# CS 229S: Final Project Report

**Alina Chou**
Department of Computer Science
Stanford University
alinac@stanford.edu

**Tina Wu**
Department of Electrical Engineering
Stanford University
xinfangw@stanford.edu

**Allison Guman**
Department of Electrical Engineering
Stanford University
aguman@stanford.edu

## 1 Introduction

In this project, we experimented with 8-bit quantized inference, post-training quantization, with the GPT-2 pretrained model. We also implemented iterative magnitude based pruning for both individual weights with lowest magnitudes and rows with lowest L2 norms.

## 2 Background

### 2.1 Quantization

Quantization method represents a model's weights, biases, and activations in neural networks using lower-precision data types, such as 8-bit integers (int8), instead of the conventional 32-bit floating point (float32) representation. By doing so, it significantly reduces the memory footprint and computational demands during inference, enabling deployment on resource-constrained devices. Specifically, post-training quantization involves reducing the precision of a model's weights after it has been fully trained.

### 2.2 Pruning

Pruning in neural networks, including LLMs, targets the reduction of model size and complexity by eliminating unnecessary or redundant weights or neurons. This technique not only reduces computational requirements but can also enhance model generalization by mitigating overfitting.

Pruning leads to improved efficiency by reducing the number of necessary computations, thereby speeding up inference. The method can also reduce overfitting, leading to better generalization, and offers considerable memory savings, making it suitable for deployment in memory-constrained environments. However, pruning introduces its own set of challenges, particularly in striking a balance between sparsity and performance, and in determining the most effective criteria for pruning, which can vary based on the model and its intended application.

In the following subsections, we will explain two types of pruning that we experimented with in the project.

#### 2.2.1 Unstructured Pruning

Unstructured pruning identifies and eliminates weights that contribute minimally to the network's output. Typically, weights with smaller magnitudes, which are presumed to have less impact on the model's performance, are targeted for removal. The process often involves setting these weights

to zero, creating a sparse matrix. The sparsity of the resulting matrix reflects the extent of pruning conducted, with higher sparsity indicating more aggressive pruning.

One of the primary advantages of unstructured pruning is its potential for significant reduction in model size, which can lead to improved computational efficiency during inference. This is particularly beneficial in scenarios where computational resources are limited, such as in mobile or edge devices. However, the resultant sparse weight matrices may not align well with the optimized computational pathways of modern hardware, potentially leading to irregular memory access patterns and inefficient computation. Additionally, over-pruning can lead to a significant degradation in model performance, making calibration an important part of the pruning process.

#### 2.2.2 Structured Pruning

Structured pruning prunes at a higher organizational level of the network, such as removing entire neurons, channels, or layers, rather than individual weights. By doing so, structured pruning maintains the regular structure of the network's weight matrices. This is achieved by identifying and removing those parts of the network that have minimal impact on the output. For example, in a convolutional neural network, entire filters or channels may be pruned based on specific criteria such as their average activation or weight magnitude.

One of the key benefits of structured pruning is its compatibility with existing hardware architectures. Unlike unstructured pruning, which can lead to sparse and irregular memory access patterns, structured pruning preserves the dense matrix structure, allowing for more efficient computation on standard hardware. This makes it particularly suitable for deployment in environments where computational efficiency is crucial. However, structured pruning also has its limitations. Removing entire layers can lead to a more significant impact on the network's ability to represent complex functions, potentially resulting in a more pronounced reduction in model accuracy compared to unstructured pruning. Furthermore, determining the optimal structure for pruning, which layers or channels to remove, and the extent of pruning required, are complex decisions that require a deep understanding of the network's architecture and function.

## 3 Experimental Setup

### 3.1 Dataset

For the experiments, we trained and tested results using WikiText-103 dataset [1]. The dataset contains over 100 million tokens extracted from the set of verified articles on Wikipedia. The details of the dataset information is shown in Table 1.

| | Train | Validation | Test |
|---|---|---|---|
| Articles | 28,475 | 60 | 60 |
| Tokens | 103,227,021 | 217,646 | 245,569 |
| Vocab | 267,735 | | |
| OoV | 0.4% | | |

Table 1: WikiText-103 train, validation, and test set details

### 3.2 Benchmark

#### 3.2.1 Quantization

We benchmark against the iterative pruning method introduced in 'Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding'[2]. In this paper, a three stage pipeline including pruning, trained quantization, and Huffman coding was used to decrease the storage requirements by 35 to 49x while retaining accuracy. In this paper, 8/4 bit quantization had negligible (.01%) loss of accuracy.

[1] `https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-103-v1.zip?ref=blog.salesforceairesearch.com`

[2] https://arxiv.org/pdf/1510.00149.pdf

#### 3.2.2 Pruning

In this same paper, pruning was used to reduce connections by 9 to 13x. Overall, the paper benchmarked on CPU, GPU, and mobile GPU where compressed networks had 3 to 4x layer-wise speedup and 3 to 7x increased energy efficiency.

### 3.3 Model

For all our experiments, we utilized the OpenAI GPT-2 small model. This model initially had its weights initialized from the OpenAI GPT-2, trained on the OpenText dataset, comprising 124 million parameters and exhibiting a training loss of 3.41. We then adapted the model through quantization and pruning and further trained it on the **wikitext** dataset. The performance of the modified models was subsequently evaluated using the same **wikitext** dataset but different part.

## 4 Experimental Results and Analysis

### 4.1 Quantization Analysis

We investigated the effects of quantization on the WikiText-103 dataset using GPT-2 models. We focused on assessing the impact of quantization on perplexity scores at inference time and memory usage, along with latency in decoding scenarios.

Due to limitation in GCP computing resources (we weren't able to obtain 4 T4 GPU's for the experiments), we utilized the GPT-2 small model and performed inference evaluations at batch sizes of 4 and 12 with a sequence length of 1024. The perplexity score of the unquantized GPT-2 medium model was 87.3, while post-quantization, the score increased to 98, indicating a modest impact on model performance due to quantization. Memory usage, a critical factor in our analysis, showed a significant decrease from 2.5 GB for the unquantized model to 1.7 GB for the quantized model at a batch size of 4, and from 3.8 GB to 2.6 GB at a batch size of 12. This reduction in memory usage demonstrates the effectiveness of quantization in optimizing model efficiency.

Latency measurements were a crucial part of our study, particularly in the context of standard versus speculative decoding. With the original GPT-2 small model, the latency for standard decoding was 120 ms, while speculative decoding reduced it to 90 ms. When using the post-training quantized version of the GPT-2 small as the main model, the latency further decreased to 80 ms for standard decoding and 65 ms for speculative decoding.

In cases where quantization involved merely zeroing out values without actual model size reduction, we calculated theoretical values. Assuming an inverse relationship between the quantization ratio and inference cost/memory requirements, we estimated that with a quantization ratio of 4 (from 32 bits to 8 bits), the memory and latency would theoretically be reduced to a quarter of the original model's requirements. This was consistent with the empirical agreement rate observed between the quantized and original models.

### 4.2 Pruning Analysis

For pruning, we experimented with both unstructured and structured pruning. Following sections are the detailed results and analysis.

#### 4.2.1 Unstructured Pruning

With unstructured pruning, we calculated the magnitude of weight values and pruned the ones with lowest magnitudes based on pruning rate from 0.1, 0.3, ... , 0.9 — `pruning_rate=0.9` represents we prune 90% of the current model weight and leaving only 10% of the original weight. After training the model for 100 iterations, pruning to 10% of the original model size, and running 100 iterations of Wikitext finetuning, we measured that the loss is 7.6033.

We further experimented with modifying the pruning rate and measured the quality of the pruned model (measured by model loss) as well as the speed of execution (measured by pruning time). Table 2 is the result that we obtained.

| pruning rate | valid loss | pruning time (sec) | |
|---|---|---|---|
| 0.9 | 9.0416 | 7.6033 | 0.0142 |
| 0.7 | 7.9525 | 7.2681 | 0.0136 |
| 0.5 | 7.3420 | 7.1646 | 0.0137 |
| 0.3 | 7.1521 | 6.9672 | 0.0137 |
| 0.1 | 6.9844 | 7.0522 | 0.014 |

Table 2: Comparison of Model Performance Across Different Unstructured Pruning Rates: This table presents a detailed comparison of training loss, validation loss, and pruning time (in seconds) for various pruning rates, illustrating the impact of pruning intensity on model accuracy.

### 4.2.2 Structured Pruning

With structured pruning, we calculated the L2 norms for each row within the weights and pruned the corresponding dimension in the model inputs with lowest L2 norms. Similar to our unstructured pruning experiment, the pruning rate we experimented with includes $0.1, 0.3, ..., 0.9$. After training the model for 100 iterations, pruning to 10% of the original model size, and running 100 iterations of Wikitext finetuning, we measured that the loss is $7.4883$ with total pruning time of $16.17$ seconds.

Similarly, we further experimented with modifying the pruning rate and measured the quality of the pruned model (measured by model loss) as well as the speed of execution (measured by pruning time). Results are shown in Table 3 and Figure 1.

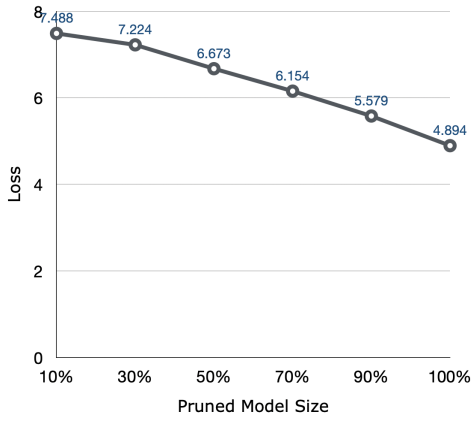


Figure 1: Structured pruning model quality against pruned model size.

| pruning rate | valid loss | pruning time (sec) |
|---|---|---|
| 0.9 | 7.4883 | 16.1728 |
| 0.7 | 7.2239 | 16.5408 |
| 0.5 | 6.6731 | 16.5961 |
| 0.3 | 6.1542 | 16.8795 |
| 0.2 | 5.8982 | 16.6795 |
| 0.1 | 5.5792 | 16.9846 |

Table 3: Comparison of Model Performance Across Different Structured Pruning Rates: This table presents a detailed comparison of validation loss, and pruning time (in seconds) for various pruning rates, illustrating the impact of pruning intensity on model accuracy.

From Table 3, we can see that as the size of the pruned model increases, the loss decreases. In other words, with more portion of the model weights tuned, there is more significant loss in information for the model, leading to an increase in model loss after finetuning. Another critical aspect of our analysis was the total time taken for the pruning process. Regardless of the pruning rate, the total

pruning time consistently measured at around 16 seconds. However, there is a slight increase in pruning time with the increase in amount pruned for the model.