

Exploring Parameter-Efficient Fine-Tuning for Paraphrase Detection with GPT-2

Stanford CS224N {Default}

Krishna Chokshi

Department of Computer Science
Stanford University
krishac@stanford.edu

Abstract

Fine-tuning large language models like GPT-2 for downstream tasks usually requires updating all model parameters, which is computationally very expensive and also memory-intensive. This poses challenges for deployment in resource-constrained settings and limits the ability to maintain multiple task-specific models. In this project, I investigate whether Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning method, can achieve comparable performance to full fine-tuning on paraphrase detection while training significantly fewer parameters. Through systematic experiments varying LoRA rank ($r \in \{4, 8, 16\}$) and training data fraction (10%–100%), I find that LoRA achieves up to 88.0% accuracy while training only 0.47% of the parameters, compared to 89.7% for full fine-tuning. Data scaling experiments reveal that full fine-tuning still achieves 1–3 percentage points higher accuracy than LoRA across all data regimes; however, LoRA still scales effectively with data (83.3% at 10% data to 87.9% at 100%) while using over $200\times$ fewer trainable parameters. These results demonstrate that LoRA provides an effective parameter-accuracy tradeoff, making it a practical choice when compute or storage efficiency is prioritized over maximum accuracy.

1 Key Information to include

- Mentor: Tristan Thrush
- External Collaborators (if you have any): No
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
Krishna Chokshi	6
Total pooled	0
Team size	1
Late days applied for the report	0

2 Introduction

Large language models (LLMs) like GPT-2 have demonstrated strong capabilities across a wide range of natural language processing tasks. But, adapting these models to specific downstream tasks traditionally requires fine-tuning all model parameters, which presents several practical challenges. For GPT-2 small alone, this means updating approximately 124 million parameters, requiring substantial computational resources and memory. When users need to deploy models for multiple tasks, storing and serving separate fully fine-tuned copies becomes too expensive. Beyond storage costs,

full fine-tuning also introduces the risk of forgetting and having to restart with retraining from scratch when the base model is updated, which furthers the need for more modular adaptation approaches.

Parameter-efficient fine-tuning (PEFT) methods have come up as a promising solution to these challenges. Out of these, Low-Rank Adaptation (LoRA) [1] has gained particular attention for its simplicity and effectiveness. LoRA freezes the pretrained model weights and injects trainable low-rank decomposition matrices into transformer layers, which dramatically reduces the number of trainable parameters while still maintaining competitive performance.

In this project, I investigate the effectiveness of LoRA for paraphrase detection using GPT-2. Specifically, I aim to answer: (1) How does LoRA performance compare to full fine-tuning on paraphrase detection? (2) How does the choice of rank r affect the parameter-accuracy tradeoff? (3) How do LoRA and full fine-tuning compare when training data is limited? I implement LoRA from scratch following the original paper and conduct systematic experiments comparing full fine-tuning against LoRA with ranks 4, 8, and 16, as well as data scaling experiments with 10%, 25%, 50%, and 100% of the training data.

My results show that LoRA achieves 87.4–88.0% accuracy across different ranks while training only 0.12–0.47% of the total parameters, compared to 89.7% for full fine-tuning. This represents a reasonable tradeoff: a modest 1.7–2.3 percentage point accuracy drop in exchange for over $200\times$ reduction in trainable parameters. Data scaling experiments reveal that full fine-tuning maintains a 1–3 percentage point advantage across all data regimes, though LoRA scales effectively from 83.3% (10% data) to 87.9% (100% data) while preserving its parameter efficiency. I also complete the sonnet generation task as part of the default project requirements, achieving a chrF score of 41.349.

3 Related Work

Parameter-Efficient Fine-Tuning. Many approaches have been proposed to reduce the cost of adapting large pretrained models. Adapter layers [2] insert small bottleneck modules between transformer layers, training only these new parameters. Prefix tuning [3] adds learnable continuous vectors to the input, optimizing only these soft prompts. LoRA [1] takes a different approach by representing weight updates as low-rank matrices, allowing the adapted weights to be merged with the original weights at inference time with no additional latency.

Low-Rank Adaptation. LoRA was introduced by Hu et al. [1] and has since become one of the most popular PEFT methods. The key insight is that the weight updates during fine-tuning have low intrinsic rank, so they can be parameterized using two matrices. The original paper demonstrates strong results on GPT-3 and RoBERTa across various NLU tasks. Additional work has extended LoRA in various directions, including adaptively selecting rank [4] and quantization-aware variants [5]. My work applies LoRA to GPT-2 for paraphrase detection and also studies the effect of rank on performance.

Paraphrase Detection. Paraphrase detection is a fundamental NLP task with applications in duplicate question detection, plagiarism detection, and semantic search. The Quora Question Pairs dataset used in this Default Project extension provides a benchmark for this task. Many modern approaches usually fine-tune pretrained transformers like BERT [6] or GPT-2 [7] on this dataset. I use the GPT-2-based approach provided in the CS224N default project as my baseline.

4 Approach

4.1 Baseline: Full Fine-Tuning

My baseline follows the standard approach of fine-tuning all parameters of a pretrained GPT-2 model for paraphrase detection. Given a pair of sentences (s_1, s_2) , I format the input as a cloze-style prompt:

```
Question 1: "{s1}"
Question 2: "{s2}"
Are these questions asking the same thing?
```

The model processes this input and produces hidden states for each token. I extract the hidden state corresponding to the last token, $h_{\text{last}} \in \mathbb{R}^d$, and pass it through a linear classification head:

$$\hat{y} = \text{softmax}(W_c h_{\text{last}} + b_c) \quad (1)$$

where $W_c \in \mathbb{R}^{2 \times d}$ and $b_c \in \mathbb{R}^2$. During training, all GPT-2 parameters and the classification head are updated to minimize cross-entropy loss.

4.2 LoRA: Low-Rank Adaptation

LoRA [1] is a parameter-efficient fine-tuning method that freezes the pretrained model weights and injects trainable low-rank decomposition matrices. Instead of learning a full weight update $\Delta W \in \mathbb{R}^{d \times d}$, LoRA represents the update as a product of two low-rank matrices:

$$W' = W + \Delta W = W + BA \quad (2)$$

where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times d}$, with rank $r \ll d$. This reduces the number of trainable parameters per adapted layer from d^2 to $2dr$.

During the forward pass, the output is computed as:

$$h = W'x = Wx + BAx \quad (3)$$

Following the original paper, I apply LoRA to the query (W_Q) and value (W_V) projection matrices in each self-attention layer. The key projections, feed-forward layers, and all other weights remain frozen. Matrix B is initialized to zeros and A with Kaiming uniform initialization, ensuring the LoRA contribution starts at zero and the model begins from the pretrained state.

My implementation adds LoRA layers to all 12 attention layers in GPT-2 small, modifying only the Q and V projections. The classification head remains trainable. I implemented all LoRA code from scratch using the original paper as guidance.

4.3 Sonnet Generation

For sonnet generation, I fine-tune GPT-2 on a corpus of Shakespeare’s sonnets using standard language modeling (next-token prediction). The model learns to predict the next token given all previous tokens, minimizing cross-entropy loss over the training sonnets. I use the default project’s provided training setup and evaluate using the chrF metric.

5 Experiments

5.1 Data

Paraphrase Detection. I use the Quora Question Pairs dataset as provided in the default project, containing 283,003 training pairs and 40,429 development pairs. Each example has two questions and a binary label indicating whether they are paraphrases or not. The input to the model is the two questions formatted as a prompt, and the output is a binary classification prediction.

Sonnet Generation. I use the Shakespeare sonnets dataset provided in the default project. At evaluation time, the model is given the first few tokens of a held-out sonnet as a prompt and must generate the remainder. Performance is measured using the chrF metric, which computes overlap between generated and reference text.

5.2 Evaluation Method

Paraphrase Detection. I evaluate using classification accuracy, reporting results on both the development set and test leaderboard to make an assessment on the trade-off between LoRA and full fine-tuning. I compare LoRA at multiple ranks ($r \in \{4, 8, 16\}$) against full fine-tuning to assess the parameter-accuracy tradeoff, and understand which rank is most suitable. To understand the relationship between data availability and method performance, I conduct data scaling experiments comparing full fine-tuning and LoRA at 10%, 25%, 50%, and 100% of the training data. This reveals whether the two methods scale differently with data, and which one is preferable per percentage of

data usage. Additionally, I conduct a qualitative error analysis by examining the 3,535 examples (8.74% of dev set) where LoRA with rank 8 and full fine-tuning disagree to understand the types of errors each method makes and if reduced parameter capacity introduces any specific failure modes or patterns.

Sonnet Generation. I evaluate using the chrF metric, which measures character n-gram F-score between generated and reference sonnets.

5.3 Experimental details

All experiments use GPT-2 small (124M parameters) trained on a single NVIDIA L4 GPU.

Paraphrase Detection:

- Batch size: 8
- Epochs: 10
- Optimizer: AdamW (no weight decay)
- Full fine-tuning learning rate: 1×10^{-5}
- LoRA learning rate: 1×10^{-4}
- LoRA scaling factor α : 16
- LoRA ranks tested: $r \in \{4, 8, 16\}$
- Data fractions (10%, 25%, 50%, 100%)

Practical guidance for LoRA training suggests using learning rates roughly an order of magnitude larger than those used for full fine-tuning because only a small subset of parameters are optimized [8]. For example, the Hugging Face TRL documentation reports using $1e-5$ for LoRA and $1e-6$ for full fine-tuning. In my experiments, I use the default project baseline learning rate of $1e-5$ for full fine-tuning and set the LoRA learning rate to $1e-4$, maintaining the same $10\times$ ratio between the two settings.

Sonnet Generation:

- Batch size: 10
- Epochs: 10
- Learning rate: 1×10^{-5}
- Generation: temperature sampling with $T = 1.0$

5.4 Results

5.4.1 Test Leaderboard Results:

- Paraphrase Detection (Full Fine-Tuning): 89.5% compared to Staff Baseline of 85.6%
- Sonnet Generation: 41.349 chrF compared to Staff Baseline of 41.208

Full fine-tuning achieved the highest development accuracy, so this model was used for test leaderboard submission, with parameters as specified above. These results closely match development set performance.

5.4.2 Paraphrase Detection

Table 1 shows the main results for paraphrase detection. Full fine-tuning achieves 89.7% development accuracy, serving as our upper bound. LoRA with rank 16 achieves the best parameter-efficient result at 88.0%, only 1.7 percentage points below full fine-tuning while training just 0.47% of the parameters.

The results show a clear trend: accuracy increases with rank, but with diminishing returns. Going from $r = 4$ to $r = 8$ improves accuracy by 0.5 percentage points while doubling parameters. Going from $r = 8$ to $r = 16$ yields only 0.1 percentage point improvement despite another doubling of parameters. This suggests $r = 8$ may offer the best tradeoff for this task.

Method	Trainable Params	% of Total	Dev Accuracy
Full Fine-Tuning	$\sim 125\text{M}$	100%	89.7%
LoRA ($r = 4$)	$\sim 148\text{K}$	0.12%	87.4%
LoRA ($r = 8$)	$\sim 296\text{K}$	0.24%	87.9%
LoRA ($r = 16$)	$\sim 590\text{K}$	0.47%	88.0%

Table 1: Comparison of full fine-tuning vs. LoRA with different ranks on paraphrase detection. LoRA achieves competitive accuracy while training less than 0.5% of the parameters.

5.4.3 Data Scaling

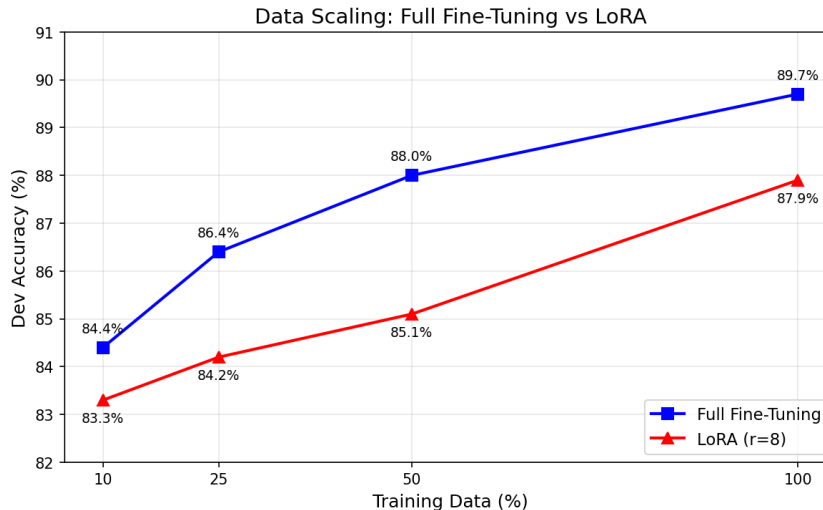


Figure 1: Data scaling comparison between full fine-tuning and LoRA ($r = 8$).

Full fine-tuning consistently outperforms LoRA across all data fractions, with the gap widening from 1.1 to 2.9 percentage points as data increases from 10% to 50%. However, at 100% data, LoRA narrows the gap to 1.8 percentage points, suggesting that LoRA benefits proportionally more from the full dataset. In practice, LoRA is preferable in resource-constrained settings where the 1–2% accuracy trade-off is acceptable for over $200\times$ fewer trainable parameters, while full fine-tuning remains the better choice when maximum accuracy is the priority and compute resources are available.

5.4.4 Discussion

The LoRA results are largely in line with the expectations set from prior work [1], which reported that LoRA can match full fine-tuning performance on many tasks. Achieving 88.0% accuracy with only 0.47% of the parameters demonstrates that weight updates during fine-tuning have low intrinsic rank which holds for paraphrase detection. The 1.7 percentage point gap is slightly larger than what I expected, possibly because paraphrase detection requires nuanced semantic understanding that benefits from full model capacity.

The data scaling results were somewhat surprising to me. I expected the gap between methods to remain constant across data fractions, but instead observed it widening from 10% to 50% before narrowing at 100%. This suggests that full fine-tuning is more data-efficient in the mid-range, but LoRA catches up when sufficient data is available. One interpretation is that LoRA’s constrained capacity requires more examples to learn effective representations, but ultimately converges to near-comparable performance.

Overall, these quantitative results make LoRA a strong alternative to full fine-tuning for paraphrase detection, particularly in settings where memory or storage constraints make maintaining separate full models impractical.

6 Analysis

6.1 Parameter Efficiency vs. Accuracy Tradeoff

The most striking finding was the extreme parameter efficiency of LoRA. With $r = 8$, LoRA trains only $\sim 296\text{K}$ parameters (0.24% of the model) yet achieves 87.9% accuracy, which is within 1.8 percentage points of full fine-tuning. This represents approximately a $420\times$ reduction in trainable parameters for less than 2% accuracy loss. This has massive practical implications since someone maintaining multiple task-specific models could store a single frozen GPT-2 base model plus small LoRA adapters for each task, which would dramatically reduce storage requirements.

6.2 Effect of Rank

The diminishing returns with increasing rank align with the hypothesis from the original LoRA paper that fine-tuning updates have low intrinsic dimensionality. For paraphrase detection with GPT-2 small, most of the task-relevant information can be captured with just 4–8 dimensions per adapted weight matrix. Beyond $r = 16$, we would expect minimal further gains.

6.3 Training Efficiency

Beyond parameter efficiency, LoRA also provides training speedups. Full fine-tuning required approximately 50 minutes per epoch, while LoRA training completed each epoch in approximately 23 minutes which is a 54% reduction. This speedup comes from reduced gradient computation and optimizer state memory.

6.4 Data Scaling

The data scaling experiments reveal an interesting non-linear relationship between data availability and the performance gap. At 10% of training data, full fine-tuning leads by only 1.1 percentage points (84.4% vs 83.3%). This gap widens to 2.9 percentage points at 50% data (88.0% vs 85.1%), before narrowing to 1.8 percentage points at 100% data (89.7% vs 87.9%). As stated previously, this pattern suggests that full fine-tuning is more data-efficient in the mid-range, extracting more signal from moderate amounts of data. But, LoRA catches up when given access to the full dataset, likely because its constrained parameter space requires more examples to learn effective representations but can ultimately converge to comparable performance.

6.5 Qualitative Error Analysis

To understand the types of errors each method makes, I examined some of the 3,535 examples (8.74% of the dev set) where LoRA ($r = 8$) and full fine-tuning disagree. Of these, full fine-tuning was correct in 2,135 cases while LoRA was correct in 1,400 cases, which is consistent with the overall accuracy gap. Breaking down LoRA’s errors: 1,037 were false negatives (incorrectly predicting non-paraphrase) and 1,098 were false positives (incorrectly predicting paraphrase). Manual inspection of these cases revealed two patterns:

- **False negatives:** LoRA tends to miss paraphrases involving lexical variation, where the same meaning is expressed with different vocabulary (e.g., “How do I write an Android application?” vs “How do I create an Android application?”).
- **False positives:** LoRA over-matches questions that share topic or structure but differ in intent (e.g., “How hard is it to double major in engineering and physics?” vs “Is it wise to double major in engineering and physics?”).

These patterns suggest that LoRA’s reduced parameter capacity limits its ability to capture the nuanced lexical-semantic mappings, making it rely more heavily on surface-level similarity. Full fine-tuning, with access to all parameters, can learn finer-grained distinctions between true paraphrases and just related but semantically distinct questions.

7 Conclusion

This project investigated Low-Rank Adaptation (LoRA) as a parameter-efficient alternative to full fine-tuning for paraphrase detection using GPT-2. Through systematic experiments comparing LoRA ranks, data scaling, and qualitative error analysis, I found that LoRA achieves 87.9% accuracy with only 0.24% of trainable parameters which is within 1.8 percentage points of full fine-tuning’s 89.7% while reducing parameters by over 420 $\times$.

Key findings:

1. LoRA provides an effective parameter-accuracy tradeoff, with rank $r = 8$ offering the best balance of efficiency and performance.
2. Data scaling experiments revealed that full fine-tuning is more data-efficient at moderate data sizes, but LoRA closes the gap with sufficient training data.
3. Qualitative analysis alluded to the idea that LoRA’s errors stem from reduced capacity to capture nuanced lexical to semantic mappings, causing too much reliance on surface-level similarity.

This project deepened my understanding of parameter-efficient fine-tuning and the practical tradeoffs involved in adapting large language models. In terms of limitations, the persistent 1.7% accuracy gap may be significant for high-stakes applications. My experiments were limited to GPT-2 small so larger models may exhibit different tradeoffs. Additionally, I only applied LoRA to Q and V projections, leaving other configurations unexplored. Promising next steps include exploring LoRA on larger models, combining LoRA with other PEFT methods like adapters, and investigating task-specific rank selection strategies.

References

- [1] Edward 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. In *International Conference on Learning Representations (ICLR)*, 2022.
- [2] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.
- [3] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL*, 2021.
- [4] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.
- [5] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.
- [6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019.
- [7] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.
- [8] Hugging Face. Lora without regret. https://huggingface.co/docs/trl/en/lora_without_regret, 2025. Accessed: 2026-03-02.