

Parameter-efficient fine-tunings for Downstream Adaptation of GPT-2

Stanford CS224N Default Project

Peiyu Li

Department of Computer Science
Stanford University
peiyu1@stanford.edu

Zefang Zhou

Department of Computer Science
Stanford University
zefangzhou@stanford.edu

Abstract

Fine-tuning on downstream tasks on large language models is very effective, but expensive, because all parameters are updated. Parameter efficient fine-tuning (PEFT) methods like Low-Rank Adaptation (LoRA) reduce trainable parameters while still adjusting weights on many layers. Representation fine-tuning (ReFT) instead acts on hidden representations at particular positions, offering a different trade-off between capacity and efficiency. Our project aims to compare the differences between these fine-tuning approaches on different downstream tasks. We develop a 12-layer GPT-2 model with masked multi-head self-attention and an Adam optimizer, then compare three fine-tuning strategies: full-model fine-tuning, LoRA (PEFT), and Low-rank Linear Subspace ReFT (LoReFT). We apply them to two downstream tasks: paraphrase detection via a cloze-style yes/no formulation, and Shakespearean sonnet generation from three-line prompts. On paraphrase detection, we find dev accuracies of 0.885 for the full model, 0.873 for LoRA, 0.815 for ReFT. For sonnet generation with GPT-2 large (10 epochs, batch size 2 for full-model and 4 for LoRA/ReFT), the best dev ChrF scores are 42.40 (full-model), 42.28 (LoRA), and 41.21 (ReFT). Full-model fine-tuning achieves the highest ChrF score on the test leaderboard (41.226), while LoRA and ReFT provide competitive performance with lower memory use.

1 Key Information to include

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

Team member	Late days remaining
<Peiyu Li>	<3>
<Zefang Zhou>	<0>
Total pooled	<3>
Team size	<2>
Late days applied for the report	<1.5>

2 Introduction

Before the advent of LLMs, fine-tuning was the method of choice for small-scale models (<300M parameters). Top of the class domain applications utilized supervised fine-tuning (SFT), i.e., to train the pre-trained model using the annotated datasets for specific domain and downstream tasks. Nonetheless, with the emergence of larger models (> 1B parameters), fine-tuning has become more

complicated. Most importantly, bigger models need larger resources and commercial hardware for fine-tuning.[1] PEFT techniques aim to address these shortcomings by training on a small number of parameters. In this project, a central question is how to balance task performance with the computational cost of large language model fine-tuning. We focus on two downstream tasks - paraphrase detection, which requires semantic understanding of sentence pairs, and sonnet generation, which requires creative, stylized text completion.

Paraphrase detection supports applications such as duplicate detection, question answering, and semantic search. Sonnet generation illustrates controlled creative generation and style transfer. These tasks differ in nature — one discriminative, one generative—and in data scale, from hundreds of thousands of paraphrase pairs to roughly a hundred sonnets. Understanding how different fine-tuning strategies behave across differences type of downstream tasks is important for practitioners who must choose between performance and efficiency.

PEFT methods such as Low-Rank Adaptation (LoRA)[2] and Representation Fine-tuning (ReFT)[3] have been proposed to lower training cost. However, systematic comparisons of full-model fine-tuning, LoRA, and ReFT on paraphrase detection and sonnet generation — with shared evaluation of task performance and efficiency — are limited. We conduct such a comparison using GPT-2 large version, measuring accuracy, ChrF, peak GPU memory, and training time.

We implement and compare three fine-tuning strategies: full-model fine-tuning with the course-provided GPT-2 implementation, LoRA via Hugging Face PEFT, and LoReFT via pyreft. We evaluate them on Quora paraphrase detection and Shakespeare sonnet generation, using accuracy and ChrF as task metrics and peak GPU memory and training time as efficiency metrics. On paraphrase detection, dev accuracies are 0.885 (full-model), 0.873 (LoRA), and 0.815 (ReFT). For sonnet generation with GPT-2 large, we use 10 epochs, batch size 2 for full-model and 4 for LoRA/ReFT, with LoRA bottleneck rank 64 and ReFT rank 8. Best dev ChrF scores are 42.40 (full-model), 42.28 (LoRA), and 41.21 (ReFT). Full-model fine-tuning achieves the highest test leaderboard ChrF (41.226). LoRA and ReFT provide competitive performance while reducing peak GPU memory (e.g., 10.3 GB and 6.5 GB vs. 13.5 GB for full-model on GPT-2 large).

3 Related Work

The rapid scaling of Large Language Models (LLMs) has made full-parameter fine-tuning increasingly impractical due to prohibitive computational and storage costs. This has catalyzed the field of Parameter-Efficient Fine-Tuning (PEFT), which aims to achieve performance comparable to full fine-tuning by modifying only a small fraction of the model’s total parameters.

3.1 Low-Rank Adaptation (LoRA)

LoRA [2] represents a milestone in Parameterization-based PEFT. It is motivated by the hypothesis that the change in weights during task adaptation resides in a low "intrinsic dimension". By approximating weight updates through the product of two low-rank matrices ($W' = W + AB$), LoRA dramatically reduces the number of trainable parameters. A distinct advantage of LoRA is its mathematical transparency; the low-rank matrices can be merged with the original weights after training, ensuring zero additional latency during inference.

3.2 Representation Fine-Tuning (ReFT)

While LoRA operates in the weight space, a more recent frontier has emerged in Representation-based PEFT, exemplified by ReFT [3]. ReFT is grounded in the principles of Mechanistic Interpretability, specifically the theory that LLMs encode semantic concepts within linear subspaces of their hidden representations. Unlike weight-based methods, ReFT employs interventional training, where a frozen model’s hidden states are modified during the forward pass using low-rank interventions. By steering the "internal state" of the model rather than altering its stored knowledge, ReFT achieves state-of-the-art performance with a parameter footprint much smaller than LoRA.

4 Approach

Our approach consists of three major components: (1) implementation of the GPT-2 model and verification of correctness; (2) establishment of baseline performance with complete fine-tuning; and (3) application of LoRA and ReFT to evaluate the paraphrase detection and sonnet generation.

4.1 Baseline GPT-2 Implementation

Our baseline consists of a decoder-only Transformer architecture, following the “pre-train, then fine-tune” paradigm. We implement a small-sized GPT-2 (124M) and load the official pre-trained weights from HuggingFace, providing the model with general language representations before task-specific adaptation. The backbone follows the architecture of Vaswani et al.[4]: L stacked transformer layers, each with masked multi-head self-attention followed by a position-wise feed-forward network. For GPT-2 small, we use hidden size $d = 768$, 12 attention heads (head dimension 64), feed-forward intermediate size $4d = 3072$, and maximum context length 1024 tokens. For larger variants (GPT-2 medium, large), we use the corresponding HuggingFace configurations ($d \in \{1024, 1280\}$, $L \in \{24, 36\}$). We adopt full-model fine-tuning as our baseline: all parameters are updated end-to-end with cross-entropy loss and the AdamW optimizer[5][6]. This baseline serves as the reference for comparing parameter-efficient methods. We compare three fine-tuning strategies: full-model (baseline), LoRA, and ReFT.

4.2 Downstream Tasks

4.2.1 Sentiment Analysis (Base Model Evaluation)

The sentiment classification task was conducted on the Stanford Sentiment Treebank[7] [8] and CFIMDB[9]. The corpus consists of movie review comments and we label them according to a five-class sentiment intensity scale: negative, somewhat negative, neutral, somewhat positive, and positive. Given an input sentence, the model predicts one of C sentiment labels. We extract the hidden state of the last non-padding token $h_T \in \mathbb{R}^d$ and apply a linear classification head:

$$\hat{y} = Wh_T + b, \quad W \in \mathbb{R}^{C \times d}$$

We evaluate two strategies: (1) last-linear-layer, where only the classification head is trained, and (2) full-model, where all parameters are updated. Both use cross-entropy loss with the AdamW optimizer[5][6].

4.2.2 Paraphrase Detection

Paraphrase detection measures how effectively systems can map different surface forms to the same semantic representation[10]. The task aims detect the semantic duplication over Quora[11] dataset, with models trained to differentiate between topical similarity in cloze style, constructing prompts such as: Is “[Sentence B]” a paraphrase of “[Sentence A]”? GPT-2 is trained to generate either the token corresponding to “yes” (ID 8505) or “no” (ID 3919). We experimented with three strategies:

- **Full Fine-Tuning** Using standard cross-entropy loss, we updated all parameters of GPT-2 with AdamW, using 10 epochs, batch size 4 and learning rate 1×10^{-5}
- **LoRA-Enhanced on GPT-2** Injecting low-rank matrices ($r = 8$, $\alpha = 2r$) into projections of the attention layers (*query*, *key*, *value*) and Multi-Layer Perceptron(MLP) layers(*interm_dense*, *out_dense*). With Base model frozen we use 10 epochs, batch size 4 and similar recommended learning rate recommended as 1×10^{-4} [12]
- **ReFT-Enhanced on GPT-2** We applied LoReFT to modify internal hidden representations at selected layers, as described in the ReFT paper[3]. Injecting interventions at layer $\ell = \lfloor L/2 \rfloor - 1$ with rank $r = 8$ at the last position. With Base model frozen we use 10 epochs, batch size 4 and similar recommended learning rate as 1×10^{-3} [3]

4.2.3 Sonnet Generation

Following the project instructions, we condition on the first three lines of each sonnet and let GPT-2 generate the remaining tokens until 14 lines are complete. We use nucleus (top- p) sampling with

temperature scaling and top- k filtering at inference: $\tau = 1.2$, $p = 0.9$, $k = 50$, and maximum generation length 128 tokens. We evaluate with the ChrF score against gold completions.

Training Configurations. We trained three versions of GPT-2 large on the Shakespeare sonnets corpus:

- **Full Fine-Tuning:** Standard next-token cross-entropy loss; all parameters updated with AdamW. We use 10 epochs, batch size 2, and learning rate 1×10^{-5} . Batch size 2 is used to fit GPT-2 large within GPU memory.
- **LoRA-Enhanced GPT-2 large:** Low-rank adapters ($r = 64$, $\alpha = 2r$) added to attention projections (c_attn, c_proj) and feed-forward layers (c_fc). Base model frozen. We use 10 epochs, batch size 4, and learning rate 1×10^{-5} .
- **ReFT-Enhanced GPT-2 large:** LoReFT intervention at layer $\ell = \lfloor L/2 \rfloor - 1$ with rank $r = 8$, applied at the last prompt position (intervene_on_prompt). Base model frozen. We use 10 epochs, batch size 4, and learning rate 1×10^{-5} .

4.3 Main Approach

The architecture of the model in our main approach is the transformer architecture, following the seminal work by Vaswani et al[4]. We then fine-tune the GPT-2 baseline on downstream tasks, including sentiment analysis and paraphrase detection.

4.3.1 Multiheaded Self-Attention

Each transformer layer uses masked multi-head self-attention[4]. Multiheaded self-attention uses a scaled dot-product attention function. Scaled dot-product attention works as follows: For input $X \in \mathbb{R}^{n \times d}$, we project to query, key, and value:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where $W_Q, W_K \in \mathbb{R}^{d \times d_k}$, $W_V \in \mathbb{R}^{d \times d_v}$, and $d_k = d_v = d/h$ for h heads. The scaled dot-product attention for each head is:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} + M \right) V$$

where M is the causal mask (upper-triangular $-\infty$). The h head outputs are concatenated and projected:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O$$

We apply dropout to the attention weights and use a residual connection plus layer normalization around the attention and feed-forward sublayers.

4.3.2 Full-Model Fine-Tuning

In full-model fine-tuning, we update all parameters θ of the pre-trained model. Given loss $\mathcal{L}$, we compute:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}$$

where η is the learning rate. For GPT-2 large (774M parameters), this requires storing gradients for all parameters, leading to high peak GPU memory usage. We use the course-provided GPT-2 implementation (models/gpt2.py) for the small model and Hugging Face’s AutoModelForCausalLM for larger variants when combined with LoRA or ReFT.

4.3.3 LoRA (Low-Rank Adaptation)

LoRA[2] injects trainable low-rank matrices into frozen pre-trained weights. For a linear layer $W_0 \in \mathbb{R}^{d \times k}$, the forward pass becomes:

$$h = W_0 x + \Delta W x = W_0 x + B A x$$

where $A \in \mathbb{R}^{r \times k}$, $B \in \mathbb{R}^{d \times r}$, and $r \ll \min(d, k)$ is the bottleneck rank. Only A and B are trained; W_0 remains frozen. The number of trainable parameters is $r(d + k)$.

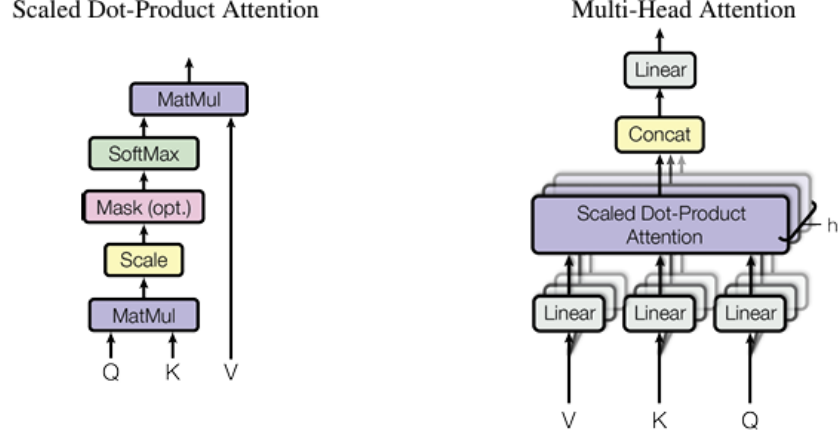


Figure 1: Scaled Dot-Product and Multi-Head Self-Attention. Figure from [4]

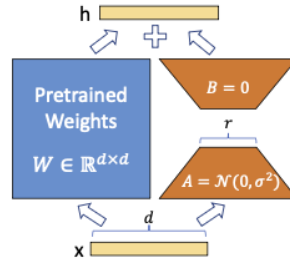


Figure 2: Reparametrization of LoRA. Figure from [2]

4.3.4 ReFT (Representation Fine-Tuning)

ReFT[3] learns interventions on hidden representations instead of modifying weights. An intervention is defined as $I = (\Phi, P, L)$, where Φ is the intervention function, P is the set of positions at which to intervene, and L is the layer. The base model remains frozen; only the parameters of Φ are trained. We use LoReFT (Low-Rank Representation Fine-Tuning), the primary instance of ReFT, via the StanfordNLP pyreft library. As illustrated in Figure 3, the intervention function Φ_{LoReFT} computes an edit vector and adds it back to the hidden state:

$$\Phi_{\text{LoReFT}}(h) = h + R^\top (Wh + b - Rh) \quad (1)$$

where $h \in \mathbb{R}^d$ is the hidden representation, $R \in \mathbb{R}^{r \times d}$ has orthonormal rows spanning the edit subspace, $W \in \mathbb{R}^{r \times d}$ is a linear projection, and $b \in \mathbb{R}^r$ is a bias. The learned parameters are $\Phi = \{R, W, b\}$; the edit is confined to the r -dimensional subspace spanned by the rows of R .

5 Experiments

This section contains the following.

5.1 Data

We use four datasets across three tasks: sentiment classification, paraphrase detection, and sonnet generation.

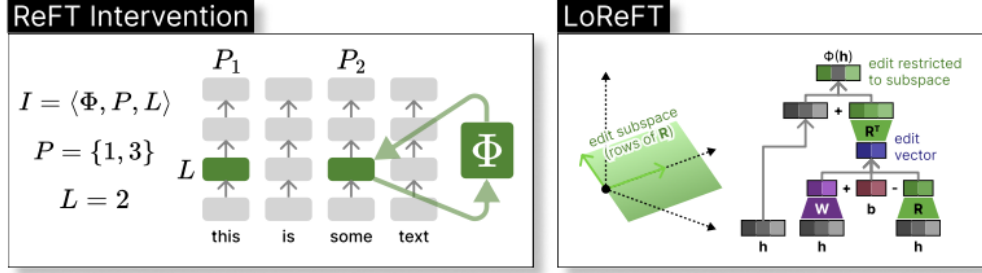


Figure 3: Illustration of ReFT. Figure from [3]

5.1.1 Sentiment Analysis

Stanford Sentiment Treebank (SST)[8]. The SST is a phrase-level sentiment dataset built from movie reviews. Each phrase is labeled with one of five sentiment classes from 0 (very negative) to 4 (very positive). We use the standard split with train, dev, and test sets. Let $x = (w_1, \dots, w_n)$ denote the input token sequence. The model predicts a sentiment label $\hat{y} \in \mathcal{Y} = \{0, \dots, 4\}$ (very negative to very positive).

CFIMDB[9]. The CFIMDB dataset is a sentiment dataset derived from IMDb movie reviews. We use the provided train, dev, and test splits. Let $x = (w_1, \dots, w_n)$ denote the input token sequence (a movie review). The model predicts a sentiment label $\hat{y} \in \mathcal{Y}$, where $\mathcal{Y}$ is binary value of 0 or 1 indicating negative or positive ($|\mathcal{Y}| = \{0, 1\}$).

5.1.2 Paraphrase Detection

We use the Quora Question Pairs dataset[11], which contains question pairs from Quora with binary labels indicating semantic equivalence. Each example has two questions and a label (1 = paraphrase, 0 = non-paraphrase). We follow a cloze-style formulation: the model is prompted to generate either “yes” or “no” to indicate whether the two questions ask the same thing. **Input (training):** Question 1: “{sentence₁}” \n Question 2: “{sentence₂}” \n Are these questions asking the same thing? \n

Input (test): Is “{sentence₁}” a paraphrase of “{sentence₂}”? Answer “yes” or “no”:

Output: the next-token prediction is restricted to the BPE tokens for “yes” or “no”; the predicted token is mapped to the binary label. Sentences are lowercased, and punctuation is tokenized. We use the standard train, dev, and test splits. The model is trained to predict the next token, which should be “yes” (BPE token 8505) for paraphrases and “no” (BPE token 3919) otherwise. We project the last token’s hidden state to the full vocabulary using the weight-tied output matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d}$:

$$\text{logits} = h_T \cdot E^\top$$

and train with cross-entropy over the vocabulary.

5.1.3 Sonnet Generation

We use William Shakespeare’s sonnets. Each sonnet is a 14-line poem in iambic pentameter with a fixed rhyme scheme. Given the first three lines of a sonnet as a prompt, the model must generate the remaining lines in Shakespearean style.

5.2 Evaluation method

We evaluate full fine-tuning, LoRA, and ReFT on two dimensions: (1) task performance and (2) training efficiency. These metrics show whether PEFT methods match or improve performance while reducing overfitting and resource usage.

Table 1: Dataset sizes used in this work.

Dataset	Train	Dev	Test
SST (sentiment)	8,544	1,101	2,210
CFIMDB (sentiment)	1,707	245	488
Quora (paraphrase)	283,011	40,430	80,860
Shakespeare Sonnets (generation)	131	12	12

For task performance, we report accuracy as the main metric for paraphrase detection:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^N [\hat{y}_i = y_i]. \quad (2)$$

We evaluate on the dev set during training and on the held-out test set for the leaderboard.

We use ChrF (character n-gram F-score) [13] for sonnet generation:

$$\text{ChrF}_\beta = (1 + \beta^2) \frac{P_{\text{chr}} \cdot R_{\text{chr}}}{\beta^2 P_{\text{chr}} + R_{\text{chr}}}, \quad (3)$$

with default $\beta = 2$. ChrF measures character-level overlap between generated and reference sonnets. We compute ChrF on the dev set (12 sonnets with gold references) each epoch for model selection; the best model by dev ChrF is used for test-set generation.

For training efficiency, Table 3 reports peak GPU RAM usage (in GB), measured during training, and total training time per epoch (in sec). These metrics reflect the resource cost of each finetuning approach.

5.3 Experimental details

We describe the setup used for all experiments, including model configurations, hyperparameters, and hardware. Paraphrase detection experiments were run on an NVIDIA L4 GPU; sonnet generation experiments on an NVIDIA T4 GPU.

Across all methods we used, we use the AdamW optimizer with learning rate 1×10^{-5} . For LoRA, we use rank $r = 64$ and scaling $\alpha = 2r$, applied to attention projections (`c_attn`, `c_proj`) and feed-forward layers (`c_fc`). For ReFT, we place the LoReFT intervention at layer $\ell = \lfloor L/2 \rfloor - 1$ with rank $r = 8$, intervening at the last token position (paraphrase) or last prompt position (sonnet). Paraphrase detection uses 10 epochs with batch size 8. Sonnet generation uses 10 epochs with batch size 2 for full fine-tuning (to fit in GPU memory) and batch size 4 for LoRA and ReFT.

Table 2 and Table 3 record the configurations we used for each downstream tasks.

5.4 Results

The hyperparameter experiment shows for sentiment analysis, full-model fine-tuning, $\text{lr} = 5 \times 10^{-5}$ has the best results, exceeding the provided baseline on both SST (51.4% vs 51.3%) and CFIMDB (98.4% vs. 97.6%). For the last-linear-layer, the best result comes from the configuration of a dropout rate of 0.1 and 50 epochs. We observed a strong learning rate sensitivity, suggesting future work to explore learning rate warm-up and decay schedules rather than a fixed rate. See Table 2 for more details.

For paraphrase detection, our model reaches the best dev accuracy of 0.885 for Full-model Fine-tuning. The LoRA and ReFT significantly lower the parameters and thus reduced the GPU Memory usage, with slightly lower dev accuracy of 0.873(LoRA) and 0.815(ReFT). For the same model and batch size, LoRA significantly faster for each epoch than Full Fine-tune, while ReFT runs slower than LoRA can be explained by ReFT requires additional intervention and implementation overhead defined in pyreft library.

Table 2: Sentiment Analysis Dev Accuracy (%)

Configuration	SST		CFIMDB	
	Last-linear	Full-model	Last-linear	Full-model
Provided Baseline	46.2	51.3	86.1	97.6
$LR = 1 \times 10^{-5}$	—	50.8	—	97.3
$LR = 5 \times 10^{-5}$	35.8	51.4	58.8	98.4
$LR = 1 \times 10^{-3}$	41.1	—	73.5	—
$LR = 5 \times 10^{-3}$	36.9	—	75.5	—
Dropout 0.05 (50 ep.)	44.7	—	84.5	—
Dropout 0.1 (50 ep.)	47.7	—	88.2	—
Batch size 4	42.1	37.8	—	—
Batch size 8	41.1	37.1	—	—
Batch size 16	41.5	39.8	—	—
Batch size 64	39.1	34.5	—	—

Table 3: Comparison of Full Fine-Tuning, LoRA, and ReFT performance across tasks and GPU architectures. Memory usage and training time are reported per epoch.

Task (GPU / Model)	Fine-Tune Method	Accuracy	Memory Usage	Ave Runtime (per epoch)
Paraphrase Detection (L4 GPU / GPT-2)	Full	0.885	21.8 GB	163.05 min
	LoRA	0.873	3.1 GB	84.53 min
	ReFT	0.815	2.5 GB	131.48 min
Task (GPU)	Fine-Tune Method	ChrF	Memory Usage	Ave Runtime (per epoch)
Sonnet Generation (T4 GPU / GPT-2 large)	Full	42.40	13.5 GB	51.43 sec
	LoRA	42.28	10.0 GB	29.12 sec
	ReFT	41.21	6.4 GB	18.83 sec

6 Analysis

Our experimental results demonstrate a clear trade-off between task performance and computational efficiency across different fine-tuning strategies. Notably, LoRA and ReFT achieved competitive results while significantly reducing resource requirements. Specifically, on the paraphrase detection task, the baseline GPT-2 model with full-model fine-tuning achieved a dev accuracy of 0.885 but requires 21.8GB graphics memory, the LoRA achieved competitive dev accuracy of 0.859 with significantly less memory of 3.1GB and significantly shorter runtime for training. On Sonnet generation with GPT-2 large, ReFT reduced peak GPU memory usage to 6.5 GB compared to the 13.5 GB required for full-model fine-tuning, full-model fine-tuning retained a slight edge in creative quality, achieving the highest ChrF score of 42.40 on the development set, though LoRA remained highly competitive at 42.28.

However, we observed an unexpected runtime phenomenon on the paraphrase detection task: ReFT runs slower per epoch than LoRA on the same hardware. This discrepancy can be attributed to the additional intervention and implementation overhead inherent in the `pyreft` library, where dynamic activation steering requires more sequential processing than the reparameterized weight updates of LoRA and prevents the same level of hardware optimization. Meanwhile, ReFT achieves a competitive but less dev accuracy may be attributed to the sub-optimized hyperparameters which are worth exploring in future work with more computing resources.

7 Conclusion

This project provided a systematic comparison of full-model fine-tuning, LoRA, and ReFT across discriminative and generative downstream tasks. Our findings confirm that while full-model fine-tuning often achieves the absolute highest performance metrics, Parameter-Efficient Fine-Tuning (PEFT) methods offer a more sustainable path for scaling to larger models like GPT-2 large. ReFT, in

particular, proved to be the most memory-efficient method, enabling large-model adaptation with a minimal parameter footprint. Future work could explore optimizing the intervention overhead of ReFT to improve training performance and investigating more complex cloze-style prompts for semantic tasks. Ultimately, the choice between these methods depends on the specific constraints of the resource and task requirements, with LoRA offering a balance of speed and performance, and ReFT providing a solution for extreme memory-limited environments.

Team contributions

Peiyu worked on the GPT implementation, sentiment analysis, paraphrase detection with Full-model Fine-tuning and sonnets generation with Full-model Fine-tuning, LoRA and LoReFT. Zefang worked on GPT implementation, sentiment analysis and paraphrase detection with Full-model Fine-tuning, LoRA and LoReFT.

References

- [1] Amir Maleki Aditya Jain and Nathalie Saade. To fine-tune or not to fine-tune. *Meta.ai*, 2024.
- [2] Phillip Wallis Zeyuan Allen-Zhu Yuanzhi Li Shean Wang Lu Wang Weizhu Chen Edward J. Hu, Yelong Shen. Lora: Low-rank adaptation of large language models. *arXiv:2106.09685*, 2021.
- [3] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. ReFT: Representation finetuning for language models. *arXiv preprint arXiv:2404.03592*, 2024.
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [6] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [7] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.
- [8] Jean Wu Jason Chuang Christopher D Manning Andrew Y Ng Richard Socher, Alex Perelygin and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631-1642, 2013.
- [9] Peter T. Pham Dan Huang Andrew Y. Ng Andrew L. Maas, Raymond E. Daly and Christopher Potts. Learning word vectors for sentiment analysis. *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*, 2011.
- [10] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. *Proceedings of the 11th annual research colloquium of the UK special interest group for computational linguistics*, pages 45–52, 2008.
- [11] Nikhil Dandekar Shankar Iyer and Kornél Csernai. First quora dataset release: Question pairs. *Quora*, 2017.
- [12] John Schulman. Lora without regret. <https://thinkingmachines.ai/blog/lora/>, 2025.
- [13] Maja Popović. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, 2015.