Efficient Fine-tuning and Inferencing of ELECTRA

Allen Chen
Department of Computer Science
Columbia University
New York, USA
atc2160@columbia.edu

***Abstract*—Small organizations and independent researchers, constrained by limited computational resources, often encounter obstacles when seeking to utilize large language models (LLM) for developing practical AI applications. This paper seeks to alleviate some pain points by exploring a cost-effective and efficient large language model development framework at its different stages. At the pre-training stage, we leverage the language model ELECTRA, which is trained in a way that requires less than one fourth of the compute of other similar LLMs such as RoBERTa and XLNet while achieving comparable performance. This makes ELECTRA an ideal candidate for AI applications that require to train a language model from scratch. Next at the fine-tuning stage, we explore Low-Rank Adaptation (LoRA) approach and its quantized variant (QLoRA) to efficiently fine-tune the ELECTRA model. At the inferencing stage, we experiment with the performance optimization methods of quantization and pruning to demonstrate the framework of finding the optimal variant. By combining these efficient approaches, this LLM model development framework aims to help AI applications decrease LLM development time and memory footprint.**

***Index Terms*—Performance Optimization, ELECTRA, LLM, LoRA, QLoRA, Quantization, PTSQ, QAT, Pruning**

I. Introduction

Large language models (LLM) have gained wide popularity recently due to its ability to understand and exhibit strong performance across various downstream tasks. To achieve state-of-the-art results for practical user applications, the LLMs oftentimes require significant computational resources. This would not be possible in resource-constrained environments, especially for smaller institutions and independent research. In this paper, we aim to lower these computational resource requirements by applying different performance optimization approaches to efficient LLMs development.

The rise of pre-trained models has significantly transformed the landscape of building AI applications with LLM. Without having to train LLM from scratch, researchers can now directly utilize pre-trained models, released by larger institutions, to generalize across various NLP tasks and domains, such as text classification and question answering. However, to tailor the pre-trained models to a specific application's task, researchers need to further fine-tune on their specific dataset to achieve satisfactory performance. This still presents a challenge to many. In this study, we apply the techniques of Low-Rank Adaptation (LoRA) and its quantized variant (QLoRA) to efficiently fine-tune LLM. By using LoRA, we reduce the number of trainable parameters by more than 90 percent while maintaining the sub-task's performance.

After fine-tuning of LLm, we deploy the model into production and serve the model predictions for the AI application. Currently there exists a bottleneck during inferencing where these to-be-deployed LLMs are way too big that demand impossible compute requirements. Oftentimes, they need substantial memory resources to store their parameters, large memory bandwidth to move them between CPUs and GPUs, and large cache to load them. This becomes even more important when the models are deployed on resource-constrained devices such as mobile phones or IoT devices. To solve this problem, we turn to quantization, which is a memory-efficient optimization technique that quantizes a floating-point neural network model to its lower-precision representation while keeping its model performance. In this study, we look at Post-Training Static Quantization and Quantization-Aware Training and compare their performances against the standard one.

Another memory-efficient optimization technique that we explore is pruning. It is also a technique to reduce the size and complexity of neural network models by randomly or selectively removing a fraction of the trainable parameters and connections. We specifically look at unstructured and structured pruning with different variants of granularities and criteria and similarly compare their performances against the standard one.

With low-rank adaptation, quantization and pruning, we see a reduced model size for all experiments, which translates to reduced memory footprint, improved computational efficiency, and potentially enhanced energy efficiency.

II. Models and Data Description

A. Model

There are cases where the LLMs need to be trained from scratch. This can happen due to AI applications operating in a specific domain that involves specialized knowledge or subjects to security concerns. It is no secret that this can be wildly expensive. This motivates the selection of the ELECTRA language model in the study.

The ELECTRA language model, which originates in the paper by Clark et al. [1] and stands for Efficiently Learning an Encoder that Classifies Token Replacements Accurately, is a pre-trained model similar to other language models like BERT, but it was trained in a more efficient manner. Instead of the masked language modeling pre-training method utilized by models like BERT, it uses replaced token detection, which trains a discriminative model that predicts a binary outcome whether each token was replaced by a generator sample. This trains the pre-trained model more efficiently at only one fourth of compute of RoBERTa and XLNet while performs comparably. It also mentions that the gains are particularly strong for small models. With these computational advantages, ELECTRA is chosen as our pre-trained model for our study, and we load its small version provided by Google from Huggingface model repository – electra-small-discriminator [5].

B. Data

This study uses the sub-task MRPC of the GLUE benchmark dataset. GLUE stands for General Language Understanding Evaluation, and it provides a collection of datasets for training, evaluating, and analyzing natural language understanding systems [11]. MRPC, which stands for Microsoft Research Paraphrase Corpus [7], is one the benchmark dataset, and the sub-task is to provide a binary prediction of whether the pair of given sentences captures a paraphrase equivalence relationship.

III. Training Methodology

All models in this study are fine-tuned for 20 epochs and run on one T4 GPU. One exception is the inferencing runs on CPU from the quantization experiments due to limited support from PyTorch.

A. Low-Rank Adaptation

In this study, we apply the low-rank adaptation (LoRA) approach proposed by Hu et al. [3] to the ELECTRA model. Traditionally, full fine-tuning of LLM trains all model parameters, and that is roughly 13.5 million parameters in our study with ELECTRA. LoRA proposes a different alternative. It freezes the pre-trained model weights and inserts trainable rank decomposition matrices into each attention layer. In our ELECTRA model, there are 12 “electra layers”, each of which includes a self-attention module and two variants of MLP modules. We follow the standard implementation from the transformers and peft [4] packages developed by Huggingface, inserting the rank decomposition matrices at the attention weights but not at the MLP modules. QLoRA, which is its quantized version and proposed by Dettmers et al. [2], adds 4-bit quantization to further decrease its memory footprint. This also follows the standard implementation from the transformers and peft packages.

B. Quantization

Quantization reduces the numerical precision of the model parameters to decrease model size at the inferencing stage. For both quantization variants in this study, we leverage the quantization module in PyTorch [10] to quantize the model. We originally plan to quantize the whole pre-trained model. However, due to the limited support of the matrix multiply implementation for quantized tensors and the complexity of the custom quantized module’s implementation, we have decided to only quantize the MLP modules and not the self-attention and embedding modules.

Post-Training Static Quantization (PTSQ), as the name suggests, maps the parameters to their lower-bit representations on a trained model. We first specify where to quantize the models, and then we calibrate the trained model with the training dataset to learn the mapping. Finally we use the scales and zero-point learned from calibration to quantize the trained model. For Quantization-Aware Training (QAT), we follow a similar approach. The major difference is that the quantized modules were added before fine-tuning the model, and these quantized modules are actually fake where the data is quantized and immediately dequantized, so the training loss can include quantization error. We follow through with the training and convert to quantized model afterward.

C. Pruning

This study explores multiple variants of pruning techniques with PyTorch [6]. The experiments can be categorized into unstructured and structured pruning, meaning the parameters are eliminated either based on the structure of the model or not. Within each category, we also run additional experiments to explore different criteria and dimensions to formulate the optimal pruning strategy. For unstructured pruning, we first establish baselines with a globally pruned strategy with random and the lowest L1 criteria, and then we increase the granularity by running another experiment with a prune-by-layer strategy. Now with structured pruning, we decide to employ the vector-based pruning granularity because most of the layers in ELECTRA is linear. By varying the pruning criteria of L1 and L2 and pruning vector dimensions of row and column, we run a total of 4 experiments for structure pruning.

D. Profiling

Total fine-tuning time, training data loading time, training time, total inferencing time, testing data loading time are logged in all experiments in this study. We use the time module and its `perf_counter()` function to measure the duration of each stage of interest, and we also place `torch.cuda.synchronize()` to synchronize the threads inside GPU before logging the time. We have decided to abandon Pytorch Profiler [9] for this study because we encounter out of memory error many times and its running time takes a surprising long time.

IV. Performance Tuning Methodology

A. Low-Rank Adaptation Tuning

The parameters Rank and Alpha are tuned in the LoRA experiment. Rank is the hidden dimension at which the weight matrix uses to break into two rank decomposition matrices. This means higher the Rank, higher the number of trainable parameters for the LoRA model. Alpha is a scaling parameter. It is usually a multiple of Rank. The ratio of Alpha / Rank tells us how much to weight the LoRA matrices when they are to be merged with the original weight matrix.

We tune the parameters Rank and Alpha for optimal performance. We start by a low rank value of 16 and stopping

TABLE I
LoRA Results

Stage	**Fine-tune Time**			**Inference Time**		**Model Size**		**Performance Metrics**	
Method	***Dataload***	***Training***	***Total***	***Dataload***	***Total***	***Total Params***	***GPU Mem (MB)***	***Accuracy***	***F1***
Un-finetuned	-	-	-	0.0157	1.1740	13549314	54.2	0.679	0.806
Original	0.0876	11.5591	11.8527	0.0133	0.5331	13549314	54.2	0.870	0.906
LoRA	0.0857	16.3347	16.5231	0.0136	0.7534	1245954	59.2	0.855	0.897
QLoRA	0.0888	16.7899	16.9843	0.0154	1.9507	1213186	22.2	0.684	0.812

TABLE II
Quantization Results

	Inference Time		**Model Size**	**Performance**	
Method	***Dataload***	***Inference***	***GPU Mem***	***Accuracy***	***F1***
Original	0.0246	4.874	54.2	0.870	0.906
PTSQ	0.0229	4.681	39.4	0.833	0.874
QAT	0.0236	4.727	39.4	0.833	0.872

at the value of 128 with an interval of two times the previous number. Figure 1 provides the tuning results. The performance at rank 128 and alpha 256 is only slightly higher than the performance at rank 64 and alpha 64 or 128. We conclude that this slight improvement does not warrant an sizable increase in trainable parameters, and rank 64 is the lowest value that maintains the model performance, so we choose rank 64 and alpha 64 for the reported ELECTRA with LoRA. We further reason that the sub-task and the pre-trained model are equally important as both tasks train to improve its general natural language understanding without focusing on a specific domain. We then select the value of 64 for Alpha. The rank is relatively high because ELECTRA is a relatively smaller model, still requiring a decent number of trainable parameters to perform well.

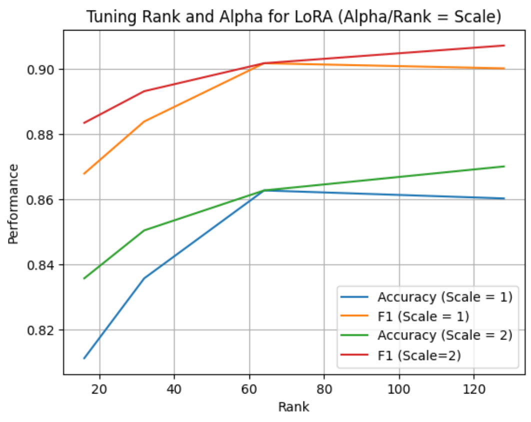

Fig. 1. Plotting model performance vs Rank. Scale is defined as Alpha / Rank.

B. Pruning Tuning

Previously all models are pruned at a pruning amount of 25 percent. Now we take the best performing model, which uses unstructured prune-by-layer pruning strategy with L1 criterion, and tune on its pruning amount. Figure 2 provided the pruning rate vs accuracy chart. After the model was pruned at 35 percent, the performance decreased rapidly. We conclude that the pruning amount of 25 percent remains optimal. For future experiments, the pruning rate tuning strategy can be more refined by tuning-by-layer and using iterative pruning.

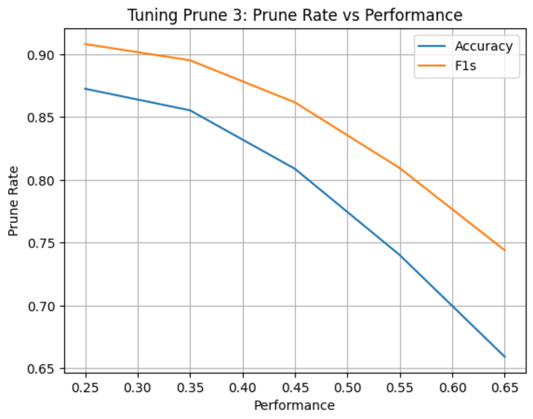

Fig. 2. Plotting model performance vs prune rate for Method Prune 3 (See Table III)

V. Experimental Results

Looking at Table I, ELECTRA with LoRA maintains the original performance while using 9.2% of total trainable parameters from the original model. However, the performance for ELECTRA with QLoRA drops significantly as the addition of 4-bit quantization introduces more errors. One interesting observation is that the fine-tune and inference time for the model with LoRA are slower than those of the original model. This is different from what we expect as reducing the number of trainable parameters does not lower or maintain the computation time. We propose couple hypotheses. This result could be specific to ELECTRA or/and the fine-tune dataset is too small to observe the effect. We test addition runs with a similar but much bigger model BERT and a bigger training dataset by replicating the original one multiple times. When comparing again with those additional changes, the computation times for the model with LoRA remains higher. We believe these hypotheses could further be tested with different models and increasing training dataset.

Table II provides the results for the quantization experiments. Both PTSQ and QAT have a smaller model size and achieve slightly faster inference time while losing just a bit of accuracies and f1 scores. However, it is surprising to see

TABLE III
PRUNING RESULTS

	Strategy					**Inference**	**Performance**	
Method	***Granularity***	***Prune-by***	***Criterion***	***Amount***	***Dimension***	***Total Time***	***Accuracy***	***F1***
Original	-	-	-	-	-	0.5331	0.870	0.906
Prune 1	Unstructured	Global	Random	25	-	0.7281	0.632	0.7180
Prune 2	Unstructured	Global	L1	25	-	0.7062	0.858	0.901
Prune 3	Unstructured	Layer	L1	25	-	0.6999	0.873	0.908
Prune 4	Structured	Layer	L1	25	Row	0.7052	0.654	0.740
Prune 5	Structured	Layer	L2	25	Row	0.7038	0.672	0.752
Prune 6	Structured	Layer	L1	25	Col	0.7269	0.581	0.635
Prune 7	Structured	Layer	L2	25	Col	0.7560	0.650	0.740

PTSQ and QAT achieve the same performance as we expect QAT to yield higher accuracies.

Lastly, we look at Table III for the pruning results. Overall, Prune method number 2 and 3 outperform the rest of the methods and maintain the same accuracies and f1-scores. These results show that Unstructured pruning with L1 pruning criteria is more effective for the linear layers in ELECTRA. On the other hand, we see poor performances across runs with structured pruning along both dimensions.

VI. Discussion

Although all optimization techniques decrease the model size, we are only able to see quantization speed up the inference time while low-rank adaptation and pruning do not. It may seem that it would be inappropriate to optimize the models with low-rank adaptation and pruning in this study. However, there could be other advantages to adding these optimizations, such as efficient storage and data transfer, as long as the marginal increase for the inference time stays reasonable. Therefore, it is crucial for researchers to closely evaluate their AI applications and understand the limitations at different stages. This would allow them to consider all the trade-offs, and thus add the appropriate optimization techniques to the model.

VII. Conclusion

In this study, we investigate three different memory-efficient optimization techniques to reduce large language model size, thereby decreasing its memory footprint and computation time. In the experiments with low-rank adaptation (LoRA), we break the weight matrix into rank decomposition matrices, and demonstrate the fine-tuning on these smaller matrices can also achieve comparable performances. We see the same results for the quantization experiments as well. The reduction of the numerical precision of model's weights yields a faster inference time. Finally, we look at pruning, which adds sparsity by directly eliminating a proportion of model's weights. We learn unstructured pruning with L1 criterion is the most effective pruning method.

We apply these techniques on ELECTRA, a language model trained in one-fourth of the computation time compared to similar large language model that uses the masked language modeling task. By demonstrating different optimization techniques at different stages of large language model development, we hope this creates an efficient framework for future developers who wish to build efficient AI applications.

References

[1] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-trainig text encoders as discriminators rather than generators. *ICLR*, 2020.

[2] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *NIPS*, 2023.

[3] 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. *ICLR*, 2021.

[4] Peft library documentation. https://huggingface.co/docs/peft/main/en/index. Accessed: 2024-05-06.

[5] Electra model on huggingface. https://huggingface.co/google/electra-small-discriminator. Accessed: 2024-05-06.

[6] Pruning tutorial. https://pytorch.org/tutorials/intermediate/pruning_tutorial.html. Accessed: 2024-05-06.

[7] Microsoft research paraphrase corpus. https://www.microsoft.com/en-us/download/details.aspx?id=52398. Accessed: 2024-05-06.

[8] Github repository electra-pytorch. https://github.com/lucidrains/electra-pytorch. Accessed: 2024-05-06.

[9] Pytorch profiler. https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html. Accessed: 2024-05-06.

[10] Practical quantization in pytorch. https://pytorch.org/blog/quantization-in-practice/. Accessed: 2024-05-06.

[11] Alex Wang, Amanpreet Singh, Julian, Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *ICLR*, 2019.