HWPQ: Hessian-free Weight Pruning-Quantization For LLM Compression And Acceleration

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have achieved remarkable success across numerous domains. However, the high time complexity of existing pruning and quantization methods significantly hinders their effective deployment on resource-constrained consumer or edge devices. In this study, we propose a novel Hessian-free Weight Pruning-Quantization (HWPQ) method. HWPQ eliminates the need for computationally intensive Hessian matrix calculations by introducing a contribution-based weight metric, which evaluates the importance of weights without relying on second-order derivatives. Additionally, we employ the Exponentially Weighted Moving Average (EWMA) technique to bypass weight sorting, enabling the selection of weights that contribute most to LLM accuracy and further reducing time complexity. Our approach is extended to support 2:4 structured sparsity pruning, facilitating efficient execution on modern hardware accelerators. Experimental results demonstrate that HWPQ significantly enhances the compression performance of LLaMA2. Compared to state-of-the-art quantization and pruning frameworks, HWPQ achieves average speedups of 5.97× (up to 20.75×) in quantization time and 12.29× (up to 56.02×) in pruning time, while largely preserving model accuracy. Furthermore, we observe a 1.50× inference speedup compared to the baseline.

1 Introduction

Recent years have witnessed an explosive growth in the capabilities of Large Language Models (LLMs). However, this advancement comes at the cost of exponentially growing model sizes, leading to substantial monetary and energy costs (Zhao et al., 2023). Consequently, there have been growing efforts to reduce these costs through model compression, with pruning and quantization emerging as the two most popular approaches. Pruning removes network weights by setting them to zero, while quantization reduces the precision of neural network weights during storage and computation. A key finding is that these two widely used methods are not orthogonal and exhibit a convergent trend (van Baalen et al., 2020; Hu et al., 2021; Schaefer et al., 2023).

Despite impressive progress, compression remains a labor-intensive process. Pruning and quantization are typically performed independently, and many methods require Recovery Fine-Tuning (RFT) to maintain performance post-compression (van der Ouderaa et al., 2024), rendering the entire process computationally expensive and difficult to scale. Current mainstream approaches circumvent the need for RFT by leveraging Hessian matrix computations (Frantar et al., 2022; Frantar and Alistarh, 2022; Fang et al., 2023; Sawmya et al., 2024). Although these methods exhibit mathematical elegance, their practical implementation faces significant computational challenges. Specifically, computing second-order derivatives across all network weights generates a massive Hessian matrix, whose dimensionality scales quadratically with the number of parameters, resulting in prohibitive computational complexity. In real-world deployment scenarios, resource limitations and stringent time constraints often hinder the effective implementation of quantization and pruning strategies. Moreover, the emergence of advanced GPU architectures highlights the need for structured hardware-aware pruning methodologies that achieve genuine performance acceleration while maintaining computational efficiency (Tang et al., 2022; Xia et al., 2024; Lu et al., 2022; Liu et al., 2017). This underscores the importance of advancing pruning methods for the 2:4 sparse format.

In this study, we propose HWPQ, a novel compression method designed to circumvent the high computational complexity associated with Hessian matrix and its inverse calculations by developing

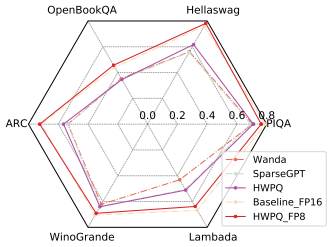

Figure 1: HWPQ achieves better compression in less time

an alternative algorithm. We anticipate that our method will advance future research in domains where Hessian matrix computations are critical. First, our observations indicate that identifying weights with minimal loss contribution depends more on their relative importance than on absolute values. Consequently, HWPQ replaces Hessian matrix computations by constructing a numerically preserved sequence as contribution-oriented weight metrics, derived from a series of loss values. Additionally, we introduce the Exponentially Weighted Moving Average (EWMA) method, borrowed from the Transmission Control Protocol (TCP), to replace traditional sorting methods, further reducing computational complexity. Moreover, we extend this approach to support 2:4 structured sparsity pruning. By eliminating inverse quantization operations and incorporating efficient execution kernels, we achieve a more streamlined and efficient computational process. As shown in Figure 1, our method demonstrates exceptional compression performance, achieving state-of-the-art results. The key contributions of our work are summarized as follows:

- We propose a computationally efficient weight metric that eliminates costly Hessian matrix calculations, reducing the time complexity from $O(n^3)$ to $O(n)$ while preserving model accuracy. This advancement achieves significant speedups: 4.88× faster quantization than AutoGPTQ, 2.82× faster than AutoAWQ, and 10.21× faster than SpQR. For model pruning, our method achieves an average speedup of 43.75× over SparseGPT and 12.29× over Wanda. When extended to 2:4 sparsity, further acceleration is achieved.

- We develop an innovative mixed pruning-quantization approach using FP8 precision, dynamically identifying and removing model weights based on their impact magnitude. This unified framework enables simultaneous pruning and quantization, significantly streamlining the compression process.

- We implement dequantization-free inference in FP8 precision, optimized for Tensor Cores with 2:4 sparsity support. This optimization delivers significant performance improvements, achieving 1.50× speedup on Attention layers and 1.60× on MLP layers in LLaMA2-7B, while reducing dequantization overhead by over 80

2 Motivation & Relate Work

2.1 Motivation

Post-training compression has become a widely used technique, initially developed and extensively applied in quantization research (Banner et al., 2019; Zhao et al., 2019; Nagel et al., 2020). Recently, it has been successfully adapted to network pruning, demonstrating promising results (Kwon et al., 2022; Fu et al., 2022; Sun et al., 2023). Common methods include magnitude-based (Gale et al., 2019), first-order (Kurtic et al., 2022), and second-order (Sanh et al., 2020) pruning approaches.

Among these approaches, Hessian-based methods, which compute the second-order derivatives of weights, have demonstrated exceptional performance in preserving model accuracy without requiring retraining. However, current mainstream approaches still require hundreds of seconds to sparsify a 2.7B model, even on high-performance GPUs. For a 175B Transformer model, processing time increases linearly, reaching hundreds of hours. This substantial time requirement significantly delays the deployment of personalized models, highlighting the challenges of scaling post-training compression to extremely large models. The primary bottleneck lies in constructing the Hessian matrix $H = \left[\frac{\partial^2 E}{\partial w_i \partial w_j}\right]$, which has a time complexity of $O(n^3)$. Therefore, this work focuses on developing a novel compression approach that bypasses the high computational cost of the Hessian matrix and its inverse while closely approximating its accuracy-preserving performance, achieving a balance between runtime efficiency and precision, and enabling scalability to very large models.

2.2 Related Work

The most fundamental sparsification approach is magnitude-based pruning, which achieves sparsity by setting the smallest weights to zero (Han et al.,

2015; Zhu and Gupta, 2017). Although these methods scale well, they often cause significant performance degradation in LLMs (Frantar and Alistarh, 2023; Harma et al., 2024). To improve sparsification, researchers proposed the Optimal Brain Surgeon (OBS) method (Hassibi et al., 1993), which innovatively uses the inverse of the Hessian matrix to update unpruned weights, thereby compensating for errors caused by weight removal. However, OBS faces computational bottlenecks in practical applications - calculating and storing the inverse Hessian matrix is computationally infeasible for models with millions of parameters. To address this challenge, recent research has proposed two improvement approaches: one approximates the inverse Hessian matrix calculation, such as the WoodFisher method (Singh and Alistarh, 2020); the other performs layerwise pruning, known as Optimal Brain Compression (OBC) (Frantar and Alistarh, 2022). While these methods perform well on medium-scale networks, they struggle with larger language models (Frantar et al., 2022).

GPTQ (Frantar et al., 2022) addresses this issue by quantizing weight matrices using a grouping scheme and compensating updates to all yet-unquantized weights in the next column of that group through the Hessian matrix. SparseGPT (Frantar and Alistarh, 2023) applies the same pruning idea and uses unstructured and semi-structured pruning to simplify large language models, while Sparse Expansion (Sawmya et al., 2024) improves inference efficiency by computing a separate Hessian matrix for each input cluster to allow specialists to specialize, then using the SparseGPT pruning algorithm with that matrix to prune the expert weight matrices. Wanda (Sun et al., 2024) simplified this idea by using only the diagonal of the Hessian.

Simultaneously, to achieve tangible speed improvements in practical applications, there has been a rowing recognition of the need to apply pruning in a structured and hardware-compatible manner (Santacroce et al., 2023; Ma et al., 2023a; Li et al., 2023; Xia et al., 2024). This approach is typically followed by additional training (or fine-tuning) to restore any diminished performance. For example, the LLM-pruner (Ma et al., 2023b) eliminates specific connection structures within LLMs prior to further training. Similarly, the Large Language Model Surgeon (van der Ouderaa et al., 2024) interleaves recovery fine-tuning with pruning.

3 The HWPQ Method

3.1 Contribution-Oriented Weight Metrics

Our objective is to identify weights that make minimal contributions to the loss function, such that their removal would not substantially affect the model's output. In this regard, our main focus lies in analyzing the relative importance of different weights rather than their absolute values, an aspect that has been largely neglected in previous research. Previous studies have quantified the influence of individual weights on the variation of E by precisely computing their contributions through the Hessian matrix. The supplementary term in the loss function is expressed as follows:

$$L = \frac{1}{2}\frac{w_q^2}{H_{qq}^{-1}} \tag{1}$$

When applied to quantization, the expression becomes:

$$L_q = \frac{1}{2}\frac{(w_q - \text{quant}(w_q))^2}{H_{qq}^{-1}} \tag{2}$$

A crucial issue arises from the fact that the matrix $(2XX^\top)$ is not positive definite, as its determinant is zero, meaning it does not possess an inverse. To address this, we introduce a small perturbation term, denoted as:

$$H = 2XX^\top + \sum_i \text{diag}(2XX^\top)I \tag{3}$$

Where I represents the identity matrix. This ensures that matrix operations can be performed safely. When using PyTorch, numerical methods are used for matrix computation, and due to errors in floating-point calculations, $2XX^\top$ can result in matrices with extremely large values, leading to instability. By incorporating these small perturbations, we achieve stability in numerical computations with almost zero overhead.

However, computing Δw and L for every weight can be computationally expensive. The time complexity of quantization primarily lies in computing the inverse matrix H^{-1}, which typically has a complexity of $O(n^3)$. Even with the capability to compute Hessian matrices for each row in parallel, the total time complexity remains at $O(n^3) + O((\frac{n}{m})^3) = O(n^3)$.

To reduce the overall time complexity, the key is to avoid the computation of H and H^{-1}. Our

goal is not to obtain the exact value of L for each weight at this stage, but rather to construct a numerically stable sequence as contribution-oriented weight metrics and to derive numerical characteristics among a series of L values (such as magnitudes, variance, and averages).

Denoting $\sum x_i^2$ as S, in Formula 3 that we constructed, $H_{qq} = 2(x_q^2 + \frac{1}{n}S)$. Noticed that H_{qq}^* is independent of x_q, H_{qq}^* can actually be written as $\mathbf{det}\left(2X_0X_0^\top + \frac{2S}{n}I\right)$, where X_0 is the original X without the q^{th} element. Since $H_{qq}^{-1} = \frac{H_{qq}^*}{\mathbf{det}(H)}$, and

$$\begin{aligned} H_{qq}^* &= 2(\frac{2S}{n})^{n-2}(\frac{S}{n} + S - x_q^2) \\ \mathbf{det}(H) &= 2(\frac{2S}{n})^{n-1}(\frac{S}{n} + S) \end{aligned} \tag{4}$$

Thus, we can express H_{qq}^{-1} as:

$$H_{qq}^{-1} = \frac{\frac{S}{n} + S - x_q^2}{\frac{2S}{n}(\frac{S}{n} + S)} = \frac{nS + n^2(S - x_q^2)}{2S(S + nS)} \tag{5}$$

Then, we can simplify further:

$$\begin{aligned} \frac{H_{qq}^{-1}}{1 - x_q^2/S} &= \frac{nS + n^2(S - x_q^2)}{2S(S + nS)} \cdot \frac{S}{S - x_q^2} \\ &= \frac{n}{2(1+n)} \cdot \frac{1}{S - x_q^2} + \frac{n^2}{2S(1+n)} \end{aligned} \tag{6}$$

In LLMs, n is sufficiently large(e.g., 4096 in LLaMA2-7B), ensuring $S >> x_q^2$. Therefore, we can approximate $S - x_q^2 \sim S$, leading to:

$$\frac{H_{qq}^{-1}}{1 - x_q^2/S} \sim \frac{n^2 + n}{2S(1+n)} = C \tag{7}$$

Now we observe that $\frac{H_{qq}^{-1}}{1 - x_q^2/S}$ approaches a constant. Since we are concerned with the comparative magnitudes of values rather than the exact value of each L, we replace H_{qq}^{-1} with $(1 - x_q^2/S)$ to avoid computations involving the Hessian matrix. Thus, we compute L as follows:

$$L = \frac{1}{2}\frac{w_q^2}{1 - x_q^2/S} \tag{8}$$

Where S represents the sum of all x_i^2 for every x_i in X.

In this formulation, the Hessian matrix is no longer needed. To determine which weights should be removed, we can simply sort the L values of all weights and eliminate those with the smallest L values. As we demonstrated earlier, smaller L values indicate that the removal of those weights will have a minor effect on the loss function. The time complexity of computing all L values is $O(n)$, while the cost of the most common sorting algorithms is $O(n \log n)$, thus reducing the overall time complexity to $O(n \log n)$. It is worth noting that we perform this operation simultaneously across different rows, where n represents the number of weights in a row.

3.2 EWMA Adaption

To further reduce the time complexity, our next objective is to find an alternative method to replace sorting, allowing us to assess where a particular L value stands among all L values.

The Exponentially Weighted Moving Average (EWMA) is a technique used for estimating the mean and variance of a sequence of data points. In the context of Transmission Control Protocol (TCP), it is employed to estimate the round-trip time (RTT) of a connection (Paxson et al., 2011).

In the practical implementation of TCP, the EWMA method exhibits strong adaptability by dynamically estimating the mean and L1-mean norm error of the recent RTT over time. We apply this method to evaluate L. For each row, we treat the weights as a sequential list.

First, after calculating S as outlined in Step 1 of Figure 2 (Algorithm 1, line 1), we initialize a tensor state for each weight in a row. This tensor state consists of the following components: the dynamically updated S, the estimated mean (denoted as est), and the L1 mean norm error (denoted as dev). Subsequently, following Step 2 of Figure 2 (Algorithm 1, line 4), we sequentially compute a series of L_i values. If L_i satisfies the condition $L < \mathrm{est} - \mathrm{la} \times \mathrm{dev}$ (Algorithm 1, line 8), we consider its contribution to the loss function to be minimal and prune it; otherwise, we symmetrically quantize it to FP8 format, as shown in Step 3 of Figure 2 (Algorithm 1, lines 9 and 12).

Next, we update the tensor state according to the procedure outlined in Table 1, as illustrated in Step 4 of Figure 2 (Algorithm 1, lines 10, 13, 16, and 17), until all weights in the row are compressed. Throughout this process, the overall time complexity is reduced to $O(n)$, demonstrating that we can

State	Updating Method	Initial Value
est	$\leftarrow (1-\alpha)est+\alpha L_i$	L_0
dev	$\leftarrow (1-\beta)dev+\beta\,\lvert est-L_i\rvert$	0
S	$\leftarrow S-x_i^2$ (if pruned) $\leftarrow S+(v_i^2-x_i^2)$ (if quanted)	$\sum_{i=0}^{n-1}(x_i^2)$

Table 1: The method for tensor state update. Parameter α, β and la can be tuned for different level of sparsity. v_i is the quantized value of x_i.

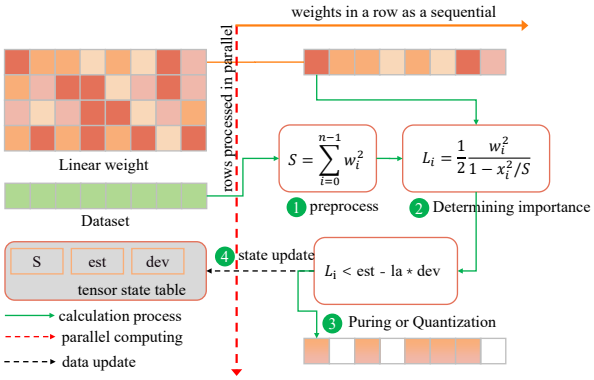

Figure 2: Our design of novel hybrid pruning and quantization method, using EWMA criteria.

evaluate the contribution of each weight to the loss function and quantize the model to sparse FP8 in linear time.

The Full Algorithm. Finally, we present the full pseudocode for HWPQ in Algorithm 1, including the optimizations discussed above.

3.3 2:4 Sparsification

To achieve efficient computation of structured sparse matrices on dedicated accelerators (Tang et al., 2022; Liu et al., 2023), our pruning method supports hardware-friendly structured sparsity. In implementation, we adopt fine-grained selection to support the 2:4 structured sparsity pattern. By leveraging Tensor Cores' native support for this pattern, we partition each row of weights into groups of four and identify the two smallest weights in each group through five-way comparison on average. This approach maintains the time complexity of pruning and quantization at $O(n)$ while achieving weight structured sparsity without introducing additional overhead, enabling a 2× improvement in inference throughput.

4 Intergration of HWPQ into Taichi framework

HWPQ significantly reduces LLMs size. However, during inference, many frameworks acquire de-quantized weights by multiplying a scaling factor (floating-point) with quantized integers (Lin et al., 2023; Lee et al., 2023). This process inevitably leads to extra time and memory overhead.

Algorithm 1 The HWPQ algorithm. We prune the matrix W to $sp\%$ sparsity

Input: $W_{nrow\times ncol}$, $X_{1\times n}$, sp
Parameter: α, β, la
Output: $C_{nrow\times ncol}$

Let $S=\sum_{i=0}^{n-1}(x_i^2), dev=0$
Parallel calculation for each row
for $i=0,1,...,n-1$ **do**
  $L_i=\frac{1}{2}\frac{w_i^2}{1-x_i^2/S}$
  if $i==0$ **then**
    $est=L_0$
  end if
  if $L_i<\text{est}-\text{la}\times\text{dev}$ **then**
    $w_i=0$ //Puring
    $S=S-w_i^2$
  else
    $v_i=\text{FP8}(w_i)$ //Quantization
    $S=S+(v_i^2-w_i^2)$
    $w_i=v_i$
  end if
  $est=(1-\alpha)est+\alpha\,L_i$
  $dev=(1-\beta)dev+\beta\,|est-L_i|$
  $c_i=w_i$
end for
return $C_{nrow\times ncol}$

To solve this problem, we improved the operator on the Taichi framework. During inference procedure, the input activation is quantized into FP8. In the Attention mechanism, the linear layers generating the query, key, and value, as well as the MLP, are replaced with FP8 precision, while other components retain FP16 precision. Within Tensor Cores, FP8 matrix operations and accumulations are performed with FP16 bit-width. The computations following the linear layers, often Root Mean Square Normalization (RMSNorm), are not linear layers, and FP16 can conveniently utilize existing operators. Finally, when the resultant output is fed into the subsequent linear layer, we re-quantize it to FP8.

Since the numerical format we use for quantization is fully consistent with the one used in hardware computations, we can directly utilize the quantized weights. In other words, we can deliver the weights directly to GPUs without transforming them again. This avoids redundant dequantization and enhances performance by leveraging the high throughput of FP8 computations on GPUs.

5 Experiments

5.1 Experimental setup

Models. We evaluate two model families: Pythia and LLaMA(including LLaMA2 and LLaMA3). Pythia is a collection of models focused on LLM interpretability, developed as a variant of GPT-

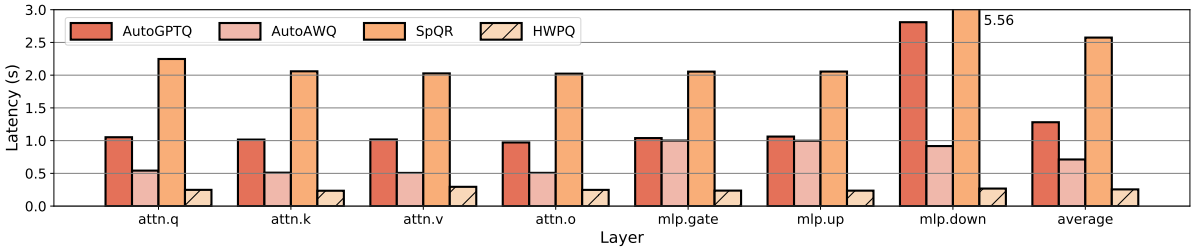

Figure 3: Comparative time analysis of compression across attention and perception layers in LLaMA2-7B model on a single RTX 4090 GPU

NeoX. LLaMA represents a series of open-source pre-trained models, with LLaMA3 being the latest iteration.

Datasets. We evaluate pruning and quantization using zero-shot perplexity (PPL) on WikiText2 (Merity et al., 2016). For task-agnostic performance, we adopt LLaMA's evaluation approach, testing on OpenCompass (Contributors, 2023) and Lm-evaluation-harness (Gao et al., 2024) benchmarks. These benchmarks offer a comprehensive assessment for LLMs. The datasets encompassed in this assessment are as follows: ARC(Easy and Challenge) (Boratko et al., 2018), WinoGrande (Sakaguchi et al., 2021), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019) and OpenbookQA (Mihaylov et al., 2018).

5.2 Evaluation of HWPQ Algorithm

Efficiency: The HWPQ algorithm provides a significant speedup. Our performance improvements stem from two primary factors. First, we have made an algorithmic advancement. The $O(n)$ algorithm offers remarkable scalability for pruning and quantization LLMs, allowing us to efficiently evaluate the importance of each weight without substantially increasing time consumption as the model size grows. Second, we have developed customized GPU operators using Taichi. Because the parameter matrix's rows are independent, we fully exploit row-wise vector parallelism to enhance pruning and quantization efficiency on GPUs.

Our experiments systematically demonstrate that compared with the state-of-the-art quantization methods such as AutoGPTQ, AutoAWQ, and SpQR (detailed in Figure 3), the average speedup reaches 4.88× (up to 10.49×), 2.82× (up to 4.23×), and 10.21× (up to 20.75×), respectively. When compared to pruning methods like SparseGPT and Wanda (detailed in Table 2), the average speedups reach 43.75× and 12.29×, respectively. The primary overhead in our approach stems from the just-in-time (JIT) compilation of kernel functions, which introduces a cold-start delay occurring only during the initial pruning-quantization phase. This characteristic makes our algorithm particularly advantageous for LLMs, where the continuous growth in model size necessitates more efficient methodologies for assessing weight contributions to final outputs. The HWPQ method, with its $O(n)$ pruning-quantization time complexity, effectively addresses this challenge, offering a scalable solution for modern LLMs.

Accuracy: Zero-shot performance in LLaMA2-7B. We conducted comprehensive fine-grained pruning experiments on the LLaMA2-7B model, systematically evaluating pruning ratios ranging from 10% to 50%, including structured 2:4 structured pruning configurations. Model performance was rigorously assessed using the Lm-evaluation-harness framework. As detailed in Table 2, our experimental results demonstrate that a model with a 20% pruning ratio successfully maintains 99.4% of the baseline model's performance without requiring any post-training. Remarkably, even at a 50% pruning ratio, the model retains 91.57% of its original performance. A comprehensive comparative analysis demonstrates that although our method may exhibit slightly inferior performance compared to well-established approaches such as Wanda and SparseGPT (which rely on computationally intensive Hessian matrix calculations) under certain pruning ratios, the performance gap remains negligible. More importantly, our method consistently achieves significantly faster pruning speeds across all pruning ratios, demonstrating its superior computational efficiency. These results validate the efficacy of HWPQ in enabling efficient model compression without the need for training data, delivering exceptional results in dramatically less time. To further substantiate the generalizability of our approach, we replicated the

Pruning Ratio	Method	Latency(s)↓	WikiText2↓	ARC_c	ARC_e	WinoGrande	PIQA	HellaSwag	OpenbookQA	Average↑
0%	LLaMA2-7B	_	9.36	43.51	71.54	70.48	78.94	76.13	44.00	64.10
10%	magnitude	2.43	9.54	43.94	**71.88**	**70.63**	**78.23**	76.04	**45.20**	64.32
	SparseGPT	371.83	10.44	43.86	71.42	70.24	77.52	76.19	42.40	63.60
	Wanda	103.32	**9.38**	**44.11**	71.54	**70.63**	76.12	78.78	45.00	**64.36**
	HWPQ (ours)	**9.55**	9.88	44.02	71.62	70.53	76.73	**78.80**	44.31	64.33
20%	magnitude	2.30	10.00	**44.54**	70.24	**69.69**	**78.40**	75.55	44.20	63.77
	SparseGPT	371.34	**9.56**	43.60	70.79	69.53	78.29	**76.12**	**45.20**	63.92
	Wanda	103.51	9.57	44.03	**71.42**	69.29	78.24	76.04	44.80	**63.97**
	HWPQ (ours)	**9.37**	9.67	43.42	70.33	69.37	78.31	76.01	44.88	63.72
30%	magnitude	2.76	11.40	42.83	69.57	68.90	77.58	73.73	42.20	62.46
	SparseGPT	357.03	**9.86**	43.68	70.07	69.13	78.07	75.17	44.20	63.38
	Wanda	100.04	9.90	**44.11**	**70.37**	**69.29**	**78.29**	**75.30**	**45.00**	**63.72**
	HWPQ (ours)	**9.12**	10.02	43.91	70.01	69.11	78.04	75.22	45.01	63.55
40%	magnitude	2.28	15.49	39.25	64.01	65.66	75.63	69.83	40.20	59.09
	SparseGPT	357.03	**9.39**	**43.83**	**69.69**	**69.13**	**78.84**	73.15	**45.40**	**63.34**
	Wanda	100.04	10.55	42.75	69.14	68.74	77.91	73.55	43.00	62.51
	HWPQ (ours)	**8.62**	10.34	42.84	69.01	68.09	78.01	**73.91**	42.69	62.43
50%	magnitude	2.29	44.37	36.77	53.78	59.74	70.73	60.88	36.20	53.01
	SparseGPT	361.29	**7.91**	39.33	66.65	**66.61**	76.44	68.84	**39.40**	**59.54**
	Wanda	108.96	8.01	**39.59**	64.85	65.90	**76.61**	**69.96**	38.40	59.21
	HWPQ (ours)	**7.85**	8.23	38.40	**67.32**	65.27	75.14	67.18	38.90	58.70
50%(2:4)	magnitude	14.99	120.90	30.12	48.86	59.58	68.77	56.30	34.01	49.60
	SparseGPT	410.10	**17.30**	32.34	53.57	63.93	69.21	55.64	34.80	51.58
	Wanda	114.26	20.49	30.55	53.45	62.19	70.35	56.17	**35.40**	51.35
	HWPQ (ours)	**7.73**	18.21	**32.42**	**56.48**	**64.01**	**71.00**	**61.72**	34.60	**53.37**

Table 2: Zero-shot performance of the pruned LLaMA2-7B. "Pruning Ratio" refers to the proportion of parameters removed relative to the original number of parameters. "Latency(s)" indicates represents the time overhead required for overall model pruning (excluding communication time such as loading to GPU). "Average" is calculated among six classification datasets. Bold indicates the best performance at the same compression rate; However, note that for Latency(s), it represents the best performance excluding the cost associated with magnitude.

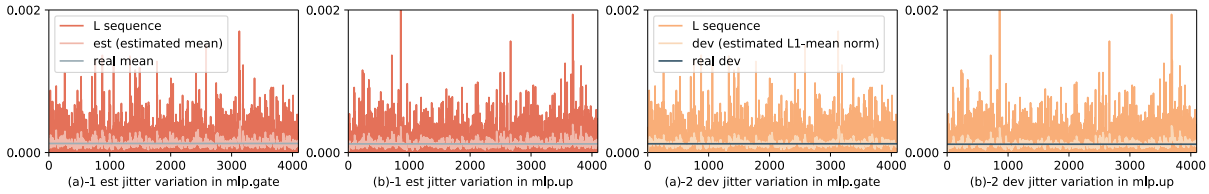

Figure 4: Statistical magnitude detection of L with EWMA method in LLaMA2-7B MLP blocks. x axis presents the sequence number of each weight, and y axis presents the numerical values. Ideal algorithms should show est approaches real mean and dev approaches real dev.

experiments on two additional model architectures: Pythia-2.8B and LLaMA3.1-8B. The consistent performance outcomes across these diverse models provide compelling evidence for the effectiveness and universal applicability of the HWPQ method in various model architectures.

Reliability: HWPQ adapts to weight changes and approaches global expectations. In Figure 4, we demonstrate how our HWPQ method consistently and accurately predicts the mean and variations of weights. As the weight sequence lengthens, HWPQ exhibits improved responsiveness and faster convergence. By adjusting the smoothing factors (α, β, and la), we can fine-tune the algorithm's responsiveness and stability to align with specific network characteristics. This capability enables us to determine whether the current row weight significantly impacts the final output, thereby deciding whether to prune or quantize it.

The data presented in Figure 4, derived from a layer of LLaMA2-7B, indicate that we can consistently approach the global weight mean shortly after an initial startup period. For the results in Figure 4, we set $\alpha = 0.125$, $\beta = 0.125$, and $\mathrm{la} = 4$, which is consistent with RFC 6298 (Paxson et al., 2011). This configuration remains robust even as the parameters undergo significant changes, with fluctuations staying relatively small. Our predictions consistently vary between the global variance and the global L1-mean norm, showing a pattern similar to the predicted mean. The experiments also show that the method maintains its effectiveness as the model weight length increases, showcasing high scalability and validating the feasibility of our introduced EWMA approach as a viable alternative to traditional sorting methods. We also conducted

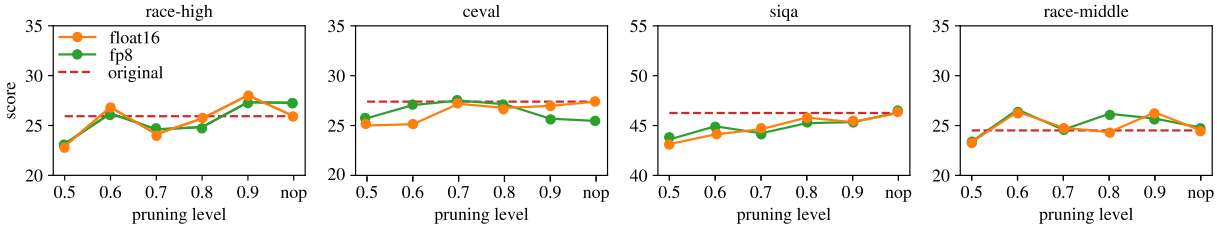

Figure 5: An evaluation using OpenCompass on the wikitext2 dataset with LLaMA2-7B at various sparsity levels shows that FP8 operators have a negligible impact on inference accuracy compared to FP16. Similar results were observed with Pythia-2.8B.

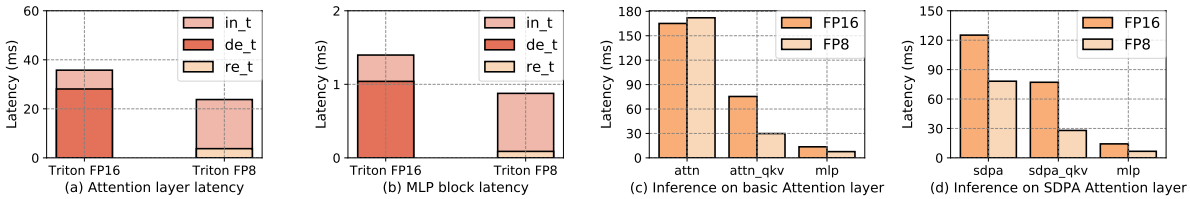

Figure 6: (a, b) In well-optimized frameworks, dequantization constitutes the majority of the time consumed in processing a linear layer. where in_t, de_t, and re_t represent inference time, dequantization time, and requantization time respectively. (c, d) FP8 demonstrates significant potential for acceleration.

the same experiments on the Pythia-2.8B model, achieving equally strong performance and further validating the generalizability of HWPQ.

5.3 Evaluation of FP8 transformers

Accuracy: Minimal impact when substituting linear layers with FP8 precision. In Figure 5, we present a comparative analysis of the evaluation scores for the LLaMA2-7B model, utilizing FP8 linear layers at varying sparsity levels, against the scores achieved by the original FP16 model. Our findings indicate that the evaluation scores between the FP8 and FP16 precision models are generally comparable. This similarity in performance underscores the potential for inference acceleration through the adoption of FP8 precision in linear layers. Our approach demonstrates not only consistent accuracy with LLaMA2 but also exhibits strong performance robustness on Pythia, particularly for certain datasets.

Efficiency: The adoption of FP8 operators significantly reduces latency in linear blocks, with requantization introducing only minimal overhead. The advantages of employing FP8 over FP16 for linear layer computations are clearly demonstrated. On Tensor Cores, data can be processed in FP8 and accumulated in FP16, producing a result tensor in FP16 format. Although requantizing this result tensor back to FP8 for the next layer may introduce slight precision loss and additional overhead, our experiments reveal that even basic truncation quantization to FP8 maintains acceptable accuracy levels. The requantization overhead constitutes merely 10% of the total processing time. This is a notable improvement over traditional methods, where dequantization can consume over 50% of the total time, as illustrated in Figure 6 (a) and (b).

Figure 6 compares cuBLAS FP8 and FP16 latency in PyTorch. In the original attention network (Vaswani, 2017), the query, key, and value computations achieve a 2.53× speedup, while the MLP block achieves a 1.77× speedup. Similarly, in scaled dot-product attention (SDPA), we achieve 2.76× and 2.13× speedups, respectively. These results highlight the efficiency gains achieved with FP8, underscoring its potential for accelerating deep learning computations while maintaining accuracy.

6 Conclusion

In this paper, we propose Hessian-free Weight Pruning-Quantization method, a hardware-friendly approach for low-bit weight-only quantization of LLMs. The core innovation of our study is the development of a novel Hessian-free LLM pruning and quantization method, which significantly reduces time complexity from $O(n^3)$ to $O(n)$ compared to mainstream algorithms. This theoretical breakthrough ensures that our method consistently outperforms existing approaches in terms of both computational efficiency and scalability. Built on a rigorous mathematical foundation, HWPQ demonstrates exceptional effectiveness and relevance, particularly as the scale of future LLMs continues to expand. By significantly reducing computational resource demands and energy consumption, our method contributes to a more sustainable future for high-performance computing (HPC).

7 Limitations

Although this study has achieved promising results and made significant contributions to the field, several limitations of our approach need to be further addressed. Firstly, the performance of the method may be constrained by various external factors, including but not limited to hardware configurations, software dependencies, and runtime environmental conditions, which could significantly impact the stability and efficiency of the method in practical applications. Secondly, while our approach has optimized memory usage compared to existing methods, the improvement is limited. Taking the LLaMA2-7B model as an example, our method requires approximately 20 GB of memory, which is slightly lower than the 22.79 GB required by Wanda and the 30.82 GB required by SparseGPT. However, further optimizing memory consumption to support the deployment of larger models on resource-constrained devices remains a key focus for our future research. Additionally, although FP8 operators in inference and optimizations for 2:4 sparse tensors theoretically offer significant acceleration potential, these aspects were not the core focus of this study. As a result, we have not fully exploited the acceleration capabilities of the new architecture, leaving room for potential technological breakthroughs in future research.

This study addresses critical challenges in large language model (LLM) compression, aiming to facilitate broader adoption and practical implementation of LLM technologies. In light of growing concerns regarding ethical implications associated with LLMs, particularly the potential presence of latent biases embedded within these models, we have conducted comprehensive investigations to ensure the integrity of our proposed methodology. Our findings demonstrate that the developed compression approach not only maintains model performance but also adheres to ethical standards by preventing the amplification of existing biases or introduction of new discriminatory patterns.

References

Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, 32.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, Pritish Yuvraj, Rajarshi Das, Andrew McCallum, Maria Chang, Achille Fokoue-Nkoutche, Pavan Kapanipathi, Nicholas Mattei, et al. 2018. A systematic classification of knowledge, reasoning, and context within the arc dataset. *arXiv preprint arXiv:1806.00358*.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass.

Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Elias Frantar and Dan Alistarh. 2022. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488.

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

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

Yonggan Fu, Haichuan Yang, Jiayi Yuan, Meng Li, Cheng Wan, Raghuraman Krishnamoorthi, Vikas Chandra, and Yingyan Lin. 2022. Depthshrinker: a new compression paradigm towards boosting real-hardware efficiency of compact neural networks. In *International Conference on Machine Learning*, pages 6849–6862. PMLR.

Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks.(2019). *arXiv preprint cs.LG/1902.09574*.

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. 2024. A framework for few-shot language model evaluation.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Simla Burcu Harma, Ayan Chakraborty, Elizaveta Kostenok, Danila Mishin, Dongho Ha, Babak Falsafi,

Martin Jaggi, Ming Liu, Yunho Oh, Suvinay Subramanian, et al. 2024. Effective interplay between sparsity and quantization: From theory to practice. *arXiv preprint arXiv:2405.20935*.

Babak Hassibi, David G Stork, and Gregory J Wolff. 1993. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE.

Peng Hu, Xi Peng, Hongyuan Zhu, Mohamed M. Sabry Aly, and Jie Lin. 2021. Opq: Compressing deep neural networks with one-shot pruning-quantization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35(9), pages 7780–7788.

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

Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems*, 35:24101–24116.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2023. Owq: Lessons learned from activation outliers for weight quantization in large language models. *arXiv preprint arXiv:2306.02272*.

Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Losparse: Structured compression of large language models based on low-rank and sparse approximation. In *International Conference on Machine Learning*, pages 20336–20350. PMLR.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *ArXiv*, abs/2306.00978.

Zhiqiang Liu, Yong Dou, Jingfei Jiang, Jinwei Xu, Shijie Li, Yongmei Zhou, and Yingnan Xu. 2017. Throughput-optimized fpga accelerator for deep convolutional neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 10(3):1–23.

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

Kai Lu, Yaohua Wang, Yang Guo, Chun Huang, Sheng Liu, Ruibo Wang, Jianbin Fang, Tao Tang, Zhaoyun Chen, Biwei Liu, et al. 2022. Mt-3000: a heterogeneous multi-zone processor for hpc. *CCF Transactions on High Performance Computing*, 4(2):150–164.

X Ma, G Fang, and X Wang. 2023a. On the structural pruning of large language models. *NeurIPS, Llm-pruner*.

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

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR.

Vern Paxson, Mark Allman, Jerry Chu, and Matt Sargent. 2011. Rfc6298: Computing tcp's retransmission timer. Technical report.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389.

Michael Santacroce, Zixin Wen, Yelong Shen, and Yuanzhi Li. 2023. What matters in the structured pruning of generative language models? *arXiv preprint arXiv:2302.03773*.

Shashata Sawmya, Linghao Kong, Ilia Markov, Dan Alistarh, and Nir Shavit. 2024. Sparse expansion and neuronal disentanglement. *arXiv preprint arXiv:2405.15756*.

Clemens JS Schaefer, Pooria Taheri, Mark Horeni, and Siddharth Joshi. 2023. The hardware impact of quantization and pruning for weights in spiking neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(5):1789–1793.

Sidak Pal Singh and Dan Alistarh. 2020. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109.

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*.

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*.

Minjin Tang, Mei Wen, Yasong Cao, Junzhong Shen, Jianchao Yang, Jiawei Fei, Yang Guo, and Sheng Liu. 2022. Mentha: Enabling sparse-packing computation on systolic arrays. In *Proceedings of the 51st International Conference on Parallel Processing*, pages 1–11.

Mart van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. 2020. Bayesian bits: Unifying quantization and pruning. In *Advances in Neural Information Processing Systems*, volume 33, pages 5741–5752. Curran Associates, Inc.

Tycho F. A. van der Ouderaa, Markus Nagel, Mart Van Baalen, and Tijmen Blankevoort. 2024. The LLM surgeon. In *The Twelfth International Conference on Learning Representations*.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. 2019. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pages 7543–7552. PMLR.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

A Other experimental setups

Platforms. We carried out our experiments on RTX 4090 GPUs. Given that the Tensor Cores of the RTX 4090 support FP8 computations, our goal is to demonstrate the computational benefits of leveraging the FP8 format. The specific experimental environment includes 2× Intel(R) Xeon(R) Platinum 8358 CPUs @ 2.60GHz, 8× RTX 4090 GPUs with 24GB each, GCC 7.5.0, NVIDIA CUDA release 12.1, and Python 3.11.5 with Anaconda 23.9.0. We utilized PyTorch version 2.3.0.dev20240220+cu121, incorporating float8 support to take advantage of FP8 (E5M2) computing via Tensor Cores on the RTX 4090.

Target Scenarios. Our focus is on AI-driven personal computers (PCs), as LLMs demand substantial computational resources even during inference. The RTX series, being a consumer-level GPU, is a common accelerator for PCs, and we hope our work will enhance the performance of compressed LLM inference on these platforms. All of our experiments were conducted on RTXs with the Ada Architecture, utilizing Tensor Cores that support FP8 GEMM. Given the generic nature of our method, which compresses the weights of the model without altering its computational pattern during inference, it can be applied to other scenarios (such as embedded devices) as long as the hardware supports FP8 or sparse computations.

B Correspondence between la and sparsity

In Algorithm 1, it was mentioned that different la values are set according to sparsity. The corresponding relationship is shown in the following table 3.

Pruning Ratio	90%	80%	70%	60%	50%
la	-1.5	-0.9	-0.2	0.2	0.5

Table 3: Adjust la according to sparsity.

C More experimental results

Pruning Ratio	Method	Latency(s)↓	WikiText2↓	ARC_c	ARC_e	WinoGrande	PIQA	HellaSwag	OpenbookQA	Average↑
0%	Pythia-2.8B	_	12.69	32.76	59.01	58.17	74.10	59.41	35.00	53.07
10%	HWPQ	4.23	13.21	32.08	57.66	59.12	73.23	58.42	35.8	52.71
20%	HWPQ	4.33	13.83	31.23	56.69	57.77	72.63	57.49	34.20	51.66
30%	HWPQ	4.10	15.03	30.89	54.76	58.88	71.87	56.28	35.2	51.31
40%	HWPQ	4.57	17.45	30.2	53.11	58.88	71.6	54.37	32.6	50.12
50%	HWPQ	3.99	21.69	29.18	51.94	57.22	70.29	52.14	31.2	48.66
50%(2:4)	HWPQ	3.83	23.3	27.28	46.81	56.12	69.83	49.13	29.2	46.39

Table 4: Zero-shot performance of the pruned Pythia-2.8B models. "pruning ratio" refers to the proportion of parameters removed relative to the original number of parameters. "Latency(s)" indicates represents the time overhead required for overall model pruning (excluding communication time such as loading to GPU). "Average" is calculated among six classification datasets..

Pruning Ratio	Method	Latency(s)↓	WikiText2↓	ARC_c	ARC_e	WinoGrande	PIQA	HellaSwag	OpenbookQA	Average↑
0%	LLaMA2-13B	_	8.04	48.98	76.94	71.74	80.41	79.57	45.40	67.17
50%	SparseGPT	759.13	**9.82**	43.60	69.53	70.88	78.35	75.13	44.00	63.58
	Wanda	146.63	10.03	**46.76**	72.85	**71.03**	77.71	76.12	**45.60**	**65.01**
	HWPQ (ours)	**16.60**	10.27	45.73	**73.74**	69.38	**78.43**	**76.29**	42.60	64.36
50%(2:4)	SparseGPT	912.81	13.27	38.65	66.62	68.67	73.83	64.54	41.00	58.88
	Wanda	190.02	15.61	37.71	65.49	66.77	75.41	62.65	39.00	57.83
	HWPQ (ours)	**16.39**	**9.42**	**42.30**	**75.72**	**70.40**	**79.38**	**77.28**	**45.20**	**65.04**
0%	LLaMA3.1-8B	_	7.93	53.50	81.10	73.56	81.23	78.90	44.80	68.84
50%	SparseGPT	558.57	12.54	43.26	67.34	70.09	76.82	68.90	40.60	61.16
	Wanda	99.98	11.26	**45.73**	**69.61**	69.77	76.88	**71.39**	43.20	**62.76**
	HWPQ (ours)	**9.49**	**10.96**	44.70	68.10	**70.24**	**77.25**	70.31	**43.82**	62.40
50%(2:4)	SparseGPT	613.13	17.76	33.96	57.24	63.46	69.42	55.22	33.60	52.15
	Wanda	125.64	29.95	29.95	52.15	59.27	67.85	48.69	31.40	48.21
	HWPQ (ours)	**9.28**	**15.02**	**35.27**	**59.19**	**65.84**	**73.17**	**63.10**	**35.02**	**55.26**

Table 5: Zero-shot performance of the pruned LLaMA2-13B and LLaMA3.1-8B. "Pruning Ratio" refers to the proportion of parameters removed relative to the original number of parameters. "Latency(s)" indicates represents the time overhead required for overall model pruning (excluding communication time such as loading to GPU). "Average" is calculated among six classification datasets. Bold indicates the best performance at the same compression rate.