Exploring Methods for Efficient Large Language Model Inference

John Wang
Department of Computer Science
Stanford University
jwang003@stanford.edu

Aaron Wan
Department of Computer Science
Stanford University
aaronwan@stanford.edu

1 Introduction

Large Language Models (LLMs) have displayed remarkable capabilities in text generation, complex reasoning, and decision-making [1][2]. Advances in deep learning architectures, particularly the Transformer model introduced by Vaswani et al. [3], are the backbone of these large models. Furthermore, the performance of an LLM is limited by the model size, training set size, and compute [4].

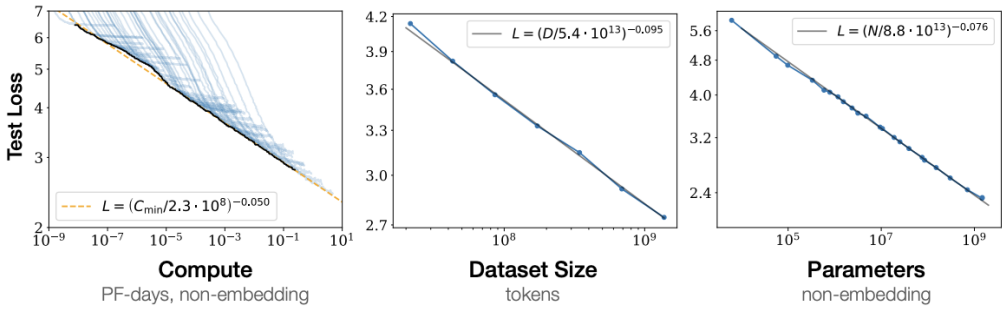

Figure 1: LLM performance displays a power-law relationship between compute, dataset size, and model size [4]

In recent years, the size of LLMs has grown by several orders of magnitude, requiring more compute, time, and space for training and inference. Therefore, addressing the scalability of LLMs is an important challenge in fully realizing the potential of LLMs for beneficial social impact. While the transformer has proven to be the best known architecture for language model performance, several components can lead to slow text generation and large memory footprint [5]. In this project, we aim to improve the latency and memory usage of LLMs during inference by employing several known techniques for efficient LLM inference.

We focus on implementing the following efficiencies for LLM inference on GPT-2 (small) [6]:

- **8 Bit Quantization:** Quantization is a popular method for reducing model size by reducing the precision of the weights in the model [7]. Originally proposed with neural networks for visual tasks, quantization of LLM weights from floating-point 32 (fp32) to 8 bit integers (int8) has shown great success in decreasing model footprint and latency while preserving accuracy [8]. Many quantization methods exist, but most rely on some scaling function $Q(v) = \text{round}\left(\frac{v}{s}\right)$, where v represents the original parameter values and s the scaling factor for a weight matrix.
- **Speculative Decoding:** In speculative decoding, a LLM typically leverages a smaller draft model to predict x tokens into the future. If the tokens that are "speculated" by the draft model match the softmax probabilities of the main model, the LLM essentially decodes x tokens in the time usually taken for a single token [9]. the number of tokens to speculatively decode typically ranges from 2 to 10.

- **Iterative Magnitude-based Pruning:** We experiment with two magnitude-based pruning strategies:
  - **Absolute Value Pruning:** For absolute value pruning, we fine-tune the full model for a fixed number of iterations, then we prune a certain percentage of weights with the lowest magnitudes, and the resulting model is further fine-tuned on the dataset.
  - **L2 Norm Pruning:** Here, we follow a similar process to absolute value pruning, except in the pruning step we prune entire rows with the smallest L2 norm instead of individual weights with the lowest magnitude.

We aim to implement these methods for reducing latency and memory usage while maintaining equivalent quality in GPT-2 (small) inference. The project consists of three parts. In the first part, we explore the effects of quantization and speculative decoding on perplexity, memory usage, and latency. In the second part, we explore the effects of pruning on output quality. In the third part, we experiment with different quantization techniques to find the smallest model while maintaining similar quality.

2 Background and Methods

For each part, we define LLM inference quality by perplexity, or the exponential of the cross entropy loss. Given the cross entropy loss for decoding token x_{t+1}

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} -log(\mathbf{y}^{(t)}_{x_{t+1}})$$

where $\mathbf{y}^{(t)}_{x_{t+1}}$ is the softmax probability decoding the correct token x_{t+1}. Then, perplexity is defined by

$$PPL(\theta) = exp(J(\theta))$$

The dataset we used to evaluate our LLM was wikitext-103, a dataset of over 100 million tokens extracted from good and verified Wikipedia pages [1]. Wikitext-103 was a popular benchmark for LLMs such as GPT-2, where reported metrics are readily available. For evaluation, we calculate perplexity on the validation set of wikitext-103 over 100 iterations.

2.1 Part 1: Quantization and Speculative Decoding

In part 1, we explore the effect of 8-bit quantization on perplexity. Namely, we implement post-training absmax quantization of the 12 transformer layers in GPT-2 (small). In 8-bit absmax quantization, the weights of a specific projection are scaled to $[-127, 127]$ by calculating a linear scale factor based on the weight with the largest magnitude and rounding the scale factors to integers. For each weight matrix X, absmax quantization operates via the following formulas:

$$\begin{aligned} \text{scale} &= \frac{127}{\max|X|} \\ X_{\text{quantized}} &= \text{scale} * X \\ X_{\text{de-quantized}} &= \frac{X}{\text{scale}} \end{aligned}$$

Outside of the provided project requirements, we also experiment with zero point quantization. Zero point quantization is another simple method for quantization, where the difference between the minimum and maximum weights are scaled to 255, and the midpoint is centered to 0 to achieve 8-bit quantization [2]. This method may generate a more representative mapping of the raw weights to 8 bits because the scale is calculated with the range of the weights in mind.

[1] Wikitext: https://blog.salesforceairesearch.com/the-wikitext-long-term-dependency-language-modeling-dataset/

[2] https://towardsdatascience.com/introduction-to-weight-quantization-2494701b9c0c

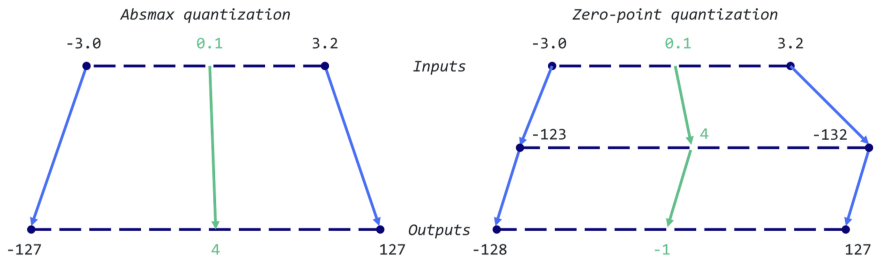

Figure 2: Absmax and zero point quantization are two simple methods to map fp32 weights to 8 bits. In zero point quantization, the scale relies on the range of weights instead of the weight with largest magnitude

As a benchmark result, we calculate the perplexity and memory usage of un-quantized GPT-2 (small) with batch sizes 4 and 12. Then, all parameters in the 12 transformer layers are quantized and the same evaluation is run. In order to satisfy PyTorch operations, which naturally occur in fp32, we de-quantize each transformer layer immediately prior to the forward pass and re-quantize the model immediately afterward. With this approach, we primarily decrease the memory usage needed for inference.

Furthermore, we explore the effect of absmax quantized models on speculative decoding, a method for faster inference via predicting batches of tokens via a draft model. In speculative decoding, generates several tokens into the future. If the main model samples the same tokens as the draft model, it is able to decode as many correct tokens as the draft model provides until a disagreement in approximately the same time as one token decoding.

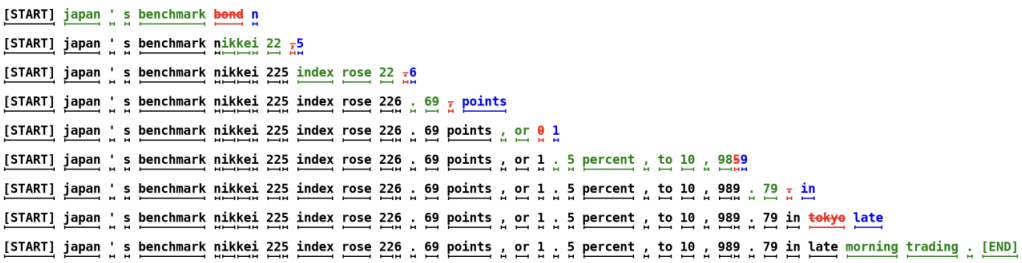

Figure 3: Speculative decoding leverages a draft model to predict several tokens in advance. The green tokens are predictions by the draft model which are accepted by the main model, while the red token indicates a disagreement. Speculative decoding can provide “free" tokens for the main LLM; in step one, four “free" tokens are generated in one forward pass of the main model [9].

We measure the effect of two speculative decoding setups and evaluate the latency of each when generating 50 new tokens from a prompt of length 1024. In both setups, the draft model predicts 4 speculative tokens with each call. In the first set up, we use GPT-2 (medium) as the main model and GPT-2 (small) as the draft model. In the second, we use GPT-2 (small) as the main model and our 8-bit quantized GPT-2 (small) as the draft model. For each setup, we compare the latency against standard decoding with the main model.

2.2 Part 2: Iterative Magnitude-based Pruning

In part 2, we investigate the impact of iterative magnitude-based pruning on inference latency and model quality for GPT2 (small). To prune the model, we zero out the pruned parameters and their corresponding gradients, thereby increasing the sparsity of the model in hopes of increasing its inference efficiency. For each of our two pruning approaches (absolute value and L2 norm), we explore two strategies for fine-tuning the pruned model:

- **Separate Pruning** In this strategy, we first fine-tune the full model on the training split of the wikitext-103 dataset for 100 iterations. Then, we prune 10% of the model using the respective pruning approach (absolute value or L2 norm), and then we fine-tune the pruned model for another 100 iterations. This process of pruning and fine-tuning is repeated until the number of pruned parameters is 90% of the original model size.
- **Simultaneous Pruning:** As with the first strategy, we first fine-tune the full model on the training split of the wikitext-103 dataset. Then, we fine-tune the model for another 100 iterations while simultaneously pruning the model down by 0.1% per iteration, resulting in 10% of the model being pruned at the end of the 100 iterations. This process of simultaneously pruning and fine-tuning is repeated until we reach 90% of the original model size.

In all our approaches, we prune locally within each layer, meaning that for each stage of pruning we compare either the absolute value of the parameters or the L2 norms of the rows within each layer to decide what to prune for that layer.

At every 10% increment of pruning, we measure the inference latency by computing the execution time per batch during inference as well as the model quality by computing the model loss on the validation set. The inference tokens per second and validation loss of the full model after being fine-tuned on the dataset for 100 iterations serve as our benchmarks for model performance after pruning. For all experiments, we use batch size 8, block size 1024, and gradient accumulation 40.

For L2 pruning, we achieved poor initial results when pruning the entire model (validation loss 9+ after pruning just 10%), and we achieved better results when skipping over the embedding layers during the pruning process. Ultimately, we decided to focus on pruning non-embedding layers for our L2 pruning experiments, and we pruned each non-embedding layer all the way down to 10% of the original layer size instead of the entire model down to 10% of the model size.

2.3 Part 3: Leaderboard

For the Leaderboard component of the project, **we focus on optimizing the inference-time memory usage** of the model while maintaining model quality. To this end, we experimented with the following techniques for the leaderboard submission:

- **Absolute Value Pruning:** We experiment with pruning to introduce sparsity into our model in hopes of reducing the memory footprint. Per the Leaderboard requirements, we fine-tuned the GPT-2 (small) model for 500 iterations on the wikitext dataset. During each iteration of this fine-tuning process, we used absolute value pruning to prune 0.05% of the model at each iteration, resulting in 25% of the model being pruned at the end of training.
- **Absmax Quantization:** After fine-tuning the GPT-2 (small) model on the wikitext dataset for 500 iterations, we implement absmax quantization on the model, as described in 2.1. Since absmax quantization compresses the parameters into smaller, 8-bit units, we expect this will be beneficial for reducing the memory footprint.
- **Zero Point Quantization:** We also experiment with zero point quantization on the fine-tuned GPT-2 (small) model, as described in 2.1. Since this method represents the parameters with fewer bits centered around a zero point, we expect this to decrease memory usage.

All fine-tuning on the wikitext dataset was conducted with 500 iterations, batch size 8, sequence length 1024, and gradient accumulation of 40. We evaluate the memory footprint by observing the maximum memory allocated from conducting inference with batch size 1 on the validation of wikitext. We also record the average validation loss of each experiment on the wikitext dataset to measure model quality.

3 Results

3.1 Part 1: Quantization and Speculative Decoding

We observe the following increase in perplexity and decrease in memory usage when implementing the two 8-bit quantization methods on GPT-2 (small):

	Metrics	GPT-2 (small)	GPT-2 (small, 8-bit)	GPT-2 (small, zero point)
B=4	Perplexity	**22.699**	26.093	24.630
	Memory Usage (GB)	4.606	3.010	**3.008**
B=12	Perplexity	**22.628**	26.105	24.647
	Memory Usage (GB)	9.494	7.995	**7.993**

Table 1: Perplexity and Memory Usage for GPT-2 (small), the absmax, and the zero point quantized models

We observe an increase in perplexity of about 4.5 and a corresponding decrease in memory usage by about 1.5GB when testing the quantized model on wikitext-103 validation. Additionally, we observe that the decrease in memory usage seems to be fixed around 1.5GB instead of a proportional decrease.

For zero point quantization, we observe similar improvements in memory usage to absmax quantization. However, with zero point quantization, model quality is better maintained, as the perplexity only increases by around 2.

For our speculative decoding experiments, we report the results in Table 2 below. We observe increased latency with both speculative decoding systems. We observe 1.82x slower inference when using GPT-2 (small) as a draft model for FPT-2 (medium), while we observe 1.58x slower inference when using GPT-2 (small, quantized) as a draft model fro GPT-2 (small).

Metric	GPT-2 (medium)	GPT-2 (small)	GPT-2 (M+S)	GPT-2 (S + Quantized)
Latency (s)	**13.833**	**7.254**	25.199	11.490

Table 2: Speculative decoding latency compared to standard decoding with the main model. We surprisingly observe greater latency for both speculative decoding setups.

3.2 Part 2: Iterative Magnitude-based Pruning

3.2.1 Absolute Value Pruning

The results for absolute value pruning on GPT2 (small) can be seen in Figure 4. As the model was pruned down to 10% of the original model, we observe an increase in loss from 3.37 to 8.04 for the Separate strategy and 3.29 to 7.10 for the Simultaneous strategy.

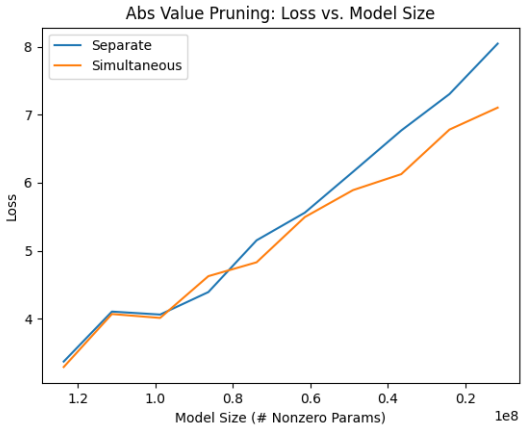

Figure 4: Absolute Value Pruning Results

3.2.2 L2 Norm Pruning

The results for L2 norm pruning on GPT2 (small) can be seen in Figure 5. For L2 norm pruning, we were able to prune to 38% of the original model size, as we applied pruning to all but the embedding layers. For the Separate strategy, the loss increased from 3.43 to 12.7 from start to finish, while the loss increased from 3.37 to 11.35 for the Simultaneous strategy. For inference, the execution time per batch decreased by an average of roughly 12.8% across the two strategies from start to finish.

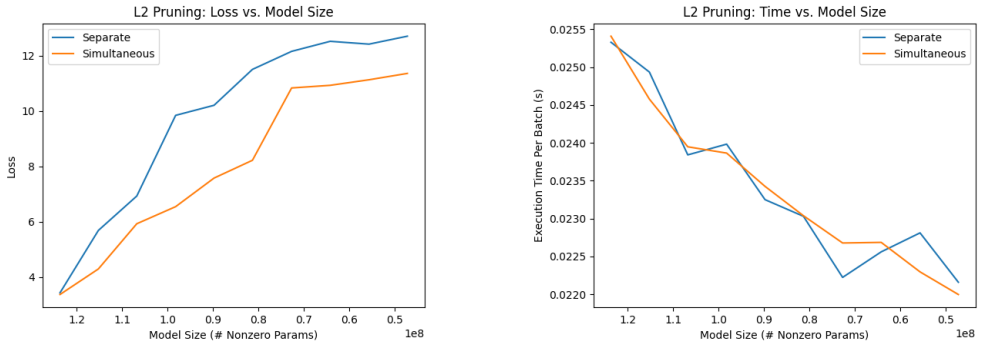

Figure 5: L2 Pruning Results

3.3 Part 3: Leaderboard

From the Leaderboard results in Table 3, we see that the pruning approach failed to reduce memory usage, while the two quantization methods both reduced memory usage by over 52.26%. Like in our results in 3.1, we see that zero point quantization was more effective at maintaining model quality than 8-bit quantization.

Metric	Baseline (GPT-2 Small)	Abs Val Pruning	Absmax	Zero Point
Memory Usage (GB)	2.836	2.836	**1.344**	**1.344**
Loss	3.094	3.591	3.269	**3.185**

Table 3: Results for Leaderboard Experiments

4 Discussion

4.1 Quantization and Speculative Decoding

The goal of quantization is to reduce model size with minimal tradeoff in performance due to lower-precision weights. To this end, we observe results that confirm this expectation. When comparing perplexity and memory usage of GPT-2 (small) with and without absmax quantization, we observe a 4.5 increase in perplexity for a drop of 1.5GB in memory required for inference. Moreover, compared to absmax quantization, zero point quantization provides nearly identically memory improvements while better maintaining model quality. Since the scale for zero point quantization is calculated with the range of the weights in mind, this may create a more representative mapping of the weights. However, we also experience slower inference with quantization despite lower memory usage. We hypothesize that this is because we implement post-training quantization on the fly, where every weight in the transformer layer is de-quantized to fp32 immediately before being used in the forward pass, and re-quantized after being used. This added computation before and after each transformer layer in the model leads to greater latency compared to the standard model.

Furthermore, other complex methods exist to quantize weights. A seminal method is described in `llm.int8()`, where large magnitude features (outliers) are treated with fp16 precision, and more typical weights are quantized with absmax quantization and multiplied in int8 before translation to floating point [8].

With speculative decoding, we observe that both setups with a draft model exhibit greater latency that standard decoding. We hypothesize that both setups fail because of two separate reasons: discrepancy between pretrained and finetuned models and the added compute time of quantized models. When using GPT-2 (small) as the draft model for GPT-2 (medium), we experience 82% slower inference. This is because the main model, GPT-2 (medium), was pre-trained and downloaded from Huggingface with no finetuning on wikitext-103. Due to compute limitations, we were limited to 1 T4 GPU and were not able to finetune GPT-2 (medium). However, because we use GPT-2 (small), finetuned on wikitext-103, we observe a large discrepancy between the datasets both models were trained on. Since speculative decoding relies on the models to agree to a degree that outweighs the added latency of the draft model, having two models in this case results in minimal agreement and increased latency for generating tokens. On the other hand, we observe that using the quantized version of GPT-2 (small) as a draft model also increases the latency compared to standard decoding with GPT-2 (small). This is because we have shown that the quantized model has greater latency for generating tokens in the first place, even if it is smaller than GPT-2 (small). Because of this, even if there is 100% agreement between the two models, the greater latency of quantized GPT-2 (small) will lead to slower generation compared to standard decoding with GPT-2 (small).

4.2 Pruning

The purpose of pruning is to reduce model size and increase inference speed while maintaining model quality. The results of our experiments with respect to this objective are mixed. We do observe a modest speed up, as Figure 4 shows us that execution time per batch decreases by $\approx 12.8\%$ as we L2 prune to 38% of the original model size. However, model quality does not appear to be effectively sustained. Absolute value pruning appears to maintain reasonable model quality (validation loss ≈ 4) up until over 20% of the model is pruned. On the other hand, with L2 pruning, we observe a sharper decline model quality, despite the fact that we do not prune the embedding layers. The sharper decline in quality can be attributed to the fact that L2 pruning has a greater impact on the model structure compared to absolute value pruning, as L2 pruning involves pruning entire rows as opposed to individual weights.

An important observation is that based on the results for both absolute value pruning and L2 norm pruning, the "Simultaneous" pruning strategy is more effective for maintaining model quality than the "Separate" pruning strategy. Pruning a small amount of the model at each fine-tuning iteration is more effective than pruning a larger chunk of the model in between fine-tuning steps. While we were limited to only 100 fine-tuning iterations per pruning stage in this project, these results indicate that spreading out the pruning across a larger number of fine-tuning iterations may be an effective strategy to maintaining model quality. In fact, our Leaderboard experiments reveal that when we prune at 0.05% per iteration instead of 0.1%, we observe empirically better model quality.

5 Future work

As discussed above, many of the limitations in this project were due to compute constraints and time for manual implementation. To truly see the value of quantization, additional methods such as quantization aware training and `llm.int8()` should be explored. The functional downside of post-training quantization with on-the-fly de-quantizing and re-quantizing necessitates methods where we can perform matrix multiplications with lower precision. Additionally, other areas of research include experimenting with different types of precision, such as 8 bit floating point or even int4 [10]. For pruning, we want to investigate pruning beyond 100 fine-tuning iterations to better maintain model quality, as longer pruning periods showed better results in our Leaderboard tests.

Other methods for efficient inference of LLMs can also be explored. In this project, we focus on quantization, speculative decoding, and pruning. Further methods like KV Caching and Mixture of Experts (MoE) have also shown to improve LLM inference latency through eliminating repeat memory reads and introducing sparse model routing instead of a dense decoder [11]. Given more time, it would also be interesting to see how those methods affect latency and memory usage. While we have previously implemented KV caching, we chose not to incorporate it into this project because of our focus on reducing memory usage. In conclusion, the array of available optimizations for LLM inference is diverse and vast; as LLMs scale with data, these methods will grow all the more important for scaling compute and model size as well.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[2] OpenAI. Gpt-4 technical report, 2023.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[4] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[5] Quentin Fournier, Gaétan Marceau Caron, and Daniel Aloise. A practical survey on faster and lighter transformers. *ACM Computing Surveys*, 55(14s):1–40, July 2023.

[6] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[7] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations, 2016.

[8] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

[9] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.

[10] Sergio P. Perez, Yan Zhang, James Briggs, Charlie Blake, Josh Levy-Kramer, Paul Balanca, Carlo Luschi, Stephen Barlow, and Andrew William Fitzgibbon. Training and inference of large language models using 8-bit floating point, 2023.

[11] William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning, 2022.