# Accelerating Large Language Model by partial head quantization

by

Tuo Yang

BASc., The University of British Columbia, 2024

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHEOR OF APPLIED SCIENCE

in

The Faculty of Applied Science

(Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April 24 2024

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Accelerating Large Language Model by partial head quantization

submitted by Tuo Yang in partial fulfillment of the requirements for

the degree of Bachelor of Applied Science in Faculty of Applied Science

**Examining Committee:**

Co-supervisor: Prashant Nair

Co-supervisor: Muhammad Abdullah Adnan

Supervisory Committee Member:

Additional Examiner:

Additional Supervisory Committee Members:

Supervisory Committee Member:

# Abstract

Large Language Models (LLM) have been widely used in Natural Language Processing (NLP) applications. However, there is a great need for a general computing platform to perform inference models of LLMs with higher speeds and smaller model sizes. Because it is difficult for CPUs or embedded processors to perform the data-intensified computation for attention models. On the other hand, the accuracy should be maintained after the acceleration of the inference models.

Considering this trade-off, we propose a method to partially quantize the attention heads according to head importance. We localize less significant attention heads in the sentence and then quantize all the weights and activations in both the attention layers and the fully connected layers of the model related to this head. By the method we propose, we can keep the accuracy as we only quantize those less important attention heads, and we can quantize both weights and activations to reduce the attention computation and KV cache usage.

We performed experiments on the OPT-1.3B model due to limited hardware support.

# Lay Summary

In this thesis, we are inspired by the observation that some of the heads are less important than others due to the superfluity of human language. We sum up the head importance across layers and sort the head values to identify the less significant heads. Then, we pass the head indices back to quantize the selected heads and corresponding weights and activations. Importantly, outliers are the dominant factor that reduces the accuracy of activation quantization in the attention model. Therefore, we transfer quantization difficulty from activations to weights because weights between channels have fewer outliers and are easier to quantize. In this way, we can reduce the KV cache size and memory bandwidth usage while maintaining model accuracy. We evaluate the method on the OPT-1.3B model and verify that we can maintain the accuracy.

# Table of Contents

# List of Figures

# List of Programs

# Chapter 1

# Introduction

Natural Language Processing (NLP) has emerged tremendous potential in various applications [4], such as machine translation, text classification, and text summary, since the advent of attention mechanism [6]. However, due to the massive amount of parameters, it has been an issue to reduce memory usage and latency while maintaining accuracy when deploying the model to an embedded processor or a general computing platform. For instance, models of size ranging from at least 350 GB for GPT-3 175B in floating point 16 bits [1] to at least 1050 GB for MT-NLG 530 GB in floating point 16 bits [10] are too large that they require 5x to 15x A100 GPUs with 80 GB memory.

In this thesis, we propose an algorithm to efficiently perform attention model inference while maintaining accuracy. We propose to combine two algorithms. SpAtten [8] introduces an algorithm-architecture co-design using cascade head pruning to reduce memory usage and latency. We use the head importance sorting algorithm to identify less important heads. SmoothQuant [10] describes an algorithm that solves the issue of activation quantization. We choose this algorithm to quantize both weights and activations to preserve the accuracy of the model while reducing memory usage and latency.

Since attention models use multiple heads to determine the dependencies at different dimensions and some heads have less influence on the outputs [7], we can prune those heads and all the corresponding weights and activations. Also, cascade means that once a head is quantized at a certain layer, the same head at all the deeper layers will be pruned. But we will do quantization instead of pruning utilizing the algorithm SpAtten [8] uses to identify the less significant heads. The sorting of the head importance will use QuickSort [5]. The sorting process is performed in software instead of the hardware architecture that SpAtten [8] uses due to hardware support consideration but is sufficient for model accuracy verification. More specifically, we calculate the head importance by accumulating the outputs of the self-attention layer for each head. A sorting algorithm will be applied to these summed values for each head. Finally, heads that have less influence

on the outputs are identified.

The head indices will be passed to Q, K, V, and output matrices to slice the weight matrices to quantize. We select heads to quantize when the model calculates activation matrices. However, when the model size rises to more than 6.7 billion parameters [3], there will be outliers in the activation matrix that exacerbate degraded accuracy. Other quantization methods such as ZeroQuant [11], applying dynamic activation quantization and group-wise weight quantization, and LLM.int8() [2], introducing mixed precision that has outliers in FP16 and inliers in INT8, either cannot maintain its accuracy when the model is very large, or not hardware friendly. Therefore, we choose the SmoothQuant algorithm to smooth the outliers by migrating the outliers from activations to weights. It is based on the fact that different tokens tend to have the same pattern throughout the channel. Hence, we can do a scale transformation to the entire channel of an activation matrix and then do an inverse scale transformation to the same channel of the weight matrix. In this way, the product of the activation and weight matrix should be kept mathematically equivalent [10].

In summary, this thesis is a proof of concept to the algorithm that combines the two algorithms above. We are only able to use an OPT 1.3B model to verify that there is an accuracy improvement compared to only using the SmoothQuant to do quantization when predicting the last word of the inputs.

# Chapter 2

# Background and Motivation

## 2.1 Attention Mechanism

For this thesis, we only evaluated the algorithm on OPT-1.3B. OPT stands for Open Pretrained Transformer. It belongs to the family of decoder-only models like GPT-3. Attention-based NLP tasks consist of discriminative tasks and generative tasks. OPT models handle generative tasks with summarization and generation stages. The summarization stage is processing the whole sentence at the same time. OPT generates one more new token at the end of the summarization stage and enters the generation stage. In the generation stage, the model processes one token at a time. Also, it concatenates all the K (key) and V (value) matrices from the summarization stage to the current K and V values. In this way, it can provide dependencies when generating new tokens.

The attention mechanism [6] improves the speed of translating one sentence to the other. It captures dependencies between tokens in the long or short term with multiple heads. The attention mechanism first embeds the input tokens into vectors. They are then processed by a linear layer to generate Q (query), K (key), and V (value) matrices for each token. We will get an attention score by calculating $attention\_score = Q * K^T / sqrt(D_k)$ where $D_k$ is the dimension of the key vector. After this, we take a softmax function to get the attention probability of the token. It indicates how likely each word contributes to this token. At the end, the attention probability is multiplied by the V matrix. Then we get the attention output of the current head. We concatenate all the attention heads to get the final outputs of the self-attention module. For OPT-1.3B, it has 32 heads for each layer and a total of 24 layers.

## 2.2 Quantization

The quantization method used in this thesis is scaled quantization, which can be written as [9]

$$s = \frac{(2^{b-1} - 1)}{\alpha} \tag{2.1}$$

s is the scale factor of the quantization equation. b is the number of bits to quantize. If we are quantizing the number to INT8, b equals 8. $\alpha$ is the maximum absolute value of the numbers in the range that are quantized. It uses maximum absolute value to calculate $\alpha$ such that it preserves the outliers in activation, which is important for maintaining the accuracy [2].

$$x_q = quantize(x, s) = round(x * s) \tag{2.2}$$

$$x_d = dequantize(x_q, s) = \frac{1}{s} * x_q \tag{2.3}$$

It is a symmetric quantization method which means we assume that the tensor is symmetric around 0 for simplicity.

## 2.3 Motivation

### 2.3.1 Motivation for quantization

SpAtten [8] has profiled the end-to-end latency of a GPT-2 model on various CPU and GPU platforms. It has shown that CPU and GPU platforms are optimized on matrix multiplication but are poor at performing complex memory operations like splitting heads, Ks, and Vs concatenations, reshape, and transpose. As a consequence, around 73% of time is spent on data movements. Therefore, it would be a huge performance improvement if we could accelerate this process. Since the fully connected layers are linear algebra multiplications and CPUs and GPUs are already optimized for matrix multiplications, we can focus on accelerating the attention layers and quantize the weights and activations to improve the latency.

### 2.3.2 Motivation for activation quantization

Downstreaming tasks of LLMs, such as chatbots, might require storing long context of Ks and Vs in the generation stage. Because the generation stage needs to concatenate all the Ks and Vs from the summarization stage. In this case, there would be massive memory access to the KV cache and the

memory usage of the KV cache can no longer be neglected [10]. Activation quantization will provide a solution to the memory cost and latency degradation caused by long context stored in the KV cache.

# Chapter 3

# Partial Quantization algorithm

In each of the Q, K, and V matrices, multiple heads contribute to different dimensions of dependencies. But in those heads, some of them have negligible influence on the final outputs. Head importance is calculated by accumulating the absolute value of the attention output of each head among all the layers. A larger magnitude of the attention output of the head will have more influence on the final output after the fully connected layer that concatenates the attention outputs of all the heads [8]. Also according to the cascade strategy, once we quantize a head, we will quantize this head in all the following layers.

## 3.1 Head Importance Difference

As demonstrated in the figure 3.1, there are differences between heads for each layer. When the layers are getting deeper, we can observe that the accumulated head importance between heads is becoming more diverged. Therefore, we can quantize those less significant heads as they appear to have less influence on the final results.

## 3.2 Code to Accumulate Head Importance

The pseudo-code to accumulate the head importance is exhibited in the program 3.1. We extract the *attention_output* of each head from the inference outputs of the first batch that the model runs. For each layer, we sum up the values from all the head dimensions and sequence lengths for each head. Then the algorithm adds the current layer's head importance list to the previous layer's head importance list to accumulate the head importance. Next, we sort the current accumulated head importance to get the less $top - k$ heads. We concatenate those heads to the less significant

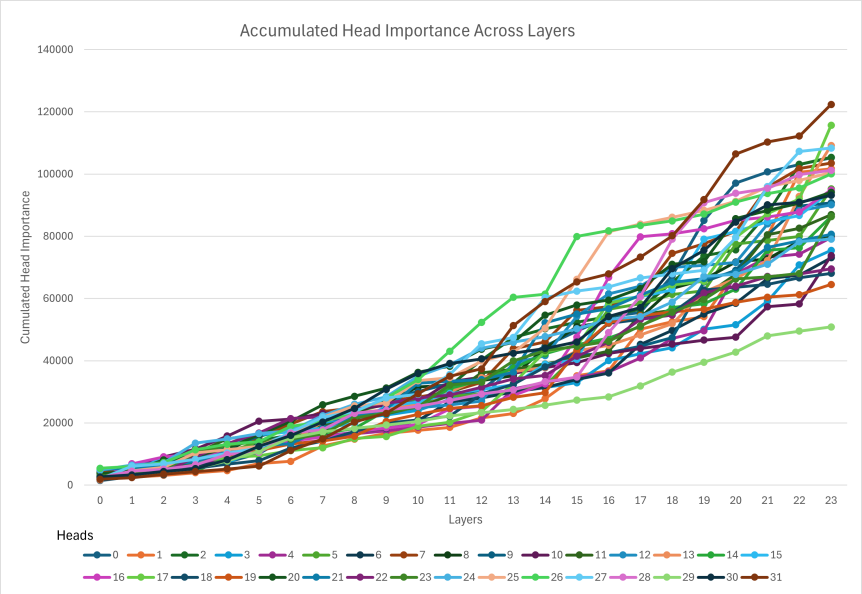


Figure 3.1: This is a figure that shows how head importance is accumulated when an OPT-1.3B is inferring some data from the lambda dataset

heads from the previous layers and store them as the heads to quantize for the current layer because we need to do cascade head quantization.

**Program 3.1** Head Quantization

```
input: num_heads, model, dataset, top_k
# top_k is the number of heads to quantize at each layer
outputs = model(input_tokens)
multihead_sum_array_across_layers = zeros(1, num_heads)
sum_head_indices_across_layer = []
    for layer in layers of the model:
        head_matrix = outputs.attention_output[layer]
        # head_matrix is in the size of 1 * 32 * 512 * 64
        which are batch_size, num_heads, sequence length,
        and head dimension respectively
        multihead_sum_array =
                            sum(abs(head_matrix), (3, 2))
        # sum the absolute value of the fourth and third
        dimension of the head matrix. The result
        multihead_sum_array is in the size of (1, 32)
        multihead_sum_array_across_layers +=
                                    multihead_sum_array
        _, sorted_indices =
            quick_sort(multihead_sum_array_across_layers)
        sum_head_indices_across_layer.append(
                            sorted_indices[0][0:top_k])
        sum_head_indices_across_layer[layer] =
        sum_head_indices_across_layer[layer - 1]
        .cat(sum_head_indices_across_layer[layer])
    model = quantize_model_less_important_heads(model,
                            sum_head_indices_across_layer)
```

# Chapter 4

# Evaluation

## 4.1 Evaluation methodology

We implemented the partial quantization algorithm in the environment of the SmoothQuant [10] repository. It is an evaluation by predicting the last word of the inputs in PyTorch. I ran the evaluations on my laptop with an 8GB GPU. The dataset we use is lambda and the model we are evaluating is OPT-1.3B pulled from Hugging Face.

## 4.2 Evaluation results

The Evaluation results that compare how partial quantization on heads maintains the accuracy against all the previous methods are listed in the figure 4.1. It shows that if we randomly selected 16 heads to quantize, it gains a performance improvement from the SmoothQuant W8A8 model. Importantly, if we use our algorithm to choose 16 less important heads rather than random heads, it gains a further accuracy boost.

The figure 4.2 has shown that when we have different numbers of heads to quantize, we have different accuracy. It exhibits that if we quantize 24 less significant heads, we will have the highest accuracy. It seems a bit unusual to me because it should be more accurate if we quantize fewer heads. I would say it is because we are using only the first 1000 data batches from the lambda dataset and the model size is too small that it could not show a general theory. However, the evaluation is sufficient to prove that partial head quantization based on head importance can improve accuracy.

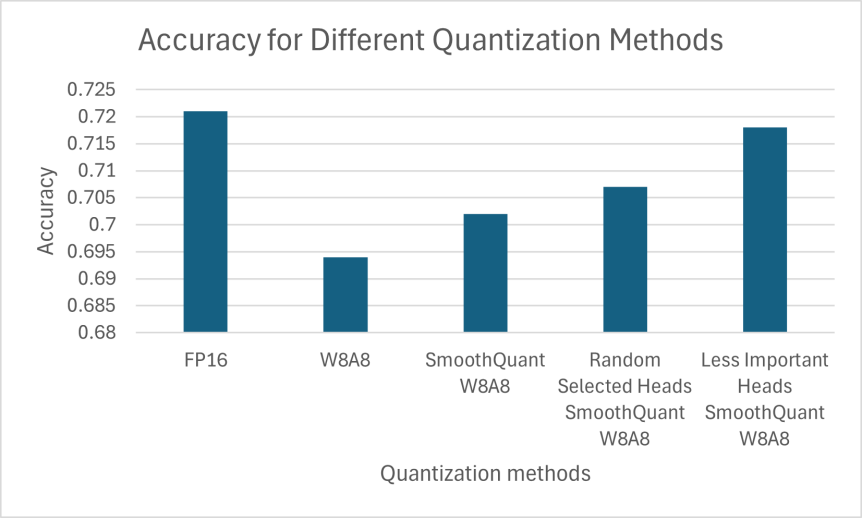


Figure 4.1: This is a figure that shows accuracy of OPT-1.3B predicting the last word of the input tokens from lambda dataset with different quantization method

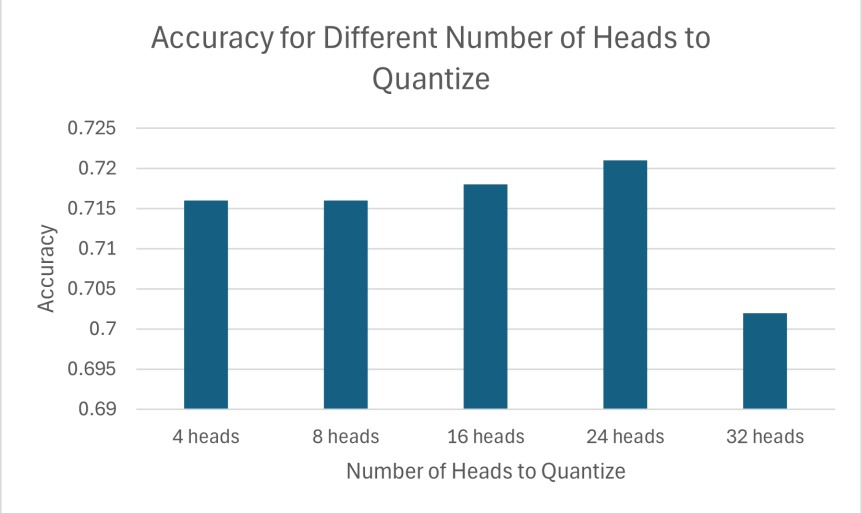


Figure 4.2: This is a figure that shows accuracy of OPT-1.3B predicting the last word of the input tokens from lambda dataset with different number of heads to quantize

# Chapter 5

# Future Work

This thesis has limitations on the evaluation results. For future work, we can test the thesis on a model with a larger size, which means the model has more layers and heads. On the other hand, we can try models of different categories, such as Llama-2. In addition, this thesis is only a verification for maintaining the accuracy with partial head quantization. It does not verify the performance of latency improvement. Lastly, it would be adequate if this thesis can be experimented with memory saving. Due to the hardware limitations, we leave these evaluation criteria for future exploration.

# Bibliography


[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale, 2022. *CoRR abs/2208.07339*, 2022.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention in natural language processing. *IEEE transactions on neural networks and learning systems*, 32(10):4291–4308, 2020.

[5] Charles AR Hoare. Quicksort. *The computer journal*, 5(1):10–16, 1962.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[7] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[8] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021.

[9] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.

[10] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[11] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183, 2022.