

ONE-SHOT SENSITIVITY-AWARE MIXED SPARSITY PRUNING FOR LARGE LANGUAGE MODELS

Hang Shao, Bei Liu, Yanmin Qian[†]

MoE Key Lab of Artificial Intelligence, AI Institute
X-LANCE Lab, Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai, China

ABSTRACT

Various Large Language Models (LLMs) from the Generative Pre-trained Transformer (GPT) family have achieved outstanding performances in a wide range of text generation tasks. However, the enormous model sizes have hindered their practical use in real-world applications due to high inference latency. Therefore, improving the efficiencies of LLMs through quantization, pruning, and other means has been a key issue in LLM studies. In this work, we propose a method based on Hessian sensitivity-aware mixed sparsity pruning to prune LLMs to at least 50% sparsity without the need of any retraining. It allocates sparsity adaptively based on sensitivity, allowing us to reduce pruning-induced error while maintaining the overall sparsity level. The advantages of the proposed method exhibit even more when the sparsity is extremely high. Furthermore, our method is compatible with quantization, enabling further compression of LLMs.

Index Terms— model compression, sparsity pruning, large language models, mixed sparsity

1. INTRODUCTION

Large language models (LLMs) from the GPT family have demonstrated exceptional performances across a wide range of tasks. However, due to their large sizes and high computational costs, model deployment face new challenges upon GPU memory consumption as well as inference latency. Consequently, there’s a prominent need to compress LLM models to a feasible range in order to make use of them in real applications. In the literature, there have been various mainstream model compression techniques available, for pre-trained models in particular, including knowledge distillation [1–4], model quantization [5–9], model sparsity pruning [10–14], and *etc.*

Knowledge distillation focuses on transferring knowledge from a large teacher model to a smaller student model by predicting pseudo-labels or leveraging the knowledge from intermediate hidden layers. This process enables the compact student model to achieve approximate accuracy comparable to that of the teacher model. Model quantization involves replacing high-precision floating-point parameters with lower-precision integer parameters, thereby reducing the model’s storage capacity and inference latency. Quantization methods can be categorized into Post-Training Quantization (PTQ) [15] and Quantization-Aware Training (QAT) [16] approaches. At present, 4-bit quantization [6] and mixed-precision quantization [9] techniques are able to reduce memory cost of each weight element from 16-bit as a float point number to equivalently 3-4 bits.

In addition, Model sparsity pruning method mainly involves removing network elements by individual weights (unstructured pruning) or by entire rows and columns of weight matrices (structured pruning). Pruning can also be applied to various parts of the model, including entire layers [17], heads, intermediate dimensions [18], and blocks of weight matrices [19]. The study of model pruning emerged since the 1980s’ [20–22], and it has demonstrated favorable outcomes across Computer Vision and NLP tasks. However, these methods require extensive retraining of pruned models to restore the performance level of the original models [13]. While this is feasible for smaller-scale models, it becomes prohibitive for LLMs due to the associated computational costs. While there are some one-shot pruning methods that don’t require retraining [22, 23], their effectiveness are limited. Therefore, extending the existing pruning methods to models with 10-100+ billion parameters is still an open and challenging problem.

Recently, SparseGPT [14] further developed the Optimal Brain Surgeon (OBS) [21] approach. It achieved at least 50% pruning ratio on LLMs with over a billion parameters in one shot, minimizing performance loss without requiring any retraining. Based on previous studies in this area, we believe there is still considerable room for further improvements. Firstly, saliency criterion for weight element selection, such as OBS and OBD [21], each captured one aspect of the weight element’s saliency. We show that fusing the two criteria of OBS and OBD yields a better option for selecting the weights to be pruned, as observed in our experiments.

Moreover, SparseGPT assumes a uniform sparsity across all layers. Inspired by the analysis of weight sensitivity in mixed-precision quantization methods [24, 25], we investigated the sensitivity properties of different LLM layers. The magnitude of pruning ratios (sparsity) for each layer are then determined based on their respective sensitivities. Specifically, we utilize second-order information (the Hessian matrix) to assess the sensitivity of each layer.

Furthermore, we extend pruning to individual weight-level, an even finer granularity, for more precise trade-off. This means that each weight matrix is assigned a distinct sparsity level according to its sensitivity. In experiments with multiple cutting edge LLMs, our approach beats SparseGPT in terms of both perplexity and zero-shot downstream task performances.

In summary, our contributions are as follows: (1) We introduce a more comprehensive criterion for selecting the pruned weight elements. (2) We propose a novel sensitivity-aware mixed sparsity pruning strategy based on Hessian information. (3) To the best of our knowledge, this work achieves a new state-of-the-art performance in the field of LLM pruning.

[†] corresponding author

2. METHODOLOGY

This section describes the Improved Saliency Criterion (ISC) and the sensitivity-aware mixed sparsity pruning method. ISC serves the purpose of selecting the optimal weight elements to be pruned, and the sensitivity level facilitates the allocation of the pruning ratios for different layers.

2.1. Mask Selection and Weight Reconstruction Based on OBS

In this sub-section, we review the OBS algorithm [21] adopted in SparseGPT [14], which serves as a part of our saliency criterion. The OBS algorithm firstly prunes some weight elements, then updates the rest for error compensation. Denoting the original weights as W , the objective is to ensure that the optimally pruned weights W^* satisfy equation (1), where X is the input data used for calibration purpose.

$$W^* = \arg \min_{\hat{W}} \|WX - \hat{W}X\|_2^2, \quad (1)$$

In the first step, to-be-pruned weight elements are determined through the saliency criterion, denoted *e.g.*, ε_m for the m -th row. In OBS, ε_m is given by (2), where w_m represents the original weight elements of the m -th row. H^{-1} denotes the inverse matrix of $H = XX^T$ (the Hessian matrix of the objective function). One can select the weight elements associated with the smallest $p\%$ (sparsity level) of ε_m , and set them to 0.

$$\varepsilon_m = \frac{w_m^2}{[H^{-1}]_{mm}} \quad (2)$$

Secondly, one need to update the remaining weights to compensate for the error. Let $\delta W = W^* - W$ be the optimal adjustment of the remaining weight elements. The objective then becomes minimizing the expectation of $\|WX - (W + \delta W)X\|_2^2$. Following Taylor expansion, the update formula is given by (3) for the m -th row of W .

$$\delta W_m = -\frac{w_m}{[H^{-1}]_{mm}} H_{:,m}^{-1} \quad (3)$$

2.2. Improved Saliency Criterion

An alternative saliency criterion s had been proposed in the Optimal Brain Damage (OBD) [20] work, where $s = w_m^2 H_{mm}$. The first step involves computing the second-order derivative of the loss with respect to the weights W , which is the Hessian matrix $H = XX^T$, derived from (1). The weight elements are then sorted according to the saliency levels, and elements with lower saliency are pruned. This approach aimed to achieve the minimal error in the pruning process. However, it only focused on minimizing the direct error and did not take error compensation into account.

Building upon the OBS and OBD approaches, this work considers combining the saliency criteria ε and s , in order to balance the different aspects of previously used methods. We also apply error compensation following the approach in 2.1. We define a more comprehensive saliency criterion – ISC (Improved Saliency Criterion) in (4). This criterion allows for a more precise determination of pruning targets.

$$ISC = w_m^2 (H_{mm} + \frac{1}{[H^{-1}]_{mm}}) \quad (4)$$

2.3. Mixed Sparsity Pruning Based on Hessian Sensitivity Awareness

In Section 2.1 we assumed a constant sparsity ratio p . In this section, we propose a sensitivity-aware pruning strategy.

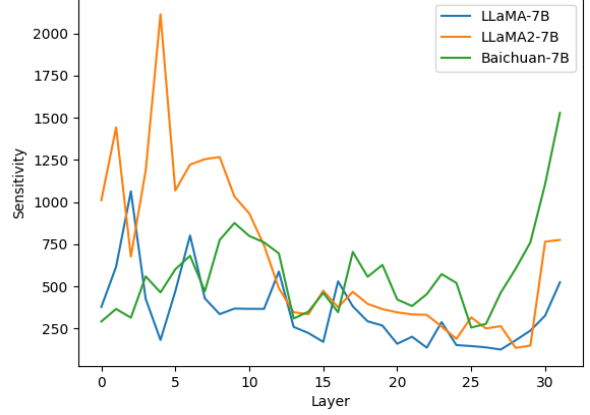


Fig. 1: Sensitivity level of different layers for three LLM models: LLaMA-7B, LLaMA2-7B, and Baichuan-7B.

2.3.1. Calculation of Sensitivity Based on Hessian Average Trace

Sensitivity level is defined as the degree to which perturbations in model weights would affect the output error. Inspired by mixed-precision quantization [25], we extend the sensitivity of model weights in mixed-precision quantization to the field of LLM pruning. The sensitivity level is derived from the trace of the Hessian matrix H . Direct computation of H for LLM is challenging because explicit calculation the Hessian matrix under cross-entropy loss would be time-and-space consuming. Nevertheless, we can address this issue using stochastic linear algebra [26] based methods for fast trace estimation. As shown in (5), I denotes identity matrix, and z is a random vector with *i.i.d.* components, sampled from a standard Gaussian distribution. N denotes the number of sampled data points. The second line in (5) uses the Hutchinson algorithm [27] for efficient approximation, which we follow in the implementation.

$$\begin{aligned} \text{Tr}(H) &= \text{Tr}(HI) = \text{Tr}(HE[z z^T]) = E[\text{Tr}(H z z^T)] \\ &= E[z^T H z] \approx \frac{1}{N} \sum_{i=1}^N z_i^T H z_i \end{aligned} \quad (5)$$

For each weight matrix, we select the average trace of the Hessian as its sensitivity sen , as shown in (6), where $\text{Tr}(H)_{i,j}$ represents the trace of the Hessian matrix for the j -th weight matrix in the i -th layer, and n denotes the number of diagonal elements in the Hessian matrix. $sen_{i,j}$ represents the sensitivity of the corresponding weight matrix.

$$sen_{i,j} = \frac{1}{n} \text{Tr}(H)_{i,j} \quad (6)$$

2.3.2. Rational Mixed Sparsity Assignment Based on Sensitivity

After obtaining the sensitivity level of weight matrices, we can allocate sparsity ratio individually to each one. This is referred to as **weight-level** mixed sparsity allocation. Pruning targets are selected adaptively within each weight matrix. We can also sum the sensitivities of all weight matrices in the same layer to obtain the sensitivity level of that layer. This is referred to as **layer-level** mixed sparsity allocation.

Layer-level sensitivity analysis can be seen in Figure 1. The three different models exhibit some commonalities in that the bottom and

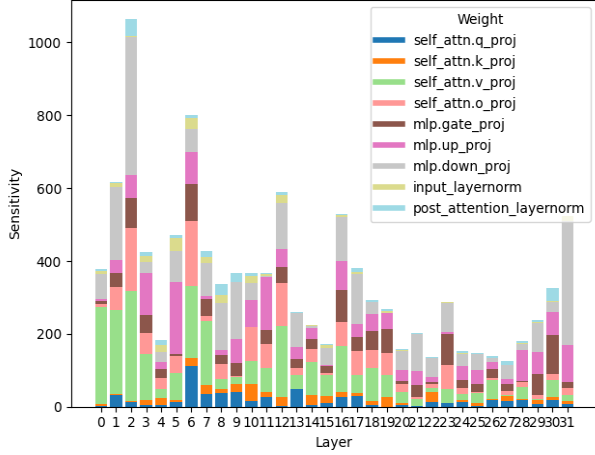


Fig. 2: Sensitivity level of different weight components in LLaMA-7B.

top layers are more likely to show higher sensitivities, while the intermediate layers have lower sensitivities. This may be attributed to the fact that the bottom layers determine the model’s initial encoding of the input embeddings, and their variations can have a significant impact on the intermediate layers, thereby affecting the overall model performance. The top layers are also sensitive since they have a direct influence on the model’s output. Weight-level sensitivity analysis can be seen in Figure 2. The value matrices (plot in green color) in self-attention modules and the down projection weights (plot in gray color) in MLP modules are relatively more sensitive.

We use Algorithm 1 to determine the sparsity of each unit (layer or weight) based on their sensitivities. We rank the sensitivity of the weight matrices, then assign a corresponding sparsity to each one, ensuring that the overall sparsity of this mixed sparsity approach equals a pre-defined overall sparsity level p . For example, the typical value for sparsity interval’s half-width α is 0.1. When the overall sparsity p is 0.5, L and R would be 0.4 and 0.6, respectively. This means that the sparsity of each weight falls within the interval $[0.4, 0.6]$, and the total sparsity of all weights adds up to 0.5.

Algorithm 1 Sparsity Assignment Based on Sensitivity

Input: sen , The sensitivity of each weight within each layer.

Input: Overall sparsity p , sparsity interval’s half-width α .

Output: spa , The sparsity of each weight within each layer.

```

1: function SPARSITY-ASSIGNMENT( $sen, p, \alpha$ )
2:   total_weight  $\leftarrow []$ 
3:   Append(all elements of  $sen$ ) to total_weight
4:   total_weight  $\leftarrow$  sorted(total_weight)
5:   for  $i \leftarrow 0$  to length( $sen$ )-1 do
6:     for  $j \leftarrow 0$  to length( $sen[i]$ )-1 do
7:       id  $\leftarrow$  rank(total_weight,  $sen[i,j]$ ) # base 0
8:        $L \leftarrow 1 - p - \alpha$ 
9:        $R \leftarrow 2 * (1 - p) - L$ 
10:       $spa[i,j] \leftarrow L + id * (R - L) / (\text{len}(\text{total\_weight}) - 1)$ 
11:   return  $spa$ 
```

3. EXPERIMENTS

3.1. Experiment Setup

3.1.1. Model, Dataset and Evaluation

We employ several cutting edge LLM models in the experiments, including LLaMA [28], LLaMA2 [29], and Baichuan [30]. We prioritize the study on 7B/13B models as these are more widely adopted in research and applications. We evaluate the pruned models on standard benchmarks such as raw-WikiText2 [31], PTB [32], and C4 [33] (a subset of the validation sets). As usual, we employ perplexity as the evaluation metric.

To examine the pruned models’ performances on downstream tasks, we conduct zero-shot accuracy tests on the OpenLLM LeaderBoard¹ by utilizing the popular open-source auto-regressive language model evaluation framework EleutherAI-evalharness². We employ a variety of tasks including ARC (Challenge) [34], HellaSwag [35], TruthfulQA (Multiple-Choice2) [36], and PIQA [37] to compare the pruned models with their corresponding original versions. This allows us to assess the performance of the model across a variety of tasks and gain insights into its generalization capabilities. The experimental results of our approach are primarily compared with the relative values of dense and baseline models.

3.1.2. Setup

We implement our method using PyTorch and HuggingFace’s Transformers library. The most computation-heavy step in our experiment was around calculating the trace of the Hessian matrix, which required 8 NVIDIA A100 GPUs for 13B-parameter models. All pruning experiments are one-shot and do not require any fine-tuning or retraining. For calibration data, we utilized 128 2048-token segments. These segments are randomly selected from the first partition of the C4 dataset. We ensure that no task-specific data is seen during the pruning process to guarantee zero-shot conditions. The baselines we compare against mainly consist of the standard magnitude pruning [38] and SparseGPT [14] methods.

3.2. Experiment Results and Analysis

3.2.1. Evaluation on Improved Saliency Criterion

In this experiment, we verify the effectiveness of the proposed saliency criterion. In Table 1, we compare the perplexity for the two LLM models under different sparsity strategies and saliency criteria. The experiment results have validated the effectiveness of ISC as well as the mixed sparsity strategy, as the perplexity of the pruned models were lower in comparison to Uniform sparsity and OBS saliency on all of the examined data sets.

3.2.2. Evaluation on Mixed Sparsity Pruning

We adopt a mixed sparsity approach to prune different base models, as shown in Table 2. It can be seen that our method outperforms SparseGPT in three mainstream 13B LLM models. The experiment results demonstrate that our method can further reduce the performance degradation caused by pruning, bringing it closer to the dense model. Since the overall sparsity is set at 50%, our method and SparseGPT achieve an equal model compression ratio and inference acceleration effect.

¹https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

²<https://github.com/EleutherAI/lm-evaluation-harness>

Table 1: Performance perplexity comparison under the condition of an overall sparsity level of 50%. “Uniform” denotes a uniform sparsity level; “Mixed” denotes the proposed sensitivity-aware pruning strategy.

Model	Sparsity	Saliency	WikiText2↓	PTB↓	C4↓
LLaMA7B	Uniform	OBS	7.79	65.77	10.17
		ISC	7.11	61.42	9.27
	Mixed	OBS	6.77	52.10	8.88
		ISC	6.76	50.12	8.82
Baichuan7B	Uniform	OBS	16.74	43.82	27.43
		ISC	14.77	38.54	24.54
	Mixed	OBS	14.43	38.50	24.13
		ISC	12.29	33.04	20.63

Moreover, we conduct tests on zero-shot downstream NLP tasks. As shown in Table 3, with an overall sparsity level of 50%, the proposed approach outperforms SparseGPT in terms of accuracy. These results validate that our method is effective in reducing the errors introduced by pruning.

Table 2: Under the condition of an overall sparsity rate of 50%, the three base models have different perplexity results under various methods.

Model	Method	WikiText2↓	PTB↓	C4↓
LLaMA-13B	Dense	5.03	32.26	6.80
	Magnitude	20.25	149.43	25.11
	SparseGPT	6.12	43.92	8.13
	Ours	5.68	38.69	7.56
LLaMA2-13B	Dense	4.88	40.99	6.73
	Magnitude	6.77	117.69	9.38
	SparseGPT	5.99	59.03	8.23
	Ours	5.51	53.03	7.57
Baichuan-13B	Dense	5.61	16.49	8.23
	Magnitude	15.64	48.74	22.23
	SparseGPT	7.52	21.46	10.95
	Ours	6.63	19.25	9.71

Table 3: Comparison of the accuracy results of the different methods for zero-shot with an overall sparsity of 50%, where HS stands for HellaSwag and TQ stands for TruthfulQA.

Model	Method	ARC↑	HS↑	TQ↑	PIQA↑	Avg↑
LLaMA-7B	Dense	51.0	77.8	34.3	79.2	60.6
	SparseGPT	39.5	69.2	36.2	76.6	55.4
	Ours	40.2	71.2	38.1	77.5	56.8
LLaMA-13B	Dense	47.6	79.1	39.9	80.1	61.7
	SparseGPT	41.8	73.9	37.5	79.0	58.1
	Ours	43.8	76.5	40.4	79.8	60.1
LLaMA2-7B	Dense	46.3	75.9	38.9	79.1	60.1
	SparseGPT	41.3	71.1	38.3	76.9	56.9
	Ours	44.1	72.8	39.4	77.5	58.5
LLaMA2-13B	Dense	48.9	79.4	36.9	80.5	61.4
	SparseGPT	44.9	74.3	37.9	77.9	58.8
	Ours	47.5	76.9	39.4	79.5	60.8

3.2.3. Evaluation on Very High Sparsity Models

Figure 3 shows that performance degradation expands when the sparsity level increases, which is a common effect for all pruning methods examined (results for LLaMA2-13B model plotted). Nevertheless, the proposed method helps mitigating the degradation even

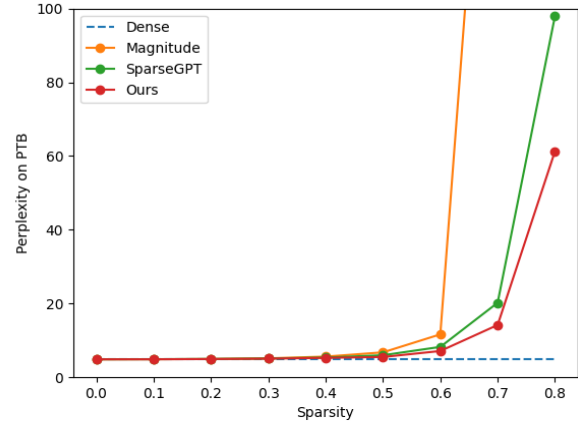


Fig. 3: Perplexity of different methods with varying sparsity levels evaluated on PTB.

more as the sparsity level becomes higher.

3.2.4. Joint Mixed Sparsity Pruning and Quantization

In this experiment, we carry out joint quantization and pruning of the model, to see if their effects can be combined. We follow the approach of SparseGPT, where pruning is applied prior to quantization. As shown in Table 4, 3-bit quantization using GPTQ method yields much heavier loss in comparison to 4-bit quantization, with only minor increase of compression ratio [6]. However, when we combine SparseGPT with 50% sparsity and 4-bit quantization, higher compression ratio and lower performance loss are achieved together. Moreover, as indicated in the last row, the proposed method further improves the performance compared to SparseGPT at the same compression ratio, confirming the effectiveness of the mixed sparsity approach. It offers a new SOTA performance via combining quantization and pruning.

Table 4: The perplexity performance of LLaMA-7B after compression under different methods. Ratio represents the theoretical model compression ratio.

Method	WikiText2↓	PTB↓	C4↓	Ratio↑
Dense	5.63	35.79	7.34	1.0
4bit GPTQ [6]	7.07	34.34	8.19	4.0×
3bit GPTQ	35.25	226.29	17.92	5.3×
SparseGPT 50% + 4bit GPTQ	12.80	88.60	11.37	8.0×
Ours 50% + 4bit GPTQ	10.23	65.49	9.91	8.0×

4. CONCLUSIONS

In this paper, we have proposed a mixed sparsity pruning method based on Hessian sensitivity awareness, which assigns sparsity levels to each weight by discerning their sensitivity through the average trace of the Hessian matrix. Under the same overall sparsity constraint as SparseGPT, our mixed sparsity allocation scheme further reduces the error introduced by pruning. Moreover, our method exhibits even more significant advantages in scenarios with extremely high overall sparsity. Additionally, our approach is compatible with quantization, achieving higher compression ratios while minimizing model performance degradation. Future work will explore the combination of our approach with mixed-precision quantization to achieve further improved results.

5. REFERENCES

- [1] S. Sun *et al.*, “Contrastive distillation on intermediate representations for language model compression,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Jan 2020.
- [2] H. Pan *et al.*, “Meta-kd: A meta knowledge distillation framework for language model compression across domains,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Jan 2021.
- [3] S. Sun *et al.*, “Patient knowledge distillation for bert model compression,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Jan 2019.
- [4] V. Sanh *et al.*, “Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,” *arXiv preprint arXiv:1910.01108*, 2019.
- [5] H. Bai *et al.*, “Binarybert: Pushing the limit of bert quantization,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Jan 2021.
- [6] E. Frantar *et al.*, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” Oct 2022.
- [7] Z. Yao *et al.*, “Zeroquant: Efficient and affordable post-training quantization for large-scale transformers,” Jun 2022.
- [8] G. Xiao *et al.*, “Smoothquant: Accurate and efficient post-training quantization for large language models,” Nov 2022.
- [9] T. Dettmers *et al.*, “Spqr: A sparse-quantized representation for near-lossless llm weight compression,” Jun 2023.
- [10] Z. Wang *et al.*, “Structured pruning of large language models,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Jan 2020.
- [11] E. Kurtic *et al.*, “The optimal bert surgeon: Scalable and accurate second-order pruning for large language models.”
- [12] O. Zafrir *et al.*, “Prune once for all: Sparse pre-trained language models.”
- [13] X. Ma *et al.*, “Llm-pruner: On the structural pruning of large language models.”
- [14] E. Frantar *et al.*, “Sparsegpt: Massive language models can be accurately pruned in one-shot,” Jan 2023.
- [15] Z. Liu *et al.*, “Post-training quantization for vision transformer,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 092–28 103, 2021.
- [16] S. A. Tailor *et al.*, “Degree-quant: Quantization-aware training for graph neural networks,” *arXiv preprint arXiv:2008.05000*, 2020.
- [17] H. Sajjad *et al.*, “Poor man’s bert: Smaller and faster transformer models,” *arXiv preprint arXiv:2004.03844*, vol. 2, no. 2, 2020.
- [18] Z. Wang *et al.*, “Structured pruning of large language models,” *arXiv preprint arXiv:1910.04732*, 2019.
- [19] F. Lagunas *et al.*, “Block pruning for faster transformers,” *arXiv preprint arXiv:2109.04838*, 2021.
- [20] Y. LeCun *et al.*, “Optimal brain damage,” *Neural Information Processing Systems, Neural Information Processing Systems*, Jan 1989.
- [21] B. Hassibi *et al.*, “Second order derivatives for network pruning: Optimal brain surgeon,” *Neural Information Processing Systems, Neural Information Processing Systems*, Nov 1992.
- [22] E. Frantar *et al.*, “Optimal brain compression: A framework for accurate post-training quantization and pruning,” Aug 2022.
- [23] I. Hubara *et al.*, “Accelerated sparse neural training: A provable and efficient method to find n:m transposable masks,” *Neural Information Processing Systems, Neural Information Processing Systems*, Dec 2021.
- [24] Z. Dong *et al.*, “Hawq: Hessian aware quantization of neural networks with mixed-precision,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019.
- [25] Z. Dong *et al.*, “Hawq-v2: Hessian aware trace-weighted quantization of neural networks,” *arXiv: Computer Vision and Pattern Recognition, arXiv: Computer Vision and Pattern Recognition*, Nov 2019.
- [26] M. Mahoney, “Randomized algorithms for matrices and data,” Mar 2012.
- [27] H. Avron *et al.*, “Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix,” *Journal of the ACM*, p. 1–34, Apr 2011.
- [28] H. Touvron *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [29] H. Touvron *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2307.09288*, 2023.
- [30] Baichuan-Inc, “Baichuan-7b,” 2023, accessed on Sep 4th, 2023.
- [31] S. Merity *et al.*, “Pointer sentinel mixture models,” *arXiv: Computation and Language, arXiv: Computation and Language*, Sep 2016.
- [32] M. Marcus *et al.*, “The penn treebank: annotating predicate argument structure,” in *Proceedings of the workshop on Human Language Technology - HLT '94*, Jan 1994.
- [33] C. Raffel *et al.*, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *arXiv: Learning, arXiv: Learning*, Oct 2019.
- [34] M. Boratko *et al.*, “A systematic classification of knowledge, reasoning, and context within the arc dataset,” Jun 2018.
- [35] R. Zellers *et al.*, “Hellaswag: Can a machine really finish your sentence?” *arXiv preprint arXiv:1905.07830*, 2019.
- [36] S. Lin *et al.*, “Truthfulqa: Measuring how models mimic human falsehoods,” *arXiv preprint arXiv:2109.07958*, 2021.
- [37] S. Tata *et al.*, “Piqa: an algebra for querying protein data sets,” in *15th International Conference on Scientific and Statistical Database Management, 2003.*, Dec 2003.
- [38] M. Zhu *et al.*, “To prune, or not to prune: exploring the efficacy of pruning for model compression,” Oct 2017.