

Parameter-Efficient Fine-Tuning of GPT-2 for Classification and Text Generation

Stanford CS224N Default Project

Ryan He

Department of Computer Science
Stanford University
ryanhe@stanford.edu

Abstract

Large pretrained language models such as GPT-2 can be adapted to downstream tasks through fine-tuning, but updating all model parameters can be computationally expensive. Parameter-efficient methods such as Low-Rank Adaptation (LoRA) address this limitation by introducing a small number of trainable parameters while keeping most model weights frozen. In this project, we compare full fine-tuning and LoRA when adapting GPT-2 small (117M parameters) to two tasks: paraphrase detection and sonnet generation. We evaluate four training configurations combining different fine-tuning strategies and optimization methods. Across both tasks, full fine-tuning achieves the best performance, with AdamW and cosine scheduling reaching a dev F1 of 0.8913 on paraphrase detection and a perplexity of 70.92 on sonnet generation. However, LoRA achieves competitive performance while training only 296,450 parameters compared to approximately 125M parameters for full fine-tuning. These results highlight the trade-off between model performance and training efficiency in parameter-efficient fine-tuning.

1 Key Information to include

- Mentor: Chenglei Si
- External Collaborators (if you have any): N/A
- Sharing project: No

2 Introduction

Pretrained transformer language models have become a dominant paradigm in natural language processing. Models such as GPT-2 demonstrate that large neural networks trained on large text corpora can learn general language representations that transfer effectively to many downstream tasks through fine-tuning. Although full fine-tuning typically achieves strong performance, it requires updating all model parameters, which can be computationally expensive as model sizes grow. This has motivated the development of parameter-efficient fine-tuning methods that adapt pretrained models while updating only a small subset of parameters. Low-Rank Adaptation (LoRA) is one such approach. Instead of modifying the original weight matrices of a pretrained model, LoRA introduces trainable low-rank update matrices while keeping the pretrained parameters frozen. This significantly reduces the number of trainable parameters while maintaining competitive performance on downstream tasks.

In this project, we compare full fine-tuning and LoRA for adapting GPT-2 small to two tasks from the CS224N default project: paraphrase detection and sonnet generation. We also evaluate how different optimization strategies, including Adam and AdamW with cosine learning-rate scheduling, affect performance. Our experiments aim to answer the following questions:

1. How does LoRA compare with full fine-tuning in terms of performance?
2. How do different optimization strategies affect fine-tuning results?
3. How much performance is lost when reducing the number of trainable parameters?

Our results show that full fine-tuning achieves the best performance, but LoRA provides strong parameter efficiency while maintaining competitive results.

3 Related Work

Large pretrained language models have become a central paradigm in natural language processing. GPT-2 [1] demonstrated that large transformer-based language models trained on large web-scale corpora can learn general language representations that transfer effectively to a wide range of downstream tasks through fine-tuning. The model architecture follows the decoder-only transformer introduced by Vaswani et al. [2], trained using a causal language modeling objective.

While full fine-tuning of pretrained models often yields strong performance, it requires updating all model parameters. As model sizes continue to grow, this motivates the development of parameter-efficient fine-tuning methods that adapt models using only a small number of trainable parameters. Low-Rank Adaptation (LoRA) [3] is a parameter-efficient technique that introduces trainable low-rank update matrices to existing weight matrices while keeping the pretrained parameters frozen. This approach drastically reduces the number of trainable parameters while maintaining strong downstream performance across a variety of NLP tasks.

Optimization strategies also play an important role in training and fine-tuning deep neural networks. Adam [4] is a widely used adaptive optimization algorithm for training neural networks. AdamW [5] improves upon Adam by decoupling weight decay from the gradient update, which has been shown to improve generalization and training stability when training transformer models. Learning-rate scheduling strategies such as cosine decay have also been widely used to stabilize training and improve convergence for large neural networks.

In this work, we evaluate how parameter-efficient fine-tuning via LoRA compares with full model fine-tuning when adapting GPT-2 to classification and generation tasks. We further analyze how different optimization strategies, including Adam and AdamW with cosine learning-rate scheduling, influence fine-tuning performance.

4 Approach

4.1 Model

All experiments use **GPT-2 small**, which contains approximately **117 million parameters**. GPT-2 is a decoder-only transformer model trained using a causal language modeling objective.

The model is adapted to two tasks:

1. Paraphrase detection (classification)
2. Sonnet generation (language modeling)

4.2 Full Fine-Tuning

In full fine-tuning, all model parameters are updated during training. The model weights are initialized from the pretrained GPT-2 checkpoint and optimized using gradient descent.

Given input tokens $x_1, x_2, \dots, x_T$, GPT-2 models the probability:

$$P(x_1, x_2, \dots, x_T) = \prod_{t=1}^T P(x_t | x_1, \dots, x_{t-1})$$

Training minimizes the cross-entropy loss:

$$L = - \sum_{t=1}^T \log P(x_t | x_{<t})$$

4.3 Low-Rank Adaptation (LoRA)

Instead of updating a weight matrix W , LoRA learns low-rank updates:

$$W' = W + \frac{\alpha}{r} BA$$

where

$$A \in \mathbb{R}^{r \times d}, \quad B \in \mathbb{R}^{d \times r}$$

Here r is the rank of the low-rank decomposition and α is a scaling factor that controls the magnitude of the LoRA update.

Unlike many implementations that rely on external libraries, we implemented the LoRA modules from scratch and inserted them directly into the query and value projection layers of the transformer self-attention blocks. Restricting LoRA to the Q and V matrices follows a common parameter-efficient adaptation design: these projections strongly influence how tokens attend to one another, while avoiding modifications to the full set of transformer weights. Implementing this ourselves also allowed us to precisely control how the low-rank updates were applied and to verify the resulting parameter counts and training behavior. The LoRA parameters were initialized using:

```
nn.init.normal_(self.lora_A.weight, std=0.02)
nn.init.zeros_(self.lora_B.weight)
```

This initialization ensures that the initial update BA is approximately zero, since the matrix B is initialized to zeros. As a result, the modified weight matrix initially satisfies

$$W' \approx W$$

meaning the model begins training with behavior identical to the pretrained GPT-2 weights. This helps stabilize training and prevents the LoRA updates from immediately perturbing the pretrained representations. The matrix A is initialized with a small Gaussian distribution so that once B begins updating during training, the low-rank adaptation can gradually learn task-specific updates.

This reduces the number of trainable parameters to **296,450**, compared to approximately **125M parameters** in full fine-tuning.

4.4 Last-Layer Fine-Tuning

As an additional parameter-efficient baseline, we evaluated a last-layer fine-tuning strategy in which only the final output layer of the model is updated while all earlier transformer layers remain frozen. This approach represents an extreme form of parameter-efficient adaptation, since only a very small number of parameters are trained compared to full fine-tuning.

In this setting, the pretrained GPT-2 representations remain fixed and the final layer is optimized to map these representations to the downstream task. While this approach significantly reduces the number of trainable parameters and computational cost, it limits the model’s ability to adapt internal representations to new tasks.

We include this baseline to evaluate whether minimal adaptation of the final layer alone can achieve competitive performance compared to LoRA and full-model fine-tuning.

4.5 Hyperparameter Search

To ensure a fair comparison between training strategies, we performed systematic hyperparameter tuning prior to the final experiments. The search was conducted in multiple stages using reduced training splits to quickly evaluate candidate configurations.

Hyperparameter	Values Explored
Fine-tuning Strategy	Full FT, LoRA, Last-layer-only
Optimizer	Adam, AdamW
Learning Rate (Full FT)	$\{1e^{-5}, 2e^{-5}, 3e^{-5}\}$
Learning Rate (LoRA)	$\{5e^{-5}, 1e^{-4}, 2e^{-4}, 3e^{-4}\}$
Learning Rate (Last Layer)	$\{5e^{-5}, 1e^{-4}, 2e^{-4}, 5e^{-4}, 1e^{-3}\}$
Weight Decay	$\{0.0, 0.01, 0.05\}$
Learning Rate Scheduler	None, Cosine
Warmup Steps	$\{0, 10, 20, 50\}$
LoRA Rank (r)	$\{4, 8, 16\}$
LoRA Scaling (α)	$\{16, 32\}$
LoRA Dropout	$\{0.0, 0.05, 0.1\}$
Random Seed	11711

Table 1: Hyperparameter search space explored across all experiments.

We first ran **39 trials** exploring optimizer and learning-rate configurations for full-model fine-tuning. Because LoRA introduces additional parameters, we conducted **24 additional trials** to tune LoRA-specific hyperparameters. Finally, we evaluated a last-layer-only baseline with **17 trials** in which all transformer layers were frozen and only the final layer was trained.

Across all experiments we evaluated **80 configurations for the paraphrase task and 31 configurations for the sonnet generation task**. The best-performing configurations identified during tuning were then used for the final experiments reported in Section 5. The explored hyperparameters and their ranges are summarized in Table 1.

5 Experiments

5.1 Tasks and Data

We evaluate our methods on two tasks from the CS224N default project.

Paraphrase Detection (Quora Question Pairs)

Determine whether two questions from the Quora dataset express the same meaning. Each example contains a pair of questions with a binary paraphrase label.

Sonnet Generation (Shakespeare Sonnet Dataset)

Generate Shakespearean-style sonnets using a dataset of Shakespeare’s 154 sonnets.

5.2 Evaluation Metrics

For the paraphrase detection task we report:

- Accuracy
- F1 score

For the sonnet generation task we report:

- Perplexity (lower is better)
- CHRF score (higher indicates better similarity to reference text)

5.3 Experimental Setup

All experiments use **GPT-2 small**.

Training runs use:

- 20 epochs with early stopping (patience = 3)

- Batch size: 8
- Random seed: 11711

To ensure a fair comparison, the final experiments were conducted using the best-performing hyperparameters identified during tuning. Because the two tasks differ in objective and optimization behavior, separate configurations were selected for paraphrase detection and sonnet generation.

Paraphrase Detection

- **Full FT + Adam:** learning rate 3×10^{-5} , no weight decay.
- **Full FT + AdamW + Cosine:** learning rate 3×10^{-5} , weight decay 0.01, cosine scheduler with 50 warmup steps.
- **LoRA + Adam:** learning rate 2×10^{-4} , LoRA rank $r = 8$, $\alpha = 16$, dropout 0.0.
- **LoRA + AdamW + Cosine:** learning rate 2×10^{-4} , weight decay 0.05, cosine scheduler with 0 warmup steps, LoRA rank $r = 8$, $\alpha = 32$, dropout 0.0.
- **Last-layer + Adam:** learning rate 5×10^{-4} , no weight decay.
- **Last-layer + AdamW + Cosine:** learning rate 5×10^{-4} , weight decay 0.01, cosine scheduler with 50 warmup steps.

Sonnet Generation

- **Full FT + Adam:** learning rate 3×10^{-5} , no weight decay.
- **Full FT + AdamW + Cosine:** learning rate 2×10^{-5} , weight decay 0.01, cosine scheduler with 20 warmup steps.
- **LoRA + Adam:** learning rate 3×10^{-4} , LoRA rank $r = 16$, $\alpha = 16$, dropout 0.0.
- **LoRA + AdamW + Cosine:** learning rate 3×10^{-4} , weight decay 0.05, cosine scheduler with 10 warmup steps, LoRA rank $r = 8$, $\alpha = 32$, dropout 0.05.

All models were trained for up to 20 epochs with early stopping (patience = 3) and batch size 8.

5.4 Paraphrase Detection Results

We compare six training strategies: full fine-tuning, LoRA fine-tuning, and last-layer-only training, each evaluated with Adam and AdamW + cosine learning-rate scheduling.

Method	Dev Accuracy	Dev F1
Full + AdamW + Cosine	0.8938	0.8913
Full + Adam	0.8872	0.8886
LoRA + AdamW + Cosine	0.8867	0.8808
LoRA + Adam	0.8830	0.8753
Last-layer + AdamW + Cosine	0.715	0.666
Last-layer + Adam	0.715	0.665

Table 2: Development performance on the paraphrase detection task.

Method	Test Accuracy
Full + AdamW + Cosine	0.877
LoRA + AdamW + Cosine	0.866

Table 3: Test accuracy on the paraphrase detection task from the leaderboard.

Full-model fine-tuning achieves the strongest performance, with AdamW and cosine learning-rate scheduling producing the best results. LoRA achieves comparable performance while updating far fewer parameters, demonstrating the effectiveness of parameter-efficient fine-tuning.

In contrast, last-layer-only fine-tuning performs substantially worse. Although it trains only a very small number of parameters, freezing the transformer layers prevents the model from adapting internal representations to the paraphrase detection task.

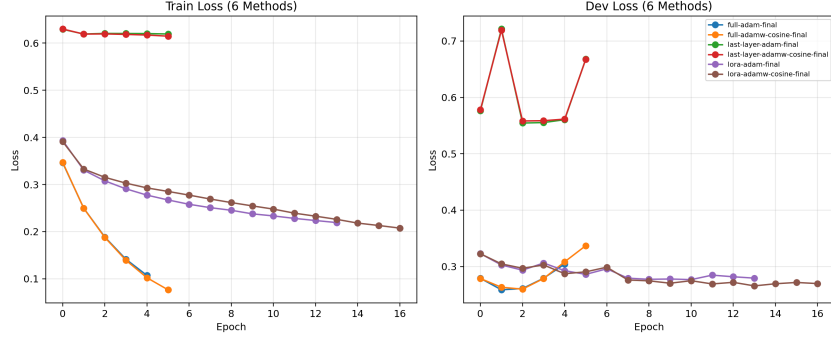


Figure 1: Training and validation loss curves for paraphrase detection.

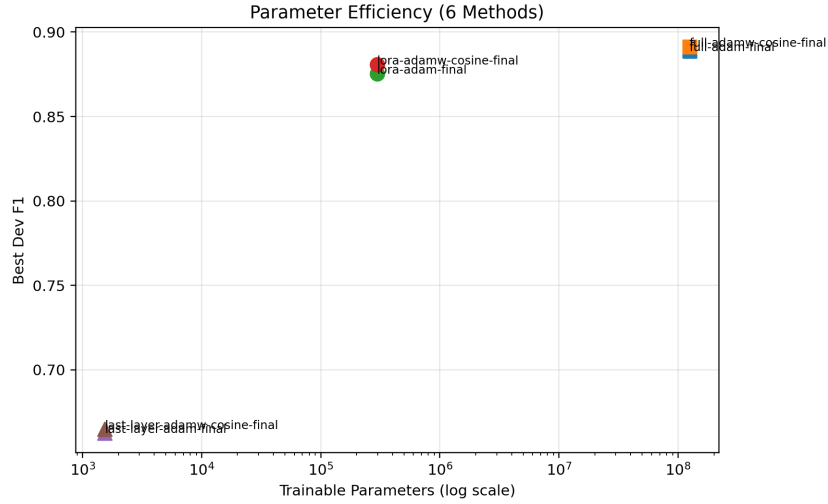


Figure 2: Parameter efficiency comparison across training strategies.

Figure 1 shows the training and validation loss curves for the different methods. Full fine-tuning converges quickly and achieves the lowest validation loss, while LoRA converges more gradually but reaches similar performance.

Finally, Figure 2 compares model performance with the number of trainable parameters. Full fine-tuning provides the highest performance but requires updating over 100 million parameters. LoRA achieves competitive results while training fewer than one million parameters. Last-layer fine-tuning requires only a few thousand parameters but suffers from a large drop in accuracy.

5.5 Sonnet Generation Results

Table 4 reports development perplexity for the sonnet generation task.

Method	Dev Perplexity
Full + AdamW + Cosine	70.92
Full + Adam	71.14
LoRA + Adam	76.09
LoRA + AdamW + Cosine	81.20

Table 4: Development perplexity for sonnet generation.

For the final evaluation we report CHRF similarity to reference sonnets.

Method	Test CHRF Score
Full + AdamW + Cosine	42.010
LoRA + Adam	37.998

Table 5: Test CHRF scores for sonnet generation from the leaderboard.

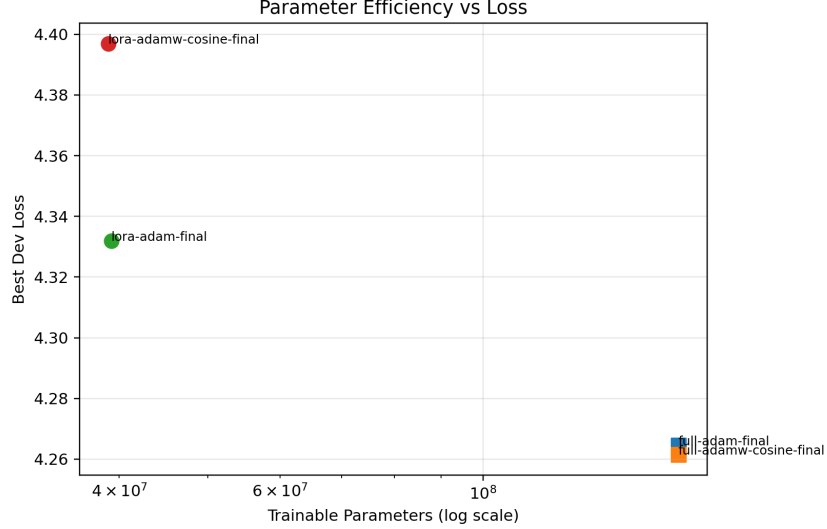


Figure 3: Sonnet generation parameter efficiency comparison.

Table 4 and Table 5 compare generation performance across fine-tuning strategies, while Figure 4 shows training loss curves.

Overall, the results across both tasks show that full fine-tuning achieves the best performance, while LoRA provides a strong albeit noticeably weaker parameter-efficient alternative with competitive results.

6 Analysis

To better understand the differences between training strategies, we conduct qualitative analysis of model predictions and generated text across both tasks.

6.1 Qualitative Analysis

We performed qualitative analysis for both tasks to better understand the behavioral differences between models.

Paraphrase Detection For paraphrase detection, qualitative inspection of individual examples revealed that most configurations behaved similarly on many sentence pairs. In many cases, both the fully fine-tuned models and the LoRA models produced identical predictions, suggesting that the decision boundary learned by the models is largely consistent across training methods. This aligns with the relatively small differences in accuracy reported in Table 3.

However, the slight performance gap indicates that full fine-tuning may capture subtle semantic distinctions slightly better than LoRA. These differences often arise in examples with strong lexical overlap but subtle semantic differences, where deeper representation learning may be beneficial.

Sonnet Generation Qualitative differences were more apparent in the sonnet generation task. We compared generated continuations for held-out prompts with the ground truth Shakespeare sonnets. For example, the reference text for Sonnet 132 begins:

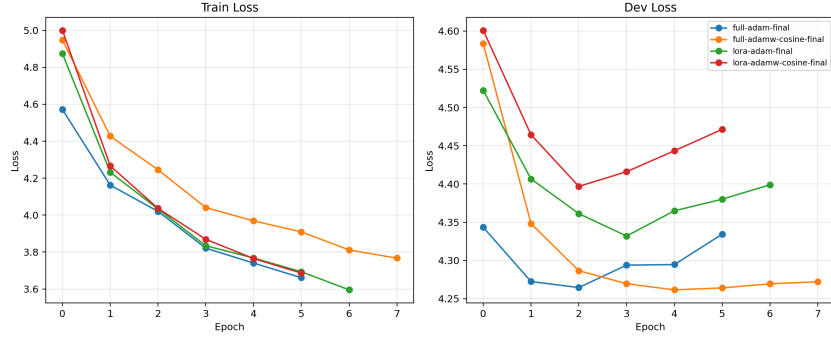


Figure 4: Training loss curves for sonnet generation.

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be,

which exhibits the characteristic vocabulary and imagery of Shakespearean poetry.

All models successfully reproduce the prompt lines, but their continuations differ in coherence and stylistic consistency.

For example, the **LoRA + Adam** model generates:

With silver's wax abaduces and tenured skirts scornful of kindred song;
Then lose pitying beauty I covet,
Effice on growing pains, when they year do pass;

while the **Full Fine-Tuning + AdamW + Cosine** model produces:

The grievful side kisses yet longer than thee.
Throw eyes oft and signs to patience,
Be fruitful sweet as suitors!

Although both outputs attempt to imitate poetic language, the LoRA model introduces more malformed phrases, while the fully fine-tuned model produces more grammatically coherent lines. These qualitative observations are consistent with the quantitative results: full fine-tuning achieves slightly higher CHRF scores on the sonnet generation task, while LoRA maintains competitive performance despite updating far fewer parameters.

7 Conclusion

In this project, we evaluated parameter-efficient fine-tuning methods for adapting GPT-2 to classification and generation tasks. We compared full fine-tuning with LoRA under different optimization strategies.

Our results show that full fine-tuning achieves the best performance overall, while LoRA provides a highly efficient alternative that reduces the number of trainable parameters by over two orders of magnitude with only a modest performance drop. These findings highlight the effectiveness of parameter-efficient fine-tuning when computational resources are limited.

One limitation of this study is that experiments were conducted on a relatively small model (GPT-2 small) and limited training data. Additionally, the qualitative evaluation of sonnet generation focused primarily on stylistic coherence rather than strict adherence to Shakespearean meter or rhyme structure. Future work could explore larger models and evaluate LoRA on additional generation tasks to determine whether similar trade-offs hold across different domains.

Team contributions (Required for multi-person team)

All parts were completed by Ryan He.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.
- [4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.
- [5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

A Appendix (optional)

If you wish, you can include an appendix, which should be part of the main PDF, and does not count towards the 6-8 page limit. Appendices can be useful to supply extra details, examples, figures, results, visualizations, etc., that you couldn't fit into the main paper. However, your grader *does not* have to read your appendix, and you should assume that you will be graded based on the content of the main part of your paper only.