

Fine-tuning GPT-2 with LoRA

Stanford CS224N Default Project

Manish Agarwal & Pierce Mullin

manish13@stanford.edu & pmullin@stanford.edu

Abstract

This project extends the GPT-2 architecture to conduct parameter-efficient fine-tuning to specific tasks using LoRA and investigates compression using quantization. The primary method of this extension was by implementing a GPT-2 version of the LoftQ methodology [1]. Our core goal is to achieve optimal performance in line with Full Fine-Tuning of the model, but with materially less compute requirements. We were able to achieve this using LoRA, and ultimately applied no material quantization due to the impact it had on performance. We also conducted hyperparameter search and in-context learning to incrementally improve performance beyond our structural improvements. From our hyperparameter search we determined learning rates were the most impactful hyperparameter to tune in our baseline vs. LoftQ models for optimized performance. We achieved 0.843 accuracy on the Paraphrase Detection’s test dataset using our baseline model and 0.42671 on the Sonnet Generation task using our LoftQ model.

1 Key Information to include

- Title: Fine-tuning GPT-2 Architecture LoRA
- Team: Manish Agarwal (manish13@stanford.edu) & Pierce Mullin (pmullin@stanford.edu)
- Mentor: Luke Bailey
- External Collaborators: None
- Custom or Default Project: Default
- Sharing project: No

2 Introduction

Pre-trained Large Language Models have been transformative in recent years, and their high performance across a range of applications has revolutionized the AI and NLP spaces. These models often have millions or billions of parameters, and while the size of these models is what has facilitated their performance and reasoning, it also leads to significant memory and computational cost (both during training and inference).

Real-world realities of resource constraints have motivated interest and development in the spaces of Quantization and Low-rank Adaptation of large pre-trained language models, in the post-training step. Quantization is the process of compressing model size by converting high-precision numerical values of weights into a discrete set of values or reducing the format of float values [2, 3]. Low-rank Adaptation (LoRA [4]) is the process of fine-tuning large pre-trained models to specific tasks using low-rank matrices as adapters; the optimization of fine-tuning then involves training the weights of these low-rank adapter matrices while keeping the larger pre-trained model fixed, which significantly reduces the computational expense.

These two ideas were combined in QLoRA [5]. However, because quantization and the initialization of LoRA matrix weights were not coordinated together there was a consistent gap in the performance on downstream tasks between fully fine-tuned models vs. quantization plus LoRA fine-tuned

approaches. The current work uses the LoftQ methodology [1], which seeks to more optimally and jointly address both quantization and downstream fine-tuning of large pre-trained models.

LoftQ (LoRA Fine-Tuning-Aware Quantization) alleviates that discrepancy by providing a promising initialization for LoRA fine-tuning, and improves generalization in downstream tasks significantly.

We show empirical results on the efficacy of using LoftQ against full fine tuning in this project. The remaining sections are organized as follows: Section 3 gives a brief literature survey on similar ideas, Section 4 gives details on the approach, Section 5 describes our datasets, experiment setup, evaluation methods, as well as our results in detail. We give a brief analysis of these results in Section 6 and conclude in Section 7.

3 Related Work

Core Architecture: Generative Pre-trained Transformers (GPT) represent a class of large language models based on the Transformer architecture introduced by Vaswani et al. [6]. GPT-2, introduced by Radford et al. [7], is a large autoregressive language model trained on large-scale web text using self-supervised next-token prediction. The model demonstrated that large-scale pretraining followed by task-specific adaptation can yield strong performance across a wide range of natural language processing tasks without task-specific architectural modifications.

GPT-2 uses a stack of Transformer decoder blocks with masked self-attention and position-wise feed-forward layers. The model learns contextual representations through large-scale unsupervised training, enabling downstream applications such as text generation, translation, summarization, and question answering.

Low-Rank Adaptation: Full-scale fine-tuning, which updates all parameters of a pretrained model, becomes increasingly expensive as model sizes grow into the billions of parameters. Low-Rank Adaptation (LoRA) addresses this by freezing pretrained weights and introducing trainable low-rank matrices that approximate weight updates [4]. Specifically, the weight update ΔW is parameterized as a product of two low-rank matrices A and B .

This approach significantly reduces the number of trainable parameters and GPU memory requirements during fine-tuning. In experiments on models such as RoBERTa [8], DeBERTa [9], GPT-2 [7], and GPT-3 [10], LoRA reduces trainable parameters by up to $10,000\times$ and lowers GPU memory usage by approximately $3\times$ while achieving comparable or superior performance to full fine-tuning.

LoRA is motivated by the observation that updates to large pretrained models often lie in a low-dimensional subspace, suggesting that the intrinsic rank of adaptation is small. By exploiting this property, LoRA provides a parameter-efficient method for adapting large models while maintaining inference efficiency.

Quantization: Quantization techniques reduce the memory footprint and computational cost of neural networks by representing weights and activations using lower numerical precision. Dettmers et al. introduced 8-bit optimizers that maintain the performance of 32-bit optimizers while significantly reducing memory consumption [11]. Their method uses block-wise dynamic quantization, where tensors are partitioned into blocks that are quantized independently. This allows the optimizer states to be stored in 8-bit precision without sacrificing numerical stability.

Quantizing large language models such as GPT-2 presents challenges due to activation outliers that cause large quantization errors. Quadapter addresses this issue by introducing a small set of learnable parameters that transform activations into quantization-friendly representations [12]. This reduces overfitting and improves quantization performance when fine-tuning quantized models.

GPTQ is a post-training quantization method designed specifically for large autoregressive transformer models [13]. The method performs one-shot weight quantization using approximate second-order information derived from the Hessian of the loss function. By optimizing quantization error with respect to this curvature information, GPTQ achieves high accuracy even at extremely low bit-widths.

This approach enables running extremely large models, such as 175B parameter models, on a single GPU while maintaining acceptable inference accuracy. GPTQ can operate effectively in aggressive quantization regimes including 3-bit and 2-bit representations.

Combining LoRA with Quantization: QLoRA extends the LoRA framework by enabling efficient fine-tuning of quantized large language models [14]. The method allows models with up to 65B parameters to be fine-tuned on a single 48GB GPU while preserving the performance of full 16-bit fine-tuning. QLoRA backpropagates gradients through a frozen 4-bit quantized pretrained model into low-rank adapter parameters. The method introduces several innovations: *NormalFloat (NF4)*: a 4-bit floating-point representation that is information-theoretically optimal for normally distributed weights. *Double Quantization*: quantization of the quantization constants themselves to further reduce memory usage. *Paged Optimizers*: memory-aware optimizers that manage GPU memory spikes during backpropagation.

While quantization and LoRA both aim to reduce the computational cost of adapting large language models, applying quantization independently before LoRA fine-tuning can introduce discrepancies between the quantized weights and the original high-precision model. These discrepancies are particularly pronounced at low-bit regimes such as 2–4 bits.

LoftQ addresses this issue by jointly optimizing the quantized weights and the initialization of LoRA adapters [15]. For an N -bit quantized weight matrix $Q \in \mathbb{R}_N^{d_1 \times d_2}$ and low-rank matrices $A \in \mathbb{R}^{d_1 \times r}$ and $B \in \mathbb{R}^{d_2 \times r}$ approximating the original weight matrix $W \in \mathbb{R}^{d_1 \times d_2}$.

This formulation jointly optimizes the quantized backbone and the low-rank adapters, producing a better initialization for subsequent LoRA fine-tuning. This framework will be detailed in the following section. Empirical results show that this initialization significantly reduces the performance degradation commonly observed when combining quantization with LoRA.

Recent research has further explored this direction through methods such as L4Q (parameter-efficient quantization-aware fine-tuning), AutoMixQ (layer-wise adaptive quantization), and hardware-aware numerical formats such as NF2.

Overall, this line of work contributes to bridging the gap between two major efficiency techniques for large language models: quantization and parameter-efficient fine-tuning. In particular, LoftQ builds upon QLoRA [14] by addressing the quantization-induced initialization gap and, in some cases, even improving performance beyond fully fine-tuned models, suggesting potential regularization and generalization benefits. Our work is inspired by this approach and we empirically verify some of the observations found in this work.

4 Approach

Our core methodology was to implement and leverage a Decoder-Only GPT-2 model architecture. A diagram of this model structure is provided in the Appendix, which is a Decoder-Only diagram based on the well known Attention is All You Need [6] Encoder-Decoder diagram. Having replicated this structure, we then augmented and added to it in order to allow the core GPT-2 model to perform specific down-stream tasks.

For our baseline implementations we leveraged the structure of our GPT-2 model and affixed task-specific heads to conduct specific tasks. For the Classification and Paraphrase Detection tasks we used projection matrices to create multi-category or binary outputs from tokens. For the Sonnet Generation task we transformed the full sequence hidden state. We conducted Last Linear Layer and Full Fine-Tuning to optimize the models for these tasks (with Full Fine-Tuning outperforming, which was not surprising).

We also implemented a LoftQ extension of the model. In this extension we injected LoRA matrices into the Q , K , V and dense attention layers and the MLP layers while keeping the initial weights of the model intact (this process is the LoRA component of QLoRA and LoftQ methodologies). We also explored quantization of the models under 16-bit and 4-bit scenarios. Using LoRA allowed us to achieve very similar optimization results to Full Fine-Tuning while only training 0.611% of parameters and with training times that were substantially lower.

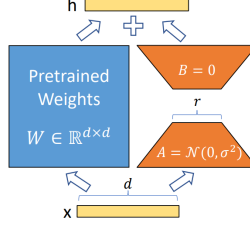


Figure 1: LoRA architecture.

LoftQ is an iterative algorithm that repeats steps of alternating initialization of the quantized weight Q_t and fine-tuning the low-rank approximation matrices A_t, B_t . This is a generalization of QLoRA, which is equivalent of a 1-iteration implementation of LoftQ. This algorithm, outlined in the original paper [1], is included below.

Algorithm 1 LoftQ

input Pre-trained weight W , target rank r , N -bit quantization function $q_N(\cdot)$, alternating step T

1: Initialize $A_0 \leftarrow 0, B_0 \leftarrow 0$

2: **for** $t = 1$ to T **do**

3: Obtain quantized weight $Q_t \leftarrow q_N(W - A_{t-1}B_{t-1}^\top)$

4: Obtain low-rank approximation $A_t, B_t \leftarrow \text{SVD}(W - Q_t)$ by

$$A_t = [\sqrt{\sigma_{t,1}} u_{t,1}, \dots, \sqrt{\sigma_{t,r}} u_{t,r}],$$

$$B_t = [\sqrt{\sigma_{t,1}} v_{t,1}, \dots, \sqrt{\sigma_{t,r}} v_{t,r}]$$

5: **end for**

output Q_T, A_T, B_T

For N -bit quantized weight $Q \in \mathbb{R}_N^{d_1 \times d_2}$ and low-rank approximations $A \in \mathbb{R}^{d_1 \times r}$ and $B \in \mathbb{R}^{d_2 \times r}$, LoftQ approximates the original high-precision pre-trained weight $W \in \mathbb{R}^{d_1 \times d_2}$ as the initialization of LoRA fine-tuning by minimizing the following objective

$$\min_{Q, A, B} \|W - Q - AB^\top\|_F,$$

where $\|\cdot\|_F$ denotes the Frobenius norm. This jointly optimizes the initial values of the quantized backbone Q and low-rank adapters A, B . Not doing this leads to notable performance degradation in downstream tasks arising from the quantization discrepancy. This optimization is solved via the above iterative algorithm. The method is then evaluated on natural language understanding, question answering, summarization, and natural language generation tasks. The method outperforms previous quantization methods especially in challenging 2/4-bit regimes.

We are not aware of a pre-existing GPT-2 implementation of LoftQ. The LoftQ paper does have a Decoder-Only implementation which is closely related, but it uses Llama-2 (which is a significantly larger model).

5 Experiments

5.1 Data

Our datasets are the default project datasets which we describe here:

- Classification, Stanford Sentiment Treebank (SST): Consisting of 11,855 single sentences and 215,154 unique parsed phrases. The training dataset we used has 8,544 observations consisting of an id, sentence, and a sentiment score from 0-4. The dev dataset had 1,101 observations and the test dataset had 2,210 observations.
- Classification, CFIMDB Dataset: Consisting of 2,434 highly polar movie reviews. The training dataset we used has 1,707 observations consisting of an id, sentence, and a senti-

ment score of 0 or 1. The dev dataset had 245 observations and the test dataset had 488 observations.

- Paraphrase Detection, Quora Dataset: Consisting of 400,000 question pairs with labels indicating whether instances are paraphrases. The training dataset we used has 283,003 observations consisting of an id, sentence1, sentence2, and an is_duplicate indicator of 0 or 1. The dev dataset had 40,429 observations and the test dataset had 80,856 observations.
- Sonnet Generation, Sonnet Dataset: consisting of 154 sonnets written by Shakespeare. Each sonnet consists of 14 lines of text following a rhyme scheme.

5.2 Evaluation method

Accuracy was the main evaluation metric we used for the Classification and Paraphrase Detection tasks. For the Sonnet Generation task we used cross entropy loss as a training & fine-tuning metric. For Sonnet Generation evaluation we used chrF (a character level n-gram comparison metric similar to BLEU score [16]) to assess the similarity between our generated sonnets and the originals; chrF is the default evaluation metric, and we also appreciate that it has more tolerance to morphology differences that may occur in a sonnet context.

We also considered training time and the ability to utilize large models as additional success measures for our Baseline vs. LoftQ comparisons. Full Fine-Tuning of the Baseline models will naturally result in strong performance, but it can place a limitation on training and inference at larger model sizes. For this reason, we considered training time (faster being better) and the model sizes that could practically be implemented on commercial hardware. Training time was measured in average seconds per training epoch, whereas the ability to use large models on commercial hardware was qualitative.

5.3 Experimental details

We have conducted three experiments. The main parameters we varied were learning rate and weight decay, and we used batch size of 8 throughout. Details about parameter values, and training times are present in the next section. We also investigate model size for the smaller dataset of sonnet generation.

- For Classification we extracted the final token from GPT-2, applied dropout, and then applied a 5-dimensional linear layer to project this token onto 5 categorical sentiment classifications
- For Paraphrase Detection we extracted the final token from GPT-2, we have not currently applied dropout to that, and then used a 2-dimensional linear layer to project this token onto 2 categorical classifications representing yes / no
- For Sonnet Generation we generated the full hidden state up to the latest point in the sequence and multiplied these by the transposed embeddings to extract the full sequence logits. This process is sequentially repeated to generate a string of text. We used a rank of 16 and quantization of 16 bit.

5.4 Results

Classification, SST and CFIMDB Datasets:

The below table outlines the performance of our two Baseline models, Last Linear Layer (LLL) and Full Model Fine-Tuning (FM), and our LoftQ models. All models used 10 epochs. We observe a need for slower learning rates and more regularization in the Full Fine-tuning models across both Classification datasets.

Dataset	Fine-tune type	LR	Weight Decay	Dev Accuracy
SST	LLL	10^{-3}	0	0.462
SST	FM	2×10^{-6}	5×10^{-3}	0.514
SST	LoftQ	10^{-4}	0	0.511
CFIMDB	LLL	10^{-3}	0	0.861
CFIMDB	FM	2×10^{-5}	10^{-3}	0.976
CFIMDB	LoftQ	10^{-4}	0	0.976

We also conducted a learning rate sweep on the SST Classification dataset. We used 3 epochs per learning rate trial because we began to see performance plateau after this point, so it was an efficient place to iterate to in our trials. We found a learning rate of 3×10^{-4} to be optimal across our sweep vs. our default LoftQ learning rate of 10^{-4} , and that there is significant improvement in LoftQ results when scaling learning rate from 10^{-5} to 10^{-4} , after which improvements begin to level off. This is in line with findings from Thinking Machines Lab [17].

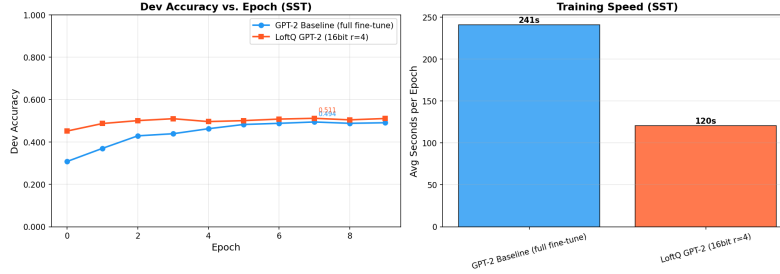


Figure 2: Classification Task (SST), Full Fine-Tuning vs. LoftQ

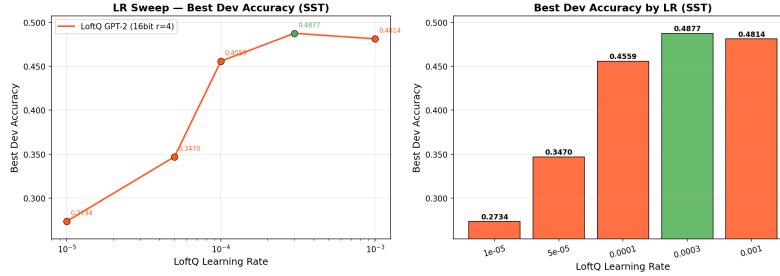


Figure 3: Classification Task (SST), Learning Rate Sweep on LoftQ Model

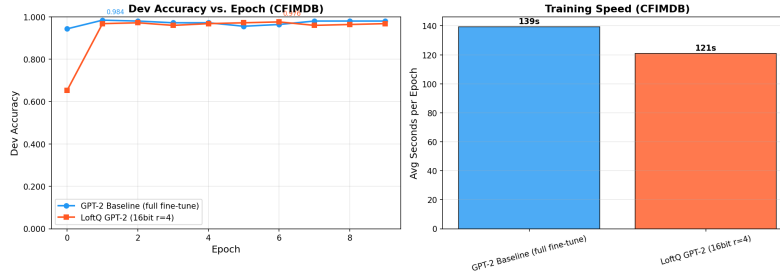


Figure 4: Classification Task (CFIMDB), Full Fine-Tuning vs. LoftQ

Paraphrase Detection, Quora Dataset:

For the Paraphrase Detection task, we achieved the highest Dev Accuracy with the Full Fine-Tuning model, and we achieved 0.843 accuracy on the test set with this model. Training time on commercial hardware was significantly longer for this task, although we did experience a marked reduction in training time when using LoftQ instead of Full Fine-Tuning. We used 3 training epochs for this task.

Model type	Fine-tune type	LR	Weight Decay	Dev Accuracy	Test Accuracy
GPT-2	FM	5×10^{-4}	0.0	0.882	0.843
GPT-2	LoftQ	5×10^{-3}	0.0	0.866	NA

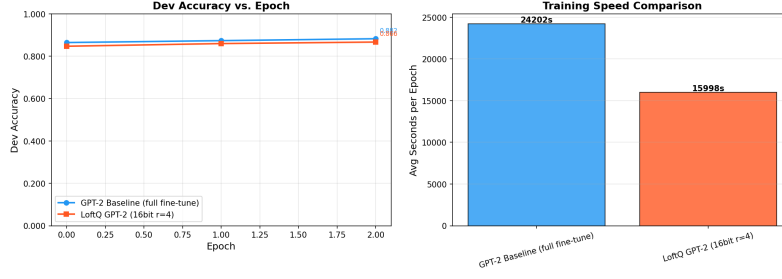


Figure 5: Paraphrase Detection Task, Full Fine-Tuning vs. LoftQ

Sonnet Generation:

This task had the least amount of data, and hence was the fastest to iterate over. We conducted 10 epochs of trials in our baseline model and also in our LoftQ model (fig.6). At such small data sizes we noted that performance and training speed was very similar between the Baseline vs. LoftQ GPT-2 models. Thus, we also conducted the same experiment using the GPT2-Large model as shown in fig.7. The run time results show that LoRA saves a considerable amount of training time while producing better results. The fact that full fine-tuning was slightly worse could be because we did not explore the parameter space enough, so we expect the results could have been more similar if that was further optimized.

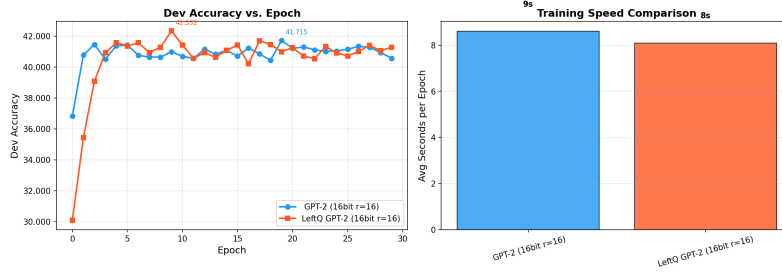


Figure 6: Sonnet Generation Task (GPT-2), Full Fine-Tuning vs. LoftQ

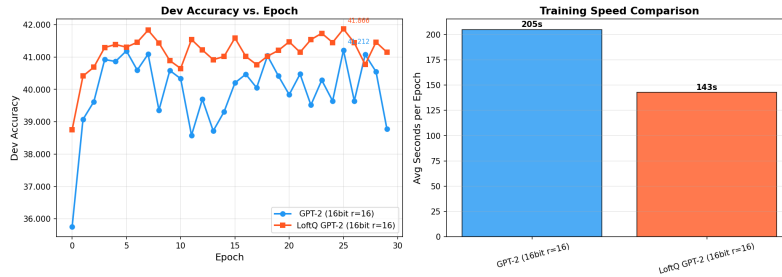


Figure 7: Sonnet Generation Task (GPT-Large), Full Fine-Tuning vs. LoftQ

Model type	Fine-tune type	LR	Weight Decay	Dev chrF	Test chrF
GPT-2	FM	5×10^{-4}	0.1	0.41715	NA
GPT-2	LoftQ	5×10^{-4}	0.1	0.42718	0.42671

6 Analysis

Overview of LoftQ / LoRA System and Strengths: Our extension of the default project was centered on implementing the LoftQ methodology. We found that quantization at significant levels such as 4-bit did meaningfully reduce performance, so for this reason we believe most of the high-performance improvements we developed should be attributed to LoRA.

Quantization approach: We implemented quantization at the numerical level but did not implement the low-level GPU memory operations required for true inference speedup. To really get the full benefit of quantization an implementation has to be done that can load the quantized weights in GPU, unpack it and then after calculations packs it back. We did not attempt this part in the interest of time.

Limitations of Commercial Hardware: Part of the challenge we gave ourselves was to train locally in most places (and in all of the versions used in this final report). We used a Macbook Pro with an M4 Pro chip to train the Classification and Paraphrase Detection tasks, and a Windows laptop with a CPU for the Sonnet Detection task. One of our motivations for implementing LoftQ was to try and make small models powerful and show these workflows can be achieved with commercial products. While we believe we were able to do this, we did meet roadblocks in some instances (such as heat throttling issues on GPU training for the Paraphrase Detection model).

Importance of Learning Rate: While iterating through models and hyperparameter selections, we immediately noticed that one of the most important factors in performance was using the appropriate learning rate. Also notable was the difference in learning rate required for Full Fine-Tuning vs. LoftQ / LoRA. Full Fine-Tuning needed lower learning rates to prevent the model overtraining and "forgetting" the language reasoning embedded in its weights, whereas LoftQ / LoRA requires a higher learning rate but does not suffer from the "forgetting" problem because large portions of the original model are frozen.

Convergence in Performance for Focused Tasks: We found that we achieved similar performance across architectures for each task as long as a sensible learning rate was used. It was particularly notable that our LoftQ / LoRA implementations achieved very similar results to Full Fine-Tuning while only training a small fraction of the total parameters. We also explored using model sizes ranging from the original GPT-2 to the Large & XL versions for the Sonnet Generation task, and size did not make a meaningful difference.

In-Context Learning: In the Sonnet Generation task, we also explored the use of prompt wrappers in order to improve results via in-context learning. We found that prefixing the standard prompt with an "ABAB CDCD EFEF GG" wrapper, while training, was able to improve performance. In this approach the trained model is used to evaluate the non-appended text. This was conducted on the LoftQ version of the model.

7 Conclusion

Through this project we have been able to develop a much stronger understanding of the Transformer architecture and LoftQ. We found quantization helped with compressing model size and loading larger models into RAM, but the LoRA component of LoftQ was more impactful in allowing us to achieve Full Fine-Tuning level performance with reduced training time. We come away from this project with the conclusion that LoRA can be used to achieve strong results on focused tasks, even on commercial hardware. This could lead to the optimistic interpretation that with careful structuring of tasks and prompting, high performance can be achieved with less compute, democratizing AI.

Team contributions

Both team members contributed to all portions of the project and in equal contributions. Both team members independently coded their own implementations of the core GPT-2 model and the LoftQ stretch goal, and then compared & aligned results for the final project code base. Manish Agarwal led fine-tuning and benchmarking for the Sonnet Generation task, and Pierce Mullin led fine-tuning and benchmarking for the Classification and Paraphrase Detection tasks, but both members were involved in both tasks.

References

- [1] Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. Loftq: Lora-fine-tuning-aware quantization for large language models, 2023.
- [2] Minseop Park, Jaeseong You, Markus Nagel, and Simyung Chang. Quadapter: Adapter for GPT-2 quantization. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2510–2517, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.
- [3] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.
- [4] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [5] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc., 2023.
- [6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [7] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.
- [8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.
- [9] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced BERT with disentangled attention. *CoRR*, abs/2006.03654, 2020.
- [10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher 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. *CoRR*, abs/2005.14165, 2020.
- [11] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2022.
- [12] Qingru Zhang et al. Quadapter: Efficient quantization for gpt-2 via trainable adapters. *arXiv preprint arXiv:2211.10438*, 2022.
- [13] Elias Frantar and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2023.
- [14] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 2023.
- [15] Zhen Li et al. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.
- [16] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics.
- [17] John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. <https://thinkingmachines.ai/blog/lora/>.

A Appendix

Decoder-Only GPT-2 Architecture [6, 7]

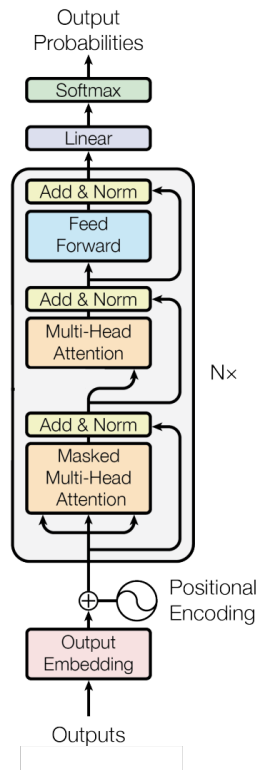


Figure 8: Decoder-Only Transformer