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

Junhui He[1,2], Shangyu Wu[3,4], Weidong Wen[2], Chun Jason Xue[3], Qingan Li[2] [*]

[1] Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University,
[2] School of Computer Science, Wuhan University,
[3] MBZUAI,
[4] City University of Hong Kong

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 resource 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, they do not model the impact of activation sparsification on performance, resulting in suboptimal performance degradation. To address the limitations, this paper reformulates the activation sparsification problem to explicitly capture the relationship between activation sparsity and model performance. Then, this paper 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 the feed-forward network (FFN) layers. Then, selective sparsification involves applying thresholding-based activation sparsification to specific layers within the attention modules. Finally, we detail the implementation of sparse kernels to accelerate LLM inference. Experimental results demonstrate that the proposed CHESS achieves lower performance degradation over eight downstream tasks while activating fewer parameters than existing methods, thus speeding up the LLM inference by up to 1.27x.

1 Introduction

Large Language Models (LLMs) have become integral to diverse applications, including code generation, office assistance, voice assistance, and assistive applications 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., 2023; 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., 2024; Song et al., 2024) are proposed to increase sparsity by applying regularization to the activation function and employing continuous training. However, those works require fine-tuning the LLMs, which entails significant overheads. To avoid these overheads and increase activation sparsity in modern LLMs, 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, which results in suboptimal 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. First, this paper reformulates the activation sparsification problem to explicitly capture the relationship between activation sparsity and model performance. Then, this paper proposes channel-wise thresholding for FFN modules in LLMs, which determines a unique threshold for each activation channel. Furthermore, this paper proposes selective sparsification, which applies thresholding-based activation sparsification to the target layers in the attention module. Finally, this paper presents the implementations of sparse kernels to accelerate the inference based on the sparse activations.

[*] Corresponding author

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 18658–18668
November 12-16, 2024 ©2024 Association for Computational Linguistics

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 CHESS can achieve lower performance degradation with better end-to-end inference speedup. The code is publicly available [1].

The main contributions of this paper are,

- This paper reformulates the activation sparsification problem and establishes a connection between sparsity and model performance.
- This paper proposes two activation sparsification methods, 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 details the algorithm design and implementations of the efficient sparse kernels.
- Experimental results demonstrate the performance and efficiency 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 to address gradient vanish issues (Zhang et al., 2022). Another benefit of ReLU is introducing the sparsity into the activation tensors. Recent studies (Li et al., 2023; Liu et al., 2023) have demonstrated 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, 2016), SiLU (Ramachandran et al., 2018), 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 increase activation sparsity 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 set of sample input 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 increases activation sparsity, it only focuses on the statistics of the activation tensors without modeling the impact of activation sparsification on the model performance, leading to suboptimal performance drop.

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 modules. The formal expression of the FFN module is defined as,

$$\text{FFN}(x) = \left(\sigma(xW^{\text{gate}}) \odot (xW^{\text{up}})\right) W^{\text{down}} \quad (1)$$

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

$$a^{\text{up}} = xW^{\text{up}}, \quad a^{\text{gate}} = \sigma(xW^{\text{gate}}) \quad (2)$$

Since the activation function introduces sparsity where the values of many elements in the output tensor are close to zero, we focus on pruning the output of the gate projection layer, i.e., a^{gate}. Then, the following computations, such as the matrix multiplication for a^{up}, the element-wise multiplication between a^{up} and a^{gate}, or the matrix multiplication with W^{down}, can further be optimized due to the zero elements in the pruned a^{gate}.

Inspired by layer-wise weight pruning (Frantar and Alistarh, 2023; Sun et al., 2024), this paper reformulates the activation sparsification problem to **find the optimal pruned activation tensor $\hat{a}^{\text{gate}}$ that guarantees a specified sparsity level while minimizing the output difference of the succeeding layer before and after pruning.** More formally, the problem is defined as,

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

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

We decompose all activations in the pruned tensor into two subsets, i.e., the pruned $\hat{a}^{\text{gate}}_{\mathcal{P}}$ which

[1] https://github.com/ZeonfaiHo/CHESS

are all zeros and the non-pruned $\hat{a}^{\text{gate}}_{\mathbb{U}-\mathcal{P}}$ which keep the original values in a^{gate}. Thus, we can simplify the objective defined in Equation 3 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}.

$$\arg\min_{\mathcal{P}} \sum_{i \in \mathcal{P}} \left(a_i^{\text{up}} a_i^{\text{gate}}\right)^2 \tag{4}$$

Equation 4 establishes the theoretical relationship between activation sparsification and model performance, which is ignored by previous works, e.g., CATS (Lee et al., 2024). However, finding the top-k smallest elments of $\left(a_i^{\text{up}} a_i^{\text{gate}}\right)^2$ requires the prior compuation of a^{up}, which is not all necessary. Besides, sorting across channels in each FFN layer is also a costly process. These challenges point us to propose the CHESS method.

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

In this section, this paper first introduces the channel-wise thresholding method for FFN modules. Then, it presents the selective sparsification for attention modules. Finally, it discusses the algorithm design and implementations of the efficient custom sparse kernels.

3.1 Channel-Wise Thresholding

As described in Equation 4, whether to prune an activation element is determined by both a^{up} and a^{gate}. To quantify the significance of each activation element, we introduce the *importance score* based on Equation 4,

$$\text{score}_i = \left|a_i^{\text{up}} a_i^{\text{gate}}\right| \tag{5}$$

To obtain all importance scores of elements in Equation 5, we need to compute two matrix-multiplication for a^{gate} and a^{up}. However, we can reduce the computational cost for a^{up} by leveraging the sparsity of a^{gate}. Therefore, we need to calculate the score only with a^{gate}. We observe that, for each channel i, the values of $|a_i^{\text{up}}|$ remain relatively consistent across different inputs, as shown in Figure 1. However, these values can vary significantly between different channels. Based on this observation, this paper estimates the $|a_i^{\text{up}}|$ using the expectation of $|a_i^{\text{up}}|$ over the sampled input data,

$$|a_i^{\text{up}}| \approx \mathbb{E}\left[\left|a_i^{\text{up}}\right|\right] = \frac{1}{n}\sum_j |a_{ij}^{\text{up}}| \tag{6}$$

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

$$\hat{\text{score}}_i = \mathbb{E}\left[\left|a_i^{\text{up}}\right|\right] \left|a_i^{\text{gate}}\right| \tag{7}$$

For the sorting overhead, this paper also adopts the CDF-based thresholding method following Lee et al. (2024). Specifically, we first outline the cumulative distribution function F of the proposed importance score across all channels,

$$F(t) = \mathbb{P}(\hat{\text{score}} \leq t) \tag{8}$$

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

$$t_i = \frac{\arg\min_t F(t) \geq k}{\mathbb{E}\left[\left|a_i^{\text{up}}\right|\right]} \tag{9}$$

This threshold indicates the maximal activation magnitude that should be pruned as zero. Different from CATS, this is a Channel-Wise Thresholding (CWT) technique that relates the model performance with the activation sparsity via introducing the *importance score* in Equation 5.

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

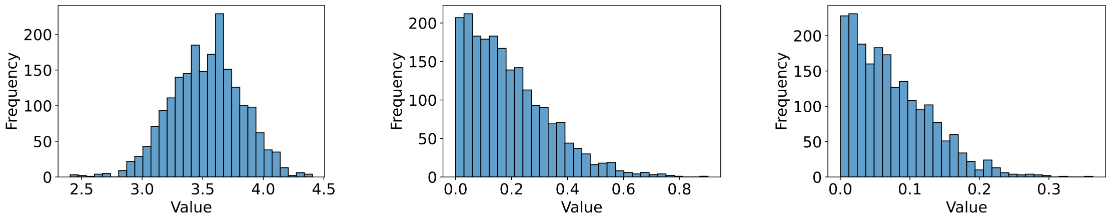
$$\text{CWT}(a_i) = \begin{cases} 0, & \text{if } |a_i| \leq t_i \\ a_i, & \text{if } |a_i| > t_i \end{cases} \tag{10}$$

and the output of the FFN modules is computed as,

$$\text{FFN}_{\text{CWT}}(x) = \left(\text{CWT}(a^{\text{gate}}) \odot a^{\text{up}}\right) W^{\text{out}} \tag{11}$$

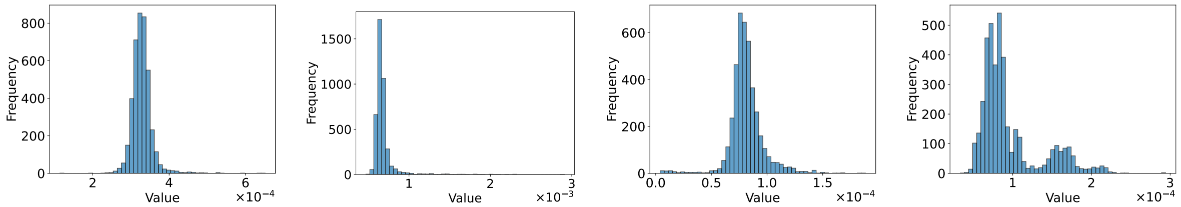
3.2 Selective Sparsification

Although the activation sparsity in attention modules is much lower than that in FFN modules, applying activation sparsification to these modules can still effectively reduce memory access and computational overhead. The standard attention mechanism involves four linear projects: query, key, value, and output projection. Similar to that in FFN modules, the objective of activation sparsification in the attention module is to **find the optimal pruned activation tensor that guarantees a specified sparsity level while minimizing the output**



(a) channel 1241 (b) channel 8718 (c) channel 12005

Figure 1: Distribution of absolute activation values $|a_i^{\text{up}}|$ across different inputs for various channels in the FFN of layer 16 of the Llama-3-8B model.



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

Figure 2: Distribution of $\|W_{i,:}\|_2^2$ of different rows i in attention projections of layer 16 of Llama-3-8B.

difference of the succeeding layer before and after pruning. More formally, the problem is defined as,

$$\arg\min_{\hat{x}} \|xW - \hat{x}W\|_2^2 \tag{12}$$

where W is the weight tensor of the projection layer.

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 and Alistarh, 2022):

$$\mathrm{E} = \mathbf{g}(\hat{x}-x)^T + \frac{1}{2}(\hat{x}-x)\mathbf{H}(\hat{x}-x)^T + O(\|\hat{x}-x\|^3) \tag{13}$$

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

$$\mathbf{g} = \left.\frac{\partial \mathrm{E}}{\partial \hat{x}}\right|_{\hat{x}=x} = 0 \tag{14}$$

$$\mathbf{H} = \left.\frac{\partial^2 \mathrm{E}}{(\partial \hat{x})^2}\right|_{\hat{x}=x} = WW^T \tag{15}$$

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 13 can be simplified as:

$$\mathrm{E} \approx \sum_{i=1}^{d} \|W_{i,:}\|_2^2 (\hat{x}_i - x_i)^2 \tag{16}$$

where $\|W_{i,:}\|_2^2$ denotes the square of 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,

$$\arg\min_{\mathcal{P}} \sum_{i\in\mathcal{P}} \|W_{i,:}\|_2^2 x_i^2 \tag{17}$$

To further simplify Equation 17, this paper analyzes the statistics of the weight matrix in the attention mechanism. Figure 2 shows the distribution of $\|W_{i,:}\|_2^2$ of different rows in projection weights. From the results, all rows from the same weight exhibit similar $\|W_{i,:}\|_2^2$, therefore we can eliminate this coefficient from Equation 17 and derive the simplified final objective:

$$\arg\min_{\mathcal{P}} \sum_{i\in\mathcal{P}} |x_i| \tag{18}$$

Based on Equation 18, this paper adopts the tensor-wise thresholiding method proposed by CATS (Lee et al., 2024) for the projection layers of attention modules:

$$\mathrm{CATS}(x_i) = \begin{cases} 0, & \text{if } |x_i| \le t \\ x_i, & \text{if } |x_i| > t \end{cases} \tag{19}$$

However, which layers 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-2-7B 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,

$$\mathrm{C}_{t_\mathrm{o}}(\mathrm{Attn}(\mathrm{C}_{t_\mathrm{i}}(x)W^{\mathrm{q}}, \mathrm{C}_{t_\mathrm{i}}(x)W^{\mathrm{k}}, \mathrm{C}_{t_\mathrm{i}}(x)W^{\mathrm{v}}))W^{\mathrm{o}} \tag{20}$$

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,

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

Experimental results (ref. Section 4.4) demonstrate that selective sparsification results in significantly 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

Algorithm 3.1 *spvmm* (sparse vector-matrix multiplication) kernel

```
Input: The sparse input vector $x \in \mathbb{R}^{1\times K}$, the
    weight matrix $W \in \mathbb{R}^{K\times N}$, the number of out-
    put 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:   for $k$ from 0 to $K$ do
 3:     if $x[k] \neq 0.0$ then
 4:       $n1_{upp} = \min(B, N - n0)$
 5:       for $n1$ from 0 to $n1_{upp}$ VECTORIZED
          do
 6:         $y[n0 + n1] \mathrel{+}= x[k] \times W[k][n0 +
            n1]$
 7:       end for
 8:     end if
 9:   end for
10: end for
11: return y
```

Algorithm 3.2 *vmmsp* (vector-matrix multiplication with output sparsity) kernel

```
Input: The input vector $x \in \mathbb{R}^{1\times K}$, the weight
    matrix $W \in \mathbb{R}^{N\times K}$, the output mask vector
    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 PAR-
    ALLEL do
 2:   $n1_{upp} = \min(B, N - n0)$
 3:   for $n1$ from 0 to $n1_{upp}$ do
 4:     if mask$[n0 + n1] \neq 0.0$ then
 5:       $accum = 0.0$
 6:       for $k$ from 0 to $K$ VECTORIZED do
 7:         $accum \mathrel{+}= W[n0 + n1][k] \times x[k]$
 8:       end for
 9:       y$[n0+n1] = accum \times$ mask$[n0+n1]$
10:     end if
11:   end for
12: end for
13: return y
```

To achieve wall-clock speedup and reduce inference latency via sparse activations, we 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 is used in FFN up projections.

Algorithm 3.1 and Algorithm 3.2 show the detailed steps of *spvmm* and *vmmsp*, respectively. Both algorithms splits the output vector into blocks of size B and accumulates the results of each block in parallel (Line 1 in Algorithm 3.1, Line 1 in Algorithm 3.2). What's more, they both reduce the latency by bypassing unnecessary weight reads and computations (Line 3 in Algorithm 3.1, Line 4 in Algorithm 3.2). Notably, although the implementation of the *vmmsp* kernel is relatively straightforward, the *spvmm* kernel requires a more complex approach because its access to each column of W is not continuous. To address this, we employ two advanced optimizations. First, we transpose the linear projection weights in advance during the model preprocessing stage, to ensure memory access continuity. Additionally, we employ loop tiling and loop reordering to make sure that each threads compute independently without the need for synchronization or atomic operations.

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 methods for the attention module and analysis on performance and efficiency over different sparsity level. Additionally, this paper conducts extended comparisons with other state-of-the-art training-free pruning methods, to validate the effectiveness of the proposed CHESS .

4.1 Datasets and Experimental Setup

Datasets We utilize ARC Challenge (Arc-C), ARC Easy (Arc-E), BoolQ, HellaSwag (HS), OpenbookQA (QA), PIQA, SCI-Q, Winogrande (WG) as benchmarks for downstream tasks, employing the Evaluation Harness library from Eleuther AI to ensure consistency with 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 conducted experiments using several state-of-the-art LLMs, including Llama-2-7B, Llama-2-13B, Llama-2-70B, Llama-3-8B and Mistral-7B. These models feature different attention mechanisms, specifically MHA and GQA, and utilize SwiGLU as the activation function for the FFN modules. We tested four different configurations across all five 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 channel-wise thresholding and selective sparsification method.

For the ablation study, we evaluate the following three models:

- **Llama-3:** the Llama-3 8B model without activation sparsification.
- **FS:** No activation sparsification applied to the FFNs; full sparsification applied in the attention modules.
- **SS:** No activation sparsification applied to the FFNs; selective sparsification applied in the attention modules.

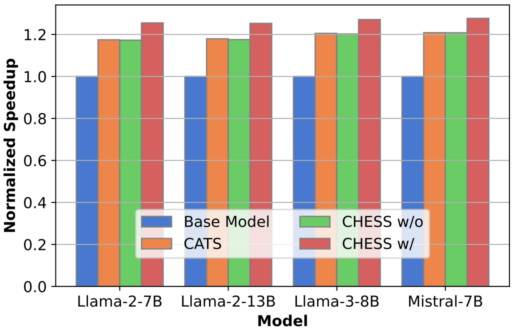


Figure 3: End-to-end inference speedup

Implementation Details For all models involving activation sparsification, thresholds are sampled from a subset of the C4 dataset (Raffel et al., 2020). Following the settings in CATS (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 (Paszke et al., 2019) and HuggingFace Transformers v4.39.3 (Wolf et al., 2019). End-to-end decoding speedups are measured on a randomly collected subset of C4 dataset. Kernel efficiency and end-to-end speedup 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 bitwidth quantizations does not materially affect our results (Lee et al., 2024).

4.2 Main Results on Downstream Tasks

Table 1 compares the accuracy of different models across eight downstream tasks and Figure 3 evaluates the end-to-end inference speedups. Experimental results draw the following conclusions.

Channel-wise thresholding can reduce accuracy degradation while achieving comparable sparsity. Achieving a comparable sparsity, the proposed CHESS w/o exhibits a smaller average

Models	**AP**$\downarrow$	**Arc-C**$\uparrow$	**Arc-E**$\uparrow$	**BoolQ**$\uparrow$	**HS**$\uparrow$	**QA**$\uparrow$	**PIQA**$\uparrow$	**SciQ**$\uparrow$	**WG**$\uparrow$	**Avg**$\uparrow$
Llama-2-7B	100%	43.43	76.26	77.68	57.15	31.40	78.07	93.90	69.14	65.87
CATS	78.16%	41.13	74.07	72.17	57.03	31.60	77.48	92.80	**66.69**	64.12
CHESS w/o	78.17%	**41.47**	**74.62**	**74.22**	**57.15**	32.40	77.20	93.20	66.61	**64.60**
CHESS w/	**70.05%**	40.36	74.37	**74.22**	56.60	**33.60**	**77.86**	**93.30**	66.22	64.56
Llama-2-13B	100%	48.38	79.38	80.61	60.06	35.00	79.05	94.60	72.22	68.66
CATS	77.97%	**46.93**	77.44	75.60	60.42	33.80	78.78	94.10	70.64	67.21
CHESS w/o	77.98%	46.67	**77.95**	**79.11**	**60.64**	34.00	78.89	**94.30**	70.09	67.71
CHESS w/	**69.82%**	46.84	**77.95**	78.50	60.47	**34.40**	**79.00**	94.20	**70.88**	**67.78**
Llama-2-70B	100%	54.44	82.70	83.76	64.77	37.40	82.21	96.90	77.98	72.52
CATS	72.96%	**54.61**	81.48	79.72	64.30	**37.20**	81.61	**96.10**	**76.32**	71.41
CHESS w/o	72.97%	54.10	**81.78**	**82.17**	**64.92**	36.60	81.12	96.00	**76.32**	**71.63**
CHESS w/	**65.24%**	54.35	81.69	81.65	64.45	36.80	**81.77**	**96.10**	76.24	**71.63**
Llama-3-8B	100%	50.17	80.22	81.07	60.15	34.60	79.60	96.30	73.32	69.42
CATS	74.96%	45.22	75.76	78.65	57.34	32.40	78.40	**94.90**	70.88	66.69
CHESS w/o	74.96%	**47.44**	**77.02**	**79.97**	**59.06**	**32.80**	78.67	94.60	**71.90**	**67.68**
CHESS w/	**67.80%**	46.67	76.85	78.04	58.62	**32.80**	**79.22**	94.20	70.17	67.07
Mistral-7B	100%	48.89	79.71	82.11	60.87	33.40	80.20	95.80	73.64	69.33
CATS	73.59%	48.29	77.40	79.42	60.65	31.60	**80.52**	94.40	70.48	67.85
CHESS w/o	73.59%	48.21	**79.71**	**80.55**	**61.70**	33.20	80.41	**95.80**	**70.88**	68.81
CHESS w/	**66.04%**	**49.32**	79.59	80.12	61.60	**34.40**	80.20	95.00	70.56	**68.86**

Table 1: Comparison of inference accuracy on downstream tasks of different models. 'AP' refers to the ratio of activated parameters.

Model	**AP**$\downarrow$	**Arc-C**$\uparrow$	**Arc-E**$\uparrow$	**BoolQ**$\uparrow$	**HS**$\uparrow$	**QA**$\uparrow$	**PIQA**$\uparrow$	**SciQ**$\uparrow$	**WG**$\uparrow$	**Avg**$\uparrow$
Llama-3-8B	100%	50.17	80.22	81.07	60.15	34.60	79.60	96.30	73.32	69.42
FS	**90.94%**	46.16	79.00	78.56	57.14	34.80	78.02	96.10	71.59	67.67
SS	92.84%	**50.17**	**79.67**	**79.57**	**59.31**	**35.00**	**79.71**	**96.30**	**72.85**	**69.07**

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

performance drop of 1.07 across five base models and eight downstream tasks, compared to the 1.70 degradation of CATS. Specifically, CHESS w/o consistently outperforms CATS on ARC Easy, BoolQ, and HellaSwag, while showing modest gains on the remaining benchmarks.

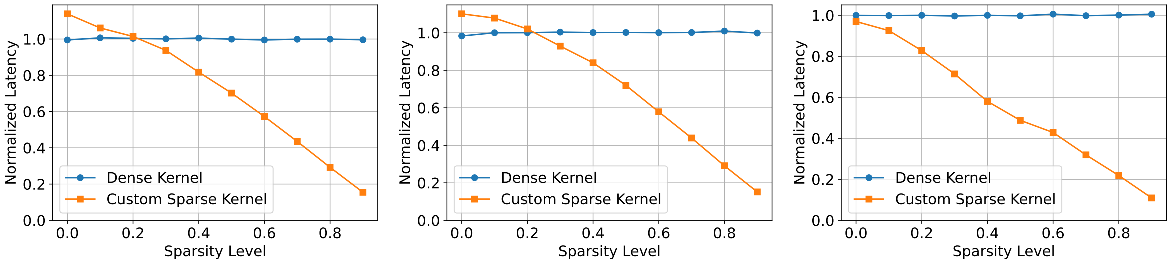
Selective sparsification of attention modules further improves sparsity while maintaining model accuracy. Compared to CHESS w/o, the average performance of CHESS w/ degrades by 0.04 on Llama-2-7B and 0.61 on Llama-3-8B, respectively. Interestingly, for larger models such as Llama-2-13B, Llama-2-70B, and Mistral-7B, CHESS w/ demonstrates comparable or even slightly superior overall performances. Specifically, CHESS w/ outperforms on OpenbookQA, but underperforms on ARC Easy, HellaSwag and BoolQ, while showing similar results on ARC Challenge, PIQA, SciQ, and Winogrande. These results demonstrate that the additional selective sparsification on attention modules has minimal impact on performance. In comparison to CATS, CHESS w/ consistently delivers superior average performance with fewer activated parameters.

4.3 End-to-End Decoding Speedup

CHESS achieves end-to-end speedups of up to 1.27x compared to Transformers baselines. 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, 1.20x on Llama-3-8B, and 1.21x on Mistral-7B, respectively. This is because of the comparable parameters activated per decoding pass of these two methods. When employing attention sparsification, 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 and Mistral-7B, respectively. Due to the limited capacity of main memory of edge devices, we did not perform the end-to-end

Model	AP↓	Arc-C↑	Arc-E↑	BoolQ↑	HS↑	QA↑	PIQA↑	SciQ↑	WG↑	Avg↑
Llama-3-8B	100%	50.17	80.22	81.07	60.15	34.60	79.60	96.30	73.32	69.42
Relufication	67.10%	20.73	24.66	38.04	25.39	17.80	53.59	1.70	49.64	28.94
Wanda	**53.49%**	30.80	62.58	**68.01**	41.23	24.40	70.73	**91.20**	62.35	56.41
CHESS	54.92%	**36.86**	**67.51**	66.91	**52.92**	**28.80**	**75.35**	89.60	**63.69**	**60.21**

Table 3: Extended comparisons with state-of-the-art training-free pruning methods. ‘AP’ refers to the ratio of activated parameters; ‘CHESS’ refers to the proposed CHESS model with a sparsity level of 0.7.



(a) Attention projection (*spvmm* kernel) (b) Down projection (*spvmm* kernel) (c) Up projection (*vmmsp* kernel)

Figure 4: Comparison between custom sparse kernels and PyTorch dense kernel on latency of linear projections

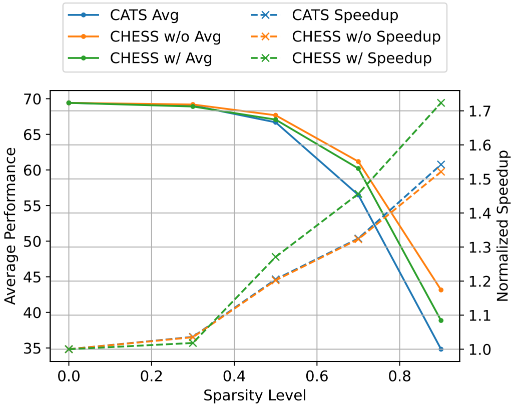


Figure 5: Downstream performance and end-to-end speedups of each method under different sparsity levels.

speedup experiment for the Llama-2-70B model. However, based on the activated parameter count per inference pass, its speedup is estimated to be similar to that of Mistral-7B and Llama-3-8B.

4.4 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.

4.5 Kernel Efficiency

As illustrated in Figure 4, this paper compares 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* kernel 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.

4.6 Impact on Different Sparsity Levels

Figure 5 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/ exhibits 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/ exhibits slightly reduced speedup, mainly due to the suboptimal efficiency of the *spvmm* kernel at lower sparsity levels.

4.7 Extended Comparisons with State-of-the-Art Training-Free Pruning Methods

To further demonstrate the effectiveness of our proposed CHESS method, we extend our comparisons to include other state-of-the-art training-free pruning approaches, such as Relufication (Mirzadeh et al., 2024) and Wanda (Sun et al., 2024). Notably, although Relufication achieves competitive performance when fine-tuned, it struggles with performance degradation in training-free scenarios. Wanda, on the other hand, focuses on weight pruning, which belongs to a different branch of work. Weight pruning typically results in unstructured sparsity or semi-structured sparsity, which is only supported by high-end NVIDIA GPUs with Ampere or Hopper architectures. In contrast, our proposed CHESS does not rely on specialized GPU architecture, making it more suitable for deploying on edge devices.

As presented in Table 3, the proposed CHESS method achieves superior performance in most benchmarks while activating comparable or fewer parameters compared to both Relufication and Wanda. Specifically, CHESS with a sparsity level of 0.7 outperforms other methods on several benchmarks including Arc Challenge, Arc Easy, HellaSwag, OpenbookQA, PIQA and Winogrande. Despite using only 54.92% of the model's parameters per decoding pass, CHESS delivers an average performance (60.21) that surpasses Wanda (56.41) and Relufication (28.94). These results emphasize the advantage of CHESS over existing methods.

5 Related Work

Various methods have been proposed to address the challenges associated with deploying LLMs locally. Weight quantization (Xiao et al., 2023; Frantar et al., 2022; Lin et al., 2024) 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 (Zhao et al., 2024; Liu et al., 2024; Hooper et al., 2024). These methods can be applied along with our proposed CHESS method.

Weight pruning (Frantar and Alistarh, 2023; Sun et al., 2024) 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 (Leviathan et al., 2023; Zhou et al., 2023) 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 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.

Limitations

The limitations of this work can be summarized in two main aspects. First, while CHESS achieves lower accuracy degradation compared to existing methods with fewer activated parameters, it still experiences a noticeable accuracy loss at higher sparsity levels. Future research could explore fine-tuning techniques to mitigate this decline in performance. Second, CHESS is optimized for inference with a batch size of one, which is suitable for edge deployment scenarios typically involving a single user. However, under larger batch sizes, the structured sparsity of activation tensors deteriorates into unstructured sparsity, limiting potential speedups and reducing effectiveness in data center deployments, where larger batch sizes are common.

Acknowledgements

We thank all the reviewers for their insightful comments. This work is supported by the National Natural Science Foundation of China (No. 62472330, U20A20177, 62272348, U22B2022), the National Key Research and Development Program of China (No. 2022YFB3104502), the State Key Laboratory of Computer Architecture (ICT, CAS) under Grant No. CARCH A202112, and Wuhan Science and Technology Joint Project for Building a Strong Transportation Country (No.2023-2-7).

References

Keivan Alizadeh, Seyed-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. *CoRR*, abs/2312.11514.

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. *CoRR*, abs/2401.10774.

Elias Frantar and Dan Alistarh. 2022. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323.

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*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org.

Babak Hassibi and David G. Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 164–171. Morgan Kaufmann.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length LLM inference with KV cache quantization. *CoRR*, abs/2401.18079.

Yann LeCun, John S. Denker, and Sara A. Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. 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.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

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

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org.

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

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. *CoRR*, abs/2402.02750.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach 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. *CoRR*, abs/1912.01703.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2018. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net.

Noam Shazeer. 2020. GLU variants improve transformer. *CoRR*, abs/2002.05202.

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. *CoRR*, abs/2402.13516.

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. 2024. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

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.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate LLM serving. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *CoRR*, abs/2310.08461.