

KVPruner: Structural Pruning for Faster and Memory-Efficient Large Language Models

Bo Lv¹, Quan Zhou^{1*}, Xuanang Ding¹, Yan Wang¹, Zeming Ma¹

¹Huazhong University of Science and Technology

{bolv, quanzhou, dingxuanang, milliewang, mzm}@hust.edu.cn

Abstract—The bottleneck associated with the key-value(KV) cache presents a significant challenge during the inference processes of large language models. While depth pruning accelerates inference, it requires extensive recovery training, which can take up to two weeks. On the other hand, width pruning retains much of the performance but offers slight speed gains. To tackle these challenges, we propose KVPruner to improve model efficiency while maintaining performance. Our method uses global perplexity-based analysis to determine the importance ratio for each block and provides multiple strategies to prune non-essential KV channels within blocks. Compared to the original model, KVPruner reduces runtime memory usage by 50% and boosts throughput by over 35%. Additionally, our method requires only two hours of LoRA fine-tuning on small datasets to recover most of the performance.

Index Terms—Structured pruning, Large language models, KV cache optimization, Inference efficiency.

I. INTRODUCTION

Large language models (LLMs) refer to natural language processing (NLP) models with a massive number of parameters [1]–[5], commonly based on the Transformer architecture [5]. These models have also found widespread applications in fields such as speech processing [6]–[8] and computer vision [9]–[11]. In recent years, LLMs have demonstrated remarkable capabilities in handling complex tasks in applications like dialogue systems [12], [13] and knowledge-based question answering [14]–[16], significantly accelerating the development of downstream applications. However, as model sizes continue to grow, the challenges related to inference efficiency have become more pronounced.

Currently, optimization methods for large models include pruning (structured pruning [17]–[19] and unstructured pruning [20], [21]), quantization [22]–[24], and distillation [25], [26]. This work focuses on structured pruning, making it more deployment-friendly and hardware-friendly. In modern structured pruning algorithms for LLMs, LLM-Pruner [17] achieves model size reduction by removing inter-group dependencies in the network. Sheared-LLaMA [19] not only removes structures within groups but also prunes less important blocks to achieve compression.

These methods can employ LoRA [27] to recover performance, but the gains in runtime memory efficiency and inference performance are still minimal. Shortened-LLM [18] aggressively removes entire blocks to speed up inference, but

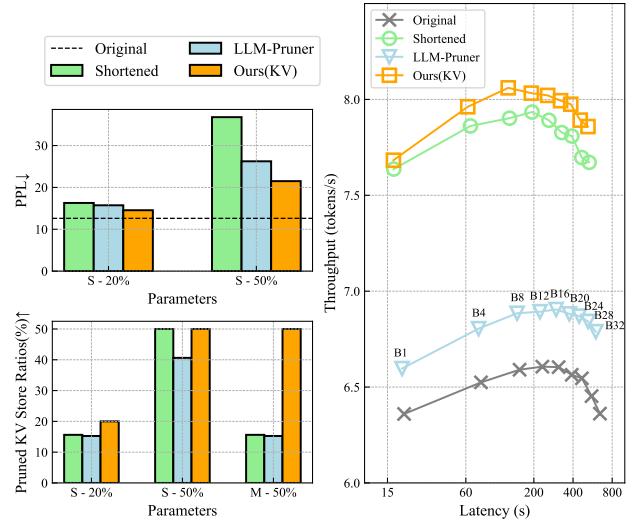


Fig. 1. The inference results of the pruned LLaMA-7B model on an NVIDIA A100 GPU, measuring computation latency and throughput with caching disabled. Left side: The top left compares perplexity(PPL) across different strategies under the same pruning ratio and fine-tuning steps, where our method demonstrates superior performance. The bottom left shows the key-value (KV) cache usage, where our approach achieves more significant KV memory pruning at both strategy-level and model parameter-level pruning ratios. Right side: Under the same model parameter settings, KVPruner achieves faster inference speeds compared to Shortened-LLM [18] and LLM-Pruner [17] pruning method.

it shows that directly removing blocks requires retraining with CPT [18], which can take several weeks to restore the pruned model.

Inference in autoregressive LLMs demands high computational resources and requires a KV cache to store previous tokens for generating new ones, increasing huge overhead on both memory and computation. A typical scenario is in Retrieval-Augmented Generation (RAG) [14]–[16], where large models must handle long prompts and generate extended responses. Recent inference optimization models, such as SGLang [28], point out that current LLM inference bottlenecks arise from the need to cache and recompute KV entries. On the GPU side, methods like FlashAttention [29], [30] reduce KV memory accesses during inference to increase throughput. In large-scale inference scenarios, DistServe [31] also highlights that KV caching significantly impacts LLM

*Corresponding author.

inference throughput.

Therefore, in the context of structured pruning for large models, KV cache in transformer is a crucial metric for evaluating runtime efficiency and throughput. This study introduces KVPruner, which focuses on KV pruning for large models (e.g., LLaMA [32]). Under similar fine-tuning conditions, the pruned model achieves performance comparable to or better than previous structured pruning methods (as shown in Fig. 1). The main contributions of this work are as follows:

- A structured pruning method for KV cache, which significantly reduces runtime memory usage.
- A perplexity(PPL)-based sensitivity analysis method for evaluating KV pruning at the block level, enabling optimal pruning ratios for each block.
- A method for intra-block evaluation of query (Q), key (K), value (V), and output (O) channels, validating the effectiveness of various pruning strategies.

II. METHODS

The Decoder-Only architecture for large models, featuring the Transformer [5] decoder structure with multiple stacked decoders, exhibits exceptional performance. Each decoder module primarily includes a normalization layer (Norm), multi-head attention (MHA), and a feedforward neural network (FFN). The main computational and storage bottleneck lies in the MHA module. Therefore, we focus on pruning KV cache to achieve better performance. As shown in Fig. 2, Our KVPruner method consists of two key phases: Phase A evaluates the optimal global pruning ratio for a given pruning target, while Phase B handles the intra-block importance of Q, K, V and O channels and executes the pruning operation. Our method performs one-shot pruning on the model, resulting in no additional overhead during runtime.

A. Global Pruning Ratio Awareness

As shown in Fig. 3, we analyze the dimensions of each block to ensure performance in both task-specific and generalization tasks. We observe that the KV channels within different blocks have varying levels of impact on PPL in LLaMA. Additionally, though the range of Q, K, and V parameters varies significantly from block to block, the overall trend aligns with the block-level PPL changes, with the first and last blocks showing the most significant impact.

Therefore, we define the PPL change for each block as ΔPPL_i , which allows us to compute the global importance ratio or priority for each block. As shown in step A of Fig. 2, we propose two methods for calculating the block pruning ratio distribution and further compute the sensitivity distribution within each block.

1) *PPL-based Sensitivity Distribution*: This method allocates the pruning ratio for each block based on its relative PPL changes, as shown in Fig. 3. The key idea is to assign lower pruning ratios to blocks with higher sensitivity, as indicated by larger PPL changes. Let ΔPPL_i represent the PPL changes for the i -th block. The pruning ratio for each block is computed by taking the inverse of the exponential change in PPL,

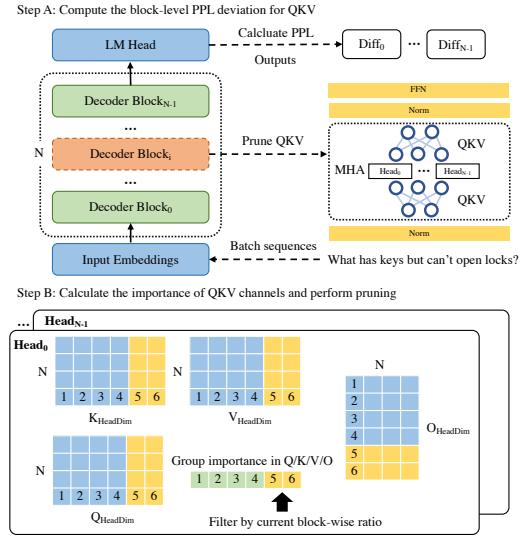


Fig. 2. Illustrates the simplified workflow of the KVPruner in LLMs. The pruning process consists of two main steps: First, global sensitivity analysis assigns the optimal pruning ratio to each block. Second, local channel sensitivity aggregates the importance of Q, K, V and O channels for evaluation and removes the less important ones. After completing these steps, LoRA is applied to quickly recovery the performance.

normalized by the total exponential change across all blocks. The pruning ratio for the i -th block is given by Eq. (1).

$$P_i = \frac{\frac{1}{e^{\Delta PPL_i} + \epsilon}}{\sum_{j=1}^N \frac{1}{e^{\Delta PPL_j} + \epsilon}} \cdot P_{\text{total}} \quad (1)$$

where P_i is the pruning ratio for the i -th block, ΔPPL_i is the PPL changes, $e^{\Delta PPL_i}$ represents the exponential change, ϵ is a small constant to avoid division by zero, P_{total} is the global pruning ratio, and N is the total number of blocks.

2) *Rank-based Sensitivity Distribution*: In this method, we prioritize blocks based on their PPL changes. Blocks are sorted by their PPL change, and pruning ratios are assigned according to this sorted order. Blocks with higher priorities are pruned, while those with lower priorities are not.

Let id_i be the ranking of the i -th block based on its PPL change. We define a threshold based on the total number of blocks and the total pruning ratio. The pruning ratio for each block is then calculated as shown in Eq. (2).

$$P_i = \begin{cases} 1, & \text{if } id_i \leq [P_{\text{total}} \cdot N] \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where P_i is the pruning ratio for the i -th block, id_i is the rank of the i -th block according to its PPL change, P_{total} is the total global pruning ratio, and N is the total number of blocks.

B. Local Channel Importance Evaluation

After determining the block-level KV pruning ratio in Section II-A, we further evaluate the intra-block importance of Q, K, V and O matrix channels, as shown in step B of Fig. 2.

First, we need to compute the total score for all elements in each channel, then we provide three methods to calculate the importance score for each channel. These methods are used to evaluate the significance of each channel, which can later be utilized for pruning purposes. Let W be the weight matrix corresponding to Q, K, V and O channels with dimensions $(C_{\text{in}}, C_{\text{out}})$, where C_{in} and C_{out} are the number of input and output channels, respectively.

1) *L1 Method*: L1 is a basic metric commonly used in pruning literature. It measures the importance of each channel by computing the sum of the absolute values of the weights within that channel. For the i -th channel, the L1 importance score $S_{L1,i}$ is computed as shown in Eq. (3).

$$S_{L1,i} = \sum_{j=1}^{C_{\text{in}}} \sum_{k=1}^{C_{\text{out}}} |W_{i,j,k}| \quad (3)$$

where the weight element $W_{i,j,k}$ represents the weight for the i -th channel. C_{in} and C_{out} denote the input and output channel dimensions, respectively.

2) *L2 Method*: The L2 method extends the L1 metric by squaring the weights, which emphasizes larger weights. The L2 importance score $S_{L2,i}$ for the i -th channel is computed as shown in Eq. (4).

$$S_{L2,i} = \sum_{j=1}^{C_{\text{in}}} \sum_{k=1}^{C_{\text{out}}} (W_{i,j,k})^2 \quad (4)$$

3) *Taylor Method*: The Taylor method leverages the error on a calibration dataset and uses the first-order term of the Taylor expansion to approximate the importance of each channel. For a given calibration dataset D , we compute the gradient of the loss function \mathcal{L} with respect to the weights. The Taylor importance score $S_{\text{Taylor},i}$ for the i -th channel is computed as shown in Eq. (5).

$$S_{\text{Taylor},i} = \sum_{j=1}^{C_{\text{in}}} \sum_{k=1}^{C_{\text{out}}} \left| \frac{\partial \mathcal{L}}{\partial W_{i,j,k}} W_{i,j,k} \right| \quad (5)$$

where \mathcal{L} is the training loss, and $\frac{\partial \mathcal{L}}{\partial W_{i,j,k}}$ is the gradient of the loss with respect to the weight $W_{i,j,k}$.

The importance score of a channel in the KV dimension is the average of the scores for the corresponding Q, K, V and O channels, calculated using the following formula:

Let M represent the method used to calculate the importance score (either L1, L2, or Taylor). For the i -th channel, the average importance score across the Q, K, V, and O matrices is given by the following unified formula as shown in Eq. (6).

$$S_{M,i}^{\text{avg}} = \frac{1}{4} (S_{M,i}^Q + S_{M,i}^K + S_{M,i}^V + S_{M,i}^O) \quad (6)$$

where $S_{M,i}^Q$ is the importance score of the i -th channel in the Q matrix using method M , $S_{M,i}^K$ is the importance score of the i -th channel in the K matrix using method M , $S_{M,i}^V$ is the importance score of the i -th channel in the V matrix using method M , and $S_{M,i}^O$ is the importance score of the i -th channel in the O matrix using method M .

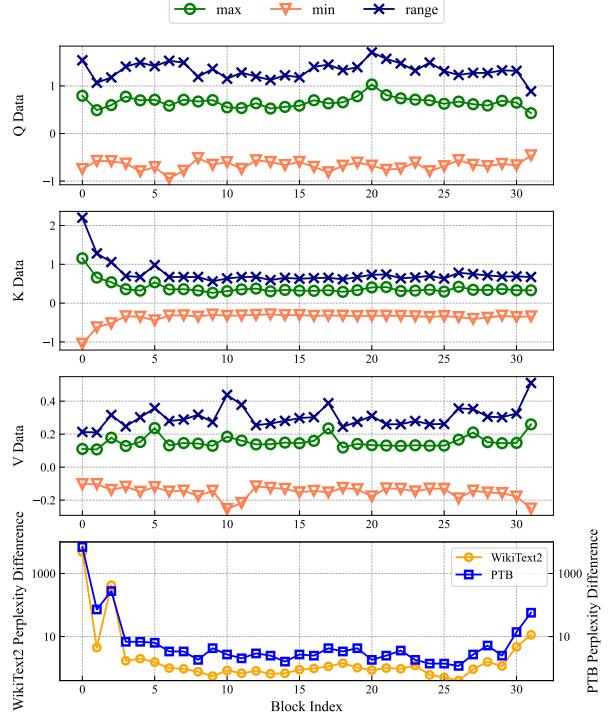


Fig. 3. The first three graphs show the range of QKV weight changes in a specific block. Estimated on the evaluation set, the fourth graph illustrates the sensitivity changes after removing the KV from the block.

C. Fine-Tuning After Pruning

For small-scale structured pruning, our method causes minimal change in PPL, but fine-tuning can still be applied to further improve model performance at a low cost. Compared to the Shortened-LLM [18] method which requires high-cost CPT [18] retraining on H100 GPUs for two weeks, our approach only requires a few hours of fine-tuning using LoRA [27], significantly reducing training costs.

III. EXPERIMENTS

A. Experimental Setup

To evaluate the performance of our method compared to existing structured pruning methods in terms of memory usage, inference latency, throughput, and PPL on the test set, we selected the LLaMA-7B [32] model as our base model. Our baseline methods include LLM-Pruner [17] and Shortened-LLM [18], representing state-of-the-art width and depth structured pruning approaches, respectively.

PPL Evaluation: We assess block-level KV sensitivity by randomly selecting a portion of the WikiText2 dataset for screening and evaluating PPL performance on the remaining WikiText2 and PTB datasets.

Inference Overhead Evaluation: To avoid the storage bottleneck mentioned in Shortened-LLM [18], we disable inference caching during the evaluation, calculating inference

TABLE I
PPL VALUES UNDER DIFFERENT PRUNING METHODS AND PRUNING RATIOS.

Method	Ratio (%)	Parameters (B)	kv store↓ (GB)	PPL↓(LoRA)	
				WikiText2	PTB
LLaMA-7B	0	7	8	12.61	22.14
LLM-Pruner	20	5.5	6.78	15.73	71.48
LLM-Pruner	50	3.5	4.75	26.24	112.02
Shortened	20	5.5	6.75	16.29	70.37
Shortened	50	3.5	4	36.79	124.97
Ours	20(qkv)	6.3	6.4	14.55	27.18
Ours	50(qkv)	5.5	4	21.5	40.33

TABLE II
LATENCY AND THROUGHPUT VALUES UNDER DIFFERENT PRUNING METHODS AND PRUNED MODEL PARAMETERS.

Method	Parameters (B)	A100 40GB		RTX 3090 24GB	
		Latency↓ (s)	Throughput↑ (tokens/s)	Latency↓ (s)	Throughput↑ (tokens/s)
LLaMA-7B	7	64.995	63.02	75.804	27.017
LLM-Pruner	5.5	64.045	63.955	66.187	30.943
Shortened	5.5	54.425	75.26	63.759	32.122
Ours	6.3	60.447	67.762	62.729	32.649
Ours	5.5	52.438	78.111	58.132	35.23

cost purely based on computation. This evaluation applies to scenarios where computation and storage are decoupled, such as in a separated architecture [31], [33], [34].

Latency and Throughput Evaluation: Following the evaluation standard of Shortened-LLM [18], we preheat the model 10 times and then measure the time T for a given batch size and output sequence length L . We calculate throughput as ML/T , where the input token size is 1024.

Hardware Environment: We perform pruning and two-batches LoRA fine-tuning on an NVIDIA A100 GPU, and test inference latency and throughput on both NVIDIA A100 GPU and NVIDIA RTX 3090.

B. Main Results

Table I and Table II show the performance comparison under 20% and 50% pruning ratios. When the remaining number of model parameters is the same after pruning, our method reduces runtime memory usage by 50%, significantly surpassing other methods. Additionally, our approach demonstrated superior inference speed and throughput on both A100 and 3090 machines. Under the same 20% and 50% pruning ratios, our method also achieved lower PPL than other methods, indicating better performance. We also observe that while Shortened-LLM approaches the performance of our method under higher pruning ratios, its coarse pruning approach prevented effective recovery within a reasonable fine-tuning time.

C. Ablation Study

We perform a detailed analysis of several metrics mentioned in the method. Table III and Table IV present comparisons of global metrics and intra-block channel importance under 20% and 50% pruning ratios, respectively. For the global metrics,

TABLE III
COMPARISON OF PRUNING STRATEGIES WITH A KV PRUNING RATIO OF 20%.

Global	Block	PPL↓(Pruned)		PPL↓(LoRA)	
		WikiText2	PTB	WikiText2	PTB
Uniform	01	17.89	32.81	15.46	64.83
	11	768.56	1010.26	245.64	327.98
	12	860.75	891.55	22.75	43.02
	taylor	564.49	566.70	24.41	45.88
PPL-based	01	15.04	29.33	14.55	27.18
	11	178.32	230.76	20.68	36.65
	12	421.14	434.50	20.48	36.58
	taylor	129.44	143.84	18.18	33.18

TABLE IV
COMPARISON OF PRUNING STRATEGIES WITH A KV PRUNING RATIO OF 50%.

Global	Block	PPL↓(Pruned)		PPL↓(LoRA)	
		WikiText2	PTB	WikiText2	PTB
Uniform	01	86.56	125.46	24.46	53.85
	11	1939.74	2539.80	57.89	121.13
	12	1552.55	1691.87	55.89	113.79
	taylor	1272.11	1564.73	88.62	186.15
PPL-based	01	52.40	115.46	21.50	40.33
	11	141.06	194.32	23.89	44.82
	12	124.48	192.06	22.80	43.61
	taylor	157.36	201.27	23.25	42.93

we compare uniform pruning (i.e., without distinguishing blocks) with PPL sensitivity-based pruning and find that the latter achieves better results. In the uniform pruning method (01, which selects based on ranked proportions), sensitive blocks with significant PPL changes are skipped, leading to better performance. However, other pruning methods that prune sensitive blocks experience a sharp increase in PPL. We also compare the 01, L1, L2, and Taylor pruning methods. The 01 method performs well at small pruning ratios by avoiding PPL-sensitive blocks, achieving strong performance even before fine-tuning. In other scenarios, the Taylor method performs best, followed by L2, with L1 showing the weakest performance. However, all methods are able to recover to good levels under the same fine-tuning conditions, offering more options for pruning in specific task scenarios.

IV. CONCLUSION

In this work, we propose KVPruner, a structured pruning method optimized for runtime KV cache during LLMs inference. By combining global sensitivity awareness with local channel sensitivity techniques, KVPruner improves model efficiency while maintaining performance. Unlike deep structured pruning methods, KVPruner recovers high performance in only two hours of LoRA fine-tuning on small datasets. Our experiments on the LLaMA-7B model demonstrate that KVPruner reduces KV memory usage by 50% and boosts throughput by over 35%, offering a balanced and effective approach to pruning non-essential KV channels, demonstrating its ability to significantly reduce KV memory usage and improve inference efficiency.

REFERENCES

- [1] T. Chen, C. Allauzen, Y. Huang, D. Park, D. Rybach, W. R. Huang, R. Cabrera, K. Audhkhasi, B. Ramabhadran, P. J. Moreno, and M. Riley, “Large-scale language model rescoring on long-form data,” in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.
- [2] S.-L. Wu, X. Chang, G. Wichern, J.-W. Jung, F. Germain, J. Le Roux, and S. Watanabe, “Improving audio captioning models with fine-grained audio features, text embedding supervision, and llm mix-up augmentation,” in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 316–320.
- [3] I. Malkiel, U. Alon, Y. Yehuda, S. Keren, O. Barkan, R. Ronen, and N. Koenigstein, “Segllm: Topic-oriented call segmentation via llm-based conversation synthesis,” in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 11 361–11 365.
- [4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: <http://arxiv.org/abs/1810.04805>
- [5] A. Vaswani, “Attention is all you need,” *Advances in Neural Information Processing Systems*, 2017.
- [6] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, “Music transformer,” *arXiv preprint arXiv:1809.04281*, 2018.
- [7] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, “Fastspeech: Fast, robust and controllable text to speech,” *Advances in neural information processing systems*, vol. 32, 2019.
- [8] Y. Gong, C.-I. Lai, Y.-A. Chung, and J. Glass, “Ssast: Self-supervised audio spectrogram transformer,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, 2022, pp. 10 699–10 709.
- [9] A. Dosovitskiy, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.
- [10] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.
- [11] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, “Emerging properties in self-supervised vision transformers,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 9650–9660.
- [12] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, “Judging llm-as-a-judge with mt-bench and chatbot arena,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [13] A. Das, S. Kottur, K. Gupta, A. Singh, D. Yadav, J. M. Moura, D. Parikh, and D. Batra, “Visual dialog,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 326–335.
- [14] A. Zhao, D. Huang, Q. Xu, M. Lin, Y.-J. Liu, and G. Huang, “Expel: Llm agents are experiential learners,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, 2024, pp. 19 632–19 642.
- [15] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, “A survey on rag meeting llms: Towards retrieval-augmented large language models,” in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 6491–6501.
- [16] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.
- [17] X. Ma, G. Fang, and X. Wang, “Llm-pruner: On the structural pruning of large language models,” *Advances in neural information processing systems*, vol. 36, pp. 21 702–21 720, 2023.
- [18] B.-K. Kim, G. Kim, T.-H. Kim, T. Castells, S. Choi, J. Shin, and H.-K. Song, “Shortened llama: A simple depth pruning for large language models,” *arXiv preprint arXiv:2402.02834*, 2024.
- [19] M. Xia, T. Gao, Z. Zeng, and D. Chen, “Sheared llama: Accelerating language model pre-training via structured pruning,” *arXiv preprint arXiv:2310.06694*, 2023.
- [20] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, “A simple and effective pruning approach for large language models,” *arXiv preprint arXiv:2306.11695*, 2023.
- [21] E. Frantar and D. Alistarh, “Sparsegpt: Massive language models can be accurately pruned in one-shot,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 10 323–10 337.
- [22] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” *arXiv preprint arXiv:2210.17323*, 2022.
- [23] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, “Awq: Activation-aware weight quantization for on-device llm compression and acceleration,” *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.
- [24] H. Zhang, X. Ji, Y. Chen, F. Fu, X. Miao, X. Nie, W. Chen, and B. Cui, “Pqcache: Product quantization-based kvcache for long context llm inference,” *arXiv preprint arXiv:2407.12820*, 2024.
- [25] X. Wang, J. Cui, Y. Suzuki, and F. Fukumoto, “Rdrec: Rationale distillation for llm-based recommendation,” *arXiv preprint arXiv:2405.10587*, 2024.
- [26] J. Saad-Falcon, O. Khattab, K. Santhanam, R. Florian, M. Franz, S. Roukos, A. Sil, M. A. Sultan, and C. Potts, “Udapdr: unsupervised domain adaptation via llm prompting and distillation of rerankers,” *arXiv preprint arXiv:2303.00807*, 2023.
- [27] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *arXiv preprint arXiv:2106.09685*, 2021.
- [28] L. Zheng, L. Yin, Z. Xie, J. Huang, C. Sun, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez *et al.*, “Efficiently programming large language models using sclang,” *arXiv preprint arXiv:2312.07104*, 2023.
- [29] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.
- [30] T. Dao, “Flashattention-2: Faster attention with better parallelism and work partitioning,” *arXiv preprint arXiv:2307.08691*, 2023.
- [31] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, “Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving,” *arXiv preprint arXiv:2401.09670*, 2024.
- [32] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [33] C. Hu, H. Huang, J. Hu, J. Xu, X. Chen, T. Xie, C. Wang, S. Wang, Y. Bao, N. Sun *et al.*, “Memserve: Context caching for disaggregated llm serving with elastic memory pool,” *arXiv preprint arXiv:2406.17565*, 2024.
- [34] R. Qin, Z. Li, W. He, M. Zhang, Y. Wu, W. Zheng, and X. Xu, “Mooncake: Kim’s kvcache-centric architecture for llm serving,” *arXiv preprint arXiv:2407.00079*, 2024.