CHESS : Optimizing LLM Inference via Channel-Wise Thresholding and Selective Sparsification

Anonymous ACL submission

Abstract

Deploying large language models (LLMs) on edge devices presents significant challenges due to the substantial computational overhead and memory requirements. Activation sparsification can mitigate these challenges by reducing the number of activated neurons during inference. Existing methods typically employ thresholding-based sparsification based on the statistics of activation tensors. However, *these methods do not model the impact of activation sparsification on performance, resulting in significant performance degradation.* To address this issue, this paper reformulates the activation sparsification problem and proposes **CHESS** , a general activation sparsification approach via **CH**annel-wise thr**E**sholding and **S**elective **S**parsification. First, channel-wise thresholding assigns a unique threshold to each activation channel in FFN layers. Then, selective sparsification involves choosing specific layers in the attention modules to apply thresholding-based activation sparsification. Finally, this paper shows the detailed implementation of sparse kernels to accelerate the LLM inference. Experimental results demonstrate that the proposed CHESS achieves lower performance degradation over 8 downstream tasks while activating fewer parameters, thus speeding up the LLM inference by up to 1.27x.

1 Introduction

Large Language Models (LLMs) have prevailed in a wide range of applications across various fields, such as code generation tools, office assistants, input method editors, voice assistants, and assistive applications designed for individuals with disabilities. However, due to the substantial computation and memory requirements of LLM inferences, deploying LLMs on edge devices is still challenging. To mitigate these overheads, utilizing the inherent activation sparsity of LLM has emerged as a promising strategy (Liu et al., 2023; Song et al., 2023; Alizadeh et al., 2023). This approach has proven effective for models with the ReLU activation function (Li et al., 2023b; Liu et al., 2023).

Contemporary LLMs demonstrate that SwiGLU or GeGLU activation functions can further boost the model performance, but they induce less activation sparsity. Consequently, several methods (Mirzadeh et al., 2023; Song et al., 2024) are proposed to explore more sparsity by regularizing the SwiGLU or GeGLU activation. However, those works require fine-tuning the LLMs, which entails significant training overhead. To avoid training overheads and improve activation sparsification in modern LLMs, Lee et al. (Lee et al., 2024) propose a thresholding-based pruning method to actively sparsify the activation tensors during the inference stage. *However, this thresholding technique focuses solely on the statistics of the activation tensors themselves, failing to model the impact of sparsification on overall model performance.* This lack of modeling results in significant performance degradation.

To address the above limitations, this paper proposes **CHESS** , a new activation sparsification optimization via **CH**annel-wise thr**E**sholding and **S**elective **S**parsification. To capture the relation between the activation sparsity and the model performance, this paper first reformulates the activation sparsification problem in each module of existing LLMs and simplifies the problem as the thresholding problem. Then, this paper proposes channel-wise thresholding for FFN layers in LLMs, which determines the unique threshold for each activation channel. Furthermore, this paper proposes selective sparsification, which applies thresholding-based activation sparsification to the target submodules in the attention module. Finally, this paper presents the implementations of sparse kernels to accelerate the inference based on the sparse activations.

To validate the effectiveness of the proposed CHESS , this paper conducts comprehensive experiments on various downstream tasks and state-of-the-art LLMs. Experimental results demonstrate that the proposed CHESS can achieve a lower performance degradation while a better end-to-end inference speedup. Codes are available in [1].

The main contributions of this paper are,

- This paper systematically formulates the activation sparsification problem and connects the activation sparsification with the model performance.
- This paper proposes two activation sparsifications, the channel-wise thresholding for FFN modules and the selective sparsification for Attention modules, which can be widely applied in existing LLMs.
- To make full use of the activation sparsity, this paper presents the detailed algorithms for implementing the sparse kernels.
- Experimental results demonstrate the efficacy and scalability of the proposed CHESS .

2 Background and Motivations

2.1 Activation Sparsification

Activation functions introduce non-linearity into neural networks, allowing networks to capture complex patterns in the data. ReLU (Glorot et al., 2011), as a popular activation function, has been widely applied in most neural networks for addressing gradient vanish issues (Zhang et al., 2022). Another benefit of ReLU is introducing the sparsity into the activation tensors. Recent studies (Li et al., 2023b; Liu et al., 2023) have demonstrated this effect, showing that up to 95% of the intermediate FFN activations in OPT models are zero. Such sparsity can be used to accelerate the model inference while maintaining comparable model performance (Liu et al., 2023; Alizadeh et al., 2023; Song et al., 2023).

Recent state-of-the-art LLMs replace the ReLU activation function with more advanced activation functions, such as GeLU (Hendrycks and Gimpel, 2023), SiLU (Ramachandran et al., 2017), or GLU-series functions (Shazeer, 2020). Although these activation functions can significantly boost the LLMs' performance (Touvron et al., 2023), they induce less activation sparsity. Previous optimizations based on activation sparsity may not be suitable for the LLMs with those activation functions.

To improve the activation sparsification in modern LLMs, existing work (Lee et al., 2024) proposes a thresholding-based pruning method called CATS on some activation tensors in FFN layers. CATS first computes the cutoff threshold over a subset of training data according to the given sparsity level, then sparsifies the activations during inference and achieves end-to-end speedup via efficient sparse kernel design. Although CATS can improve activation sparsification, it only focuses on the statistics of the activation tensors themselves without modeling the impact of activation sparsification on the model performance, leading to significant performance drop.

[1] https://anonymous.4open.science/r/CHESS-BA40

2.2 Motivation

Following the observations in CATS (Lee et al., 2024), this paper also aims to apply activation sparsification in the Gated-MLP blocks of FFN layers, which are the most common components in modern LLMs. The formal expression of the gated-MLP block is defined as,

$$\mathrm{FFN}(x) = \big(\sigma(xW^{\mathrm{gate}}) \odot (xW^{\mathrm{up}})\big)\, W^{\mathrm{down}} \quad (1)$$

where W^{up}, W^{gate}, W^{down} are parameters in MLP blocks, $\sigma(\cdot)$ is the activation function. Therefore, the activation values in FFN layers are,

$$A^{\mathrm{up}} = xW^{\mathrm{up}}, \quad A^{\mathrm{gate}} = \sigma(xW^{\mathrm{gate}}) \quad (2)$$

Inspired by layer-wise weight pruning (Sun et al., 2023; Frantar and Alistarh, 2023), this paper reformulates the activation sparsification problem. Following CATS (Lee et al., 2024), we focus on sparsifying A^{gate}. Therefore, the objective is to find the optimal pruned activation tensor $\hat{A}^{\mathrm{gate}}$, guaranteeing the sparsity level and minimizing the difference of output of the succeeding layer between before and after pruning. More formally, the problem is defined as,

$$\arg\min_{\hat{A}^{\mathrm{gate}}} \left\| A^{\mathrm{up}} \odot A^{\mathrm{gate}} - A^{\mathrm{up}} \odot \hat{A}^{\mathrm{gate}} \right\|_2^2 \quad (3)$$

where A^{up}, A^{gate} are different activation tensors in FFN layers, $\hat{A}^{\mathrm{gate}}$ is the pruned activation tensor.

We decompose all activations in the pruned tensor into two subsets, i.e., the pruned $\hat{A}^{\mathrm{gate}}_{\mathcal{P}}$ which are all zeros and the non-pruned $\hat{A}^{\mathrm{gate}}_{\mathbb{U}-\mathcal{P}}$ which are the same as the corresponding values in A^{gate}. Thus, this paper can simplify the objective as: **finding a subset of indices $\mathcal{P}$ that indicates the index of the pruned elements, and satisfies sparsity level $|\mathcal{P}| \geq k \cdot |\mathbb{U}|$, while minimizing the sparsification error illustrated in Equation 4**, where

$\mathbb{U} = \{1, \ldots, d\}$, d is the feature dimension of A^{gate}.

$$\arg\min_{\mathcal{P}} \sum_{i\in\mathcal{P}} \left(A_i^{\text{up}} \cdot (A_i^{\text{gate}} - 0)\right)^2 + \sum_{i\in\mathbb{U}-\mathcal{P}} \left(A_i^{\text{up}} \cdot (A_i^{\text{gate}} - A_i^{\text{gate}})\right)^2 \tag{4}$$

Equation 4 could be further simplified into Equation 5. An optimal solution is to sort all $\left(A_i^{\text{up}} A_i^{\text{gate}}\right)^2$ and select the top smallest elements according to the given sparsity level. However, the sorting operation requires the prior computation of $\left(A_i^{\text{up}} A_i^{\text{gate}}\right)^2$, which involves large matrix computations to obtain A^{up} and A^{gate}. Besides, sorting across channels in each FFN layer is also a costly process.

$$\arg\min_{\mathcal{P}} \sum_{i\in\mathcal{P}} \left(A_i^{\text{up}} A_i^{\text{gate}}\right)^2 \tag{5}$$

3 CHESS : Activation Sparsification via Channel-Wise Thresholding and Selective Sparsification

In this section, this paper first introduces channel-wise thresholding for FFN layers. Then, this paper presents the selective sparsification for attention layers. Finally, this paper shows the efficient implementation of the proposed custom sparse kernels.

3.1 Channel-Wise Thresholding

As described in Equation 5, whether to prune an activation element is determined by both A^{up} and A^{gate}. This implies that A^{gate} demonstrates the activation sparsity while A^{up} measures the impact of pruning the activation on performance degradation. Therefore, we introduce the *importance score* of each activation element,

$$\text{score}_i = \left|A_i^{\text{up}} A_i^{\text{gate}}\right| \tag{6}$$

Since the activation sparsity is introduced by activation functions, it can determine the elements to be pruned after obtaining the activation values A^{gate}, which can save a large amount of computation in the matrix multiplication with W^{up} and W^{down}. However, the importance score is computed by A^{up} and A^{gate}, where A^{up} is still unknown after computing A^{gate}.

To address this limitation, this paper estimates the $|A_i^{\text{up}}|$ using the expectation of A_i^{up} over sampled training data,

$$|A_i^{\text{up}}| \approx \mathbb{E}\left[|A_i^{\text{up}}|\right] = \frac{1}{n}\sum_j |A_{ij}^{\text{up}}| \tag{7}$$

where n is the number of sampled data. Therefore, the importance score is further estimated as,

$$\hat{\text{score}}_i = \mathbb{E}\left[|A_i^{\text{up}}|\right]\left|A_i^{\text{gate}}\right| \tag{8}$$

For the sorting overhead, this paper also adopts the distribution sampling method. Specifically, we first outline the cumulative distribution function F of the proposed importance score across all channels,

$$F(t) = P(\hat{\text{score}} \leq t) \tag{9}$$

Then, given a sparsity level k, we can obtain the threshold t_i for sparsifying the activation elements on channel i,

$$t_i = \frac{\arg\min_t F(t) \geq k}{\mathbb{E}\left[|A_i^{\text{up}}|\right]} \tag{10}$$

This threshold indicates the maximal activation value that should be pruned as zero. Different from CATS, **this is a Channel-Wise Thresholding (CWT) technique that relates the model performance degradation with the activation sparsity via introducing the *importance score* in Equation 6.**

Finally, based on the channel-wise thresholds, the activation values can be sparsified as,

$$\text{CWT}(A_i) = \begin{cases} 0, & \text{if } |A_i| \leq t_i \\ A_i, & \text{if } |A_i| > t_i \end{cases} \tag{11}$$

And the final output of the FFN layer is computed as,

$$\text{FFN}_{\text{CWT}}(x) = \left(\text{CWT}(A^{\text{gate}}) \odot A^{\text{up}}\right) W^{out} \tag{12}$$

3.2 Selective Sparsification

Although the activation sparsity in attention modules is much less than that in FFN modules, it is also worth applying activation sparsification to the features to reduce the memory footprint of the attention weights. The common attention mechanism has four linear projects: query, key, value, and output projection. Similarly, we can also reformulate the activation sparsification in the attention mechanism for each projection,

$$\arg\min_{\hat{x}} \|xW - \hat{x}W\|_2^2 \tag{13}$$

The objective of activation sparsification in the attention mechanism is to find the optimal pruned features for each attention layer, ensuring the given sparsity level and low model performance degradation.

The error E= $\|xW - \hat{x}W\|_2^2$ can be approximated using the Taylor series as follows (LeCun et al., 1989; Hassibi and Stork, 1992; Frantar et al., 2023b):

$$\mathrm{E} = \mathbf{g}(\hat{x}-x)^T + \frac{1}{2}(x-\hat{x})\mathbf{H}(\hat{x}-x)^T + O(\|\hat{x}-x\|^3) \tag{14}$$

where $\mathbf{g}$ and $\mathbf{H}$ denote the first-order and second-order derivatives of the error E with respect to $\hat{x}$, respectively.

$$\mathbf{g} = \frac{\partial \mathrm{E}}{\partial \hat{x}}\bigg|_{\hat{x}=x} = 0 \tag{15}$$

$$\mathbf{H} = \frac{\partial^2 \mathrm{E}}{(\partial \hat{x})^2}\bigg|_{\hat{x}=x} = WW^T \tag{16}$$

Then, we replace $\mathbf{g}$ and $\mathbf{H}$ with true values, discard the higher-order terms, and apply diagonal approximation to $\mathbf{H}$. The Equation 14 can be simplified as:

$$\mathrm{E} \approx \sum_{i=1}^{d} \|W_i\|^2 (\hat{x}_i - x_i)^2 \tag{17}$$

where $\|W_i\|^2$ denotes the ℓ_2 norm of row i in weight matrix W. As described in Section 2.2, we can also decompose the input features into pruned features (zeros) and non-pruned features (original values) and then transform the objective as follows,

$$\arg\min_{\mathcal{P}} \sum_{i \in \mathcal{P}} \|W_i\|^2 (x_i)^2 \tag{18}$$

To further simplify Equation 18, this paper analyzes the statistics of the weight matrix in the attention mechanism. Figure 1 shows the distribution of $\|W_i\|^2$ of different rows in projection weights. From the results, all rows from the same weight exhibit similar $\|W_i\|^2$, therefore we can eliminate this coefficient from Equation 18 and derive the simplified final objective:

$$\arg\min_{\mathcal{P}} \sum_{i \in \mathcal{P}} |x_i| \tag{19}$$

Based on Equation 19, this paper also adopts a similar distribution sampling strategy as that in CATS (Lee et al., 2024) to determine the thresholds

Algorithm 3.1 spvmm

Input: The sparse input vector $x \in \mathbb{R}^{1\times K}$, the weight matrix $W \in \mathbb{W}^{K\times N}$, the number of output elements N, the number of input elements K, the block size B.

Output: The output vector $y \in \mathbb{R}^{1\times N}$

for $n0$ from 0 to N with step size B in PARALLEL **do**
  for k from 0 to K **do**
    if $x[k] \neq 0.0$ **then**
      $n1_{upp} = \min(B, N - n0)$
      for $n1$ from 0 to $n1_{upp}$ VECTORIZED **do**
        $y[n0 + n1]$ += $x[k] \times W[k][n0 + n1]$
      end for
    end if
  end for
end for
return y

given a sparsity level. Different from CWT, CATS is a tensor-wise thresholding,

$$\mathrm{CATS}(x) = \begin{cases} x_i, & \text{if } |x_i| > t \\ 0, & \text{if } |x_i| \le t \end{cases} \tag{20}$$

However, which modules the CATS should be applied to becomes a challenge in terms of the trade-off between model performance and model efficiency. The search space is quite large. Taking Llama-7B as an example, which has 32 layers and four attention projections per layer, the search space is over the septillion level.

In this paper, we compare two stratagies, namely *full sparsification* and *selective sparsification*. Full sparsification refers to applying CATS to all four projections of the attention mechanism,

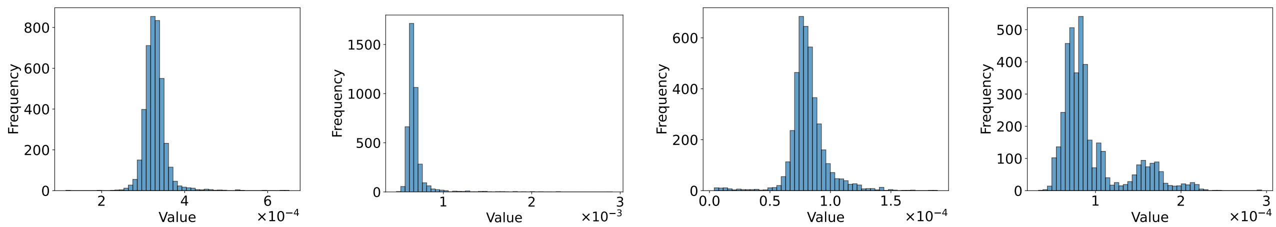
$$\mathrm{C}_{t_\mathrm{o}}(\mathrm{Attn}(\mathrm{C}_{t_\mathrm{i}}(x)W^{\mathrm{q}}, \mathrm{C}_{t_\mathrm{i}}(x)W^{\mathrm{k}}, \mathrm{C}_{t_\mathrm{i}}(x)W^{\mathrm{v}}))W^{\mathrm{o}} \tag{21}$$

where $C(\cdot)_t$ is the CATS function with the threshold t.

Conversely, selective sparsification refers to applying the CATS function to only query and output projections, while not altering key and query projections. The formal expression is,

$$\mathrm{C}_{t_\mathrm{o}}(\mathrm{Attn}(\mathrm{C}_{t_\mathrm{q}}(x)W^{\mathrm{q}}, xW^{\mathrm{k}}, xW^{\mathrm{v}}))W^{\mathrm{o}} \tag{22}$$

Experimental results (ref. Section 4.3) demonstrate that selective sparsification results in significantly



(a) query projection (b) key projection (c) value projection (d) output projection

Figure 1: Distribution of $\|W_i\|^2$ of different rows i in attention projections of layer 15 of Llama-3-8B

Algorithm 3.2 vmmsp

Input: The input vector $x \in \mathbb{R}^{1\times K}$, the weight matrix $W \in \mathbb{R}^{N\times K}$, the mask array mask $\in \mathbb{R}^{1\times N}$, the number of output elements N, the number of input elements K, the block size B.

Output: The output vector $y \in \mathbb{R}^{1\times N}$.

```
1:  for n0 from 0 to N with step size B in PARALLEL do
2:    n1_upp = min(B, N − n0)
3:    for n1 from 0 to n1_upp do
4:      if mask[n0 + n1] ≠ 0.0 then
5:        accum = 0.0
6:        for k from 0 to K VECTORIZED do
7:          accum += W[n0 + n1][k] × x[k]
8:        end for
9:        y[n0+n1] = accum × mask[n0+n1]
10:     end if
11:   end for
12: end for
13: return y
```

lower performance degradation, while achieving comparable overhead reduction when applied to GQA modules. Since the GQA modules are widely applied in modern LLMs, we utilize selective sparsification as our main method for attention modules.

3.3 Efficient Sparse Kernels

To achieve wall-clock speedup and reduce inference latency based on sparse activations, this paper developed two custom CPU kernels: *spvmm* (sparse vector-matrix multiplication) and *vmmsp* (vector-matrix multiplication with output sparsity). The spvmm kernel is optimized for cases where the input activation tensor is sparse, and it is employed in attention modules and FFN down projections. Conversely, the vmmsp kernel is designed for cases where the output activation tensor is multiplied with a sparse mask, and it is used in FFN up projections.

Algorithm 3.1 and Algorithm 3.2 show the detailed steps of spvmm and vmmsp, respectively. Algorithm 3.1 splits the input vector into blocks of size B and accumulates the vector-matrix multiplication results of each block when $x[k]$ is not 0 (Lines 5-7). Algorithm 3.2 also performs block-level vector-matrix multiplications but computes the outputs at the specific position based on the sparsity mask (Lines 5-9). Both algorithms reduce the latency by bypassing unnecessary weight reads and computations.

The implementation of the vmmsp kernel is relatively straightforward; it computes $Y = XW^T$, consistent with the definition of linear projection in PyTorch (Paszke et al., 2019). However, the spvmm operator requires a more complex approach to ensure efficient computation on multi-core CPUs while avoiding atomic operations. To this end, we employ two advanced optimizations. First, we employ loop tiling and loop reordering strategies to make sure that each threads compute independently without the need for synchronization or atomic operations. Furthermore, we transpose the linear projection weights in advance during the model preprocessing stage, to maximize memory locality and enhance cache hit rates.

4 Experiments

In this section, this paper first introduces the dataset, comparisons, and implementation details. Then, this paper presents the main results over 8 downstream tasks in terms of the model performance and model efficiency. Besides, this paper also conducts an ablation study across different sparsification module and analysis on efficiency over different sparsity level.

4.1 Datasets and Experimental Setup

Datasets We utilize OpenBookQA, ARC Easy, Winogrande, HellaSwag, ARC Challenge, PIQA,

Models	AP↓	WG↑	SciQ↑	PIQA↑	QA↑	HS↑	BoolQ↑	Arc-E↑	Arc-C↑	Avg↑
Llama-2-7B	100%	69.14	93.90	78.07	31.40	57.15	77.68	76.26	43.43	65.87
CATS	78.16%	**66.69**	92.80	77.48	31.60	57.03	72.17	74.07	41.13	64.12
CHESS w/o	78.17%	66.61	93.20	77.20	32.40	**57.15**	**74.22**	**74.62**	**41.47**	**64.60**
CHESS w/	**70.05%**	66.22	**93.30**	**77.86**	**33.60**	56.60	**74.22**	74.37	40.36	64.56
Llama-2-13B	100%	72.22	94.60	79.05	35.00	60.06	80.61	79.38	48.38	68.66
CATS	77.97%	70.64	94.10	78.78	33.80	60.42	75.60	77.44	**46.93**	67.21
CHESS w/o	77.98%	70.09	**94.30**	78.89	34.00	**60.64**	**79.11**	**77.95**	46.67	67.71
CHESS w/	**69.82%**	**70.88**	94.20	**79.00**	**34.40**	60.47	78.50	**77.95**	46.84	**67.78**
Llama-3-8B	100%	73.32	96.30	79.60	34.60	60.15	81.07	80.22	50.17	69.42
CATS	74.96%	70.88	**94.90**	78.40	32.40	57.34	78.65	75.76	45.22	66.69
CHESS w/o	74.96%	**71.90**	94.60	78.67	**32.80**	**59.06**	**79.97**	**77.02**	**47.44**	**67.68**
CHESS w/	**67.80%**	70.17	94.20	**79.22**	**32.80**	58.62	78.04	76.85	46.67	67.07

Table 1: Main results on downstream tasks of different models. 'AP' refers to the ratio of activated parameters.

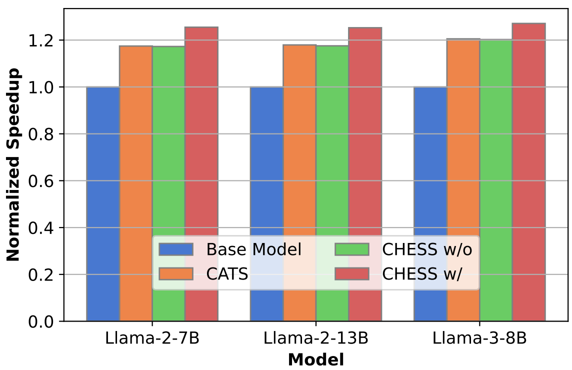


Figure 2: End-to-end inference speedup

BoolQ, and SCI-Q as benchmarks for downstream tasks, employing the Evaluation Harness library from Eleuther AI to ensure consistency with and Lee et al. (2024). These tasks are designed to assess various aspects of the language model's performance, including comprehension, common sense, and reasoning abilities, which effectively illustrate the model's capability loss with activation sparsification.

Comparisons To validate the effectiveness of the proposed CHESS , we implement the CHESS and comparisons on state-of-the-art LLMs, including Llama-2 7B, Llama-2 13B, and Llama-3 8B. These LLMs incorporate different attention mechanisms, i.e., MHA and GQA, and adopt SwiGLU as the FFN activation function. For the main results, we evaluate four models based on all three LLMs,

- **Base Model:** the LLM model without any activation sparsification.
- **CATS (Lee et al., 2024):** the state-of-the-art activation sparsification method, which applies magnitude pruning to FFN activations.
- **CHESS w/o:** the proposed method including channel-wise thresholding but without attention sparsification.
- **CHESS w/:** the proposed method including channel-wise thresholding and selective sparsification.

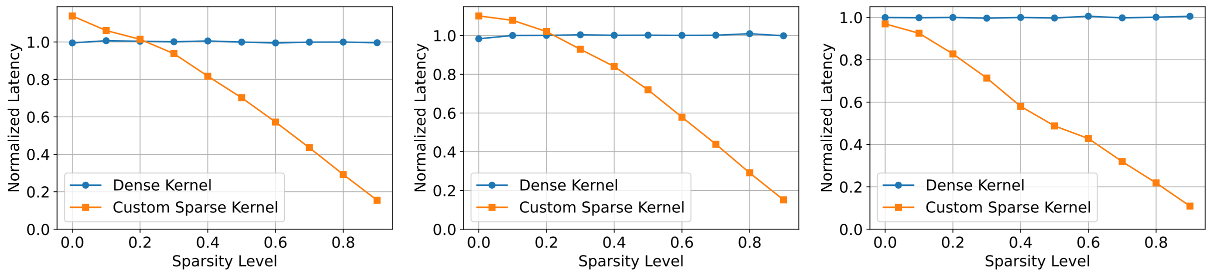
For the ablation study, we evaluate three models,

- **Llama-3:** the Llama-3 8B model.
- **FS:** the proposed method with full sparsification in attention modules.
- **SS:** the proposed method with selective sparsification in attention modules.

Implementation Details For all models involving activation sparsification, thresholds are sampled from a subset of the C4 dataset (Raffel et al.). Following the settings in (Lee et al., 2024), the sparsity level k is set to 0.5, where the accuracy drop is minimal while the inference latency significantly decreases. The proposed method was implemented using the PyTorch v2.2.2 and HuggingFace Transformers v4.39.3. End-to-end decoding speedups are measured on a randomly collected subset of C4 dataset. All experiments are conducted with FP32 precision on a personal computer equipped with an Intel Core I9-12900K CPU and 64GB of DDR4 memory. Since our work can be applied to quantized models as well, changing weight precision to FP16 or even lower bit-width quantizations does not materially affect our results (Lee et al., 2024).

Model	AP↓	WG↑	SciQ↑	PIQA↑	QA↑	HS↑	BoolQ↑	Arc-E↑	Arc-C↑	Avg↑
Llama-3	100%	73.32	96.30	79.60	34.60	60.15	81.07	80.22	50.17	69.42
FS	**90.94%**	71.59	96.10	78.02	34.80	57.14	78.56	79.00	46.16	67.67
SS	92.84%	**72.85**	**96.30**	**79.71**	**35.00**	**59.31**	**79.57**	**79.67**	**50.17**	**69.07**

Table 2: Ablation study among full sparsification and selective sparsification in attention modules. 'AP' refers to the ratio of activated parameters.



(a) Attention projection (b) Down projection (c) Up projection

Figure 3: Comparison between custom sparse kernels and PyTorch dense kernel on latency of linear projections

4.2 Main Results on Downstream Tasks

Table 1 compares the accuracy of different models across 8 downstream tasks and Figure 2 evaluates the end-to-end inference speedups. Experimental results draw the following conclusions.

Channel-wise thresholding can reduce accuracy degradation while achieving comparable sparsity. Compared to CATS, the proposed CHESS w/o achieves lower performance degradation of 1.32 on average over 8 tasks and 3 base models. Specifically, CHESS w/o achieves the lower average performance degradation with the base model Llama 3. CHESS w/o performs better on 5 tasks than CATS. Besides, CHESS w/o achieves a comparable sparsity to CATS.

Selective sparsification of attention modules further improves sparsity while maintaining model accuracy. Compared CHESS w/o on Llama-2-7B and Llama-3-8B, the average performance of CHESS w/ degrade by 0.04% and 0.61%, respectively. Interestingly, on the Llama-2-13B, CHESS w/ achieves an improvement of 0.07% over CHESS w/o. Specifically, CHESS w/ performs better on PIQA and OpenbookQA, but worse on HellaSwag, BoolQ, Arc Easy and Arc Challenge, and comparably on WinoGrande and SCI-Q. These results demonstrate the minimal impact of additional selective sparsification on performance. Compared to CATS, CHESS w/ consistently achieves better average performance with fewer activated parameters.

CHESS achieves end-to-end speedups of up to 1.27x compared to Transformers baselines. The proposed CHESS w/ achieves the highest speedup of 1.25x on Llama-2-7B and Llama-2-13B, and 1.27x on Llama-3-8B, respectively. When not employing attention sparsification, CHESS w/o achieves comparable speedups to CATS, which is 1.17x on Llama-2-7B and Llama-2-13B, and 1.20x on Llama-3-8B, respectively. This is because of the comparable parameters activated per decoding pass of these two methods.

4.3 Ablation Study

Table 2 presents the ablation study with different sparsification in attention modules. While selective sparsification achieves a comparable reduction in overhead relative to full sparsification, it significantly outperforms full sparsification across all eight benchmarks. Specifically, selective sparsification exhibits substantial improvements on the HellaSwag and Arc Challenge benchmarks, while demonstrating modest gains on the remaining benchmarks. These results underscore the advantages of selective sparsification.

4.4 Kernel Efficiency

As illustrated in Figure 3, this paper conducts a comparative analysis of the latency against sparsity level between the proposed custom sparse kernel and the dense kernel in PyTorch (Paszke et al., 2019). At a sparsity level of 0, the *vmmsp* ker-

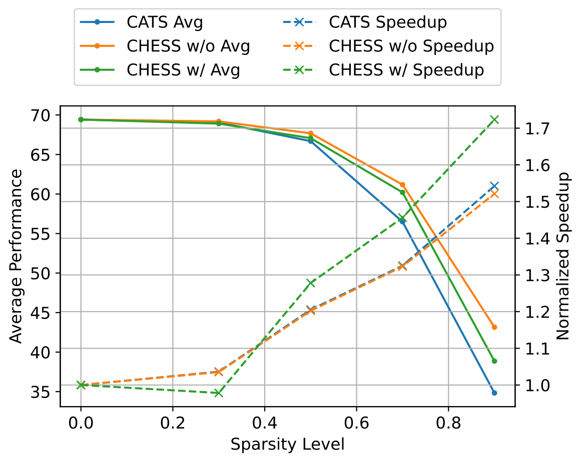


Figure 4: Average downstream performance and end-to-end speedups of each method under different sparsity levels.

nel used for up projections demonstrates slightly lower latency compared to the PyTorch dense kernel. Conversely, the *spvmm* kernel, utilized by attention projections and down projections, exhibits slightly higher latencies than the dense kernel. This increased latency is primarily due to the advanced loop tiling and reordering strategies, which cause slight performance degradation at low sparsity levels.

As the sparsity level increases, the latency of the dense kernel remains relatively constant, whereas the latency of our custom sparse kernels decreases proportionally. Notably, at a sparsity level of 0.5, our custom sparse kernels achieve latency reductions of 30%, 28%, and 51% for attention projection, FFN up projection, and FFN down projection, respectively. These findings highlight the efficiency of our custom kernels.

4.5 Impact on Different Sparsity Levels

Figure 4 shows the model performance on downstream tasks and end-to-end decoding speedups at different sparsity levels. We selected Llama-3 8B as the base model since it incorporates the contemporary GQA module.

Experimental results indicate that at lower sparsity levels (0.3 and 0.5), both CATS and CHESS maintain performance comparable to the base model, with CHESS exhibiting superior performance. At higher sparsity levels (0.7 and 0.9), these models experience noticeable performance degradation, and CHESS models, particularly CHESS w/o models, consistently outperform CATS. Specifically, at a sparsity level of 0.7, the CATS, CHESS w/o, and CHESS w/ models achieve average performances of 56.49, 61.18, and 60.21, respectively. At a sparsity level of 0.9, the corresponding performances are 34.83, 43.15, and 38.86, respectively.

Regarding end-to-end speedup, CHESS w/ models exhibit the highest speedup at all sparsity levels above 0.3, attributed to the selective sparsification of attention modules. Specifically, CHESS w/ achieves speedups of 1.46x and 1.72x at sparsity levels of 0.7 and 0.9, respectively, compared to 1.33x and 1.52x for CATS. However, at a sparsity level of 0.3, the CHESS w/ model exhibits speedup slightly below 1, primarily due to the inefficiency of our custom sparse kernels at low sparsity levels.

5 Related Work

Various methods have been proposed to address the challenges associated with deploying LLMs locally. Weight quantization (Frantar et al., 2023a; Lin et al., 2023; Xiao et al., 2022) aims to represent LLM weights using lower bit-widths, thereby reducing memory usage and access overhead. Activation quantization focuses on minimizing the memory footprint of activation tensors and KV cache (Li et al., 2023a). These methods can be applied along with our proposed CHESS method.

Weight pruning (Frantar and Alistarh, 2023; Sun et al., 2023) involves setting a portion of the LLM weights to zero to reduce computational overhead and memory requirement. However, this approach faces several challenges including noticeable degradation in performance and limited hardware support when applied on personal devices.

Non-autoregressive decoding approaches, such as speculative decoding (Chen et al., 2023, 2024) or Medusa (Cai et al., 2024), seek to convert autoregressive decoding process of LLMs into parallel decoding to mitigate memory access overhead. However, these methods simultaneously impose increased computational demands, which presents significant challenges for deployment on personal devices with limited processing capabilities.

6 Conclusion

This paper reformulates the activation sparsification problem and introduces the CHESS , a general activation sparsification via channel-wise thresholding and selective sparsification. Experiments show that the proposed CHESS can achieve a lower performance degradation and accelerate the LLM inference with sparse activations.

7 Limitations

The limitations of this paper lie in two aspects. First, although CHESS achieves lower accuracy degradation while activating fewer parameters compared to existing methods, it still incurs a noticeable accuracy loss, especially at high sparsity levels. Future research may investigate fine-tuning techniques to mitigate this performance drop. Secondly, our method performs optimally with a batch size of 1. This constraint is acceptable for edge deployment scenarios, where typically only a single user is involved. However, in data center deployments, this method does not yield significant end-to-end speedup. This is because the structural sparsity of the activation tensor deteriorates into unstructured sparsity under larger batch sizes.

References

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C. Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. *arXiv preprint*. ArXiv:2312.11514 [cs].

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. *arXiv preprint*. ArXiv:2401.10774 [cs].

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. *arXiv preprint*. ArXiv:2302.01318 [cs].

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. Sequoia: Scalable, Robust, and Hardware-aware Speculative Decoding. *arXiv preprint*. ArXiv:2402.12374 [cs].

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. In *Proceedings of the 40th International Conference on Machine Learning*, pages 10323–10337. PMLR. ISSN: 2640-3498.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023a. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *arXiv preprint*. ArXiv:2210.17323 [cs].

Elias Frantar, Sidak Pal Singh, and Dan Alistarh. 2023b. Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning. *arXiv preprint*. ArXiv:2208.11580 [cs] version: 2.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 315–323.

Babak Hassibi and David Stork. 1992. Second order derivatives for network pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann.

Dan Hendrycks and Kevin Gimpel. 2023. Gaussian Error Linear Units (GELUs). *arXiv preprint*. ArXiv:1606.08415 [cs].

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Je-Yong Lee, Donghyun Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. 2024. CATS: contextually-aware thresholding for sparsity in large language models. *CoRR*, abs/2404.08763.

Qingyuan Li, Yifan Zhang, Liang Li, Peng Yao, Bo Zhang, Xiangxiang Chu, Yerui Sun, Li Du, and Yuchen Xie. 2023a. FPTQ: fine-grained post-training quantization for large language models. *CoRR*, abs/2308.15987.

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. 2023b. The Lazy Neuron Phenomenon: On Emergence of Activation Sparsity in Transformers. *arXiv preprint*. ArXiv:2210.06313 [cs, stat].

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, Chuang Gan, and Song Han. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *arXiv preprint*. ArXiv:2306.00978 [cs].

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. 2023. Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time. In *Proceedings of the 40th International Conference on Machine Learning*, pages 22137–22176. PMLR. ISSN: 2640-3498.

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C. Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for Activation Functions. *arXiv preprint*. ArXiv:1710.05941 [cs].

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.

Noam Shazeer. 2020. GLU Variants Improve Transformer. *arXiv preprint*. ArXiv:2002.05202 [cs, stat].

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. 2024. ProSparse: Introducing and Enhancing Intrinsic Activation Sparsity within Large Language Models. *arXiv preprint*. ArXiv:2402.13516 [cs].

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2023. Powerinfer: Fast large language model serving with a consumer-grade GPU. *CoRR*, abs/2312.12456.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2023. A Simple and Effective Pruning Approach for Large Language Models. *arXiv preprint*. ArXiv:2306.11695 [cs].

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2022. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. Publication Title: arXiv e-prints ADS Bibcode: 2022arXiv221110438X.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. 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. 2022. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068.