Understanding the Impact of Post-Training Quantization on Large Language Models

Somnath Roy

Freshworks Inc

somnath.roy@freshworks.com

Abstract

Large language models (LLMs) are rapidly increasing in size, with the number of parameters becoming a key factor in the success of many commercial models, such as ChatGPT, Claude, and Bard. Even the recently released publicly accessible models for commercial usage, such as Falcon and Llama2, come equipped with billions of parameters. This significant increase in the number of parameters makes deployment and operation very costly. The remarkable progress in the field of quantization for large neural networks in general and LLMs in particular, has made these models more accessible by enabling them to be deployed on consumer-grade GPUs. Quantized models generally demonstrate comparable performance levels to their unquantized base counterparts. Nonetheless, there exists a notable gap in our comprehensive understanding of how these quantized models respond to hyperparameters, such as temperature, max new tokens, and top_k, particularly for next word prediction.

The present analysis reveals that nf4 and fp4 are equally proficient 4-bit quantization techniques, characterized by similar attributes such as inference speed, memory consumption, and the quality of generated content. Nevertheless, these quantization methods exhibit distinct behaviors at varying temperature settings, both in the context of smaller and larger models. Furthermore, the study identifies nf4 as displaying greater resilience to temperature variations in the case of the llama_2 series of models at lower temperature, while fp4 and fp4-dq proves to be a more suitable choice for falcon series of models. It is noteworthy that, in general, 4-bit quantized models of varying sizes exhibit heightened sensitivity to lower temperature settings, unlike their unquantized counterparts. Additionally, int8 quantization is associated with significantly slower inference speeds, whereas unquantized fp16 models consistently yield the fastest inference speeds across models of all sizes.

Index Terms: post-training quantization, LLM, nf4, fp4, nf4-dq, fp4-dq

1. Introduction

With the emergence of the Transformer architecture [1], a significant breakthrough was achieved, enabling the effective retention of extensive long-range dependencies in tasks related to natural language processing, speech, and vision. The transformer architecture enables highly parallel training due to sequence parallelism, which makes it possible to pretrain LLMs with hundreds of billions of parameters [2, 3, 4]. The Big-bench [5] introduced over 200 benchmarks designed to assess the capabilities of Large Language Models(LLMs) through quantification and extrapolation. This diverse and intricately elaborated set of benchmarks significantly contributed to the intensification of the race surrounding LLM development and advancement.

The widespread adoption of LLMs on a substantial scale gained traction following the successful establishment of ChatGPT (including GPT-3 and subsequent iterations) [2]. The pretraining of large transformer language models with 7 billion parameters and beyond demands a considerable amount of GPU computation, which can translate to costs amounting to millions of dollars. Such level of expenditure is beyond what academic research and small organizations can typically afford. Despite the high cost of deploying and operating large language models (LLMs), the recent release of the Falcon [6] and Llama2 [7] models has sparked optimism among small organizations and has increased their desire to deploy their own custom LLMs.

The efficient deployment of decoder only LLMs are challenging in practice because the generative inference proceeds sequentially, where the computation for each token depends on the previously generated tokens [8]. It is noteworthy that caching the attention key and value tensors of each layer can significantly improve the inference speed of smaller decoder-only models that fit on a single GPU memory. However, this is not possible for models that do not fit into the memory of a single GPU. To address the need for expensive high-end GPUs to support the deployment of these models, diverse forms of quantization have been put forward as potential solutions. The application of quantization methods to transformers emerges as a efficacious approach for mitigating sampling latency, while incurring minimal to negligible impact on overall performance [9]. Quantization techniques can be mainly characterized into three forms namely - i) quantization aware training [10, 11], ii) quantization aware fine-tuning [12, 13, 14], and iii) post training quantization (PTQ) [15, 16, 17]. In [18], the investigation primarily centers on evaluating the impact of diverse post-training quantization methods, employing perplexity scores as a benchmark. The perplexity scores are computed on datasets such as Wiki [19], PTB [20], and C4 [21], which mostl likely have served as foundational datasets during the training of most of the LLMs. It should be noted that these datasets are predisposed to exhibit favorable perplexity scores across all models, owing to their utilization in model training. Furthermore, it is acknowledged that perplexity, as a metric, may not effectively capture instances of repetitive generation within LLMs. Following outlines the primary contributions of the present study.

1. This study offers a systematic examination of the influence exerted by three pivotal hyper-parameters, namely, max new tokens, temperature, and top_k, on LLMs that have undergone quantization through widely adopted post-training quantization techniques such as [15][1] (hereafter, gptq) and

[1] https://github.com/IST-DASLab/gptq

[14, 12][2][3] (hereafter, bitsandbytes).

2. It explores how these hyper-parameters exert their influence across a range of model sizes, spanning from 3 billion to 70 billion parameters.
3. The process involves generating a total of 6,300 samples for each quantization method, achieved by constructing ten smaller prompts that encompass a diverse spectrum of domains for every model.
4. LLMs typically exhibit a tendency towards repetitive generation, and it is often challenging to discern such repetition through perplexity scores. Therefore, to identify and quantify repetitive generation, the primary metric employed is the number of duplicate content words.
5. It scrutinizes quantization methods that share similar inference speeds but manifest differing effects on accuracy.
6. Finally, it aims to discern the optimal quantization method for deployment, considering specific constraints and requirements.

2. Quantization

Quantization is a well defined mechanism for reducing the number of bits used to represent a value. In the context of large neural network models, quantization reduces the precision of the model's parameters and/or activations. Moreover, it has been found that the quantized large models are often competitive to its base ones in terms of accuracy while reducing the computational requirements.
In the context of LLMs, the quantization process can be divided into two types namely i) simulated and, ii) pure quantization. In simulated quantization, some operations are performed in floating-point arithmetic, which requires the dequantization of quantized parameters back to full precision during inference. [22, 23, 24, 25]. Pure quantization uses integer-only quantization, which eliminates the need for dequantization during inference [26, 27, 28, 12, 29]. The main difference between these two process of quantization is shown below in Table 1. However, it is crucial to note that pure quantization is a more aggressive approach and can also lead to a greater loss of accuracy. On the other hand, simulated quantization is a conservative approach and can achieve significant speedups without sacrificing too much accuracy. Pure quantization can be further categorized into W8A8 and W4A4, where the weights and activations are quantized to 8-bit integers and 4-bit integers, respectively [29] [27].

Features	**Simulated Quantization**	**Pure Quantization**
Operations	Floating and Fixed point	Fixed-point
Need for dequantization	Yes	No
Inference speed	Slower	Comparatively Faster

Table 1: *General understanding of simulated vs. pure quantization in transformer based LLMs*

[2]https://github.com/TimDettmers/bitsandbytes
[3]https://github.com/artidoro/qlora

2.1. GPTQ

It is a layer-wise quantization method based on the Optimal Brain Quantization (OBQ) [30]. The goal is to find a quantized weight matrix $\widetilde{W}$ that minimizes the squared error between the quantized layer output $\widetilde{W}X$ and the full-precision layer output WX as shown below.

$$\underset{\widetilde{W}}{\operatorname{argmin}} \|WX - \widetilde{W}X\|^2$$

The OBQ algorithm iteratively quantizes one weight at a time, while the GPTQ algorithm utilizes a vectorized implementation that allows it to efficiently handle multiple rows of the weight matrix in parallel. This makes GPTQ significantly faster than OBQ, especially for large models.

2.1.1. GPU Memory Consumption in 4-bit GPTQ Quantization

It is well-established that the goal of quantization is to deploy LLMs on consumer-grade GPUs having at most 24 GB.The distribution of GPU memory utilised by the models during GPTQ 4-bit quantization is shown below in Table 2. GPTQ quantization has following limitations.

- It is very GPU memory intensive process.
- Even 4-bit quantization of 40B model throws out of memory (OOM) on 80GB A100 GPU machine. Moreover, it is not possible to quantize 7B models on 24GB A10 GPU machines.

Model	**GPU Memory(GB)**
stablelm_3b	19.54
redpajama_3b	9.58
falcon_7b	23.64
llama2_7b	24.83
llama2_13b	40.46
falcon_40b and llama2_70b	OOM on single A100 80GB GPU

Table 2: *Distribution of GPU memory consumed by GPTQ 4-bits quantization for different models*

2.1.2. Layerwise Error induced by GPTQ

GPTQ 4-bit quantization reduces the size of a model by more than 80%, i.e., a model of 14 GB is reduced to around 2 GB post quantization. It is important to note that the quantization error introduced by GPTQ is different for different models, as shown in Table 3. This is because the models shown in Table 3 have different architectures, including the number of heads, number of layers, embedding dimension, number of query groups in multi-query attention, block size, and hidden dimension.

2.2. bitsandbytes Quantizations

bitsandbytes (bnb) provides implementation of five powerful and state-of-the-art quantization techniques namely i) int8, ii) fp4, iii) nf4, iv) fp4-dq[4], and v) nf4-dq. The int8 quantization procedure[14] uses vector-wise quantization with separate normalization constants for each inner product in the matrix multiplication. However, they have found around 0.1% dominant

[4]dq stands for double quantization

Model	mlp.proj	att.proj	mlp.fc	attn.attn
stablelm_3b	52850.7	12638.9	383200.9	844806.3
redpajama_3b	23448.1	1048.9	137061.9	138947.9
falcon_7b	19194.83	2362.39	149962.4	32886.3
llama2_7b	22773.0	3198.7	170837.6	248520.0
llama2_13b	27829.5	5470.5	247389.2	301002.0

Table 3: *Quantization error introduced by GPTQ in mlp projection, attention projection, fully connected and attention layers.*

activation outliers that has the potential to degrade the quality especially in bigger LLMs. Therefore, the precision for these dominant outliers are kept in float16. This scheme isolates the outlier feature dimensions into a 16-bit matrix multiplication, while still allowing more than 99.9% of the values to be multiplied in 8 bits.

QLoRA[12] introduced a new data type called 4-bit normalfloat (nf4), which is optimal for normally distributed weights, double quantization to reduce the memory footprint, and paged optimization to manage memory spikes. These techniques together yield excellent inference speed without sacrificing the quality of generation. In nf4 quantization, the base model weights are stored in nf4 data type and computation is performed in bfloat16. However, the model weights are dequantized to bfloat16 in the foward pass for inference [31]. The bnb quantizations compress the model footprint in the range of 40% (int8) to 70% (nf4-dq). It is important to emphasize here that int8 quantization for llama2_70B throws OOM error on A100 80GB GPU machine. Rest of the details of compressed model size corresponding to bnb quantizations are described in the following sections.

3. Experiment

This section provides a detailed description of the models, prompts, decoding approach, and related hyper-parameters used to generate the data for the analysis.

3.1. Model Description

A total of six pre-trained models with 3 billion to 70 billion parameters were selected for next-word prediction. These models are decoder-only, and their architecture-specific details are shown in Table 4. As can be seen, these models differ from each other in terms of the number of heads, number of layers, embedding dimension, number of query groups used in multi-query attention, sequence length, and intermediate size.

3.2. Prompts Selection and Proposed Hypothesis

Ten prompts are designed to access the quality and inference speed of pre-trained models for next word generation. These prompts are selected on the simple proposed hypothesis and shown below in Table 5.

Hypothesis 1: All pre-trained LLMs trained on billions or trillions of tokens can be ideally conceptualized as a large tree, where each node represents a topic and the text continuations associated with that topic. As we traverse down the tree, the text continuations become more specific and focused. Conversely, as we traverse up the tree, the text continuations become more general and abstract.

Hypothesis 2: The quality of a pre-trained model can be assessed based on its ability to accurately identify the correct topic node and then traverse to the sub-topic node for focused next word prediction.

3.3. Decoder Description

The current experiment uses a bare top_k sampling decoder without any additional features, such as repetition penalty. To assess the model's potential, we use a list of max new tokens, temperature as well as top_k. The max new tokens, temperature and top_k are [50, 100, 150, 200, 250, 300, 350, 400, 450, 500], [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0] and [1, 5, 10, 20, 50, 100, 200] respectively. The completion text is generated for every quantized models using all the combinations of max new tokens, temperature and top_k. The reason for high top_k such as 200 is that it might allow models to choose more diverse, less repetitive, and semantically coherent text.

4. Analysis

A total of 6300 (10 prompts $\times$ 10 max new tokens $\times$ 9 temperature $\times$ 7 top_k) completion text is generated for each quantized model except falcon_40b and llama2_70b for 16bit [5]. The evaluation of these completion texts is conducted through the computation of counting the duplicate content words, serving as a metric for assessing the quality of the generated text. The content words are the remaining words after removing the stop words. Additionally, the model's size in gigabytes (GB) serves as a key measure for quantifying GPU memory consumption, while tokens/sec is employed as a metric to gauge the model's inference speed.

4.1. Memory Consumption and Inference Speed

The utilization of int8 quantization demonstrates a significant reduction in memory consumption, approximately in the range of 40% to 50%, when compared to fp16, as illustrated in Table 6. Nonetheless, it is important to note that this enhancement is accompanied by a corresponding trade-off in inference speed, with int8 exhibiting a slowdown of roughly 75% to 80% in comparison to fp16, as indicated in Table 7.

When evaluating memory consumption between the fp4 and nf4 quantization approaches for model sizes up to 13 billion, their distinctions are negligible. However, nf4 quantization exhibits a slight advantage over fp4 in terms of memory consumption for larger models such as falcon_40b and llama2_70b. Nevertheless, fp4-dq is found to be better in memory consumption (i.e., takes less memory) across the models compared to its counterpart, as shown in Table 6. It is worth noting that while double quantization offers a clear advantage in memory consumption, it results in an inference speed reduction of approximately 10% to 25% compared to the absence of double quantization, as outlined in Table 7.

In conclusion, among the various quantization methods,

[5] Both falcon_40b and llama2_70b encounter OOM errors on an 80GB GPU machine. falcon_40b throws an OOM error when the max new tokens exceeds 400, while llama2_70b faces an OOM error during the loading process.

Model	n_head	n_layer	embed_dim	n_query_groups_mqa	seq_len	intermediate_size
stablelm_3b	32	16	4096	32	4096	16384
redpajama_3b	32	32	2560	32	2048	10240
falcon_7b	71	32	4544	1	2048	18176
llama2_7b	32	32	4096	32	4096	11008
llama2_13b	40	40	4096	32	5120	13824
falcon_40b	128	60	8192	8	2048	32768
llama2_70b	64	80	8192	8	5120	28672

Table 4: *Descrption of most relevant architectural specs of the pre-trained models used during the experiment*

Prompt	General Expected Continuation
Life in London	Travel/Cultural/Work-Related/London specific stuff
It is easy to be a techie	Comparison of techie with other probable roles in tech sector
Stock brokers are earning	Stock brokers and their earning style, sources, etc
It looks like written by Shakespeare	Shakespeare style text comparison
Hello, my name is	Chat or Introduction
Global warming and AI	Global Warming and AI in general as well as their +ive and -ive association
Current world order	Essay/Discussion/Power and Politics related tow world order
Percentage of people adore actors and singers	Stats on people following their favourite actors/singers and discussion on the related topic
Exercise and eating habits for	Eating habits and exercise routine in general (pros and cons)
Millennial and genz	Comparison and contrast between millennial and genz

Table 5: *Prompts Description*

fp16 stands out as the least efficient in terms of memory consumption. However, it excels in terms of inference speed, except in the case of stablelm_3b.

4.2. Temperature vs. Quality of Generation

A common pattern emerges within all quantization approaches, wherein an increase in temperature correlates with an elevation in number of duplicate content words except for fp16. However, it is worth noting that some models are more sensitive to even temperature lower than 0.5 compared to others.

When comparing the performance of stablelm_3b and redpajama_3b models, it becomes evident that the fp4 and nf4-dq quantization methods exhibit suboptimal results, characterized by an increased occurrence of duplicate words at lower temperature settings. However, the situation varies when considering falcon models, where nf4 quantization consistently demonstrates inferior performance across the entire temperature spectrum in comparison to other quantization methods.

In contrast, when assessing llama2 models, the situation becomes more nuanced, with most quantization approaches contributing significantly to repetitive generation. In this context, determining a clear front-runner among these methods proves to be a challenging task. Nevertheless, it is noteworthy that for the llama2_70b model, both fp4 and fp4-dq quantization methods outshine the others in terms of performance.

The analysis reveals that the int8-quantized model demonstrates effective control over the occurrence of duplicate content words for both llama2_13b and llama2_70b, effectively limiting them in the range of 40. In contrast, the fp16 models exhibit a characteristic of independence from temperature scaling, as they consistently generate a comparable number of repetitive words across all temperature settings except redpajama_3b.

4.3. Max Returned Tokens vs. Quality of Generation

The term max returned tokens encompasses the combined value of max new tokens and the length of the input prompt in terms of tokens. the analyis reveals that an the count of duplicate words generated linearly increases with the increase of max returned tokens across all models and quantization methods.

4.4. Top_k vs. Quality of Generation

The analysis offers a somewhat surprising insight, indicating that setting top_k equal to 1 tends to result in the lowest occurrence of duplicate words across models and quantization methods. Nonetheless, it's noteworthy that this effect reaches a point of saturation and loses distinctiveness when top_k is equal to or greater than 5.

4.5. Overall Comparison of Quality of Models

In terms of the average number of duplicate content words[6] generated in absolute terms, our analysis reveals the following insights:

- For fp4 and fp4-dq compared to nf4 and nf4-dq across various models (except llama2 series), there is a consistent reduction in repetitive generation, typically ranging from 12% to 20% relative.
- In the case of nf4 and nf4-dq for llama2 models of different sizes, there is a more noticeable advantage, with relative reduction of 9% to 11% in repetitive generation.
- Int8 quantization has a more pronounced limitation on the number of generated words, producing approximately 30-50% fewer content words than 4-bit quantization. Additionally, it produces 25-40% more duplicate content words relative to 4-bit quantization at normalized scale.
- When comparing fp16 with 4-bit quantization, it's noteworthy that fp16 generally produces more number of content words, often ranging from approximately 3% to 10%. Nonetheless, fp16 tends to generate a marginally higher du-

[6]The total number of content words generated for the unquantized model lies in the range of 1.34M to 1.45M and the maximum duplicate number of words is around 80K.

Model	bnb.nf4	bnb.nf4-dq	bnb.fp4	bnb.f4-dq	bnb.int8	fp16
stablelm_3b	3.22	3.20	3.22	3.06	4.68	7.42
redpajama_3b	2.31	2.17	2.31	2.17	3.52	5.60
falcon_7b	5.72	5.37	5.72	5.37	8.71	14.50
llama2_7b	4.58	4.27	4.58	4.27	7.82	13.53
llama2_13b	8.83	7.8	8.83	7.8	14.2	26.23
falcon_40b	26.40	24.64	26.55	24.64	44.52	80.85
llama2_70b	40.23	38.2	40.4	38.2	70.44	-

Table 6: *The distribution of memory consumed (lower is better) of all the models for different quantization.*

Model	bnb.nf4	bnb.nf4-dq	bnb.fp4	bnb.f4-dq	bnb.int8	fp16
stablelm_3b	**(37.76, 62.79)**	(38.7, 53.37)	**(42.99, 63.11)**	(38.7, 53.03)	(7.91, 16.81)	(37.76, 49.88)
redpajama_3b	(24.2, 32.29)	(22.59, 27.04)	(25.64, 31.37)	(15.49, 27.08)	(2.52, 3.24)	**(29.35, 37.85)**
falcon_7b	(29.09, 37.54)	(22.71, 30.04)	(24.77, 37.41)	(22.23, 30.7)	(3.13, 12.63)	**(35.79, 48.05)**
llama2_7b	(23.09, 29.88)	(19.32, 23.44)	(23.01, 28.65)	(17.85, 23.41)	(1.32, 8.87)	**(28.39, 36.35)**
llama2_13b	(15.9, 23.14)	(13.22, 18.84)	(12.0, 22.98)	(10.83, 18.22)	(6.49, 7.14)	**(24.12, 29.34)**
falcon_40b	(11.93, 16.59)	(11.57, 14.51)	(12.12, 16.61)	(10.42, 12.76)	(3.56, 4.63)	**(12.37, 13.99)**
llama2_70b	(8.67, 10.39)	(6.47, 9.07)	(8.52, 10.23)	(6.39, 8.82)	(2.79, 3.76)	-

Table 7: *The distribution of minimum and maximum inference speed (higher is better) in tokens/sec for different quantization.*

plicate words, indicating relative inferiority of 1% to 3.5% with 4bit quantization.

The computation of average perplexity scores, with a token stride of 512, is conducted for all quantization levels across each model. An examination of these scores reveals that the perplexity values for all models reside within a relatively constrained range, typically ranging from 12 to 15. Consequently, it is discerned that perplexity, within this context, may not serve as a suitable metric for assessing the quality of the generated text.

5. Conclusions

In scenarios where GPU memory is not a limiting factor and the utmost priority is placed on achieving both high inference speed and accuracy, it is advisable to prioritize the utilization of fp16 for models up to 7 billions. This preference arises due to its reduced susceptibility to variations in temperature and max new tokens. Moreover, model upto 7 billion size effectively fits into a consumer grade GPU machine. Alternatively, nf4 and fp4 serves as the default choice for individuals seeking a balance between GPU utilization, accuracy and inference speed, thus offering a middle-ground solution that combines all aspects effectively.

It's worth noting that the adoption of double quantization, such as fp4-dq and nf4-dq, can lead to a marginal reduction in memory footprint. However, it is accompanied by a relatively decreased inference speed. Hence, the recommendation leans toward using quantization without the doubling approach. Additionally, when considering the nf4 and fp4 precision combination, it is advisable to use temperature ≤ 0.5 to ensure optimal performance.

The current evaluation does not consider int8 to be a feasible alternative to other quantization methods. While int8 reduces memory usage, it significantly slows down inference and produces around 30-50% fewer words than other quantization methods.

It is important to note that the current experiment did not achieve satisfactory results in terms of accuracy and inference speed when using gptq 4-bit quantization. Further investigation is needed to replicate the comparable performance that has been reported in other studies[7]. Therefore, this result is not included in the analysis presented.

6. Limitations and Future Work

The current study is conducted on 7 models ranging in size from 3 billion to 70 billion parameters, and 10 prompts are used for next-word prediction using various combinations of hyperparameters. Further study with more models (at least 7 billion parameters) and prompts may provide more insights into the effects of these hyperparameters on quantized LLMs.

Future work will focus on the primary causes of repetitive generation and their relationship to Hypothesis 1 and Hypothesis 2.

7. References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[3] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[4] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti *et al.*, "Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model," *arXiv preprint arXiv:2201.11990*, 2022.

[5] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso *et al.*, "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models," *arXiv preprint arXiv:2206.04615*, 2022.

[7] https://github.com/PanQiWei/AutoGPTQ

[6] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay, "The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only," *arXiv preprint arXiv:2306.01116*, 2023.

[7] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[8] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[9] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper, "Accelerating large language model decoding with speculative sampling," *arXiv preprint arXiv:2302.01318*, 2023.

[10] G. Yang, D. Lo, R. Mullins, and Y. Zhao, "Dynamic stashing quantization for efficient transformer training," *arXiv preprint arXiv:2303.05295*, 2023.

[11] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra, "Llm-qat: Data-free quantization aware training for large language models," *arXiv preprint arXiv:2305.17888*, 2023.

[12] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023.

[13] S. J. Kwon, J. Kim, J. Bae, K. M. Yoo, J.-H. Kim, B. Park, B. Kim, J.-W. Ha, N. Sung, and D. Lee, "Alphatuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models," *arXiv preprint arXiv:2210.03858*, 2022.

[14] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm. int8 (): 8-bit matrix multiplication for transformers at scale," *arXiv preprint arXiv:2208.07339*, 2022.

[15] 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.

[16] Z. Yuan, L. Niu, J. Liu, W. Liu, X. Wang, Y. Shang, G. Sun, Q. Wu, J. Wu, and B. Wu, "Rptq: Reorder-based post-training quantization for large language models," *arXiv preprint arXiv:2304.01089*, 2023.

[17] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," *arXiv preprint arXiv:2306.00978*, 2023.

[18] Z. Yao, C. Li, X. Wu, S. Youn, and Y. He, "A comprehensive study on post-training quantization for large language models," *arXiv preprint arXiv:2303.08302*, 2023.

[19] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[20] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," 1993.

[21] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[22] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8815–8821.

[23] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[24] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "Binarybert: Pushing the limit of bert quantization," *arXiv preprint arXiv:2012.15701*, 2020.

[25] W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu, "Ternarybert: Distillation-aware ultra-low bit bert," *arXiv preprint arXiv:2009.12812*, 2020.

[26] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in *International conference on machine learning*. PMLR, 2021, pp. 5506–5518.

[27] Z. Yao, C. Li, X. Wu, S. Youn, and Y. He, "A comprehensive study on post-training quantization for large language models," *arXiv preprint arXiv:2303.08302*, 2023.

[28] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[29] X. Wu, C. Li, R. Y. Aminabadi, Z. Yao, and Y. He, "Understanding int4 quantization for transformer models: Latency speedup, composability, and failure cases," *arXiv preprint arXiv:2301.12017*, 2023.

[30] E. Frantar and D. Alistarh, "Optimal brain compression: A framework for accurate post-training quantization and pruning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 4475–4488, 2022.

[31] Y. Belkada, T. Dettmers, A. Pagnoni, S. Gugger, and S. Mangrulkar, "Making llms even more accessible with bitsandbytes, 4-bit quantization and qlora," 2023. [Online]. Available: https://huggingface.co/blog/4bit-transformers-bitsandbytes