A Survey on Compression Techniques for Large Language Models

Shiwei Chen, Mason Davis McBride
University of California, Berkeley

Abstract
Large Language Models (LLMs) have revolutionized natural language processing tasks, but their significant size and computational demands create barriers to practical deployment, particularly in resource-constrained environments. Model compression has emerged as a pivotal research field to overcome these limitations. This research project provides a survey of the domain space of compression techniques for LLMs, exploring methods such as quantization, pruning, knowledge distillation, and low rank factorization, as well as orthogonal methods combining two or more of these areas. There is an emphasis on recent developments, as well as practical implementation and performance comparisons between the various compression methods surveyed. We hope this project will serve as a valuable resource for ML practitioners aiming to improve the efficiency and practical utility of LLMs.

Contents

1 Background

LLMs are built on the Transformer architecture[9] which is based on the idea of multi-headed self-attention (MHA), which allows the model to jointly attend to information at different positions from different representation subspaces.[10] MHA is defined as

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O,$$

where $Q, K, V \in \mathbb{R}^{n \times d_m}$ are input embedding matrices, n is the sequence length, and d_m is the embedding dimension, and h is the num-

ber of heads. Each head is defined as

$$\text{head}_i = \text{softmax}\left[\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d_k}}\right]VW_i^V, \tag{1}$$

where $W_i^Q, W_i^K \in \mathbb{R}^{d_m \times d_k}$, and $W_i^V \in \mathbb{R}^{d_m \times d_v}$, $W^O \in \mathbb{R}^{hd_v \times d_m}$ are learned matrices and d_k, d_v are hidden dimension of the projection subspaces.

2 Quantization

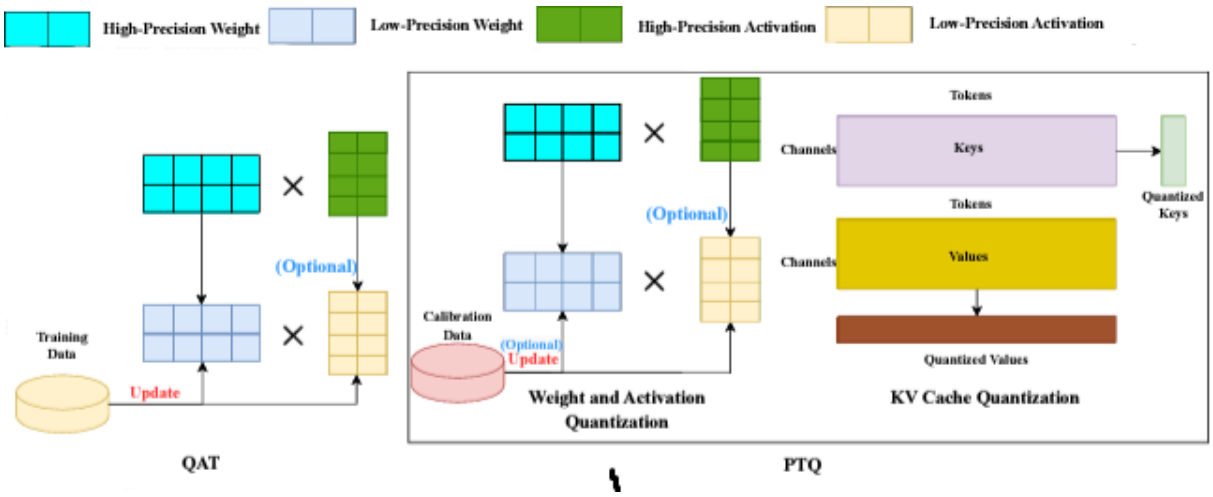

Figure 1: An illustration comparing Quantization-Aware Training (QAT) and Post-Training Quantization (PTQ).

When Large Language Models are required on mobile devices or edge devices, the memory overhead can be a major blocker. Not just memory, but also inference will be slow because the computation required to run a forward pass on hundreds of billions of parameters is too large for mobile processors. Instead of reducing the number of parameters directly, quantization is a technique that reduces the actual size of the data type the parameters are stored as. Basic methods involve reducing the 32-bit floating point numbers 16-bit floats, 16-bit bfloats, 8-bit integers, 4-bit floats, and even as small 1-bit integers.

Quantization inherently reduces the amount of information stored in each of the parameters, theoretically reducing the performance of the LLM. Researchers have tackled this problem in one of two ways: Quantization-Aware Training and Post-Training Quantization.

2.1 Quantization-Aware Training (QAT)

To regain the information lost in quantization, Quantization-Aware training retrains the quantized model on output from the original size LLM.

2.1.1 OneBit: Towards Extremely Low-bit Large Language Models

The OneBit[13] approach quantizes the weight matrices of LLMs to 1-bit using a method known as Sign-Value-Independent Decomposition (SVID). This approach decomposes the weight matrix W into a sign matrix $W_{\pm 1}$ and two value vectors g and h to maintain precision. The decomposition of W is represented as:

$$W \approx W_{\pm 1} \odot (g \cdot h^T),$$

where $\odot$ denotes element-wise multiplication. The sign matrix $W_{\pm 1}$ contains only ± 1 values, and the value vectors g and h are stored in FP16 precision. This decomposition enables efficient 1-bit quantization while preserving much of the model's predictive performance.

One of the most striking outcomes of the OneBit method is its ability to achieve a compression ratio far greater than traditional quantization methods. The experiments conducted on LLaMA models demonstrated that even at 1-bit quantization, the predictive performance was retained at 81% of the non-quantized model. This finding shows the potential of 1-bit quantization in edge devices where memory and computational efficiency are critical. Furthermore, the SVID approach ensures that the approximation of the original weight matrix is highly accurate, thereby minimizing the loss in performance. The method also employs a novel initialization strategy for the quantized weights using rank-1 decomposition to ensure better convergence during training.

Additionally, OneBit incorporates knowledge distillation to bridge the performance gap between quantized models and full-precision models. This process uses the output of the full-precision model as a teacher to guide the quantized model's training, thereby ensuring that key decision boundaries are preserved. This combination of efficiency and effectiveness makes OneBit a revolutionary advancement in LLM quantization.

2.1.2 Low-Rank Quantization-Aware Training for LLMs

The Low-Rank Quantization-Aware Training (LR-QAT)[1] method introduces low-rank adapters to reduce the memory footprint during quantization-aware training. The weight matrix W is represented as:

$$W_c = s \cdot \text{clip}\left(\frac{W_0}{s_0} + \frac{\alpha}{r}AB, -2^{b-1}, 2^{b-1} - 1\right),$$

where s is a quantization scale, A and B are low-rank matrices, and W_0 is the pre-trained weight matrix. Here, s_0 is a frozen initial scale, and the matrices A and B are updated during training. This approach reduces the memory required for training and enables the merging of the low-rank matrices into the quantized weight tensor at the end of training.

The LR-QAT method stands out due to its memory efficiency and adaptability. Unlike traditional QAT, which requires significant memory overhead, LR-QAT only requires low-rank adapters A and B, leading to a substantial reduction in training memory requirements. The method was validated on LLaMA-2 and Mistral model families, showcasing its ability to achieve performance on par with full QAT but at a fraction of the memory cost. This efficiency makes it possible to train large models on a single consumer-grade GPU.

LR-QAT also incorporates an innovative checkpointing strategy that avoids storing intermediate computation results in memory during backpropagation. Instead, it recomputes necessary elements on-the-fly, thereby significantly reducing the memory footprint. This feature allows large LLMs to be trained using consumer-grade GPUs without compromising on computational efficiency.

Another critical aspect of LR-QAT is its compatibility with other PTQ techniques, allowing for flexible integration with existing quantization pipelines. By combining low-rank decomposition with quantization-aware training, LR-QAT achieves high compression ratios while maintaining high accuracy. Additionally, it supports flexible quantization granularity and can be used for both weights and activations.

2.2 Post-Training Quantization

Post-training quantization (PTQ) reduces model size and computational cost without requiring retraining. This approach directly

quantizes the trained weights and activations of an LLM.

2.2.1 SpinQuant: LLM Quantization with Learned Rotations

SpinQuant[7] addresses the challenge of outliers in LLMs by applying rotation matrices to weight and activation tensors before quantization. By using optimized rotation matrices R_1, R_2, R_3, R_4, the method reduces the kurtosis of activation distributions, making them more Gaussian-like, which facilitates easier quantization. The rotation transformation is represented as:

$$XQ = \alpha \left\lfloor \frac{XR - \beta}{\alpha} \right\rfloor + \beta,$$

where X_Q is the quantized tensor, X_R is the rotated real-valued tensor, and α and β are scale and offset values, respectively. By optimizing the rotation matrices, the model achieves better quantization results compared to random rotation, leading to minimal performance degradation.

SpinQuant introduces two main variants, SpinQuant^{nohad} and SpinQuant^{had}. SpinQuant^{nohad} merges rotation matrices into pre-trained weights without altering the network architecture, while SpinQuant^{had} introduces online Hadamard rotations to address low-bit quantization of activations and KV-cache. This combined approach reduces the performance gap between quantized and full-precision models.

Conclusion

The quantization methods presented here illustrate the diverse strategies available for compressing large language models. From OneBit's extreme 1-bit quantization to LR-QAT's memory-efficient low-rank approach and SpinQuant's optimization of activation distributions, each method demonstrates a unique path toward efficiency. These techniques enable LLMs to be deployed on resource-constrained devices while maintaining high performance. Collectively, they highlight the potential of quantization to reshape the practical deployment of LLMs in real-world applications.

3 Pruning

Prune small models from LLMs

Neural networks are known to possess more parameters than necessary for effective generalization and accurate predictions. Research by Frankle et al.[4] highlights that only a specific subset of these parameters is crucial for making predictions. Model pruning emerges as a natural approach to compressing neural networks, based on the premise that effective compression involves eliminating weights that are not actively contributing to the model's performance.

3.1 Lottery Ticket Hypothesis

Hypothesis: *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

Identifying winning tickets: We identify a winning ticket by training a network and pruning its smallest magnitude weights. The remaining, unpruned connections constitute the architecture of the winning ticket, and each unpruned connections value is then reset to its initialization from the original network

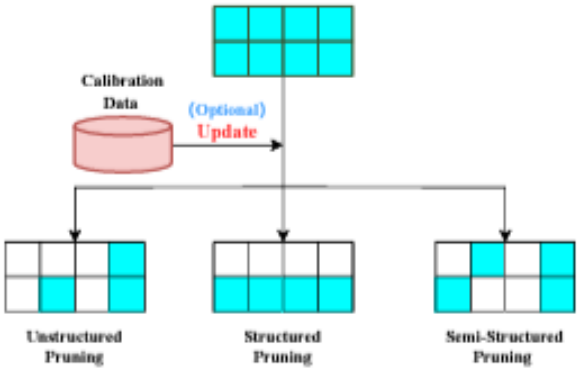

Figure 2: Pruning.

before it was trained.

We can prune the weights for n rounds for a $p\%$ pruned network by pruning $p^{\frac{1}{n}}\%$ each round. Winning tickets tend to learn faster than the original networks.

3.1.1 Interaction with Dropout

Dropout[8] improves accuracy by randomly disabling a fraction of the units on each training iteration. Since the lottery hypothesis suggests that one of these subnetworks comprises a winning ticket, it is natural to ask whether dropout and network pruning interact.

Experiments combining pruning and dropout[4] suggests that iterative pruning interacts with dropout in a complimentary way.

3.1.2 Limitations

Retraining a neural network n times is very computationally intensive, and likely not feasible on modern LLMs. It is possible to take a one-shot pruning approach, but *iterative pruning* typically produces better accuracy at smaller sizes.

3.2 LLM-shearing

LLM shearing is an algorithm which consists of the following two components: (1) *targeted structured pruning*, which prunes a source model to a specified target architecture, typically existing pre-trained models, and (2) *dynamic batch loading* which dynamically updates the composition of sampled data in each training batch based on varying losses across different domains.[11] We only cover (1) as it is most relevant to our studies.

Though aggressive pruning in (1) inevitably incurs a performance drop, continued pre-training in (2) helps the compressed model reach and occasionally exceed pre-pruned performance.

3.2.1 Targeted Structured Pruning

The goal of targeted structured pruning is to prune the source model into some desired target configuration, typically that of a pre-trained model, a decision based on the intuition that these configurations have already been well-optimized.

The LLM learns a set of pruning masks (a binary variable that decides whether a component is pruned or retained) on model parameters at different granularities, ranging from hidden dimensions to local multi-head attentions. To directly impose constraints on the pruned model shape, we use Lagrange multipliers. For example, for a target number of

heads H_τ, we have the imposed constraint on a single layer as

$$L^{\text{head}}(\lambda, \phi, z) = \lambda^{\text{head}} \cdot \left(\sum z^{\text{head}} - H_\tau \right) + \phi^{\text{head}} \cdot \left(\sum z^{\text{head}} - H_\tau \right)^2$$

Similar constraints are applied to pruning other structures. Overall, we jointly optimize the model weights and pruning masks by a min-max objective $\min_{\theta,z} \max_{\lambda,\phi} L_{\text{prune}}(\theta, z, \lambda, \phi)$:

$$L_{\text{prune}}(\theta, z, \lambda, \phi) = L(\theta, z) + \sum_{j=1}^{L_S} L_j^{\text{head}} + \sum_{j=1}^{L_S} L_j^{\text{int}} + L^{\text{layer}} + L^{\text{hidden}}.$$

where $L(\theta, z)$ is the language modeling loss computed with the masked model weights. This objective will eventually produce a model with the target shape.

Following pruning, we can finalize the pruned architecture by preserving the highest-scoring components associated with the mask variables in each substructure, and continue pretraining the pruned model with the language modeling objective.

4 Knowledge Distillation

Small Models Learn to Replicate Large Models

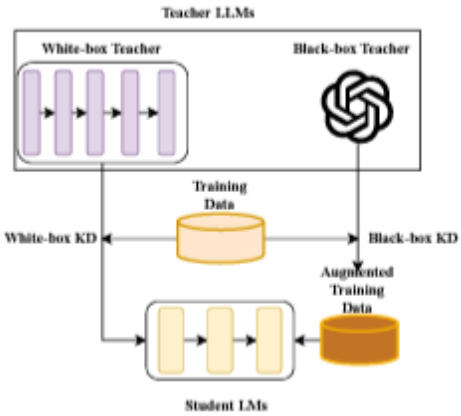

Figure 3: An illustration showing Black-box knowledge distillation and so-called Grey-box knowledge distillation.

Knowledge distillation is a model compression technique that transfers knowledge from a larger, more complex model (the *teacher model*) to a smaller, more efficient model (the *student model*. This approach to LLM compression was used to create the open-source LLaMA models and is an active area of research. Both the techniques discussed in this report and other heuristic methods are covered in a survey of knowledge distillation techniques by Xiaohan et al.[12]

4.1 Supervised Fine-Tuning

Supervised Fine-Tuning (SFT), also known as Sequence-Level Knowledge Distillation (Se-

qKD)[6], is one of the simplest yet highly effective methods for distilling black-box LLMs. SFT involves fine-tuning the student model by maximizing the likelihood of the sequences generated by the teacher LLM, ensuring that the student's predictions closely align with those of the teacher. Let p be the probability distribution of the teacher model, and q be the probability distribution of the student model. This process can be mathematically expressed as minimizing the following objective function:

$$L_{\text{SFT}} = \mathbb{E}_{x\sim\chi, y\sim p_T(y|x)}[-\log p_S(y|x)],$$

where y is the output sequence produced by the teacher model.

4.2 Divergence

Divergence-based methods minimize divergence between the probability distributions of the teacher and student models, represented by a general divergence function D:

$$L_{\text{div}} = \mathbb{E}_{x\sim\chi, y\sim\Upsilon}[D(p_T(y|x), p_S(y|x))],$$

The specific form of D varies depending on the type of divergence employed. The commonly-used standard KD objectives minimize the approximated forward Kullback-Leibler divergence (KLD) between the teacher and the student output distribution (soft labels), which is

$$D(p,q) = \sum p(t) \log \frac{p(t)}{q(t)}.$$

A drawback of this approach is that soft labels are not available for the most powerful, state of the art LLMs.[2]

4.3 Similarity

Similarity-based methods in knowledge distillation aim to align the hidden states or features of the student model with those of the teacher. These methods use various similarity metrics to measure and optimize the congruence of internal representations between the two models. The objective is to ensure that the student model not only produces outputs similar to those of the teacher, but also processes information in a comparable manner. The formulation for a similarity-based objective might look like:

$$L_{\text{Sim}} = \mathbb{E}_{x\sim\chi, y\sim\Upsilon}[L_F(\Phi_T(f_T(x,y)), \Phi_S(f_S(x,y)))]$$

where $f_T(x,y)$ and $f_S(x,y)$ are the feature maps of the teacher and student models. Transformation functions Φ_T, Φ_S are applied to these feature maps to ensure they are in the same shape, facilitating direct comparison. The similarity function L_F is used to match these transformed feature maps. Typical functions for L_F include:

- L2 Norm: $||\Phi_T(f_T(x,y)) - \Phi_S(f_S(x,y))||_w$
- L1 Norm: $||\Phi_T(f_T(x,y)) - \Phi_S(f_S(x,y))_1$

5 Low Rank Factorization

Low-Rank matrices produce comparable performance

5.1 The Linformer

The softmax equation (1) of the standard self-attention mechanism of the Transformer requires multplying two $n \times d$ matrices, which uses $O(n^2)$ time and space with respect to sequence length. The Linformer[10] reduces both complexities to $O(n)$ by approximating the self-attention mechanism with a low-rank matrix.

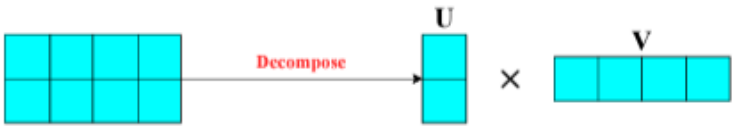

Figure 4: Low Rank Decomposition.

The Linformer paper hypothesizes that the self-attention mechanism has low intrinsic rank. Formally, $\forall Q, K, V \in \mathbb{R}^{n \times d}$ and $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d}$ for any column vector $\omega \in \mathbb{R}^n$ of matrix VW_i^V, there exists a low-rank matrix $\tilde{P} \in \mathbb{R}^{n \times n}$ such that

$$P(||\tilde{P}\omega^T - P\omega^T|| < \epsilon||P\omega w^T||) > 1 - o(1)$$

and $\text{rank}(\tilde{P}) = \Theta(\log(n))$, where the context mapping P is defined in equation (1) as the softmax function.

To calculate $P \cdot VW_i^V$ in linear time and memory complexity, we can add two linear projection matrices $E_i, F_i \in \mathbb{R}^{n \times k}$ when computing key and value. First, project the original $(n \times d)$ dimensional key and value layers KW_i^K and VW_i^V into $(k \times d)$ dimensional projected key and value layers. Then compute a $(n \times k)$ dimensional context mapping $\overline{P}$ using scaled dot product attention:

$$\overline{head_i} = \text{softmax}\left(\frac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_k}}\right) \cdot F_iVW_i^V$$

Computing context embeddings for each head using $\overline{P} \cdot (F_iVW_i^V)$ requires $O(nk)$ time and space complexity. Choosing a very small projected dimension $k << n$, we can significantly reduce the memory and space consumption.

5.2 Low Rank Adaptation

Traditional fine-tuning of LLMs for domain-specific applications typically updates all the parameters of the pre-trained model.

Low-Rank Adaptation (LoRA) is a technique which freezes the pre-trained model weights and injects trainable **low-rank** decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks[5].

It was shown in previous papers that large, over-parametrized models in fact reside on a low intrinsic dimension. LoRA takes advantage of this by finetuning matrices of comparatively tiny rank (typically $r = 8$ or 16, but sometimes as small as $r = 1$).

This makes LoRA both storage and compute efficient. We only need to optimize the injected, much smaller low-rank matrices, rather than calculating the gradients and maintaining the optimizer states for most parameters. The low-rank matrices are then **merged** with the frozen weights when deployed, *introducing no inference latency* compared to a fully-fine tuned model by construction.

5.2.1 The Method

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, we constrain its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$ where $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$, and the rank $r << \min(d, k)$. Note that both W_0 and BA are in $\mathbb{R}^{d \times k}$. We use random Gaussian initialization for A and initialize an empty matrix for B, so $\Delta W = BA$ is zero at the beginning of training.

Our forward pass becomes:

$$h = W_0 x + \Delta W x = W_0 x + BAx.$$

When deployed in production, we can explicitly compute and store $W = W_0 + BA$ and perform inference as usual.

5.2.2 LoRA on Transformers

LoRA can be applied to any weight matrix in a neural network to reduce the number of trainable parameters. In the Transformer arhitecture, there are four weight matrices in the self-attention module (W_q, W_k, W_v, W_σ) and two in the MLP module.

5.3 Quantized LoRA

Quantized LoRA (QLoRA) compresses LLMs further by introducing (a) 4-bit NormalFloat (NF4), a data type that is information theoretically optimal for normally distributed weights, (b) Double Quantization to reduce the average memory footprint by quantization the quantization constants, and (c) Paged Optimizers to manager memory spikes.[3]

5.3.1 Improvements on LoRA

4-bit NormalFloat Quantization: To ensure the entire range of a low-bit data type is used, the input data type is commonly rescaled into the target data type range through normalization by the absolute maximum of the input elements.

For example, quantizing a FP32 tensor into a Int8 tensor would be:

$$X^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(X^{\text{FP32}})} X^{\text{FP32}}\right)$$

$$= \text{round}(c^{FP32} \cdot X^{FP32})$$

where c is the quantization constant. Dequantization is the inverse:

$$\text{dequant}(c^{FP32, X^{\text{Int8}}}) = \frac{X^{\text{Int8}}}{c^{FP32}} = X^{FP32}.$$

Neural Network weights usually have a zero-centered normal distribution with standard deviation σ. We can therefore transform all weights to a single fixed distribution by scaling σ such that the distribution fits exactly into the range of our data type.

Double Quantization: Quantize the quantization constants for additional memory savings. the double quantization process will be explained further in the Implementation section

Paged Optimizers: Use the NVIDIA unified memory feature which does automatic page-to-page transfers between the CPU and GPU for error-free GPU processing in the scenario where the GPU occasionally runs out of memory.

5.3.2 Implementation

QLoRA for a single linear layer in the quantized base model with a single LoRA adapter is as follows, where f is the double Dequant function and g is the dequant function and

$$f(c_1^{FP32}, c_2^k, W^k) = q(q(c_1^{FP32}, c_2^k), W^4) = W^{BF16}$$

$$Y^{BF16} = X^{BF16} f(c_1^{FP32}, c_2^k, W^k) = W^{BF16}$$

6 Conclusions

This project has presented a comprehensive survey of compression techniques for Large Lan-

guage Models, analyzing the methodologies and impacts of quantization, pruning, knowledge distillation, and low-rank factorization. Each technique offers distinct advantages in optimizing LLMs for practical deployment.

Quantization methods, exemplified by OneBit and SpinQuant, effectively reduce model size and computational cost, enabling deployment on resource-constrained devices. Pruning strategies, including those based on the Lottery Ticket Hypothesis and targeted structured pruning, achieve model sparsity while preserving performance through iterative fine-tuning. Knowledge distillation facilitates the transfer of knowledge from large teacher models to smaller student models, bridging the gap between model complexity and deployability. Finally, low-rank factorization techniques like LoRA and QLoRA significantly improve fine-tuning efficiency by leveraging low-dimensional subspaces, reducing storage and computational overhead without substantial accuracy loss.

Ultimately, no single compression technique is universally superior. The most promising direction for future research lies in combining these orthogonal methods. For example, integrating quantized low-rank adaptation or developing hybrid approaches combining pruning and knowledge distillation could yield substantial improvements. As LLMs continue to grow in scale and complexity, model compression will remain crucial for ensuring accessibility, efficiency, and broader applicability across diverse domains.

References

[1] Yelysei Bondarenko, Riccardo Del Chiaro, and Markus Nagel. Low-rank quantization-aware training for llms, 2024. Preprint. Available at arXiv:2406.06385.

[2] Hongzhan Chen, Ruijun Chen, Yuqi Yi, Xiaojun Quan, Chenliang Li, Ming Yan, and Ji Zhang. Knowledge distillation of black-box large language models, 2024.

[3] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021.

[6] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation, 2016.

[7] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations, 2024. Preprint. Available at arXiv:2405.16406.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[10] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[11] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning, 2024.

[12] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models, 2024.

[13] Yuzhuang Xu, Xu Han, Zonghan Yang, Shuo Wang, Qingfu Zhu, Zhiyuan Liu, Weidong Liu, and Wanxiang Che. Onebit: Towards extremely low-bit large language models, 2024. Preprint. Available at arXiv:2402.11295.