, positional embedding, residual connections, etc. for simplicity. (For more details on the variants of Attention layers and those used in our models, we refer the reader to Touvron et al. (2023) and Jiang et al. (2023).)

The new equation for i-th transformer layer, where we wrap the previous layer with CATS activations, becomes:

$$\text{MLP}_i(\text{CATS}_{t_{i,1}}(\text{Attention}_i(\text{CATS}_{t_{i,2}}(\mathbf{x})))) \tag{7}$$

where $t_{i,1}$ and $t_{i,2}$ are the sparsification thresholds for the CATS operations applied before the MLP and attention layers, respectively, in the i-th transformer layer.

We verify that this operation results in sparse activations in Appendix C.

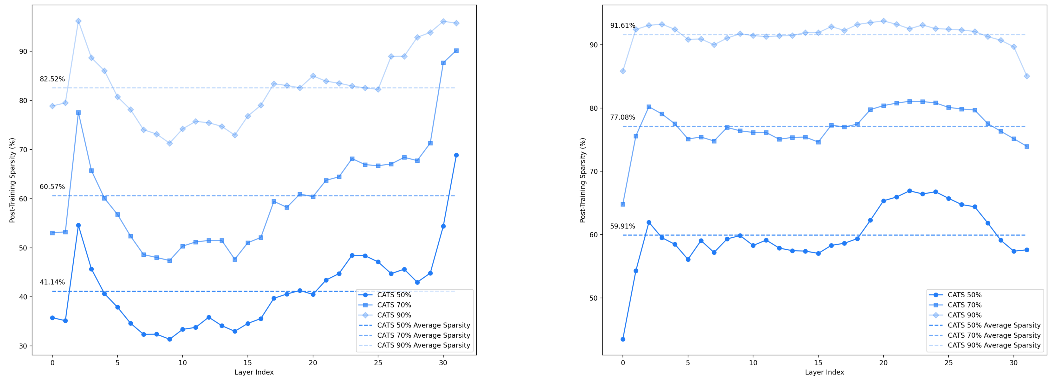
B.2 Experimental Results

CATS can also be applied to accelerate the attention blocks of LLMs: We also apply CATS to accelerate the computation of attention layers. Our approach is inspired by "Stage 2" of ReLUficiation (Mirzadeh et al., 2023).

Due to space constraints, we only measure the performance of CATS-50% applied to the base Mistral-7B model and measure zero-shot task performance. We fine-tune both models for 2000 fine-tuning batches of 16 examples each. Stage 2 CATS, which appplies CATS to both the MLP and Attention blocks, demonstrates an average downstream task performance of 66.84% across the 8 different evaluation tasks from Section 5, whereas the base Mistral-7B model demonstrates an average task performance of 69.94%. In contrast, the original CATS, applied only to the MLP layers, demonstrates an average task performance of 69.21%.

Our results demonstrate that CATS can also be applied to the attention layers of LLMs, albeit with a slight (4.3% relative) performance degradation. Future work may investigate how to apply CATS in way that better preserves the performance of the model.

C Target sparsity vs. actual sparsity



(a) Sparsity of Mistral-7B.

(b) Sparsity of Llama2-7B.

Figure 5: CATS-based models still exhibit sparsity after general fine-tuning on the Refined-Web dataset.

Dataset/Sparsity	0.5	0.7	0.9
Cola	49.629	68.926	87.6
BoolQ	49.196	68.444	87.571
SST2	48.727	68.738	87.882
Average	49.184	68.703	87.684

Table 3: CATS-based models' final sparsity after specific fine-tuning on each task. They continue to exhibit sparsity after task-specific fine-tuning.

Figure 5 demonstrates the the sparsity of each layer of Mistral-7B and Llama2-7B after CATS has been applied and fine-tuning has been performed on the RefinedWeb dataset. The average sparsity of each model (dashed lines) is roughly equal to the target sparsities (indicated by the legend).

Table 3 demonstrates the average layer sparsity of each model after task-specific fine-tuning on the 3 datasets used for this experimental setting in Section 5. The observed sparsity levels are approximately equal to the target sparsity levels.

Future work might focus on enforcing a minimum sparsity layer-wise, i.e., by zeroing out at least enough neurons to enforce the desired sparsity level k for each layer. Such work could investigate the tradeoffs between sparsity, latency, and downstream task performance.

D Details on Custom GPU Kernel Design

The previous subsections describe the procedure by which we sparsify the activations of an LLM, obviate some computations, and reduce the required number of FLOPs. Though significant recent work has focused on FLOPs as a proxy for inference cost, other work has demonstrated that reducing FLOPs is not sufficient to reduce real wall-clock inference latency Liu et al. (2023b). However, predictable sparsity patterns can be exploited to reduce floating point operations (FLOPs) during inference. We now translate the reduction in FLOPs to an actual wall-clock latency reduction via several custom GPU kernel optimization techniques.

The operations of the Gated-MLP with the CATS activation functions are:

$$v = \text{CATS}(\text{SiLU}(xW_{gate})) \tag{8}$$

$$\texttt{Mask} = \mathbb{1}_{\{|v|>t\}} \quad \text{(elementwise)} \tag{9}$$

$$y = (v' * (xW'_{\text{up}}))W'_{\text{down}} \tag{10}$$

where v', W'_{up}, and W'_{down} are v, W_{up}, and W_{down} masked by `Mask` (for the matrices W_{up} and W_{down}, the entire column j is 0 if $\texttt{Mask}_j = 0$, i.e., the mask is broadcast across columns).

If `Mask` is sparse, then Equation (10) performs two sparse matrix multiplications. In fact, only coordinates (respectively, rows) of v (respectively, W_{up} and W_{down}) corresponding to nonzero coordinates of `Mask` need to be loaded into memory. Since the MLP layer at inference time is known to be memory-bound Kim et al. (2023), the latency can be reduced if the memory access is reduced. We exploit these observations to translate the reduction in FLOPs to a real wall-clock time reduction in inference.

Custom GPU Kernel 2 MLP using CATS

1: **Input:** threshold $t > 0$, hidden layer x, weights W_{gate}, W_{down}, and W_{up}
2: $v \leftarrow$ CATS $(\text{SiLU}(xW_{gate}))$
3: $\texttt{Mask} \leftarrow 1$ if $|v| \geq t$ else 0
4: $\texttt{idcs} \leftarrow$ indices where $\texttt{Mask} = 1$
5: $x_1 \leftarrow (xW_{\text{up}}[\texttt{idcs}] * v[\texttt{idcs}])$
6: $y \leftarrow x_1 W_{\text{down}}[\texttt{idcs}]$

Custom GPU Kernel 3 MLP using CATS without atomic operations

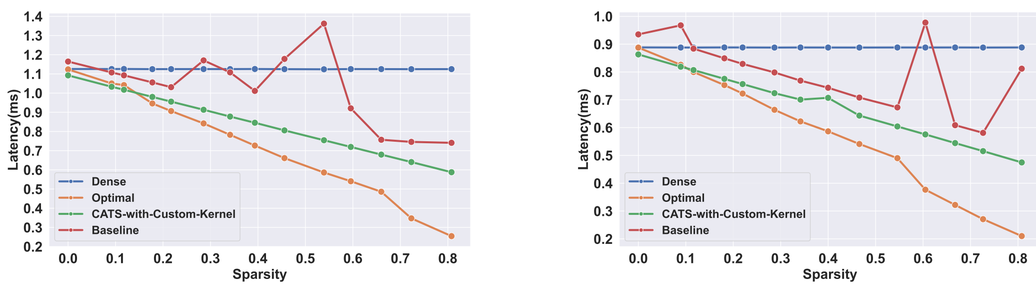
1: **Input:** threshold $t > 0$, hidden layer x, weights W_{gate}, W_{down}, and W_{up}
2: $v \leftarrow$ CATS $(\text{SiLU}(xW_{gate}))$
3: $\texttt{Mask} \leftarrow 1$ if $|v| \geq t$ else 0
4: $x_1 \leftarrow (xW_{\text{up}}[\texttt{Mask}] * v[\texttt{Mask}])$
5: $y \leftarrow x_1 W_{\text{down}}[\texttt{Mask}]$

Algorithms 2 and 3 describe Equations (8)-(10) in lower-level pseudocode. Algorithms 2 and 3 contain several optimizations.

Optimization 1: We fuse the element-wise multiplication of $v[\texttt{idcs}]$ into each tiling of $xW_{\text{up}}[\texttt{idcs}]$ as shown in Line 5 of Algorithm 2. We use an efficient algorithm from Deja Vu Liu et al. (2023b) to compute $x_1 = xW_{\text{up}}[\texttt{idcs}]$ without the element-wise multiplication by $v[\texttt{idcs}]$. In this manner, we fuse several operations and save the memory operations for storing and loading x_1 several times.

The atomic operations in Line 4 of 2, however, introduce extra overhead. Line 4 compresses a one-hot mask to a compressed coordinate array and requires atomically appending to the `idcs`. GPUs, however, cannot efficiently perform such atomic operations because of their massively parallel nature.

Optimization 2: We therefore introduce another optimization in Algorithm 3 to reduce the memory loading incurred by the atomic operations. In Algorithm 3, we directly use `Mask` to control which parts of weight matrices to load, instead of the condensed `idcs`. Algorithm 3 has more operations than Algorithm 2 because it directly assigns the unloaded elements to zero instead of squeezing out the zero values before computation. Algorithm 3 does not skip the zero operations in a fine-grained way because the sparsity in this problem is not asymptotically high Zhang et al. (2023), which means the operation reduction does not compensate for the performance loss caused by complex control logic. Figure 6 the ablation experiment results



(a) CATS of Mistral-7B's MLP.

(b) CATS of Llama2-7B's MLP.

Figure 6: Ablation study on kernel optimizations.