

Effects of Quantization on GPT-2 Small

Stanford CS224N Default Project

Isabella Jordan

Department of Electrical Engineering

Stanford University

ijordan0@stanford.edu

Abstract

This project implements GPT-2 to use across various natural language processing tasks, including sentiment analysis, paraphrase detection, and sonnet generation. It then investigates the effects of quantization on paraphrase detection and sonnet generation. Two post-training quantization approaches, Optimum Quanto and GPTQ, are evaluated across multiple weight and activation bit widths. Task accuracy, inference time, and model size are measured to evaluate the trade-offs between efficiency and model performance. Experiments show 8 bit quantization across both methods is able to maintain baseline performance across tasks while also reducing model size by at least 2x. 2 bit quantization leads to significant reductions in model performance. GPTQ, while harder to use than Optimum Quanto, results in better task performance, faster inference time, and smaller model sizes in comparison to Optimum Quanto due to its optimized kernels.

1 Key Information to include

- Mentor: Ali Sartaz Khan
- External Collaborators (if you have any): N/A
- Sharing project: N/A
- Late Days (if you are using any):

Team member	Late days remaining
Isabella Jordan	5
Late days applied for the report	1

2 Introduction

Large language models such as GPT-2 have made significant steps in language tasks. However, these models can have a large number of parameters that tend to increase with every generation of a model which leads to extremely high computational and storage costs. Quantization is a common technique that reduces the numerical precision of model weights and activations. It converts high-precision values, such as 32 bit floating point numbers, into a lower-precision (less bits) representation. Reducing the number of bits can help decrease the model size and thus memory usage while also accelerating inference by using faster, smaller hardware multipliers. However, quantization introduces approximation errors which may degrade model performance. Quantization-aware training is popular, but this requires retraining the entire model which can be expensive and time-consuming for large models. This project aims to explore different methods of post-training quantization on a GPT-2 small model to explore its effects on model performance, inference time, and model size.

3 Related Work

Quantization has become an important technique for reducing the computational and memory footprint of large language models. One approach is to perform inference using low-precision arithmetic while

preserving model accuracy. LLM.int8() [1] enables 8-bit matrix multiplication for transformers by handling activation outliers separately, resulting in minimal performance degradation.

Several works proposed improved post-training quantization methods for transformer architectures. ZeroQuant [2] is a post-training end-to-end quantization and inference pipeline for large scale transformer that reduces the precision of weights and activations to int8 with minimal accuracy impacts. It also has an inference speedup and can work well even without access to the original training data. GPTQ [3] is a method for one-shot post-training quantization based on approximate second-order information that is that maintains high accuracy while also being efficient by reducing runtime and reducing model size to be able to fit on one GPU. This approach performs quantization layer by layer and builds on Optimal Brain Quantization (OBQ) which handles each weight row independently, quantizing one weight at a time. However, GPTQ quantizes weights in all rows in the same order rather than greedily and batches this operation to have higher arithmetic intensity. SmoothQuant [4] is a training-free post-training quantization method that enables 8 bit weights and 8 bit activations. It smooths activation outliers by moving the quantization from activations to weights with a mathematically equivalent transformation.

Other work focuses on adapting transformer architectures to improve quantization performance. Quadapter [5] is a proposed quantization adapter for fine-tuning that avoids overfitting. It adds blocks into the GPT-2 model that have a set of learnable parameters which is able to maintain the original model parameter. [6] proposed a zero-shot dynamic quantization technique for transformers that does not need training data for calibration and is applicable to many tasks.

Taking inspiration from these works on performing post-training quantization, this project aims to evaluate and survey post-training quantization of GPT-2 across different NLP tasks. Different quantization methods are explored including GPTQ [5].

4 Approach

There are three tasks as specified in the handout [7] to be done for this project:

1. *Sentiment Analysis*: Classifying the sentiment of movie reviews (SST, CFIMDB)
2. *Cloze-Style Paraphrase Detection*: Determining if two different questions are paraphrases of each other (Quora)
3. *Sonnet Generation*: Generating a sonnet given the first three lines (Shakespeare)

The approach to solving these will be using GPT-2 Small. GPT-2 is implemented as defined in the default project handout [7]. This model will include tokenization, an embedding layer, transformer layers (masked multi-head attention, skip connections, a multi-layer perceptron (MLP), dropout, and layernorm layers). The Adam Optimizer will also be implemented. The baseline for these models is an implementation that reaches the teaching staff's performance metrics.

Post-training quantization will then be explored for the extensions: Paraphrase Detection and Sonnet Generation. The standard GPT-2 Small model uses 32 bit floating point number representations for model weights and activations and the goal is to reduce memory and computational costs while maintaining similar performance. Post-training quantization is implemented with HuggingFace's Optimum Quanto [8]. Weights will be quantized to float8, int8, int4, and int2. Activations will be quantized to float8 and int8. Optimum Quanto is an easy to use library that linearly quantizes weights. Linear quantization uses a high-precision (float32) number for a scale and a zero point offset which represents the quantized value that corresponds to 0. The formula for dequantizing is $x = scale * (x_{quantized} - zero_point)$. It calls these datatypes qfloat8, qint8, qint4, and qint2 to indicate that there are additional parameters for this overhead. Another version of post-training quantization, GPTQ [3], mentioned in the related work section, is also implemented using HuggingFace's Optimum integration with AutoGPTQ [9] which only runs on CUDA machines. This API only quantizes weights so weights will be quantized to 8, 4, and 2 bits.

5 Experiments

5.1 Data

1. *Stanford Sentiment Treebank (SST)*: 11,855 single sentences from movie reviews (Sentiment Analysis). The split is train (8,544 examples), dev (1,101 examples), test (2,210 examples).
2. *CFIMDB*: 2,434 highly polar movie reviews (Sentiment Analysis). The split is train (1,701 examples), dev (245 examples), test (488 examples).
3. *Quora*: 400,000 question pairs (Paraphrase Detection). The split is train (141,506 examples), dev (20,215 examples), test (40,431 examples).
4. *Shakespeare Sonnets*: 154 sonnets written by Shakespeare (Sonnet Generation). The split is train (131 sonnets), dev, (12 sonnets) test (12 sonnets).

5.2 Evaluation method

For sentiment analysis and paraphrase detection, accuracy will be used as the performance metric. The CHRF score will be used for sonnet generation. These are listed in the default project handout [7]. Inference time and model size will be used as metrics with the quantized models for paraphrase detection and sonnet generation. Training loss vs epochs and dev accuracy/CHRF score vs epochs will be analyzed to see how the model trained. For paraphrase detection, the output will be studied by examining the number of "yes" and "no"s the model classified see if there is a bias and if it matches the general trends of the data. For sonnet generation, the generated sonnets will be reviewed to see if they look reasonable from a human perspective.

5.3 Experimental details

5.3.1 Sentiment Analysis

Two experiments were run on each of the datasets (SST and CFIMDB). These were fine-tuning (1) only the last linear layer which freezes the pre-trained GPT parameters and (2) the whole model which is all parameters. Default GPT-2 small was used. The learning rates used were $1e-3$ for the last linear layer fine-tuning and $1e-5$ for the full model. The number of epochs was set to 10. The batch size was set to 8. The hidden dropout probability was set to 0.3. Last linear layer was run locally and full model was run on a NVIDIA T4 GPU.

5.3.2 Paraphrase Detection

GPT-2 small was used as the model. The Quora dataset was used. Learning rate was set to $1e-5$. The number of epochs was set to 10. Batch size was set to 64. This was run on a Modal NVIDIA H100 GPU where epoch took approximately 8 minutes to run.

The model trained as above was then quantized multiple times on an NVIDIA L4 GPU. Baseline quantization with Optimum Quanto was run, quantizing weights to qfloat8, qint8, qint4, and qint2. After, activations were also quantized to qfloat8 and qint8. All of these combinations were run together resulting in 12 quantized models. GPTQ was then used to quantize the original model weights to 8, 4, and 2 bits. GPTQ took approximately 1.5 minutes to quantize the original model for each quantization configuration.

5.3.3 Sonnet Generation

GPT-2 small was used as the model. The Shakespeare sonnet dataset was used. Learning rate was set to $1e-5$. The number of epochs was set to 20 because the dataset was much smaller meaning training took less time. Batch size was set to 8. This was run locally on a Mac M3 Max where epoch took approximately 3 seconds to run.

The model trained as above was then quantized multiple times on an NVIDIA L4 GPU, the same as paraphrase detection's model. Baseline quantization with Optimum Quanto was run, quantizing weights to qfloat8, qint8, qint4, and qint2. After, activations were also quantized to qfloat8 and qint8. All of these combinations were run together resulting in 12 quantized models. GPTQ was then used

to quantize the original model weights to 8, 4, and 2 bits. GPTQ took approximately 39 seconds to quantize the original model for each quantization configuration.

5.4 Results

5.4.1 Sentiment Analysis

Table 1 below contains the accuracies achieved:

	SST Dev	SST Test	CFIMDB Dev
Last Linear	0.473	0.467	0.865
Full Model	0.513	0.539	0.980

Table 1: Sentiment Analysis Accuracy

The results are as expected as they reached the staff’s baselines. They are slightly better than expected since some of the results exceeded the staff’s baselines. It was also easier to tune than expected since generally the default parameters seemed to work and produce these results. This is likely because of the robustness of the pre-trained GPT-2 small parameters and architecture for these specific tasks.

5.4.2 Paraphrase Detection

A dev accuracy of 0.893 was achieved on the baseline FP32 (32 bit floating point) model. This exceeded the teaching staff’s reported number so it was considered sufficient as a baseline. No other quantized model was able to reach a higher dev accuracy but qfloat8 (no activation quantization), qint8 (no activation quantization) and GPTQ with 8 bit quantization reached the same accuracy. A test accuracy of 0.839 was reached.

Fig. 1a shows the dev accuracy achieved across the various quantized models. It can be seen that generally weights that are quantized with 4 and 8 bits are able to reach the baseline FP32 accuracy with the exception of those that have qint8 quantized activations. This result is likely due to the fact that qfloat8 has a floating point format that handles a wider range of values better than integers. 2 bit weights massively decrease the performance. GPTQ with 2 bits did not perform as poorly as the Optimum Quanto models.

Fig. 1b shows the inference time across the various quantized models for producing the dev set results. With the exception of GPTQ, the quantized models take longer to produce the dev set results. This is likely due to the fact that every weight/activation needs to be dequantized (converted back to FP16 or FP32) in order for the machine to actually perform the matrix multiplication that is part of inference. This occurs because the machine either does not have these lower bit multiplication hardware units or it is not being properly mapped to them. If inference cannot use these smaller multipliers/matrix multiply units, there is no speedup for the model and there is extra overhead to dequantize to do the multiplication and then quantize after for activations. Generally, as the number of bits for the weights decrease, the inference time gets longer. Again, this is with the exception of GPTQ which has the same inference time across all weight quantizations. The faster inference speed is likely due to the fact that GPTQ [3] discussed optimizing kernels for inference and the AutoGPTQ backend that HuggingFace Optimum paired with likely implemented this.

Fig. 1c shows the model size across the various quantized models. qfloat8 and qint8 based models have the same size since they have the same number of bits for their representations. As the number of bits for weights decreases, the model gets smaller. It does not scale linearly due to the overhead of quantization parameters such as the scale and zero-point values which are needed to dequantize the quantized values. Activation quantization does not affect inference size since this is not a stored value and is what is calculated for each layer. GPTQ models are smaller likely due to having less metadata and overhead quantization parameter storage than the Optimum Quanto version.

The dev accuracy results are about what was expected with lower bit quantizations generally resulting in worse accuracy. The longer inference time was not expected because generally quantization results in less bits which should theoretically use cheaper (faster, smaller) multipliers; however, the reasoning for this deviation was explained above. Model size results are as expected with less bits resulting in a smaller model. The test accuracy was worse than expected since it was slightly under the teaching staff’s baseline and 5% lower than the dev accuracy. The best model was chosen according to dev accuracy results which may have made it fit too much to the dev set.

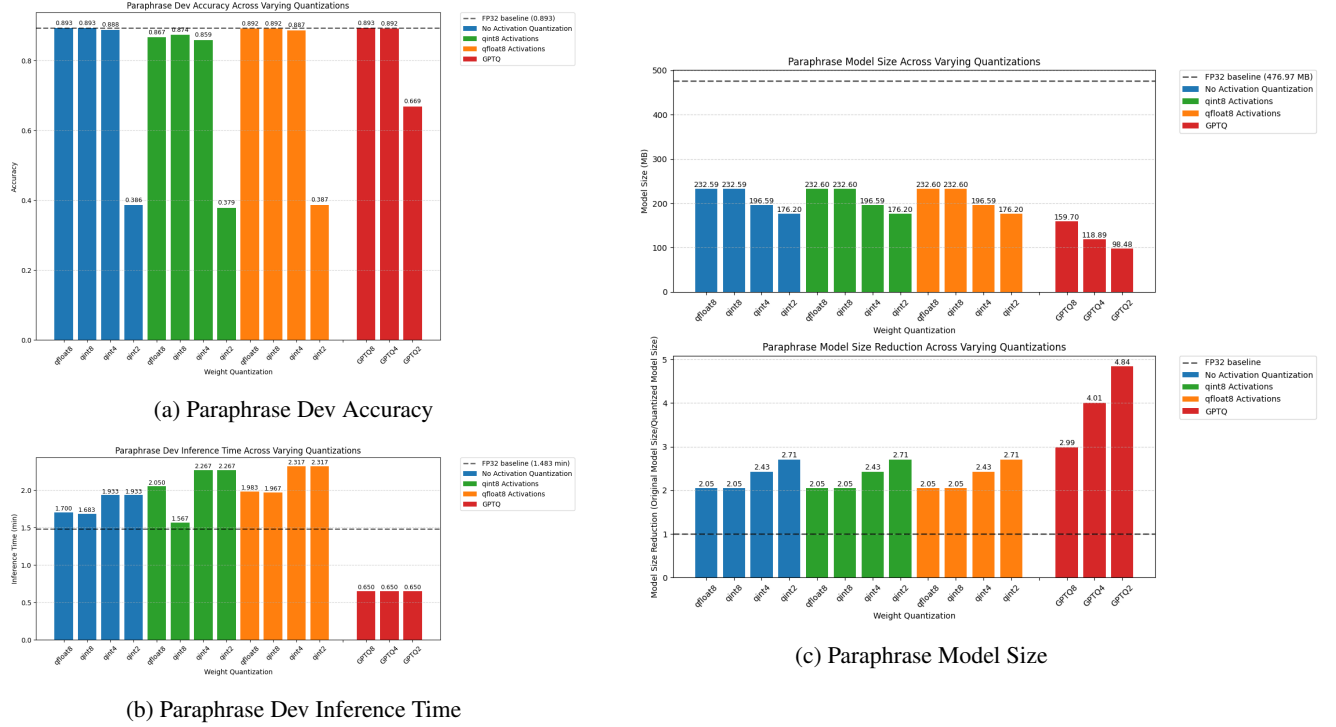


Figure 1: Paraphrase Model Evaluation Metrics

5.4.3 Sonnet Generation

A dev CHRF score of 42.217 was achieved on the baseline FP32 model. This exceeded the teaching staff’s reported number so it was considered sufficient as a baseline. The GPTQ 8 bit quantization was able to reach the highest dev CHRF score of 42.279 across all models. A test CHRF score of 41.648 was reached with this model.

Fig. 2a shows the dev CHRF score achieved across the various quantized models. Most quantized models, with the exception of qint2 weights, are able to achieve or exceed the FP32 baseline. qint2 models also do not see as significant of a drop off as in paraphrase detection. Additionally, a few quantized models even exceed the baseline which was not seen for paraphrase detection. These results are likely because sonnet generation has a smaller dataset. Quantization is likely helping with not overfitting on the small dataset and allows the model to generalize better since it has a regularization effect.

Fig. 2b shows the inference time across the various quantized models for generating the dev sonnets. All quantized models take significantly longer than the FP32 baseline. With less data for inference, the base FP32 model seems to be quite fast. The additional overhead associated with quantization and dequantization is not able to pay off with these smaller matrix multiplies. As with paraphrase detection, as the number of bits for the weights decrease, the inference time tends to get longer. However, interestingly for no activation quantization and qfloat8 activation quantization, qint2 weights are faster than qint4. Additionally, the models with activation quantization take longer for inference which is due to the extra overhead of quantizing and dequantizing the activations. GPTQ, like with paraphrase detection, has a constant inference time across all weight quantizations.

Fig. 2c shows the model size across the various quantized models. It has the same numbers as in paraphrase detection which makes sense since it is the same model architecture, just with different weight parameter values.

The dev CHRF results are slightly unexpected as quantization does help performance which is different from paraphrase detection. The longer inference time was unexpected initially, but after running paraphrase detection it was expected. However, it was longer in comparison to the baseline which was explained previously. Model size results are again as expected with less bits resulting in a smaller model. The test CHRF was about as expected, exceeding the staff’s baseline slightly.

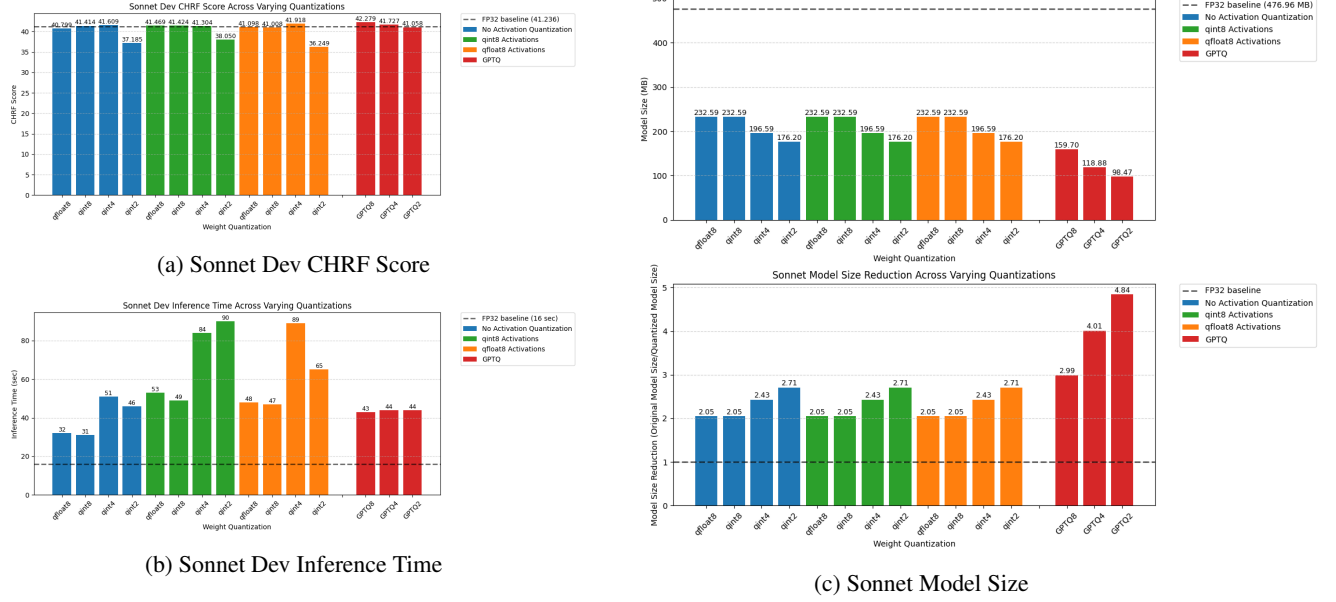


Figure 2: Sonnet Model Evaluation Metrics

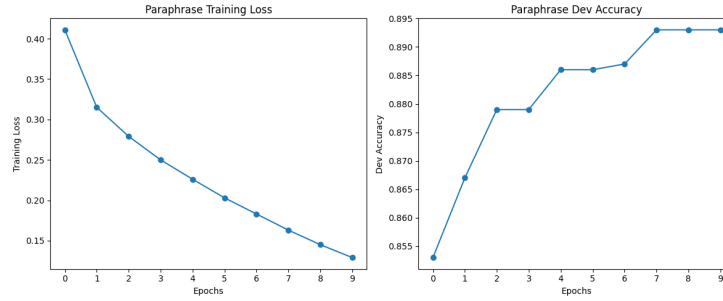


Figure 3: Paraphrase Training Metrics

However, it was below the dev more than the staff's was below their dev. Like paraphrase detection, the best model was chosen according to dev accuracy results which may have made it fit too much to the dev set. However, the quantization likely helped with generalization with the smaller datasets.

6 Analysis

6.1 Paraphrase Detection

Paraphrase training metrics which show training loss vs epoch and dev accuracy vs epoch are shown in Fig. 3. Through epoch 9, training loss continues decrease. However, after epoch 7, the maximum dev accuracy is reached and the plot becomes flat indicating training is done. Training loss was still continuing to go down during these epochs which probably indicated the model was going to start overfitting on the training data.

Fig. 4 shows the dataset classification split. 63% of the training and dev data has a "No" classification and 37% results in a "Yes" classification. Having an uneven distribution is not ideal because this can bias the model. However, there is a large amount of data which is good for training and the dev split is essentially the same as the training.

Fig. 5 shows how the FP32 model classified the test and dev sets. As can be seen, the splits of the dev closely match the actual dev split. Test is split more towards "No" classifications. As this has a fairly high accuracy on both test and dev, this is what the quantized models should generally look like as well.

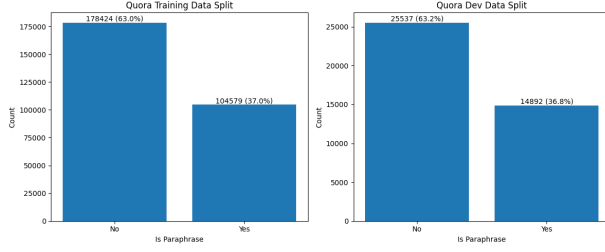


Figure 4: Quora Dataset Split

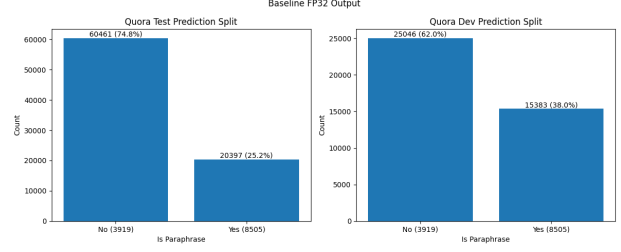
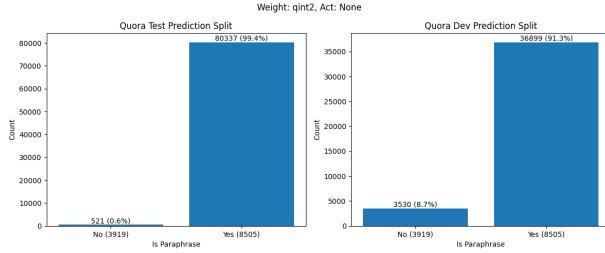
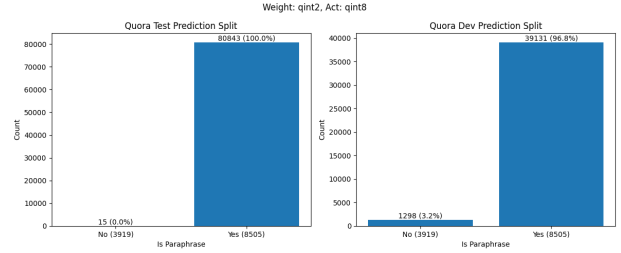


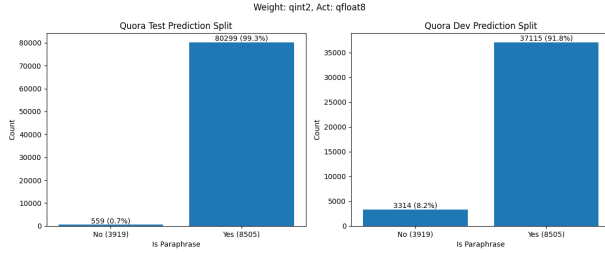
Figure 5: Quora Prediction Split FP32



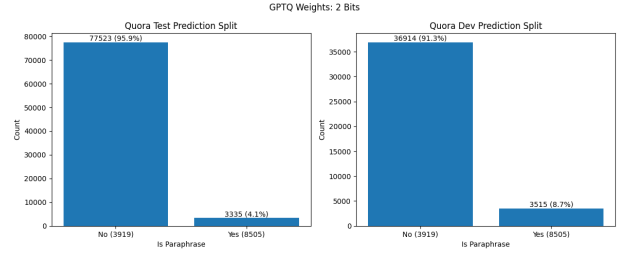
(a) Weight: qint2, Activation: None



(b) Weight: qint2, Activation: qint8



(c) Weight: qint2, Activation: qfloat8



(d) GPTQ Weight: 2 bits

Figure 6: Quora Prediction Splits for 2 Bit Weights

As noted in the Results section, models with weights quantized to 2 bits performed significantly worse than all other models. Fig. 6 shows the prediction splits for these models. It can be seen in all of these models, the model has essentially collapsed, only making one classification. For the Optimum Quanto models (Figs. 6 a-c), the model almost always predicts "Yes". This is interesting since the dataset is biased towards "No". Since it almost always classifies "Yes" and the dev dataset has 36.8% of the data being a true "Yes", this explains why the dev accuracy is around 0.3. GPTQ with 2 bit weight quantization collapses as well, but to mostly predict "No". This is why it has a dev accuracy of around 0.6. The rest of the prediction splits are in the Appendix in Fig. 11. It can be seen that every other model looks similar to the FP32 prediction split which explains why all the dev accuracies are around the same.

6.2 Sonnet Generation

Sonnet training metrics which show training loss vs epoch and CHRF score vs epoch are shown in Fig. 7. Through epoch 18, training loss continues to decrease. It goes up at epoch 19, indicating training can stop. The maximum dev CHRF score was achieved on the epoch 12. After this epoch, it starts to approximately flatten, indicating it has reached a training stopping point. Training loss likely kept decreasing because the model was beginning to overfit to the small dataset.

Fig. 8 has the first true dev sonnet. Fig. 9 has the best dev sonnet which was for GPTQ with 8 bit weights. Fig. 10 has a partial output of qint2 weight quantized model. Fig. 9 shows a sonnet that has similar formatting to Fig. 8 and upon reading, makes sense. It does tend to have a lot more punctuation at the end of each line. Fig. 10 in comparison only has three lines. It never creates a new line after the first three lines and the generated line cannot even fit on one page width in this

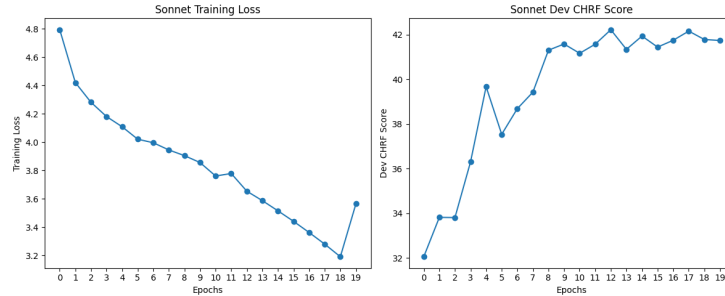


Figure 7: Sonnet Training Metrics

document. Additionally, the words themselves do not make sense. From this, it seems that the model collapsed for these qint2 quantized weights as well.

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be,
Looking with pretty ruth upon my pain.
And truly not the morning sun of heaven
Better becomes the gray cheeks of the east,
Nor that full star that ushers in the even
Doth half that glory to the sober west
As those two mourning eyes become thy face.
O, let it then as well beseem thy heart
To mourn for me, since mourning doth thee grace,
And suit thy pity like in every part.
Then will I swear beauty herself is black,
And all they foul that thy complexion lack.

Figure 8: True Sonnet Output

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be,
Which when thou helpst hate give thee grace,
Eternal before the threshold of filth.
They hold on sickbed my high talent,
And sympathize with old-day pain,
And deny my tender age;
Nor may grace grant verses moving;
Nor gladly turns them out of weight;
Yet their furrows hide grace alone;
And through that pure wrung truth I review;
Time must have me be, the muse so round
Which graces time's thought.
Time by faces,'st made me stronger;
To quote in that heart are a span

Figure 9: GPTQ8 Sonnet Output

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be, an Another as the St Campus Each Security Size...

Figure 10: qint2 Sample Output

7 Conclusion

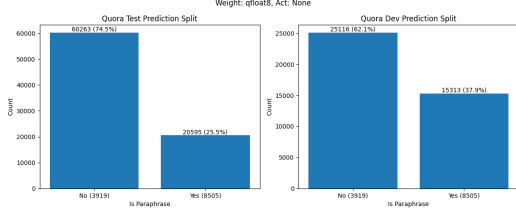
This project explored the effects of post-training quantization on an implemented GPT-2 Small. 8 bit weight quantized models were generally able to achieve the same performance metrics as the baseline FP32. 4 bit weight quantized models had a small dropoff on model performance. However, both of these had significant reductions of model size. Inference time was worsened except in the case of GPTQ for paraphrase detection. More work would need to be done to properly map models to lower bit matrix multiplication units. Aggressive quantization to 2 bits resulted in a model collapse where the model predicted only one class (paraphrase detection) or outputted close to nonsense (sonnet generation). While Optimum Quanto had a much easier implementation and much more documentation, the GPTQ version performed better in all metrics: task performance (accuracy/CHRF score), inference time, and model size.

Quantization did not help paraphrase detection's performance, but it was able to speed up inference time (GPTQ) and decrease the size of the model. Quantization did help sonnet generation's performance likely due to its regularization effects that allowed it to generalize better with a smaller dataset. It also decreased model size. However, it increased the inference time. Overall, quantization was not fully able to outperform the baseline FP32 model in all aspects across both tasks, but it provided a practical trade-off between efficiency and model performance where moderate quantization significantly reduced model size while maintaining comparable model performance.

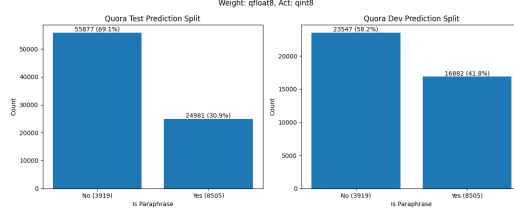
References

- [1] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.
- [2] 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, 2022.
- [3] Elias Frantar, Saleh Ashkboos, Torsten Hoeffer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.
- [4] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024.
- [5] Minseop Park, Jaeseong You, Markus Nagel, and Simyung Chang. Quadapter: Adapter for gpt-2 quantization, 2022.
- [6] Yousef El-Kurdi, Jerry Quinn, and Avirup Sil. Zero-shot dynamic quantization for transformer inference, 2022.
- [7] CS 224N. Cs 224n: Default final project: Build gpt-2. https://web.stanford.edu/class/cs224n/project/DFP_Instructions.pdf.
- [8] Hugging Face. optimum-quanto. <https://github.com/huggingface/optimum-quanto>, 2023.
- [9] Hugging Face. Quantization. https://huggingface.co/docs/optimum/llm_quantization/usage_guides/quantization.

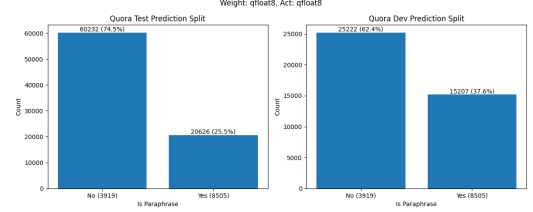
A Appendix



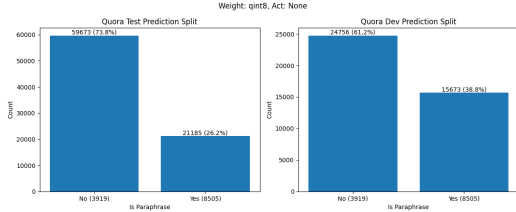
(a) Weight: qfloat8, Activation: None



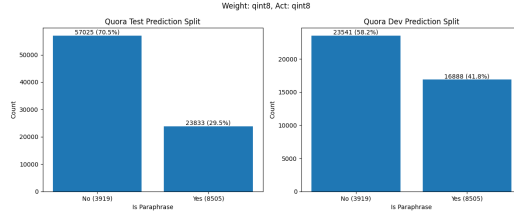
(b) Weight: qfloat8, Activation: qint8



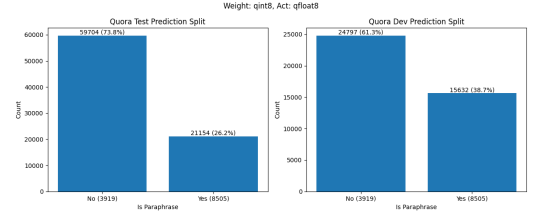
(c) Weight: qfloat8, Activation: qfloat8



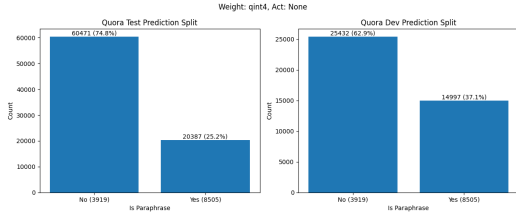
(d) Weight: qint8, Activation: None



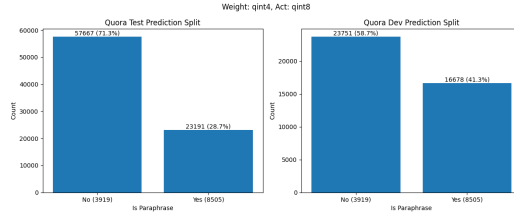
(e) Weight: qint8, Activation: qint8



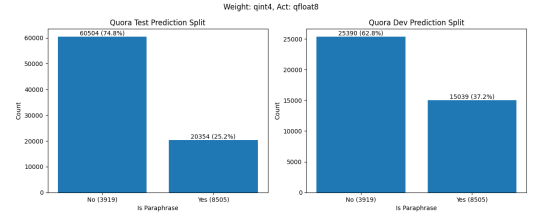
(f) Weight: qint8, Activation: qfloat8



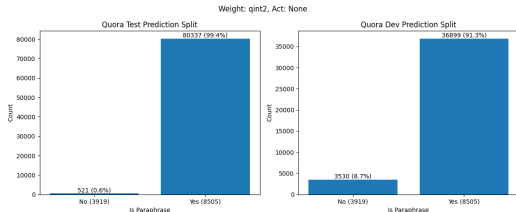
(g) Weight: qint4, Activation: None



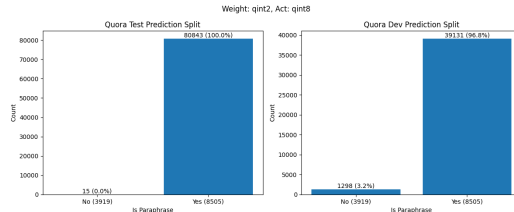
(h) Weight: qint4, Activation: qint8



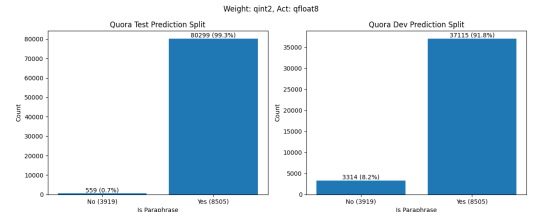
(i) Weight: qint4, Activation: qfloat8



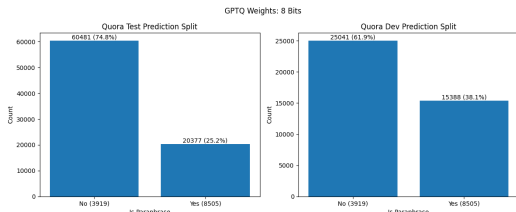
(j) Weight: qint2, Activation: None



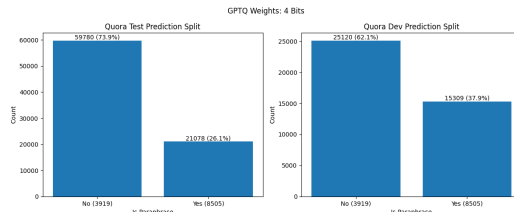
(k) Weight: qint2, Activation: qint8



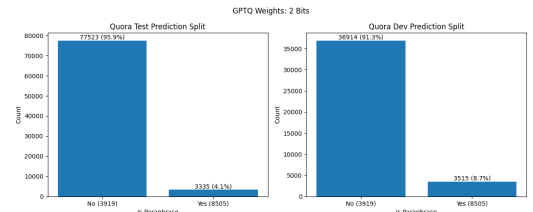
(l) Weight: qint2, Activation: qfloat8



(m) GPTQ Weight: 8 bits



(n) GPTQ Weight: 4 bits



(o) GPTQ Weight: 2 bits

Figure 11: Quora Prediction Splits