

# Parameter-Efficient Fine-Tuning for GPT-2: Comparing LoRA and ReFT on Paraphrase Detection

Stanford CS224N Default Project

**Ngoc Nguyen**

Department of Computer Science  
Stanford University  
ngocng@stanford.edu

**Matthias Jiro Walther**

Department of Computer Science  
Stanford University  
jiro@stanford.edu

## Abstract

We implement components of the GPT-2 architecture and evaluate three adaptation strategies — full fine-tuning, Low-Rank Adaptation (LoRA), and Representation Fine-Tuning (ReFT) — on two tasks: paraphrase detection and sonnet generation. On paraphrase detection, full fine-tuning achieves 88.9% development accuracy on GPT-2 small, while LoRA closely matches this at 87.3% using under 1.1% of the trainable parameters. Scaling to GPT-2 medium with LoRA reaches 89.5%, matching the small model’s full fine-tuning ceiling at a fraction of the cost. ReFT underperforms with a single-layer configuration but recovers to 81.0% when interventions are applied across all layers. We also implement LoRA from scratch and verify it against the library baseline. For sonnet generation, we fine-tune GPT-2 using autoregressive next-token prediction and generate completions using top- $p$  sampling, achieving a test chrF score of 41.24. Our results suggest that LoRA is a practical and parameter-efficient alternative to full fine-tuning, while ReFT requires careful configuration to be competitive.

## 1 Key Information to include

- Mentor: Simon Kim
- Late Days (if you are using any):

| Team member                      | Late days remaining |
|----------------------------------|---------------------|
| Ngoc Nguyen                      | 5                   |
| Matthias Jiro Walther            | 3                   |
| Total pooled                     | 8                   |
| Team size                        | 2                   |
| Late days applied for the report | 3                   |

## 2 Introduction

Full fine-tuning is a standard way to adapt pretrained language models to downstream tasks, but it becomes increasingly expensive as models grow larger. Parameter-efficient fine-tuning (PEFT) methods aim to reduce this cost by training only a small number of additional parameters while keeping most pretrained weights frozen.

In this project, we compare three adaptation strategies for GPT-2: full fine-tuning, Low-Rank Adaptation (LoRA) [1], and Representation Fine-Tuning (ReFT) [2]. We evaluate them primarily on paraphrase detection and also apply GPT-2 to sonnet generation. Our main goal is to study the tradeoff between task performance, trainable parameter count, and training cost.

We find that LoRA performs nearly as well as full fine-tuning while updating under 1.1% of the parameters on GPT-2 small, and that LoRA on GPT-2 medium matches the accuracy of full fine-

tuning on GPT-2 medium. ReFT performs substantially worse in a single-layer setup, but improves when interventions are applied across all layers.

### Summary of Contributions

- We implement core components of the GPT-2 architecture and apply the model to downstream NLP tasks.
- We fine-tune GPT-2 on paraphrase detection and sonnet generation datasets.
- We extend the baseline model with parameter-efficient fine-tuning methods including LoRA and ReFT.
- We compare three adaptation strategies for GPT-2 on paraphrase detection: full fine-tuning, LoRA, and ReFT with learning rate and PEFT hyperparameter sweeps.
- We evaluate each method in terms of development accuracy, training time, and percentage of trainable parameters.
- We perform extensive experiments including adapter placement ablations and scaling experiments with larger base models.
- We implement LoRA from scratch and verify it against the library baseline.

## 3 Related Work

Models such as BERT [3] and GPT-2 [4] are typically trained on large text corpora and later adapted to specific tasks through fine-tuning. While this approach achieves strong performance, updating all model parameters can be expensive in terms of compute and memory, especially for larger models.

To address this issue, a number of parameter-efficient fine-tuning methods have been proposed. These methods keep the pretrained weights mostly frozen and instead introduce a small number of task-specific parameters. Examples include adapter layers [5], prefix tuning [6], and prompt tuning [7].

Low-Rank Adaptation (LoRA) [1] is a PEFT approach that models weight updates using low-rank matrices. Instead of modifying the original transformer weights directly, LoRA learns small low-rank updates that are inserted into existing layers. This allows the model to adapt while training only a small fraction of the parameters.

More recently, Representation Fine-Tuning (ReFT) [2] proposed an alternative strategy. Rather than modifying weights, ReFT learns interventions on intermediate hidden representations inside the transformer. This allows the model to adapt to a task by modifying internal activations.

Although both LoRA and ReFT aim to improve parameter efficiency, there has been limited direct comparison between these approaches under the same experimental setup. In this work, we evaluate full fine-tuning, LoRA, and ReFT on GPT-2 and analyze the tradeoff between performance and training efficiency.

## 4 Approach

### 4.1 GPT-2 Backbone

Our model uses a custom implementation of GPT-2 whose parameters are initialized from pretrained Hugging Face GPT-2 weights [4]. GPT-2 is a decoder-only transformer that models each token conditioned on its left context.

The backbone consists of token embeddings, positional embeddings,  $L$  causal transformer blocks, and a final layer normalization. Each transformer block applies masked multi-head self-attention and a feed-forward network, with residual connections, dropout, and pre-layer normalization.

Given a tokenized input sequence  $x_{1:T}$ , the backbone produces contextual hidden states

$$H = f_{\theta}(x_{1:T}) \in \mathbb{R}^{T \times d},$$

where  $d$  is the hidden size and  $f_{\theta}$  denotes the GPT-2 backbone.

For downstream paraphrase detection, we do not add a separate classification head. Instead, we use GPT-2’s tied input-output embedding matrix to project the final hidden state into vocabulary logits. This preserves the pretrained vocabulary space and enables cloze-style prediction with answer tokens such as yes and no.

## 4.2 Task Formulation

After implementing the GPT-2 architecture, we fine-tune the model on tasks provided in the assignment. These include paraphrase detection using the Quora Question Pairs dataset and sonnet generation using a dataset of Shakespeare’s sonnets.

### 4.2.1 Paraphrase Detection

We treat paraphrase detection as a cloze-style language modeling task. Given a pair of questions  $(q_1, q_2)$ , we convert the pair into a prompt asking whether the two questions have the same meaning. The model then predicts a final answer token corresponding to either ‘yes’ or ‘no’.

Let  $x_{1:T}$  denote the tokenized prompt for a question pair and let

$$y^{tok} \in \{\text{id}(\text{yes}), \text{id}(\text{no})\}$$

be the target token ID.

Since sequences are padded inside each batch, we extract the hidden state at the final non-padding position, denoted  $h^*$ . Vocabulary logits are then computed using the GPT-2 embedding matrix  $E \in \mathbb{R}^{|V| \times d}$ :

$$z = h^* E^\top$$

Training minimizes cross-entropy on the correct yes/no token:

$$\mathcal{L}_{para} = -\log \frac{\exp(z_{y^{tok}})}{\sum_{v \in V} \exp(z_v)}$$

At inference time, we predict

$$\hat{y} = \arg \max_{v \in V} z_v.$$

On labeled evaluation data, the prediction is counted as correct if  $\hat{y}$  matches the token ID of the gold yes/ no label.

### 4.2.2 Sonnet Generation

For the sonnet generation task, the model is conditioned on the first three lines of a Shakespeare sonnet and asked to generate the remaining lines. Because GPT-2 is a generative language model, this task follows the standard next-token prediction setup used during pretraining. Given a full sonnet tokenized as a sequence  $x_{1:T}$ , the model is trained to minimize the autoregressive language modeling loss:

$$\mathcal{L}_{sonnet} = -\sum_{t=1}^{T-1} \log P_\theta(x_{t+1} \mid x_{1:t}) \quad (1)$$

where  $P_\theta(x_{t+1} \mid x_{1:t})$  is computed by passing the hidden state at position  $t$  through the tied embedding matrix and applying a softmax.

At inference time, the model is given the first three lines as a prompt and generates the remaining tokens autoregressively using top- $p$  sampling with  $p = 0.9$  and temperature  $\tau = 1.2$ . Generation terminates when the end-of-sequence token is produced or a maximum of 128 tokens is reached. The generated text is evaluated using the chrF metric, which measures character  $n$ -gram overlap between the generated and reference sonnet.

## 4.3 Baseline: Full Fine-Tuning

Our baseline is full fine-tuning, where all GPT-2 parameters are updated on the downstream task.

#### 4.4 Extension: Parameter-Efficient Fine-Tuning Methods

We compare two parameter-efficient alternatives that keep most GPT-2 weights frozen and train only a small number of additional parameters: LoRA [1] and ReFT [2].

#### 4.5 LoRA

LoRA modifies the model by inserting low-rank adapters into existing linear layers while keeping the original weights frozen.

For a pretrained weight matrix  $W \in \mathbb{R}^{d_{out} \times d_{in}}$ , LoRA models the update  $\Delta W$  using a low-rank factorization:

$$W' = W + \Delta W$$

$$\Delta W = \frac{\alpha}{r} BA$$

where  $A \in \mathbb{R}^{r \times d_{in}}$ ,  $B \in \mathbb{R}^{d_{out} \times r}$ ,  $r$  is the rank of the adapter, and  $\alpha$  is a scaling factor.

Since  $r$  is much smaller than the original matrix dimensions, the number of trainable parameters is significantly reduced compared to full fine-tuning.

In our implementation, LoRA adapters are added to both attention and feed-forward projections within each GPT-2 transformer block.

#### 4.6 ReFT

ReFT takes a different approach by learning low-rank interventions on internal representations rather than updating the base model’s weight matrices.

Rather than updating parameters directly, ReFT learns small intervention functions that operate on hidden states inside the transformer. If  $h_{\ell,t} \in \mathbb{R}^d$  denotes the hidden state at layer  $\ell$  and token position  $t$ , ReFT applies an intervention

$$\tilde{h}_{\ell,t} = h_{\ell,t} + g_{\phi}(h_{\ell,t})$$

where  $g_{\phi}$  is a learned function constrained to a low-rank subspace. In our experiments the intervention is applied at the output of a transformer block, typically at the final layer and final token position.

Compared to LoRA, which inserts low-rank updates into weight matrices, ReFT modifies intermediate representations inside the model.

#### 4.7 Implementation and Technical Contribution

Our project builds on the course starter code for the custom GPT-2 architecture and training pipeline. We use pretrained weights and tokenization from Hugging Face Transformers [8], the Hugging Face `peft` library for LoRA [9], and the Stanford NLP `pyreft` library for ReFT [10].

Our main technical contribution is extending this custom GPT-2 codebase to support both LoRA and ReFT inside our own training and evaluation pipeline. This required a fair amount of engineering because our setup is different from the typical Hugging Face Trainer workflow that many PEFT libraries assume: we integrate LoRA directly into the custom GPT-2 implementation, add ReFT support by exposing hookable layer outputs inside our transformer blocks, manually construct intervention locations inside our training loop, and implement checkpointing and resuming so that both LoRA and ReFT training runs can be saved and continued.

We also implemented LoRA from scratch as a correctness check and learning exercise. Under the same hyperparameters used for the library baseline (GPT-2 small, rank 8,  $\alpha = 32$ , dropout 0.05, learning rate  $10^{-4}$ , 3 epochs), our implementation reached 86.5% development accuracy compared to 87.35% for the `peft` version. For the main experiments in this paper, we use the library implementations to ensure stable comparisons.

## 5 Experiments

### 5.1 Data

We evaluate our methods on two tasks: paraphrase detection and sonnet generation.

For paraphrase detection, we use the Quora question-pair data provided with the project, which is based on the Quora Question Pairs dataset. The dataset is split into `quora-train.csv` (283,003 examples), `quora-dev.csv` (40,429 examples), and `quora-test-student.csv` (80,858 examples). Each example contains two questions, `sentence1` and `sentence2`. For the train and dev sets, the label `is_duplicate` indicates whether the two questions have the same meaning.

In our setup, we convert each pair of questions into a cloze-style prompt and ask the model to predict the final answer token, either `yes` or `no`. So the task becomes binary paraphrase detection: the input is two natural language questions and the model predicts whether they are semantically equivalent.

For sonnet generation, we use a corpus of Shakespeare’s sonnets from the Folger Shakespeare Library edition. In our project split, the file `sonnets.txt` contains 131 sonnets used for training. We also use prompt files `sonnets_held_out.txt` and `sonnets_held_out_dev.txt`, each containing 12 incomplete sonnets that include only the first three lines. The file `TRUE_sonnets_held_out_dev.txt` contains the reference completions for the dev prompts.

During training, the model is given full sonnets and trained autoregressively using next-token prediction. At generation time, the task is to complete a sonnet: the input is the first three lines of the poem, and the model generates the rest of the sonnet as a continuation.

### 5.2 Evaluation method

For paraphrase detection, we evaluate our models on the development set using accuracy and macro-F1. Accuracy measures the fraction of question pairs the model predicts correctly. Macro-F1 computes the F1 score for both classes (paraphrase and non-paraphrase) and then averages them, which helps check that the model performs reasonably on both labels.

During evaluation, the model outputs logits for each example and we choose the label with the highest value as the prediction. We then compute accuracy and macro-F1 using the `sklearn.metrics` library. We report both metrics on the development set, but we select the best model based on development accuracy. For the test set, the true labels are not provided, so we only generate predictions for submission instead of computing metrics.

For sonnet generation, we evaluate the outputs using the chrF metric. chrF is a character-level similarity metric that compares the generated sonnets with the original Shakespeare sonnets in the dataset. We compute a corpus-level chrF score using the `sacrebleu` library. Higher chrF scores mean the generated text is more similar to the reference sonnets.

### 5.3 Experimental details

All experiments were run on Modal A100-80GB GPUs, and we fixed the random seed to 11711 for reproducibility. For paraphrase detection, unless otherwise noted, runs used 3 epochs and a batch size of 32.

We first performed a learning-rate sweep on GPT-2 small for full fine-tuning, LoRA, and ReFT using learning rates  $\{10^{-3}, 5 \times 10^{-4}, 10^{-4}\}$ . The best learning rate for each method was then used as the default for later experiments. We repeated this procedure on GPT-2 medium, starting from the best GPT-2 small learning rate and additionally testing  $0.5\times$  and  $0.1\times$  of that value.

For each run, we recorded the development-set metric, the number of trainable parameters, the percentage of trainable parameters, and the wall-clock training time. GPU hours were computed by dividing wall-clock time in seconds by 3600.

For the PEFT methods (LoRA and ReFT), we used the best learning rate found in the sweep and then performed a series of ablation studies. First, we studied where the methods should be applied in the transformer. For LoRA, we compared adapters applied only to the MLP layers, only to the attention layers, and to both; full results are shown in Appendix A.2. For ReFT, we varied the intervention

location by applying it to the first half of the transformer layers, the second half, the middle layers, or all layers.

After identifying the best-performing layer configurations from these ablations, we then performed a rank sweep measured performance as a function of rank. For LoRA, we evaluated ranks  $\{4, 8, 16\}$ . For ReFT, we evaluated ranks  $\{1, 4, 8\}$ .

For sonnet generation, we trained on the full set of 131 sonnets for 10 epochs with a batch size of 8, and a learning rate of  $10^{-5}$ . We used full fine-tuning of GPT-2 small, updating all model parameters with the autoregressive next-token prediction loss. For generation, we used top- $p$  sampling with  $p = 0.9$  and temperature  $\tau = 1.2$ , with a maximum generation length of 128 tokens. The generation terminated early if the end-of-sequence token was produced. Afterwards, we evaluated the generated completions against the reference sonnets using the corpus-level chrF metric.

## 5.4 Results

**Sonnet Test chrF:** 41.244

**Paraphrase Test Accuracy:** 0.871

**Learning Rate Sweep Results.** We evaluated several learning rates for full fine-tuning, LoRA, and ReFT on GPT-2 small. We found that  $10^{-4}$  consistently produced the best development performance across all three methods, and we therefore used this value as the default learning rate for subsequent experiments.

We then performed a smaller sweep for GPT-2 medium using  $0.5\times$  and  $0.1\times$  this learning rate as we hypothesized that bigger models might perform better using smaller learning rates [11]. We actually found that  $10^{-4}$  was still the best learning rate for GPT-2 medium. The resulting performance trends are shown in Appendix 2 and Appendix 3.

| Name            | Total Params | Trainable Params | Rank | Alpha | Dropout | Dev Acc | GPU-hrs |
|-----------------|--------------|------------------|------|-------|---------|---------|---------|
| gpt2-small-full | 126,357,504  | 126,357,504      | –    | –     | –       | 0.889   | 1.07    |
| gpt2-small-lora | 126,357,504  | 1,327,104        | 8    | 32    | 0.05    | 0.873   | 0.92    |
| gpt2-small-reft | 125,036,548  | 6,148            | 4    | –     | –       | 0.685   | 0.44    |
| gpt2-med-full   | 355,872,768  | 355,872,768      | –    | –     | –       | 0.895   | 3.04    |
| gpt2-med-lora   | 359,411,712  | 3,538,944        | 8    | 32    | 0.05    | 0.895   | 2.47    |
| gpt2-med-reft   | 355,880,964  | 8,196            | 4    | –     | –       | 0.722   | 1.13    |

Table 1: Comparison of full fine-tuning, LoRA, and ReFT on the paraphrase detection task. GPU-hours report the total wall-clock training time (including validation) for three epochs on a single GPU. ReFT was only applied to the last layer.

| Configuration    | Layers      | Trainable Params | Dev Acc       |
|------------------|-------------|------------------|---------------|
| ReFT-first_half  | 0–5         | 36,888           | 79.22%        |
| ReFT-middle      | 3–8         | 36,888           | 80.30%        |
| ReFT-second_half | 6–11        | 36,888           | 80.32%        |
| <b>ReFT-all</b>  | <b>0–11</b> | <b>73,776</b>    | <b>80.98%</b> |

Table 2: ReFT layer placement ablation on GPT-2 small (rank=4, lr= $10^{-4}$ , 3 epochs). Interventions at all layers outperform any single half, motivating the use of `reft_layers=all` in subsequent rank sweep experiments.

**Rank Sweep — GPT-2 Small — Paraphrase Detection**  
 $lr=10^{-4}$ , epochs = 3, batch = 32, Quora dev set

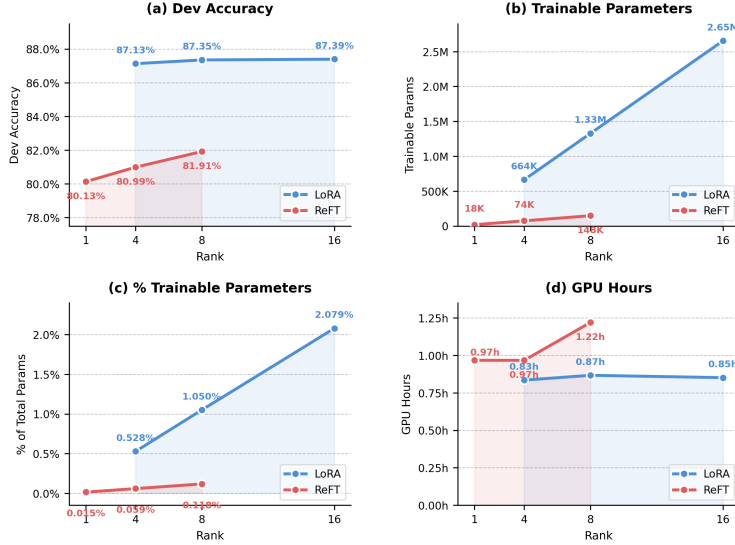


Figure 1: Rank sweep on GPT-2 small for paraphrase detection. LoRA ranks {4, 8, 16} and ReFT ranks {1, 4, 8}, all layers,  $lr=10^{-4}$ , 3 epochs.

To understand where ReFT interventions are most effective, we ran a layer placement ablation varying which of the 12 layers received interventions. As shown in Table 2, the three partial configurations perform surprisingly similarly, with all-layer coverage giving the best result (80.98%) at the cost of doubling the parameter count. Full fine-tuning of GPT-2 small achieves 88.9% dev accuracy, as expected given unrestricted parameter updates. LoRA closely matches this while updating fewer than 1.1% of parameters, which is broadly in line with what the literature would predict for low-rank adaptation on a classification-style task. More surprisingly, scaling to GPT-2 medium with LoRA reaches 89.5%, matching the small model’s full fine-tuning ceiling at a fraction of the trainable parameter count — better than we expected from PEFT alone. ReFT, on the other hand, underperforms relative to our expectations: with a single-layer intervention it reaches only 68.5%, well below the other methods. This was unexpected given ReFT’s strong reported results in the original paper, and suggests that our single-layer setup was too restricted for this task. Expanding to all 12 layers recovers much of the gap (81.0%), which confirms that the weak baseline result was a configuration issue rather than a fundamental limitation of the method.

The rank sweep (Figure 1) reinforces these findings. For LoRA, performance is largely stable across ranks {4, 8, 16}, with rank 4 already achieving 87.1% and rank 16 adding only 0.3 points despite quadrupling the parameter count. This suggests that the task does not require high-rank updates — the pretrained representations are already close to what is needed, and only a small low-rank nudge is required. For ReFT, the trend is different: accuracy improves more steadily from rank 1 (80.1%) to rank 8 (81.9%), suggesting that ReFT benefits more from additional expressiveness. However, even at rank 8, ReFT uses only 148K parameters compared to LoRA’s 664K at rank 4, and still falls short of LoRA’s accuracy by around six points. The LoRA layer ablation (Appendix A.2) further shows that applying adapters to both attention and MLP layers (87.35%) outperforms either alone (85.79% and 85.99%), suggesting that both components contribute meaningfully to adaptation.

## 6 Analysis

Looking at the dev set predictions, the model predicts “paraphrase” 61.2% of the time, even though the true positive rate in the Quora dataset is only about 37%. This suggests the model has a strong positive bias. One likely reason is the cloze-style prompt, which may push GPT-2 toward affirmative completions because of how it was pretrained. In practice, many of the errors are false positives: the model marks pairs as paraphrases when they share similar keywords or topics but do not actually

mean the same thing. This suggests the model is relying more on lexical overlap than deeper semantic reasoning.

Comparing the three training strategies also reveals some differences beyond the overall accuracy numbers. As shown in Table 1, LoRA achieves almost the same dev accuracy as full fine-tuning while training far fewer parameters. For example, GPT-2 small with full fine-tuning updates 126M parameters and reaches 0.889 dev accuracy, while LoRA trains only about 1.3M parameters but still reaches 0.873. When scaling up to GPT-2 medium, LoRA reaches 0.895 dev accuracy, matching the performance of full fine-tuning while training less than 1% of the model parameters.

LoRA likely works well here because paraphrase detection is a relatively simple task for a large pretrained language model. The final prediction is just a binary decision at a single token position, and GPT-2’s pretrained representations already capture a lot of semantic similarity information. Because of this, the model may only need a small adjustment to adapt to the task. This is supported by the rank sweep (Figure 1), where increasing the LoRA rank from 4 to 16 only improves dev accuracy by about 0.3 points. In contrast, scaling the base model from GPT-2 small to GPT-2 medium gives a larger improvement (about 2.2 points), suggesting that the pretrained model itself matters more than the adapter size.

The ReFT results show a different pattern. When ReFT is applied at only a single layer, the performance is much lower than the other methods. However, the layer ablation experiment shows that applying ReFT to all layers improves performance quite a bit. This suggests that changing the representation at only one layer may be too restrictive, and that the model needs interventions at multiple layers to effectively adapt to the task.

From a training efficiency perspective, LoRA also has a clear advantage in memory usage. Full fine-tuning requires storing gradients for every parameter in the model, which becomes very expensive as the model size increases. In contrast, LoRA freezes the GPT-2 weights and only computes gradients for the small adapter matrices. This makes it possible to train larger models like GPT-2 medium even when full fine-tuning might exceed GPU memory limits. This is especially valuable because larger pretrained models generally learn richer language representations during pretraining, which often leads to better downstream performance[12].

However, fewer trainable parameters does not always mean faster training. In our experiments, the runtime was mostly dominated by forward passes through the base model. Even with LoRA, the training loop still runs inference through the entire GPT-2 network at every step. Because of this, GPT-2 medium with LoRA can still take longer to train than GPT-2 small with full fine-tuning simply because the underlying model is larger. For ReFT, applying interventions at every layer also adds extra computation at each transformer block, which increases GPU time even though the number of trainable parameters is very small.

For the sonnet generation task, looking at the generated outputs gives some insight into what the model learned. The generated sonnets are usually strongest at the beginning, where the model is directly conditioned on the prompt. As generation continues, the rhyme scheme often breaks down and the theme becomes less coherent. For example, one generated sonnet starts with a reasonable continuation of a grief-related theme from the prompt, but by the third quatrain the imagery becomes unrelated. This suggests the model has learned local phrasing and vocabulary from the 131 training sonnets, but it struggles to maintain longer-range structure across the entire poem. The chrF score of 41.24 reflects this behavior: the generated text looks somewhat similar to the reference sonnets on a surface level but does not consistently follow the full sonnet structure.

## 7 Conclusion

In this project, we compared three strategies for adapting GPT-2 to downstream tasks: full fine-tuning, LoRA, and ReFT. Our experiments show that LoRA can achieve performance comparable to full fine-tuning while training only a very small fraction of the model parameters. In particular, LoRA matched the performance of full fine-tuning on paraphrase detection while updating less than 1% of the parameters. We also observed that scaling the base model from GPT-2 small to GPT-2 medium produced larger improvements than increasing the LoRA rank, suggesting that the quality of the pretrained representations matters more than adapter capacity for this task.

Our results also highlight important differences between PEFT methods. ReFT underperformed in our initial single-layer configuration, but improved substantially when interventions were applied across multiple layers. This suggests that broad coverage across the network may be more important than intervention expressiveness for adapting representations in this setting.

For the sonnet generation task, the model produced Shakespearean-style language in the opening lines but often lost rhyme structure and thematic coherence across the full poem. This indicates that the model learned local stylistic patterns from the training data but struggled to maintain longer range structure.

At the same time, our work has several limitations. Our experiments focused on a relatively simple classification task, and for the main experiments we relied on existing libraries for the LoRA and ReFT implementations. While this allowed us to run stable comparisons, implementing both methods fully from scratch could provide deeper insight into how these approaches modify transformer behavior.

Overall, our results suggest that parameter-efficient fine-tuning methods such as LoRA can provide strong performance while significantly reducing training cost. These approaches make it possible to adapt larger pretrained models without the computational expense of full fine-tuning.

## Team contributions

Both members contributed equally to implementing the GPT-2 architecture, the paraphrase detection and sonnet generation pipelines, the LoRA implementation from scratch, and the writing of this report. Matthias implemented the AdamW optimizer, the ReFT layer ablation, the LoRA layer ablation, and the rank sweep. Ngoc implemented the LoRA and ReFT integrations using the `peft` and `pyreft` libraries, implemented checkpointing and resuming, ran the learning rate sweep, scaled experiments to GPT-2 medium, and handled sonnet generation training and evaluation.

## References

- [1] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [2] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Reft: Representation finetuning for language models. *arXiv preprint*, 2024.
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.
- [4] Alec Radford, Jeffrey Wu, Rewon Child, and et al. Language models are unsupervised multitask learners. 2019.
- [5] Neil Houlsby and et al. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.
- [6] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL*, 2021.
- [7] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021.
- [8] Hugging Face. Transformers: State-of-the-art machine learning for pytorch, tensorflow, and jax. <https://github.com/huggingface/transformers>, 2024. GitHub repository.
- [9] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. PEFT: State-of-the-art parameter-efficient fine-tuning methods. <https://github.com/huggingface/peft>, 2022.
- [10] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. ReFT: Representation finetuning for language models. 2024.

- [11] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [12] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

## A Appendix

### A.1 Learning Rate Sweep

We performed a learning rate sweep for GPT-2 small using learning rates  $\{10^{-3}, 5 \times 10^{-4}, 10^{-4}\}$ . Across full fine-tuning, LoRA, and ReFT, we found that  $10^{-4}$  gave the best development performance.

We hypothesized that larger models may benefit from smaller learning rates, since they contain more parameters and may require more stable updates during fine-tuning. To test this, we conducted an additional sweep for GPT-2 medium starting from the best GPT-2 small learning rate and additionally evaluating  $0.5 \times$  and  $0.1 \times$  this value. The resulting plots for both GPT-2 small and GPT-2 medium are shown below.

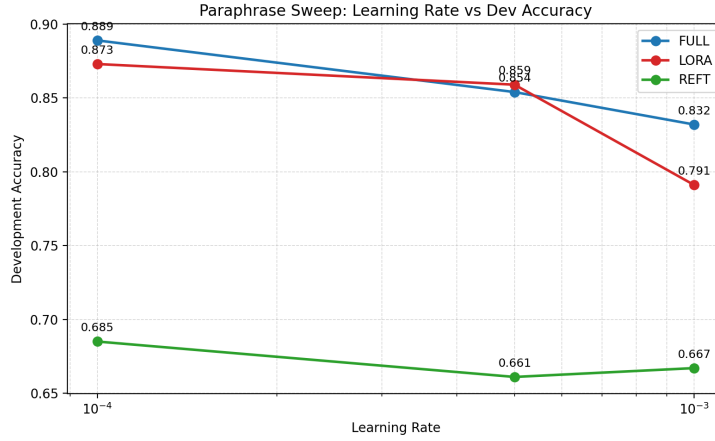


Figure 2: Learning rate sweep for GPT-2 small.

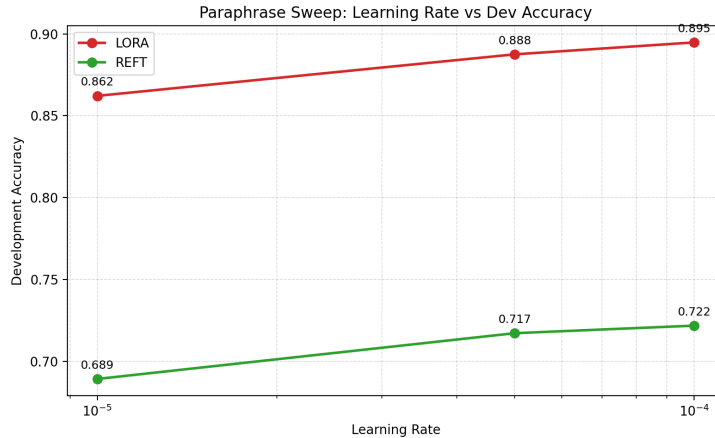


Figure 3: Learning rate sweep for GPT-2 medium using  $0.5 \times$  and  $0.1 \times$  the best GPT-2 small learning rate.

## A.2 LoRA Layer Placement Ablation

| Configuration    | Target Modules                     | Trainable Params | Dev Acc       |
|------------------|------------------------------------|------------------|---------------|
| LoRA-attn        | query, key, value, attention_dense | 589,824          | 85.79%        |
| LoRA-mlp         | interm_dense, out_dense            | 737,280          | 85.99%        |
| <b>LoRA-both</b> | <b>all six modules</b>             | <b>1,327,104</b> | <b>87.35%</b> |

Table 3: LoRA layer placement ablation on GPT-2 small (rank=8,  $\alpha = 32$ , dropout=0.05, lr= $10^{-4}$ , 3 epochs). Applying LoRA to both attention and MLP layers outperforms either alone, motivating the use of all six modules in subsequent experiments.