SwiftPrune: Hessian-Free Weight Pruning for Large Language Models

Yuhan Kang[1], Yang Shi[1]*, Mei Wen[1], Jun He[1],
Jianchao Yang[1], Zeyu Xue[1], Jing Feng[1], Xinwang Liu [1],
[1]National University of Defense Technology,
Correspondence: kangyuhan, shiyang14, meiwen, hejun19, yangjianchao16, xuezeyu18,fengjing22,xinwangliu@nudt.edu.cn

Abstract

Post-training pruning, as one of the key techniques for compressing large language models (LLMs), plays a vital role in lightweight model deployment and model sparsity. However, current mainstream pruning methods dependent on the Hessian matrix face significant limitations in both pruning speed and practical effectiveness due to the computationally intensive nature of second-order derivative calculations. This paper presents SwiftPrune, a novel Hessian-free weight pruning method that achieves hardware-efficient model compression through two key innovations: 1) SwiftPrune 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. 2) we employ the Exponentially Weighted Moving Average (EWMA) technique to bypass weight sorting, enabling the selection of weights that contribute most to LLMs accuracy and further reducing time complexity. Our approach is extended to support structured sparsity pruning, facilitating efficient execution on modern hardware accelerators. We validate the SwiftPrune on three LLMs (namely LLaMA2, LLaMA3, and Pythia), demonstrating that it significantly enhances compression performance. The experimental findings reveal that SwiftPrune completes the pruning process within seconds, achieving an average speedup of 12.29× (up to 56.02×) over existing SOTA approaches.

1 Introduction

In recent years, the capabilities of LLMs have experienced explosive growth. However, this advancement comes at the cost of exponential expansion in model scale, resulting in significant financial and energy expenditures (Zhao et al., 2023). Consequently, there has been growing effort to mitigate these costs through model compression. (Frantar et al., 2022; Lin et al., 2024; Frantar and Alistarh, 2023; Ma et al., 2023b; Sun et al., 2024; Dong et al., 2024). Among these, pruning has emerged as one of the most widely adopted techniques, with its fundamental principle involving the elimination of redundant parameters by selectively zeroing out network weights.

Contemporary pruning methods for large language models primarily eliminate retraining requirements through Hessian-based loss analysis (Frantar and Alistarh, 2022; Fang et al., 2023; Frantar and Alistarh, 2023; Sawmya et al., 2024; Shao et al., 2024). While mathematically elegant, these methods face persistent implementation challenges due to slow pruning speeds. Specifically, computing second-order derivatives across all network weights creates a Hessian matrix whose dimensionality scales quadratically with parameter count, leading to intractable computational complexity. This limitation becomes critical in emerging real-time pruning scenarios such as training sparse models from scratch (Evci et al., 2020), finding the optimal sparsity (Jin et al., 2022) and other scenarios requiring frequent pruning operations(Shen et al., 2022; Kwon et al., 2022a; Fu et al., 2024; Le et al., 2025). With existing methods requiring hundreds of seconds per pruning iteration (see Table 1), conventional approaches fail to meet real-time operational demands, making the development of efficient pruning algorithms imperative for practical deployment.

Furthermore, the emergence of advanced GPU architectures underscores the demand for structured hardware-aware pruning methods that achieve genuine acceleration while maintaining computational efficiency (Liu et al., 2017; Lu et al., 2022; Tang et al., 2022; Xia et al., 2024), thereby highlighting the importance of pruning approaches compatible with structured sparse formats.

In this study, we propose SwiftPrune, a novel pruning method designed to circumvent the high

*Corresponding author

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 12868–12879
November 4-9, 2025

Method	LLaMA2		LLaMA3.1	Pythia
	7B	13B	8B	2.8B
SparseGPT	410.10	912.81	196.38	196.38
Wanda	114.26	190.02	125.64	48.23
Pruner-Zero	143.45	165.05	129.14	59.12
SwiftPrune	**7.73**	**16.39**	**9.28**	**3.83**

Table 1: The time consumption (seconds) of mainstream methods.

computational complexity associated with Hessian matrix and its inverse calculations by developing an alternative algorithm. First, our observations indicate that identifying weights with minimal loss contribution depends more on their relative importance than on absolute values. To leverage this, SwiftPrune replaces Hessian matrix computations by constructing a numerically preserved sequence as contribution-oriented weight metrics, derived through a series of loss values. Secondly, 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 structured sparsity pruning.

We conduct comprehensive evaluations of SwiftPrune across three prominent open-source LLMs families: Pythia (Biderman et al., 2023), LLaMA2 (Touvron et al., 2023), and LLaMA3 (Dubey et al., 2024). Compared to previous state-of-the-art methods for large language model pruning (Frantar and Alistarh, 2023; Sun et al., 2024), our SwiftPrune framework achieves the pruning process within seconds, delivering an average 12.29× speedup (with peak acceleration reaching 56.02×) while maintaining comparable accuracy retention across standard benchmarks. Experimental results demonstrate that SwiftPrune can finish pruning tasks more rapidly without requiring any retraining or weight updates, thereby addressing application scenarios that necessitate frequent pruning.

2 Background

Post-training pruning has emerged as a prevalent model compression technique, originating from quantization research (Banner et al., 2019; Zhao et al., 2019; Nagel et al., 2020) and later extended to LLMs pruning (Sanh et al., 2020; Kwon et al., 2022b; Fu et al., 2022; Sun et al., 2023). In neural network optimization, the primary mechanism for minimizing the target loss function involves iterative adjustment of network weights through first-order gradient computation. However, post-training pruning methods operate under a distinct paradigm: These approaches are typically applied to models that have already converged to a local (or potentially global) minimum through standard training procedures. In such optimized states, the first-order derivatives of weights with respect to the loss function asymptotically approach zero(i.e, $\frac{\partial E}{\partial \Delta w} \approx 0$ in equation 1). This mathematical condition fundamentally shifts the optimization focus to second-order sensitivity analysis.

To formalize this concept, we employ a Taylor expansion of the loss function E around the trained network parameters. The expansion reveals:

$$\Delta E = \frac{\partial E}{\partial \Delta w}\Delta w + \frac{1}{2}\Delta w^{\top}\frac{\partial^2 E}{\partial w_i \partial w_j}\Delta w + O(\Delta w^3) \quad (1)$$

Where higher-order terms become non-negligible precisely when first-order derivatives vanish, necessitating explicit consideration of second-order derivatives for effective post-training pruning.

From this, we can infer that if a weight has a significant second-order derivative with respect to the target function, it indicates that the convergence of the weight is not yet stable. The impact of the weight change δw_i on the loss function is reflected by the second-order derivative $\frac{\partial^2 E}{\partial w_i^2}\Delta w_i$. Unfortunately, to compute the second-order derivatives of weights, we need to construct the Hessian matrix $H = \left[\frac{\partial^2 E}{\partial w_i \partial w_j}\right]$, which costs $O(N^3)$ in time complexity. Here, we use N to denote the total number of weights in the model.

To characterize the differences in outputs obtained from the compressed model and the original model under the same input, we select $E = \sum_{i=1}^{d_{\text{row}}} \|w_i x - \widehat{w}_i x\|_2$ as the loss function, where $\widehat{w}_i$ is the i_{th} weight, and d_{row} is the dimension of a row in a module's weights. Since weights from different rows act on the same input, resulting in elements in the same column in the output, we ***assume*** that for any linear layer, weights across different rows are independent of each other. Specifically, in linear layers, every row in W never multiplies with another row, so there are no cross terms in loss functions, meaning they can be optimized independently. In this scenario, $H = 2XX^{\top}$.

Building upon this theoretical foundation, 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.

3 The SwiftPrune 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 evaluated 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:

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

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 pruning 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 $O(n^3) + O((\frac{n}{m})^3) = O(n^3)$, where n represents the number of weights in a row.

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 particular, in LLMs, the dimensionality parameter n (e.g. 4096 in LLaMA2-7B) is large enough to ensure that the quadratic term S dominates over x_q^2 by orders of magnitude ($S >> x_q^2$). This significant scale disparity allows us to employ the approximation $S - x_q^2 \approx S$ with negligible error, leading to:

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

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 $(1 - x_q^2/S)$

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

Table 2: The method for tensor state update. Parameter la can be tuned for different level of sparsity.

Param	Pruning Ratio (%)				
	90	80	70	60	50
la	−1.5	−0.9	−0.2	0.2	0.5

Table 3: Parameter Adjustment under Different Pruning Ratios

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 $O(n\log n)$.

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

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.

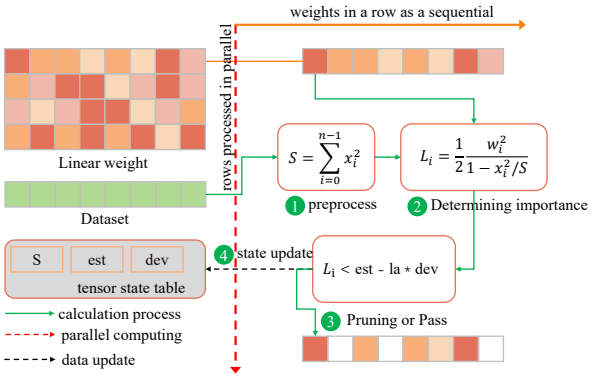

Figure 1: Our design of novel pruning method, using EWMA criteria.

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

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

1: Let $S=\sum_{i=0}^{n-1}(x_i^2), dev=0$
2: **Parallel calculation for each row**
3: **for** $i=0,1,...,n-1$ **do**
4: $\quad L_i=\frac{1}{2}\frac{w_i^2}{1-x_i^2/S}$
5: $\quad$ **if** $i==0$ **then**
6: $\quad\quad est=L_0$
7: $\quad$ **end if**
8: $\quad$ **if** $L_i<\text{est}-\text{la}\times\text{dev}$ **then**
9: $\quad\quad S=S-w_i^2$
10: $\quad\quad w_i=0$ //Pruning
11: $\quad$ **else**
12: $\quad\quad$ **pass**
13: $\quad$ **end if**
14: $\quad est=(1-\alpha)est+\alpha\,L_i$
15: $\quad dev=(1-\beta)dev+\beta\,|est-L_i|$
16: $\quad c_i=w_i$
17: **end for**
18: **return** $C_{nrow\times ncol}$

First, after calculating S as outlined in Step 1 of Figure 1 (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 1 (Algorithm 1, line 4), we sequentially compute a series of L_i values. If L_i satisfies the condition $L<\text{est}-\text{la}\times\text{dev}$ (Algorithm 1, line 8. The corresponding relationship between parameter la and sparsity is shown in Table 3), we consider its contribution to the loss function to be minimal and prune it; otherwise, we get the original weights, as shown in Step 3 of Figure 1 (Algorithm 1, lines 9 and 12).

Next, we update the tensor state according to the procedure outlined in Table 2, as illustrated in Step 4 of Figure 1 (Algorithm 1, lines 10, 14 and 16), until all weights in the row are compressed.

LLaMA Layer	Dense	2:4	Speedup
attn	165.09	110.80	1.49×
attn_qkv	75.40	51.64	1.46×
mlp	13.58	9.24	1.47×

Table 4: Comparison of inference latency(ms) between using original weights and 2:4 sparse weights for Llama2-7B on an RTX 4090 GPU

Throughout this process, the overall time complexity is reduced to $O(n)$. This indicates that we can evaluate the contribution of each weight to the loss function and prune the model into a sparse one within linear time.

3.3 Structured Sparse Support

In practical deployment scenarios, weight sparsity in large language models serves as a critical determinant for enhancing inference efficiency (Tang et al., 2022; Liu et al., 2023). To fully leverage the sparse computation capabilities of modern hardware accelerators, we extend SwiftPrune with structured sparsity support. Taking the widely adopted 2:4 fine-grained structured sparsity pattern as a representative example — a hardware-native sparse specification requiring exactly two non-zero values within every contiguous four-weight block — this design achieves deep integration with the sparse tensor computation units in NVIDIA Ampere architecture's Tensor Cores. Through instruction-level sparse format optimization, it completely eliminates format conversion overhead inherent in conventional sparsification approaches.

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 at $O(n)$ while achieving weight structured sparsity without introducing additional overhead. Compared to unstructured sparsity baselines, our method achieves 1.48× mean speedup in end-to-end inference latency (see Table 4). The regularity of the sparse pattern also reduces DRAM access conflicts by 41%, as validated through Nsight Compute memory trace analysis.

Notably, our mtehod naturally extends to 4:8 and coarser-grained structured sparsity configurations while maintaining hardware compatibility. This adaptability demonstrates our method's scalability across varying sparsity ratios without modifying the core acceleration mechanism.

4 Experiments

4.1 Experimental setup

Models. We conduct comprehensive evaluations of SwiftPrune across three prominent open-source LLMs families: Pythia, LLaMA2, and LLaMA3. As a GPT-NeoX variant specialized for interpretability analysis in autoregressive transformers, Pythia provides granular architectural insights through its controlled design. The LLaMA series represents cutting-edge pre-trained models, with LLaMA3 introducing enhanced multilingual tokenization and dynamic sparse attention mechanisms in its latest iteration. This benchmark suite spans 7B to 70B parameter scales, covering interpretability-oriented frameworks, production-optimized architectures, and next-generation multilingual models, thereby systematically validating our method's robustness across evolving transformer paradigms.

Datasets. We evaluate pruning 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).

Platforms. Our experimental platform configuration consists of 2× Intel(R) Xeon(R) Platinum 8358 CPUs @ 2.60GHz and 8× RTX 4090 GPUs (24GB VRAM each). The software stack includes GCC 7.5.0, NVIDIA CUDA 12.1, and Python 3.11.5 (Anaconda 23.9.0). For structured sparsity implementation, we utilize PyTorch 2.3.0.dev20240220+cu121 with custom kernel extensions that leverage the native 2:4 sparse tensor core operations on the RTX 4090 GPUs, enabled via the cuSPARSELt library. This implementation directly accesses the hardware's structured sparsity acceleration units, where the 2:4 compressed sparse blocks are processed through dedicated warp-level MMA (Matrix Multiply-Accumulate) instructions (SM_AMPERE_SPARSE_MMA feature).

Pruning Ratio	Method	Latency(s)↓	WikiText2↓	ARC_c	ARC_e	WG	PIQA	HS	OQ	Avg↑
Dense	**LLaMA2-7B**	_	9.36	43.51	71.54	70.48	78.94	76.13	44.00	64.10
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
	SwiftPrune (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
	SwiftPrune (ours)	**7.73**	18.21	**32.42**	**56.48**	**64.01**	**71.00**	**61.72**	34.60	**53.37**
Dense	**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**
	SwiftPrune (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
	SwiftPrune (ours)	**16.39**	**9.42**	**42.30**	**75.72**	**70.40**	**79.38**	**77.28**	**45.20**	**65.04**
Dense	**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**
	SwiftPrune (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
	SwiftPrune (ours)	**9.28**	**15.02**	**35.27**	**59.19**	**65.84**	**73.17**	**63.10**	**35.02**	**55.26**
Dense	**Pythia-2.8B**	_	12.69	32.76	59.01	58.17	74.10	59.41	35.00	53.07
50%	SparseGPT	185.17	22.53	**29.44**	51.58	56.51	69.31	50.22	30.80	47.97
	Wanda	39.77	23.30	28.16	49.07	56.04	68.93	51.01	30.80	47.33
	SwiftPrune (ours)	**3.99**	**21.69**	29.18	**51.94**	**57.22**	**70.29**	**52.14**	**31.20**	**48.66**
50%(2:4)	SparseGPT	196.38	27.12	24.83	**46.89**	54.06	65.61	40.88	28.20	43.41
	Wanda	48.23	30.69	24.15	38.89	53.75	61.43	36.81	28.40	40.57
	SwiftPrune (ours)	**3.83**	**23.30**	**27.28**	46.81	**56.12**	**69.83**	**49.13**	**29.20**	**46.39**

Table 5: Zero-shot performance of the pruned LLaMA2-7B, LLaMA2-13B, LLaMA3.1-8B and Pythia-2.8B. "Latency(s)" indicates represents the time overhead required for overall model pruning (excluding communication time such as loading to GPU). The 'Avg' denotes the average value calculated across six classification datasets (HS, WG, and OQ represent HellaSwag, WinoGrande, and OpenbookQA respectively). Bold formatting indicates the best performance under equivalent compression ratios. However, note that for Latency(s), it represents the best performance excluding the cost associated with magnitude. The magnitude pruning method is omitted for LLaMA2-13B, LLaMA3.1-8B, and Pythia-2.8B because it causes significant accuracy degradation in these models.

4.2 Evaluation of SwiftPrune Algorithm

Efficiency: The SwiftPrune algorithm provides a significant speedup. The performance gains derive primarily from algorithmic innovations. By eliminating computationally intensive Hessian matrix calculations, our $O(n)$ algorithm achieves rapid acceleration in LLMs pruning tasks without requiring retraining or weight updates (Table 6). This methodology not only enables efficient assessment of weight significance but also maintains near-constant time complexity — a critical advantage that prevents substantial increases in computational overhead as model dimensions expand.

Our experiments systematically demonstrate that the proposed method achieves an average speedup of 43.75× and 12.29× compared to state-of-the-art pruning approaches like SparseGPT and Wanda respectively (detailed in Table 5). This substantial acceleration effectively addresses the temporal overhead inherent in scenarios requiring iterative pruning applications, particularly those involving adaptive sparsity mechanisms and dynamic input pruning techniques.

Accuracy: Zero-shot performance comparison with baselines. We conducted comprehensive fine-grained pruning experiments on the LLaMA2-7B model and rigorously evaluated its average zero-shot learning accuracy across six tasks under three pruning configurations (including 50% sparsity with 2:4 structured pruning) using the lm-evaluation-harness framework.

As shown in Table 5, the experimental results demonstrate that when reaching a 50% pruning rate, SwiftPrune maintains an average performance decline within 2 percentage points compared to the original dense model. Systematic comparative analysis reveals that our method achieves significant acceleration while preserving negligible accuracy loss (average difference < 1%), outperforming existing approaches like Wanda and SparseGPT that

Method	Weight Update	Calibration Data	Pruning Metric S_{ij}	Complexity
Magnitude	NO	NO	$\lvert W_{ij}\rvert$	$O(1)$
SparseGPT	YES	YES	$\left[\lvert W\rvert^2 / \mathrm{diag}\left[(XX^T + \lambda I)^{-1}\right]\right]_{ij}$	$O(d^3_{\text{hidden}})$
Wanda	NO	YES	$\lvert W_{ij}\rvert \cdot \lVert X_j\rVert_2$	$O(d^2_{\text{hidden}})$
SwiftPrune	NO	YES	$\lvert W_{ij}\rvert \cdot n$	$O(d_{\text{hidden}})$

Table 6: Taxonomy of Pruning Methodologies: Algorithmic Properties and Computational Complexity

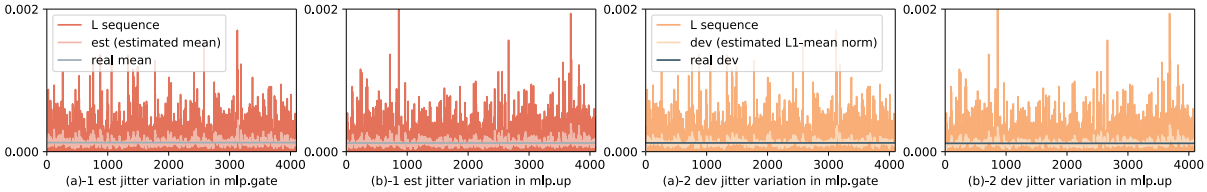

Figure 2: Statistical magnitude detection of L with EWMA method in LLaMA2-7B MLP blocks. x axis presents the sequence number of each weight, and y axis presents the numerical values. Ideal algorithms should show est approaches real mean and dev approaches real dev.

rely on computationally intensive Hessian matrix calculations. Notably, in 2:4 structured sparsity scenarios, our method achieves 3.7-12.7% accuracy improvements across multiple benchmarks through innovative fine-grained pruning strategies. These empirical findings validate the innovation of the SwiftPrune framework: It realizes intelligent model compression through algorithm-level optimizations without requiring training data, effectively balancing model performance preservation with substantial computational complexity reduction. This breakthrough provides an efficient solution for practical industrial deployment scenarios where both accuracy and processing speed are critical. Experimental results of SwiftPrune under other pruning ratio will be presented in the appendix.

Reliability: SwiftPrune adapts to weight changes and approaches global expectations. In Figure 2, we demonstrate how our SwiftPrune method consistently and accurately predicts the mean and variations of weights. As the weight sequence lengthens, SwiftPrune 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 it.

The data presented in Figure 2, 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 2, 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 the same experiments on the Pythia-2.8B model, achieving equally strong performance and further validating the generalizability of SwiftPrune.

Fine-tuning. We systematically investigated two distinct fine-tuning strategies: LoRA (Hu et al., 2022) and full-parameter dense fine-tuning (Lv et al., 2023). Experiments were conducted on the WikiText2 training dataset while strictly maintaining the structured/unstructured mask matrices generated during pruning. We validated the compatibility of pruned models with fine-tuning algorithms under two representative sparsity patterns: unstructured 50% sparsity and structured 2:4 sparsity.

As shown in Table 7, the pruned LLaMA3.1-8B model processed by SwiftPrune pruning demonstrated significant improvements in both zero-shot accuracy and perplexity metrics after fine-tuning. Experimental results confirm the strong compatibility between the adopted fine-tuning strategies and pruning methodology, effectively restoring the model's expressive power diminished during weight trimming. This finding provides crucial technical validation for efficient compression and performance preservation in LLMs.

Evaluation	Dense	Fine-tuning	50%	2:4
Zero-Shot	68.84	NO	62.40	55.26
		LoRA	63.81	58.47
		Full	66.02	63.21
Perplexity	7.93	NO	10.96	15.02
		LoRA	9.53	13.21
		Full	8.42	10.32

Table 7: Fine-tuning can recover some of the losses caused by pruning.

5 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 learned from 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).

SparseGPT (Frantar and Alistarh, 2023) tackles the Hessian computation challenge through a grouping-based pruning strategy. This approach applies compensation updates to weights in adjacent columns via Hessian matrix operations while employing unstructured and semi-structured pruning patterns to streamline large language models. Concurrently, Sparse Expansion (Sawmya et al., 2024) enhances inference efficiency by constructing dedicated Hessian matrices for distinct input clusters, enabling specialized pruning of expert weight matrices through the SparseGPT framework. In a notable simplification, Wanda (Sun et al., 2024) demonstrates that preserving only the diagonal elements of the Hessian matrix suffices for effective pruning, significantly reducing computational overhead while maintaining competitive performance.

Simultaneously, to achieve tangible speed improvements in practical applications, there has been a growing 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., 2023c) 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.

6 Conclusion

In this paper, we propose SwiftPrune, a hardware-friendly approach for pruning of LLMs. The core innovation of our study is the development of a novel Hessian-Free LLMs pruning 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, SwiftPrune demonstrates exceptional effectiveness and relevance, particularly as the scale of future LLMs continues to expand. By significantly reducing computational resource demands and energy consumption.

7 Limitations

While this study achieves promising results and makes notable contributions to the field, we acknowledge several limitations requiring further investigation. Although our method optimizes memory usage compared to existing approaches (SwiftPrune's 20.01 GB, Wanda's 22.79 GB and SparseGPT's 30.82 GB for LLaMA2-7B), the improvements remain constrained. Consequently, further optimization of memory consumption to enable deployment of larger models on resource-constrained devices constitutes a critical focus for future research.

This study tackles key challenges in LLMs compression, aiming to promote practical adoption of LLM technologies. We address ethical concerns, particularly latent biases in LLMs, through thorough investigation of our pruning method. Our results show that the approach preserves model performance while preventing bias amplification or the 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.

Stella Biderman, Hailey Schoelkopf, Quentin G. Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. *ArXiv*, abs/2304.01373.

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.

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. 2024. Pruner-zero: evolving symbolic pruning metric from scratch for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pages 2943–2952. PMLR.

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.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. 2024. Lazyllm: Dynamic token pruning for efficient long context llm inference. *Preprint*, arXiv:2407.14057.

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.

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.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Tian Jin, Michael Carbin, Daniel M. Roy, Jonathan Frankle, and Gintare Karolina Dziugaite. 2022. Pruning's effect on generalization through the lens of training and regularization. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

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

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

Qi Le, Enmao Diao, Ziyan Wang, Xinran Wang, Jie Ding, Li Yang, and Ali Anwar. 2025. Probe pruning: Accelerating LLMs through dynamic pruning via model-probing. In *The Thirteenth International Conference on Learning Representations*.

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, 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. *Proceedings of Machine Learning and Systems*, 6:87–100.

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.

Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. 2023. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782*.

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. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023c. 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*.

Hang Shao, Bei Liu, and Yanmin Qian. 2024. One-shot sensitivity-aware mixed sparsity pruning for large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 11296–11300. IEEE.

Maying Shen, Pavlo Molchanov, Hongxu Yin, and Jose M. Alvarez. 2022. When to prune? a policy towards early structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12247–12256.

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.

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

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*.

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 More experimental results

Pruning Ratio	**Method**	**Latency(s)↓**	**WikiText2↓**	**ARC_c**	**ARC_e**	**WG**	**PIQA**	**HS**	**OQ**	**Avg↑**
Dense	LLaMA2-7B	_	9.36	43.51	71.54	70.48	78.94	76.13	44.00	64.10
10%	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**
	SwiftPrune (ours)	**9.55**	9.88	44.02	**71.62**	70.53	76.73	**78.80**	44.31	64.33
20%	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**
	SwiftPrune (ours)	**9.37**	9.67	43.42	70.33	69.37	**78.31**	76.01	44.88	63.72
30%	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**
	SwiftPrune (ours)	**9.12**	10.02	43.91	70.01	69.11	78.04	75.22	45.01	63.55
40%	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
	SwiftPrune (ours)	**8.62**	10.34	42.84	69.01	68.09	78.01	**73.91**	42.69	62.43

Table 8: 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). The 'Avg' denotes the average value calculated across six classification datasets (HS, WG, and OQ represent HellaSwag, WinoGrande, and OpenbookQA respectively). Bold formatting indicates the best performance under equivalent compression ratios.

Pruning Ratio	Method	Latency(s)↓	ARC_c	ARC_e	WinoGrande	PIQA	HellaSwag	OpenbookQA	Average↑
0%	LLaMA2-7B	_	43.51	71.54	70.48	78.94	76.13	44.00	64.10
50%	Pruner-Zero	134.74	38.99	65.45	**67.09**	75.52	68.45	**42.00**	**59.58**
	RIA	134.61	**39.51**	65.36	66.93	**76.44**	**69.18**	39.60	59.50
	SwiftPrune (ours)	**7.85**	38.40	**67.32**	65.27	75.14	67.18	38.90	58.70
50%(2:4)	Pruner-Zero	143.45	29.18	55.72	63.06	70.02	56.63	33.40	51.33
	RIA	147.29	29.27	55.58	62.59	70.40	57.35	33.60	51.46
	SwiftPrune (ours)	**7.73**	**32.42**	**56.48**	**64.01**	**71.00**	**61.72**	**34.60**	**53.37**
50%(4:8)	Wanda	1139.87	34.81	58.42	65.82	73.78	63.54	36.80	55.52
	RIA	140.31	35.67	**60.40**	64.96	74.21	64.27	34.60	55.68
	SwiftPrune (ours)	**13.72**	**36.12**	59.98	**66.74**	**75.02**	**65.43**	**37.01**	**56.71**
0%	LLaMA2-13B	_	48.98	76.94	71.74	80.41	79.57	45.40	67.17
50%	Pruner-Zero	143.28	43.00	71.84	**69.69**	**78.73**	73.53	42.41	63.20
	SwiftPrune (ours)	**16.60**	**45.73**	**73.74**	69.38	78.43	**76.29**	**42.60**	**64.36**
50%(2:4)	Pruner-Zero	165.05	37.03	65.19	66.61	74.32	61.70	37.80	57.10
	SwiftPrune (ours)	**16.39**	**42.30**	**75.72**	**70.40**	**79.38**	**77.28**	**45.20**	**65.04**

Table 9: Zero-shot performance of the pruned LLaMA2-7B and LLaMA2-13B. "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.