

Adapting Pretrained GPT-2 via LoRA: How Much Fine-Tuning Do We Actually Need?

Stanford CS224N Default Project

Zengmingyu He

Department of Computer Science
Stanford University
zachx@stanford.edu

Zerong Chen

Department of Civil Engineering
Stanford University
zerongc2@stanford.edu

Abstract

Large pretrained language models are typically adapted via full fine-tuning until GPT came out in 2018, which can be very expensive. For a model like GPT-2 with 117 million parameters, training requires storing gradients and optimizer states for every single parameter. Low-Rank Adaptation (LoRA) offers a more efficient alternative by freezing the original weights and only training a small number of additional low-rank matrices. In this report, we implement GPT-2 from scratch and conduct a systematic study of how LoRA compares to full fine-tuning on two tasks: paraphrase detection and sonnet generation. We run a series of controlled experiments varying three key LoRA settings: the rank of the update matrices, which weight matrices to apply LoRA to, and which layers of the network to adapt LoRA. We also test a hybrid strategy that combines full fine-tuning in the upper layers with LoRA in the lower layers. Our results show that LoRA with all attention matrices and MLP layers achieves 88.01% accuracy on paraphrase detection, closing the gap with full fine-tuning's 89.85% while using less than 0.6% of its trainable parameters. Contrary to findings on GPT-3, we find that MLP adaptation is more impactful than attention-only LoRA at GPT-2 scale, and that the upper layers dominate task-specific adaptation for both classification and generation tasks. These findings suggest that for GPT-2-scale models, careful LoRA configuration can largely close the gap with full fine-tuning at much lower cost.

1 Key Information

- **Mentor:** Simon Kim
- **External Collaborators:** None
- **Sharing project:** No
- **Late Days:**

Team member	Late days remaining
Zengmingyu He	3 days
Zerong Chen	6 days
Total pooled	9 days
Team size	2
Late days applied for the report	3 days*2

2 Introduction

Large language models like GPT-2 [1] can acquire rich linguistic representations of language from massive amounts of unlabeled text. However, to apply them on a specific task, such as detecting

paraphrasing or generating text content according to certain format, they typically need to be fine-tuned on labeled data for the task. The most straight-forward way to achieve this is to update every parameter in the model, which is called full fine-tuning. This strategy typically works well but requires storing and computing gradients for all parameters. Also, if we want to use the same model on multiple tasks, we have to train separate copies of all those weights per task.

This cost becomes much more serious with larger models. As models have grown from GPT-2’s 117 million parameters to GPT-3’s 175 billion, the memory requirements of full fine-tuning have become prohibitive for many appliers. Researchers have been developing ways to adapt models to new tasks while updating far fewer parameters. These methods are called Parameter-Efficient Fine-Tuning (PEFT). One of the most popular and effective PEFT methods is Low-Rank Adaptation (LoRA) [2]. The core idea is: instead of training the original weight matrices directly, LoRA freezes them and adds small trainable matrices alongside them. These extra matrices are constrained to be low-rank, so the number of trainable parameters is much less than the full fine-tuning method.

LoRA comes with several design choices that can meaningfully affect its performance: How large should the rank be? Which weight matrices should have LoRA applied to them? Should we apply LoRA to all layers or just some? The original LoRA paper answers some of these questions for very large models like GPT-3, but it is less clear whether the same answers hold for smaller models like GPT-2. Our work addresses this directly through a controlled set of experiments.

Specifically, our contributions are:

1. We implement GPT-2 from scratch in PyTorch and implement LoRA as an injection into the mainly attention layers.
2. We compare LoRA against full fine-tuning on paraphrase detection (Quora Question Pairs) and Shakespearean sonnet generation.
3. We run ablation experiments varying rank ($r \in \{2, 4, 8, 16\}$), target weight matrices (various combinations of W_q , W_k , W_v , W_o , and MLP), and layer coverage (all layers, top 8, top 4).
4. We test a hybrid strategy — fully fine-tuning the top layers while applying LoRA to the lower layers — to see whether this can recover performance lost from partial fine-tuning.

3 Related Work

3.1 Pre-training and Fine-Tuning

The idea of pre-training a language model on large amounts of text and then adapting it to specific tasks was introduced by GPT-1 [3]. GPT-2 [1] showed that the same approach works even better at scale. In early exploration of post training, the standard way to adapt these models was full fine-tuning, where all model parameters were updated on the downstream task. This usually worked well but required significant compute and storage, especially as models get bigger.

A key insight from Aghajanyan et al. [4] helps explain why more efficient approaches might work: they showed that the weight changes during fine-tuning tend to lie in a surprisingly low-dimensional space. In other words, for the post training of GPT-2 model, we do not actually need to change all 117 million parameters — the useful changes concentrate in a much smaller subspace. This motivates approaches like LoRA that explicitly constrain the update to a low-dimensional space.

3.2 Low-Rank Adaptation (LoRA)

LoRA [2] works by adding a pair of small matrices to selected weight matrices in the transformer. For a frozen pretrained weight $W_0 \in \mathbb{R}^{d \times k}$, the adapted computation becomes:

$$h = W_0 x + \frac{\alpha}{r} \cdot B A x \quad (1)$$

where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ are the trainable matrices, r is the rank, and α is a scaling factor. Because $r \ll \min(d, k)$, the number of trainable parameters is much smaller than the original weight matrix. B is initialized to zero and A to a random Gaussian, so the adaptation starts from the same state as the pretrained model.

Rank. The rank r is the most important hyperparameter. Hu et al. [2] found on GPT-3 that LoRA performs competitively even at very small ranks, suggesting a useful update matrix can be very low-dimensional. It is not obvious the same holds at GPT-2 scale, since a smaller model may need relatively higher-rank updates to adapt. We test $r \in \{2, 4, 8, 16\}$ to characterize this.

Which matrices to adapt. Hu et al. [2] conduct an ablation on weight matrix selection under a fixed parameter budget, finding that concentrating all parameters in ΔW_q or ΔW_k alone yields significantly lower performance, while splitting the budget across W_q and W_v achieves the best result on GPT-3. We test a wider range of combinations, including applying LoRA to the MLP layers, to check whether this conclusion holds on GPT-2.

Which layers to adapt. Different layers of a transformer encode different kinds of information. Earlier layers tend to capture basic syntax while later layers are more sensitive to meaning and task identity [5]. This suggests that adapting only the upper layers might be enough. We test applying LoRA to only the top 8 or top 4 of GPT-2’s 12 layers.

3.3 Partial Fine-Tuning and Hybrid Strategies

An alternative to LoRA is to simply fine-tune only some layers fully while freezing the rest. We include this as a baseline and also test a hybrid approach: fully fine-tuning the upper layers while applying LoRA to the frozen lower layers. This hybrid strategy is motivated by the intuition that 1. upper layers need full flexibility for task-specific adaptation, while lower layers might benefit from lighter-weight LoRA updates rather than being frozen entirely; 2. The amount of trainable parameters for LoRA is negligible comparing to full fine-tuning.

4 Approach

4.1 GPT-2 Architecture

We implement GPT-2 (small) model architecture in PyTorch following Radford et al. [1]. The model uses embedding dimension $d = 768$, 12 transformer decoder layers, and 12 attention heads. Each transformer layer contains a masked multi-head self-attention block followed by a two-layer MLP with GELU activation, both with residual connections and pre-norm layer normalization. The model uses byte pair encoding (BPE) tokenization and processes sequences of up to 1024 tokens in a single example.

The masked multi-head self-attention prevents each token from attending to future positions, which is important for the autoregressive generation task. For a sequence of length T , the attention computation for each head is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d/h}}\right)V \quad (2)$$

where an upper-triangular mask is applied before the softmax. The implementation of model weights is completed by loading the official GPT-2 weights from HuggingFace.

4.2 LoRA Implementation

We implement LoRA by wrapping selected linear layers with a LoRA module. For each targeted weight matrix W_0 , we add trainable matrices A and B as described in Section 3. During training, W_0 is frozen and only A and B receive gradient updates. The scaling factor α/r is applied to the LoRA output before adding it to the original linear output. We implement LoRA as a modular wrapper so that we can flexibly choose which layers and which weight matrices to apply it to.

4.3 Task Formulations

Sentiment Analysis.

Input: a sentence/review x .

Output: sentiment label $y \in \{\text{very neg, neg, neutral, pos, very pos}\}$ (SST).

Goal: Trained with cross-entropy. Learn $p_\theta(y \mid x)$ and predict:

$$\hat{y} = \arg \max_y p_\theta(y \mid x), \quad (3)$$

Paraphrase Detection

Input: a sentence pair (x_1, x_2) .

Output: binary label $y \in \{0, 1\}$ where 1 means paraphrase/duplicate.

Goal: Trained with cross-entropy. Learn $p_\theta(y \mid x_1, x_2)$ and predict:

$$\hat{y} = \arg \max_y p_\theta(y \mid x_1, x_2), \quad (4)$$

Sonnet Generation

Input: context/prefix c (e.g., first 3 lines).

Output: continuation tokens $y = (y_1, \dots, y_T)$ (e.g., lines 4–14).

Goal: maximize conditional likelihood

$$\max_{\theta} \log p_\theta(y \mid c) = \sum_{t=1}^T \log p_\theta(y_t \mid c, y_{<t}), \quad (5)$$

4.4 Optimizer

We implement the Adam optimizer [6] using the efficient bias-correction formulation. For paraphrase detection, all models are trained with learning rate $\eta = 2 \times 10^{-5}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, weight decay = 0, and batch size = 32 for up to 20 epochs or convergence. For sonnet generation, batch size is set to 8, learning rate is set to $\eta = 5 \times 10^{-5}$, and $\eta = 2 \times 10^{-5}$ for LoRA. For all LoRA runs, we use $\alpha = 8$ as the scaling factor.

5 Experiments

5.1 Data

Stanford Sentiment Treebank (SST) [7]. Contains 11,855 single sentences from movie reviews, each labeled with one of five sentiment categories. Splits: 8,544 train / 1,101 dev / 2,210 test.

CFIMDB. Contains 2,434 longer movie reviews with binary sentiment labels (positive/negative). Splits: 1,701 train / 245 dev / 488 test.

Quora Question Pairs [8]. A binary paraphrase detection dataset of question pairs. We use the provided subset: 141,506 train / 20,215 dev / 40,431 test.

Shakespeare Sonnets. 154 sonnets following the ABAB CDCD EFEF GG rhyme scheme. Splits: 143 train / 12 test. At evaluation, we are given the first 3 lines of each test sonnet and asked to generate the rest.

5.2 Evaluation Method

For sentiment analysis and paraphrase detection, we report classification **accuracy**, the percentage of examples where the model’s prediction matches the gold label, on both train and development sets.

For sonnet generation, we report the **CHRF score** [9], a character-level n -gram F-score that measures how closely the generated text matches the original Shakespeare. We also qualitatively inspect generated sonnets to assess rhyme scheme and coherence.

5.3 Experimental Details

Our experiments consist of 16 total training runs with the settings described in Section 4.4 unless specifically described.

5.4 Results

Table 1 presents all results, organized into three ablation groups. The rank ablation (runs 4, 5, 6, 12) isolates the effect of rank by varying $r \in \{2, 4, 8, 16\}$ while holding all other factors fixed. The module ablation (runs 5, 7, 8, 9, 13, 14) fixes $r = 4$ and systematically varies which weight matrices receive LoRA updates, ranging from the minimal $\{W_q, W_v\}$ configuration to full attention plus MLP coverage. The layer coverage ablation (runs 1–3, 10, 11, 15, 16) compares three adaptation strategies: partial full fine-tuning, LoRA restricted to a subset of layers, and a hybrid approach combining full fine-tuning in the upper layers with LoRA in the lower layers.

Table 1: All Runs and Results for Paraphrase Detection and Sonnet Generation. Runs are grouped by ablation dimension. Trainable parameters are averaged across both tasks. FF = full fine-tuning.

Run	Configuration	α/r	Modules	Params	Dev Acc	Dev Score
<i>Baselines</i>						
—	Zero-shot	—	—	0	0.3680	39.350
1	Full FF	—	—	$\sim 125\text{M}$	0.8985	41.455
<i>Rank ablation</i>						
4	Full LoRA	8/2	W_q, W_v	$\sim 74.5\text{K}$	0.8522	41.058
5	Full LoRA	8/4	W_q, W_v	$\sim 148\text{K}$	0.8538	40.809
6	Full LoRA	8/8	W_q, W_v	$\sim 296\text{K}$	0.8558	41.012
12	Full LoRA	8/16	W_q, W_v	$\sim 590\text{K}$	0.8557	40.905
<i>Module ablation</i>						
5	Full LoRA	8/4	W_q, W_v	$\sim 148\text{K}$	0.8538	40.809
7	Full LoRA	8/4	W_q, W_k, W_v	$\sim 222\text{K}$	0.8566	41.437
8	Full LoRA	8/4	W_q, W_v, W_o	$\sim 222\text{K}$	0.8542	41.140
9	Full LoRA	8/4	W_q, W_k, W_v, W_o	$\sim 296\text{K}$	0.8568	41.323
13	Full LoRA	8/4	MLP only	$\sim 369\text{K}$	0.8679	40.226
14	Full LoRA	8/4	All attn + MLP	$\sim 664\text{K}$	0.8801	41.115
<i>Layer coverage ablation</i>						
1	Full FF	—	—	$\sim 125\text{M}$	0.8985	41.455
2	Top 8 FF	—	—	$\sim 56.7\text{M}$	0.8975	41.832
3	Top 4 FF	—	—	$\sim 28.3\text{M}$	0.8873	41.494
15	Top 8 LoRA	8/4	W_q, W_v	$\sim 99.1\text{K}$	0.8441	41.371
16	Top 4 LoRA	8/4	W_q, W_v	$\sim 49.9\text{K}$	0.8226	40.982
10	Top 8 FF + Btm 4 LoRA	8/4	W_q, W_v	$\sim 56.8\text{M}$	0.8962	41.431
11	Top 4 FF + Btm 8 LoRA	8/4	W_q, W_v	$\sim 28.4\text{M}$	0.8889	41.781

6 Analysis

6.1 Paraphrase Detection

Effect of Rank Runs 4, 5, 6, and 12 vary the rank $r \in \{2, 4, 8, 16\}$ while holding everything else fixed (full layers, $W_q + W_v$, $\alpha = 8$). Our results indicate that accuracy improves slightly from $r = 2$ (85.22%) to $r = 4$ (85.38%), but largely plateaus after that. Even when quadrupling the trainable parameters by pushing the rank to $r = 16$ (Run 12), the accuracy only marginally increases to 85.57%. This strongly supports the "intrinsic dimensionality" hypothesis: even for smaller models like GPT-2, the task-specific updates required for classification lie in an extremely low-dimensional subspace. For paraphrase detection, $r = 4$ represents the optimal trade-off between parameter efficiency and performance.

Which Weight Matrices to Adapt Runs 5, 7, 8, 9, 13, and 14 compare different combinations of weight matrices, all at $r = 4$, $\alpha = 8$, across all layers. This ablation yields our most surprising finding, which aligns with the most recent guest lecture shared by John Schulman and the Thinking Machines Lab in "LoRA Without Regret".

While the original LoRA paper found that adapting attention weights (W_q, W_v) was sufficient and adapting MLP layers was less effective, our GPT-2 results show the opposite. Applying LoRA exclusively to the MLP layers (Run 13) yields an accuracy of 86.79%, outperforming the standard

$W_q + W_v$ configuration (Run 5, 85.38%). Furthermore, when LoRA is applied to all attention matrices and the MLP (Run 14), accuracy jumps to 88.01%, significantly bridging the gap to full fine-tuning (89.85%).

We hypothesize that this discrepancy is due to model scale. While massive models like GPT-3 have sufficient semantic representations and only need attention re-routing to adapt to a new task, smaller models like GPT-2 may heavily rely on the factual and semantic knowledge stored in their feed-forward (MLP) layers to successfully identify complex paraphrases.

Layer Coverage We compare three strategies for choosing which layers to adapt: partial full fine-tuning (Runs 2-3), partial LoRA (Runs 15-16), and hybrid fine-tuning (Runs 10-11).

Restricting full fine-tuning to only the top 8 layers (Run 2) achieves 89.75%, nearly matching the 89.85% of full 12-layer fine-tuning (Run 1). However, restricting LoRA to only the top layers causes a severe degradation (From Run 4 to Run 16 drops from 85.38% to 82.26%).

The hybrid strategy reveals exactly where the model’s capacity bottleneck lies. Run 10 (Top 8 Full FF + Bottom 4 LoRA) achieves 89.62%, performing almost identically to Run 2 (Top 8 Full FF with bottom layers completely frozen). This demonstrates that for high-level semantic tasks like paraphrase detection, the lower layers that extract basic syntactic features have less impact on model performance. In other words, model performance is dictated mostly by the adaptation capacity of the upper layers, which require either full fine-tuning or extensive LoRA coverage (Attn+MLP).

6.2 Sonnet Generation: Quantitative and Qualitative Results

Quantitative. As shown in Table 1, the CHRF scores across different fine-tuning methods showed very little statistical variance. They mostly fluctuating around 40.5-41.5. At first glance, this might suggest that all configurations perform similarly. However, CHRF only measures character-level overlap. Because our dev set is extremely small, these scores can be heavily affected by random vocabulary collisions rather than true structural quality. Therefore, we rely on qualitative human inspection to truly understand how different LoRA configurations affect the generated text.

Qualitative. Pre-training Bias in Zero-Shot (Run 0). Without any fine-tuning, the pre-trained GPT-2 completely ignores the sonnet format. Because the prompt uses Elizabethan English, the model often relates this to religious texts or classic novels from its pre-training data. For example, in Sonnet 10, Run 0 abruptly transitions into quoting the Bible: *"But God gave him mercy to be his servant." (1 Pet 1) and "to come unto Christ?" (Rom 8)*. This clearly shows why we need task-specific fine-tuning to override the base model’s default behavior.

LoRA vs. Full Fine-Tuning (Run 1 vs. Run 5). Both full fine-tuning (Run 1) and our baseline LoRA ($r = 4$, W_q , W_v , Run 5) successfully fix this issue. They stop generating modern prose and naturally use Shakespearean vocabulary. For instance, Run 1 generates *"But what doth beauty think of thine eye?"* (Sonnet 0), and Run 5 generates *"And doth love more than any other light"* (Sonnet 1). It is interesting to note that Run 5 achieves almost the same stylistic alignment as Run 1, even though it only trains less than 1% of the parameters. This proves that the "style" of a sonnet can be effectively learned through low-rank updates.

The Failure of MLP-Only Adaptation (Run 13). Our results also confirm the hypothesis from previous literature that applying LoRA only to the MLP layers is not effective. While the model still uses some old English words, it loses its ability to form correct sentences. In Sonnet 1, Run 13 generates lines that awkwardly start with punctuation marks: *"? How can any thing which hath loved love stay"* and *", till all its beauty was cut down from thy sight"*. This happens possibly because the attention matrices are frozen, so the model struggles to maintain the context needed for poetry formatting.

Syntax Collapse in Top-Layer Adaptation (Run 16). One of the most interesting findings comes from Run 16, where we only apply LoRA to the top 4 layers and freeze the bottom 8 layers. The top layers attempt to enforce a poetic tone, but the frozen bottom layers (which handle basic grammar and structure) revert to generating modern prose. For example, the end of Sonnet 10 reads exactly like a modern fiction book: *"He was silent like an old friend when he saw her say these words aloud after that awful time did his father die away years ago from memory ."*. This suggests that for tasks

requiring a strict output format like poetry, updating the lower layers is necessary to reshape the foundational structure of the text.

7 Conclusion

In this work, we implemented GPT-2 and conducted a systematic comparison of LoRA against full fine-tuning on paraphrase detection and sonnet generation. Our ablations across rank, module selection, and layer coverage lead to several takeaways.

First, rank has diminishing returns beyond $r = 4$ for paraphrase detection: accuracy improves from 85.22% at $r = 2$ to 85.38% at $r = 4$, then plateaus through $r = 16$ (85.57%), consistent with the intrinsic dimensionality hypothesis of Aghajanyan et al. [4] and suggesting that task-specific updates can be useful at extremely low-dimensional even at GPT-2 scale.

Second, contrary to findings on GPT-3, MLP layers are the more impactful adaptation target at GPT-2 scale. Applying LoRA exclusively to MLP layers (86.79%) outperforms the standard $W_q + W_v$ configuration (85.38%), and combining all attention matrices with MLP achieves the best LoRA result at 88.01%. This could be that because smaller models like GPT-2 rely more heavily on MLP layers for semantic knowledge than its larger counterparts.

Third, upper layers dominate task-specific adaptation. Restricting full fine-tuning to only the top 8 layers nearly matches full fine-tuning (89.75% vs. 89.85%), while the hybrid strategy of top-8 full fine-tuning combined with bottom-4 LoRA (Run 10, 89.62%) confirms that lower-layer adaptation contributes only marginally to paraphrase detection. For sonnet generation, however, freezing the lower layers causes qualitative degradation in grammar and format, indicating that generation tasks place greater demands on the full model depth.

Overall, our best LoRA configuration (all attention + MLP, $r = 4$) closes most of the gap between zero-shot and full fine-tuning while using less than 0.6% of the trainable parameters. This demonstrates that carefully configured LoRA is a practical alternative to full fine-tuning at GPT-2 scale, particularly for classification tasks. However, the qualitative evidence from sonnet generation suggests that generation tasks with strict structural requirements may require more expressive adaptation.

One limitation of this study is the small sonnet test set (12 sonnets), which makes CHRF scores noisy. Another is that we only test GPT-2 small: it is very possible that the optimal LoRA configuration changes for larger models. Future work could explore applying these configurations to larger GPT-2 variants or other larger modern models. Combining the hybrid layer strategy with rank ablations for a more complete design space exploration can be meaningful as well.

Team Contributions

Zengmingyu He implemented the GPT-2 architecture including masked multi-head self-attention, transformer layers, and the embedding layer, implemented the Adam optimizer, implemented the sentiment classifier and sonnet generation pipelines and contributed to writing the final report.

Zerong Chen implemented the LoRA injection module, implemented the paraphrase detection pipelines, designed all ablation training runs, and contributed to writing the milestone and the final report.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.
- [2] 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.
- [3] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI*, 2018.

- [4] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Association for Computational Linguistics (ACL)*, 2021.
- [5] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Association for Computational Linguistics (ACL)*, 2019.
- [6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- [7] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.
- [8] Quora. First Quora dataset release: Question pairs. <https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs>, 2017.
- [9] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In *Workshop on Statistical Machine Translation (WMT)*, 2015.