Can we make GPT-2 better?

Stanford CS224N {Default}

Yu Jun Zheng
Stanford University
yujunz@stanford.edu

Yarong Li
Stanford University
yarongli@stanford.edu

Yifan Geng
Stanford University
yifangen@stanford.edu

Abstract

By pretraining GPT-2 on a large amount of text for next-token prediction, it was able to learn linguistic and contextual patterns to form a language understanding that serves as a good initial foundation for various downstream applications. We are going to fine-tune the GPT-2 model for different downstream tasks such as sentiment analysis, paraphrase detection, and sonnet generation. Moreover, given the increasing size of large language models, it is important to explore the benefits of using parameter-efficient fine-tuning techniques, such as LoRA, to improve the efficiency and performance of downstream tasks. Therefore, we are going to apply the LoRA technique on paraphrase detection and sonnect generation to evaluate the difference in performance and efficiency. Our work demonstrated the benefits of utilizing a pre-trained model for fine-tuning to downstream tasks and the advantage of using PEFT methods like LoRA during fine-tuning.

1 Key Information to include

- Mentor: N/A
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

One important process in natural language processing includes the adaptation of large-scale pre-trained language models to multiple downstream tasks through fine-tuning. It is important to evaluate how well we can fine-tune a large language model generalize to a specific downstream task. In addition, there can be significant computational costs when performing full fine-tuning, given that it retrains on all the model parameters. For example, there are 176 billion trainable parameters in GPT3. Hence, it is important to have a mechanism that fine-tunes on a smaller set of parameters and is able to maintain a similar model performance. However, existing techniques, such as adding adapter layers, that aim to lower trainable parameters often introduce inference latency by extending the model's depth or reducing the model's usable sequence length. They also often fail to match performance with the fine-tuning baselines.

Low-Rank Adaptation (LoRA) is a fine-tuning technique that was introduced in 2021. LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into layers of the Transformer architecture for fine-tuning. This contributed to a new theory about how changes in weights during fine-tuning exhibit a low "intrinsic rank". There are several advantages of the LoRA technique. One is that it reduces storage requirements and enhances task-switching efficiency by sharing a pre-trained model and separately fine-tunes small LoRA modules for different tasks that generate lower rank matrices. Secondly, LoRA lowers the hardware barrier in training/fine-tuning since we only need to optimize the injected smaller low-rank matrices. Moreover, by the design of this technique, the technique introduces no inference latency when compared to a fully fine-tuned model. Therefore, we want to compare the difference in performance and efficiency when fine-tuning

with LoRA vs. full fine-tuning. By fine-tuning GPT-2 models to downstream tasks such as paraphrase detection and sonnet generation with LoRA, we discovered that LoRA significantly reduces fine-tuning time by 40% - 80% for both small and medium GPT-2 models while maintaining a similar performance as the fully fine-tuned models.

3 Related Work

Our work primarily focused on re-implementing the key components of the GPT-2 model and fine-tuning the general pre-trained model for downstream tasks, including sentiment analysis, paraphrase detection, and sonnet generation. The foundation for this work was laid by OpenAI's introduction of GPT-1 in the paper *Improving Language Understanding by Generative Pre-Training*[1] in 2018. GPT (Generative Pretrained Transformer) is a decoder-only Transformer model trained using a combination of unsupervised generative pre-training on a diverse corpus of unlabeled text, followed by discriminative supervised fine-tuning. This general, task-agnostic model demonstrated significant performance improvements, highlighting the value of pre-training on a large and diverse corpus of long text. Through this pre-training, the model acquires significant language understanding and world knowledge, as well as the ability to process long-range dependencies, which can then be successfully transferred to discriminative downstream applications.

Building upon GPT-1, OpenAI released GPT-2 in 2019 in the paper *Language Models are Unsupervised Multitask Learners*[2]. By scaling up the size of GPT-1 and training the model on a larger and more diverse dataset, GPT-2 showed improved performance across a wide range of tasks and domains without the need for explicit supervision. In this project, we leverage the pre-trained GPT-2 model and fine-tune it for the three downstream tasks.

To further extend our work, we aim to fine-tune a larger version of GPT-2 (124M parameters), specifically GPT-2 Medium with 355M parameters. However, as the size of the pre-trained model increases, fine-tuning the full model for multiple downstream tasks becomes increasingly challenging. To address this, we employ the Low-Rank Adaptation (LoRA) method, introduced by Hu et al. in 2021 in paper *LoRA: Low-Rank Adaptation of Large Language Models*[3]. LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into layers of the Transformer architecture. This approach is grounded in the previous findings that the learned over-parameterized models reside on a low intrinsic dimension, and the hypothesis that the change in weights during model adaptation also has a low "intrinsic rank". By significantly reducing the number of trainable parameters, LoRA enhances task-switching efficiency and lowers the hardware requirements during the fine-tuning process. Moreover, by the design of the LoRA technique, it introduces no inference latency. In our work, we applied LoRA modules to the self-attention weights in the paraphrase detection and sonnet generation tasks.

4 Approach

We used the original GPT-2 framework to fine-tune the various downstream tasks. GPT-2 is a decoder-only Tranformer model that focuses on the task of next-token prediction. The main components of the GPT-2 model are the tokenization & embedding layer as well as the Transformer layers. We applied full-finetuning on the sentiment analysis task and used the LoRA technique to the self-attention weights when fine-tuning on paraphrase detection and sonnet generation.

4.1 Tokenization & Embedding Layer

The GPT-2 model first uses byte pair encoding (BPE) tokenization to convert input text into different tokens. Then the tokens are fed into a trainable embedding layer with an added trainable positional embedding layer. The embedding layer maps input tokens into vector representations, while the positional embedding layer furnishes the embedding vectors with positional information. Each embedding has a dimension of 768, and GPT-2 can process a maximum number of 1024 tokens in terms of context length.

4.2 Transformer Layers

Each Transforer layer is composed of masked multi-head attention, residual connections, layer norms, and a multi-layer perceptron (MLP) layer. We focus on the masked multi-headed self-attention layer here due to its importance in the Transformer architecture. The multi-head self-attention layer is the concatenation of multiple single-head self-attention output. For each head, GPT-2 computes the dot products of each position's query with all the keys across all the position, scale the scores by $\frac{1}{\sqrt{d_k}}$, then applies a softmax function to obtain the weights for all the values. The output is the weighted average of the value vectors. Along with the multi-headed feature, this self-attention mechanism enables the model to jointly attend to different information across various positions.

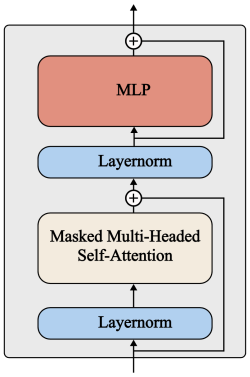

Figure 1: Transformer Layer

The following demonstrates a mathematical formulation of the multi-head attention layer:

$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, \quad W_i^K \in \mathbb{R}^{d_{model} \times d_k}, \quad W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{model}}$

$$\textit{Attention}(Q, K, V) = \textit{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Moreover, there are two masks applied to each single-head attention output. Attention mask is used to mask out padded positions so the output does not attend to information that is beyond the actual length of the current input. Causal self attention makes sure each position only attends to information at or towards its left, so that it will not have access to the actual label during training, which is the immediate next token.

4.3 LoRA Fine-tuning Algorithm

LoRA enables the fine-tuning of dense neural network layers indirectly by optimizing rank decomposition matrices of the layers during fine-tuning, while keeping the original pre-trained weights fixed on the GPT-2 model. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA constrains its update by representing the fine-tuning changes $\Delta W = BA$. A is initialized with a random Gaussian while B is initialized to 0 at the beginning of fine-tuning. The changes in weights are also scaled by $\frac{alpha}{rank}$, with rank being the rank of the two matrices. Parameters in W_0 are frozen throughout fine-tuning. LoRA significantly reduces the number of trainable parameters for downstream tasks to enable efficient fine-tuning. We utilized this technique for fine-tuning the self-attention weights for paraphrased detection and sonnet generation with the PEFT package provided by Hugging Face (https://huggingface.co/docs/diffusers/en/training/lora).

5 Experiments

5.1 Data

1. ***Sentiment Analysis*** We used two datasets, each consisting of full or extracted movie reviews, and leveraged GPT-2's embeddings of the last token of these documents to predict sentiment classifications. We then compared these predictions against the human-annotated labels.
   (a) **Stanford Sentiment Treebank (SST)**: the dataset contains 11,855 single sentences extracted from movie reviews labeled with their sentiment polarity. There are 5 sentiment classes in total, negative, somewhat negative, neutral, somewhat positive, and positive.
   (b) **CFIMDB**: the dataset contains 2,434 multiple-sentence highly polar movie reviews which are labeled as either negative or positive.

   Both datasets are split into train, dev, and test sets with approximately 70%, 10%, and 20% of the total data.
2. ***Cloze-Style Paraphrase Detection*** We used a subset of the **Quora** dataset that consist of 400,000 question pairs labeled to indicate whether one question is a paraphrase of the other. The classification task was reformulated as a binary question asking if one sentence is a paraphrase of the other for each pair, and let GPT-2 output the embeddings of the predicted next token. The embeddings are then mapped to a binary label, yes or no, to be used as the response to the classification task. The dataset is splitted into train, dev, and test sets with 141,506, 20,215, and 40,431 examples respectively.
3. ***Sonnet Generation*** We used a dataset consisting of 156 Shakespearian sonnets. The dataset is first split into training and held-out test sets with 143 and 12 sonnets respectively. We further split the training set into train and validation sets with the ratio specified in *train_size* in order to detect early stopping and avoid overfitting.

5.2 Evaluation method

1. ***Sentiment Analysis & Cloze-Style Paraphrase Detection*:** For the two classification tasks, we used **cross-entropy loss** to measure the distance between the predicted and true distributions during training, and evaluated model performance on the dev and test sets using two metrics, **F1 score** and **accuracy**. Since the true labels for the test set are unavailable to us, the predictions were uploaded to Gradescope for evaluation.
2. ***Sonnet Generation*:** In this multi-token generation task, we used **cross-entropy loss** on both the training and validation sets to measure the distribution distance between the model-generated outputs and Shakespeare's original sonnets during training. At the evaluation stage, we asked the model to complete the sonnets given their first three lines, and assessed how close the generated outputs are to Shakespeare's original sonnets using the **chrF** metric. The chrF metric is a character-level n-gram comparison metric similar to BLEU, defined as the F-score of character n-gram precision and recall.

5.3 Experimental details

1. **Sentiment Analysis** - we trained our model on Google Colab, using T4 GPU.
   (a) We began by leveraging pre-trained embeddings and only training the last linear layer. The training task on SST and CFIMDB datasets took about 30 minutes in total. We used the default configuration ($seed = 11711, epochs = 10, batch\_size = 8, hidden\_dropout\_prob = 0.3, lr = 1e-3$).
   (b) We then fine-tuned the full model using SST and CFIMDB datasets, which took in total about 1 hour. We first tried to train with default configuration, which didn't give satisfactory results. Then, we tuned the parameter to $seed = 11711, epochs = 10, batch\_size = 16, hidden\_dropout\_prob = 0.3, lr = 1e-4$, which gave reasonable results3.
2. **Paraphrase Detection** - we trained our model on GCP VM, using T4 GPU.

   We fine-tuned both GPT-2 and GPT-2 Medium models by utilizing full fine-tuning and the LoRA technique with different matrix ranks. We used the default configuration ($seed = 11711, epochs = 10, batch\_size = 8, lr = 1e-5$). For LoRA, we used $lora\_rank = 16, 32$, $lora\_alpha = 16$, $lora\_dropout = 0.1$; The training times are shown in Table 1.

3. ***Sonnet Generation*** - we trained our model on Google Colab, using T4 GPU.

   We fine-tuned both GPT-2 and GPT-2 Medium models, and experimented with both full fine-tuning and applying the LoRA method with different LoRA ranks. The training times are shown in Table 1. The hyper-parameters were set as follows: (a) Optimizer-related: $batch\_size = 8$, $lr = 1e-5$, $epochs = 20$; (b) LoRA-related: $lora\_rank = 16, 32, 64, 128$, $lora\_alpha = 16$, $lora\_dropout = 0.1$; (c) Early stop-related: $train\_size = 0.9$, $patience = 2$.

   We incorporated the **early-stopping** mechanism into the training process: if the number of consecutive epochs without improvement in validation loss exceeded the *patience* threshold, training was terminated early. During experiments, we observed varying stopping behavior across training processes. In some cases, early stopping was triggered after only very few epochs, potentially prematurely; while in others, it was not activated until the maximum number of epochs was reached. This was particularly evident when fine-tuning the GPT-2 Medium model with LoRA ranks 16, 64, and 128, where training stopped after just 3 epochs, but with LoRA rank 32, early stopping was not triggered. To investigate the impact of hyper-parameters–*patience*, *train_size*, and *learning rate*–on convergence behavior and model performance, we performed a heuristic search. Our search showed that for processes that stopped very early, increasing *patience* and decreasing *learning rate* did not affect convergence, but reducing *train_size* led to early termination in some processes that had not previously early-stopped with larger *train_size*. We will discuss more in the following sections.

Base Model	**Fine-Tuning Method**	**Training Time / Epoch**	
		Paraphrase Detection	**Sonnet Generation**
GPT-2	Full Model	1.6 h	0.7 min
	LoRA (lora_rank=16)	0.9 h	0.33 min
GPT-2 Medium	Full Model	5 h	1.8 min
	LoRA (lora_rank=16)	3 h	0.33 min

Table 1: Training Time per Epoch for Different Models and Fine-Tuning Methods

5.4 Results

1. ***Sentiment Analysis***

   For last-layer fine-tuning, we achieved dev accuracies of 0.453 on the SST dataset and 0.878 on the CFIMDB dataset, comparable to the given baselines. Consequently, we focused on using full finetuing for this downstream task.

   With the default configuration, dev accuracies were 0.410 (SST) and 0.853 (CFIMDB), unexpectedly lower than those from last-layer tuning. The training loss decreased while the training accuracy increased (Figure 2) indicative that the model is learning. However, SST dev accuracy slightly decreased with fluctuations, and significant gaps emerged between training and dev accuracies for both datasets, indicating potential over-fitting and noisy updates. To mitigate this, we decreased the learning rate to 1e-4, increased the batch size to 16, and kept other hyper-parameters unchanged. This improved dev accuracies to 0.504 (SST) and 0.967 (CFIMDB). The reduced train-dev accuracy gap for CFIMDB supported our over-fitting hypothesis.

   The results are reasonable given the label distributions: SST (5 labels, random guess 0.2 accuracy) and CFIMDB (2 labels, random guess 0.5 accuracy). For a model that can effectively learn the underlying pattern and make reasonable predictions, we expect the accuracy to be significantly higher than random guess, which is reached by full fine-tuning with the new configuration.

2. ***Paraphrase Detection***

   As shown in Figure 4, we observe that, with the same fine-tuning method–whether fine-tuning the full model or using LoRA–the larger base model performs better on the dev set. In addition, models partially fine-tuned with LoRA exhibit slightly lower performance compared to fully fine-tuned models, which aligns with the trade-off for reduced training time and lower GPU requirements. Notably, the LoRA fine-tuned model achieves 95% of the fully fine-tuned model's best dev accuracy while utilizing only about 60% of the training time. Furthermore, experiments with varying LoRA ranks show minimal performance differences, as evidenced by nearly overlapping learning curves and consistent best dev accuracies after 10 epochs. We also notice that for the

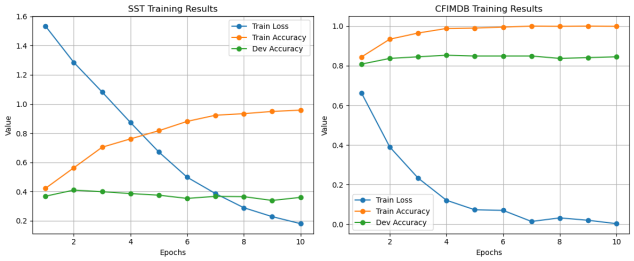

Figure 2: Training Result with Default Configuration

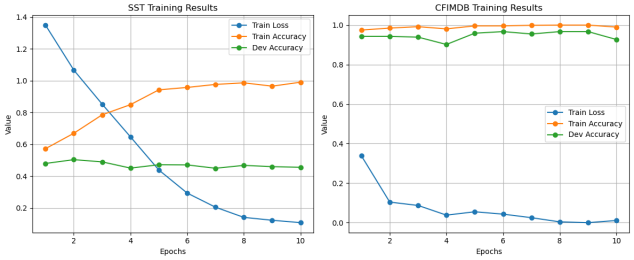

Figure 3: Training Result with New Configuration

GPT-2 model with full fine-tuning, there has been no improvement in dev accuracy after the epoch 6, suggesting potential overfitting. The fully fine-tuned GPT-2 Medium model with 4 epoches achieved dev accurracy of 0.902 and test accuracy of 0.872.The LoRA fine-tuned GPT-2 Medium model with 10 epoches and 16 rank achieved dev accuracy of 0.881 and test accuracy of 0.826.

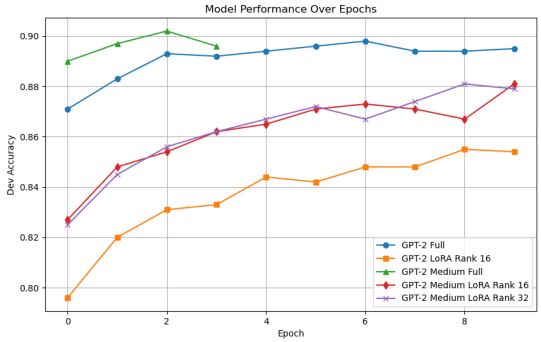

Figure 4: Dev Accuracy vs. Epochs for Paraphrase Detection.

3. ***Sonnet Generation***

First, we fine-tuned the GPT-2 model using the default settings to establish a baseline, which achieved a chrF score of 35.058 on the dev set. We then fine-tuned the GPT-2 Medium model on the same task, which resulted in a chrF score of 41.213.

Next, we explored hyper-parameter tuning, focusing on the number of training epochs. Since the early stopping mechanism with patience = 2 was not triggered in the baseline run, we set the maximum number of epochs to 20. Both models stopped training at epoch 12, with the GPT-2 model achieving a chrF score of 40.459 and GPT-2 Medium reaching 41.440 on the dev set. As a result, we selected the fully fine-tuned GPT-2 Medium model trained for 12 epochs as our final model, which achieved a chrF score of 41.859 on the test set.

Finally, we implemented the LoRA extension on both models and experimented with different LoRA ranks. The results are summarized in Table 2. We observed that the dev scores were

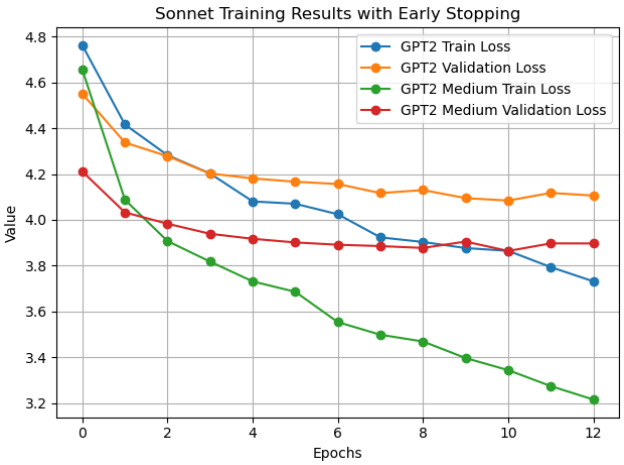

Figure 5: Train and Validation Loss for Sonnet Generation

significally lower when early stopping was not triggered (indicated by *). However, when we reduced *train_size* to 0.8 and early stopping was triggered for LoRA rank 16, the model achieved comparable results to others. This suggests that the early stop mechanism played a key role in improving model performance on the dev set during LoRA fine-tuning for sonnet generation.

Dev Score	**Paraphrase Detection**			**Sonnet Generation**				
	Full	l_rank **16**	l_rank **32**	**Full**	l_rank**16**	l_rank **32**	l_rank **64**	l_rank **128**
GPT-2	0.898	0.855	-	40.459	39.419	37.651*	39.970	36.269*
GPT-2 Medium	0.902	0.881	0.881	41.444	39.271	37.332*	40.304	38.696
GPT-2 Medium (ts 0.8)	-	-	-	-	39.899	39.123	39.824	39.571

Table 2: Dev score for different fine-tuned models in 2 down stream applications. ts = $train\_size$

6 Analysis

- ***Early Stopping in Sonnet Generation***

  The approach of increasing the maximum number of training epochs while incorporating an early stopping mechanism proved effective for this task. It led to performance improvements for both the GPT-2 and GPT-2 Medium models, with the GPT-2 model benefiting more significantly. This is expected, as smaller models tend to learn more from additional epochs compared to larger ones. In addition, we found that fine-tuning sonnet generation task is very sensitive to overfitting and the early stop mechanism helps boost the performance. Notably, validation losses fluctuated in later epochs, indicating that stopping training as soon as the validation loss increases may not always be ideal. Instead, allowing the model some patience to adjust is beneficial. This is supported by the comparison between stopping at epoch 8 (patience = 1, stopping at the first increase in validation loss) and epoch 12 (patience = 2), which resulted in chrF scores of 41.213 and 41.440, respectively, on the development set.

- ***Parameter-Efficient Fine-Tuning with LoRA***

  In this work, we also examined the impacts of fine-tuning with LoRA, with results presented in Table 2. All partially fine-tuned models using LoRA performed slightly worse than the fully fine-tuned model, but the differences were minimal. Importantly, LoRA significantly reduced both training time and memory requirements, as shown in Table 1. Therefore, LoRA proves to be an effective technique for decreasing computational costs without substantially compromising the model performance.

  A key observation is that in both paraphrase detection and sonnet generation tasks, increasing the LoRA rank led to no improvement or even negative impact on model performance, as shown in Table 2. This finding seems to contrast with the expectation that higher ranks, by more closely

approximating full updates, would yield better results. However, this behavior aligns with the hypothesis of the LoRA framework that the change in weights during model adaptation has a low "intrinsic rank". It appears that a rank of 16 may have already captured most of the necessary adaptations for the downstream tasks, with further increases in rank providing minimal benefits and potentially causing overfitting. While full fine-tuning outperformed partial fine-tuning with LoRA, seemingly contradicting the overfitting hypothesis, it's important to note that LoRA was only applied to the self-attention and the final linear layers here. Therefore, the performance gains from full fine-tuning may be attributed to updates in other layers not included in our LoRA adaptation.

7 Conclusion

In this study, we fine-tuned GPT-2 and GPT-2 Medium models on three downstream tasks:sentiment analysis, paraphrase detection, and sonnet generation. With full fine-tuning, all three tasks resulted in above based-line performance, which demonstrates the benefits of fine-tuning pre-trained large language models. Furthermore, to enhance training efficiency, we applied Low-Rank Adaptation (LoRA) method on paraphrase detection and sonnet generation, which reduced training time to 20%-60% of full fine-tuning durations shown in Table 1. With the significant decrease in training time, models fine-tuned with LoRA were able to achieve a similar performance to fully, fine-tuned models, that is around 90% - 95% in Table 2. These results underscore the advantages of Parameter-Efficient Fine-Tuning techniques (PEFT), particularly LoRA, when training models for various downstream tasks. Moreover, we observed that increasing the LoRA rank beyond 16 did not yield performance gains and sometimes negatively impacted results. This observation aligned with LoRA's hypothesis of low "intrinsic" rank in weight changes during adaptation and potential overfitting. The degraded performance of fine-tuning with LoRA compared to full fine-tuning might be attributed to our implementation choice where LoRA adaptations were applied only to the self-attention and final linear layers, leaving all other weights unchanged. Therefore, the performance gains from fine-tuning could result from updates to these unmodified layers. One potential direction for future work is to include additional layers during fine-tuning a large language model to assess their impact on model performance across diverse downstream tasks.

Team contributions (Required for multi-person team)

Yu Jun Zheng: Contributed to developing the base GPT-2 model and the optimizer algorithm for fine-tuning the downstream tasks. Implemented the LoRA fine-tuning technique for the paraphrase detection application. Contributed to fine-tuning and analyzing performance/results of the paraphrase detection task and writing the Abstract, Introduction, Approach, and Conclusion sections of the project report.

Yarong Li: Contributed to developing the base GPT-2 model. Implemented models on down stream application tasks sentiment analysis and sonnet generation application. Contributed to optimizing and evaluating model performance on sonnet generation task, and writing the Results, Analysis, and Conclusion sections of the project report.

Yifan Geng: Contributed to developing the base GPT-2 model. Implemented the paraphrase detection application and the LoRA fine-tuning logic for sonnet generation. Tested and fine-tuned the models for sonnet generation and analyzed the model behavior with different model configurations. Contributed to Related Work, Experiments, Results, Analysis, and Conclusion sections of the project report.

References

[1] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[3] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.