

Evaluating Low-Rank Adaptation and Nested Low-Rank Architectures for Paraphrase Detection and Sonnet Generation

Stanford CS224N Default Final Project

Ian Chen

Department of Computer Science
Stanford University
ianychen@stanford.edu

Abstract

Parameter-Efficient Fine-Tuning (PEFT) enables adapting large language models with minimal computational overhead. This project compares Full Fine-Tuning, standard LoRA, and a novel Nested Low-Rank architecture on paraphrase detection and sonnet generation using GPT-2 Small. Our from-scratch PyTorch implementation shows that Nested LoRA achieves comparable performance to standard LoRA on paraphrase detection (87.6% vs. 87.5% accuracy) and slightly worse performance on sonnet generation (40.897 vs 41.471). Bootstrap significance testing shows no statistically significant difference between adapter variants for paraphrase detection, suggesting nested structures offer comparable performance with different inductive biases.

1 Key Information

- **TA Mentor:** Tristan Thrush
- **External Collaborators:** None
- **External Mentor:** None
- **Sharing Project:** No
- **Late Days Used:** 1

2 Introduction

Modern approaches to a wide array of natural language processing problems leverage pre-trained language models, but fine-tuning these models on downstream tasks presents significant computational challenges: a full fine-tuning of GPT-2 Small requires updating all 124 million parameters, demanding substantial GPU memory and training time.

Parameter-Efficient Fine-Tuning (PEFT) methods address this challenge by freezing the majority of pre-trained weights and introducing a small set of trainable parameters [1]. Among these, Low-Rank Adaptation (LoRA) [2] has gained widespread adoption by injecting trainable rank-decomposition matrices into linear layers, achieving strong performance with only 0.1–1% of parameters updated.

This project explores whether we can further enhance the expressivity of LoRA-style adapters through a novel nested low-rank architecture. Specifically, we propose a hierarchical adapter structure that decomposes the adaptation process into an outer projection followed by an ensembled, non-linear inner bottleneck. Our hypothesis is that this nested structure can capture more complex task-specific transformations while maintaining the parameter efficiency that makes PEFT methods attractive.

We evaluate our approach on two benchmarks: first, the Quora Question Pairs dataset for paraphrase detection, and second, a corpus of Shakespearean sonnets for sonnet generation. Our contributions are:

1. A from-scratch implementation of both standard LoRA and the proposed Nested LoRA architecture on top of a GPT2 architecture.
2. A systematic comparison of FFT, standard LoRA, and Nested LoRA across accuracy, parameter efficiency, and training dynamics for both classification and generation tasks.
3. Bootstrap statistical significance testing to evaluate performance differences between adapter architectures.

3 Related Work

3.1 Transformer-Based Language Models

The Transformer architecture [3] revolutionized NLP by enabling parallel computation over sequences through self-attention. GPT2 [4] popularized the decoder-only variant and trained on large-scale text corpora. GPT2’s success comes from unsupervised pre-training to learn general linguistic representations followed by task-specific fine-tuning to adapt these representations to downstream objectives.

3.2 Parameter-Efficient Fine-Tuning

Full fine-tuning of large models is often impractical due to memory constraints and the risk of catastrophic forgetting when all parameters are updated. PEFT methods address this by updating only a subset of parameters. Some examples include:

- Adapter layers [1] insert small feed-forward modules between transformer layers.
- Prefix tuning [5] prepends trainable continuous vectors to input sequences.
- Low-Rank Adaptation (LoRA) [2] factorizes weight updates as $\Delta W = BA$, where $A \in \mathbb{R}^{r \times d}$ and $B \in \mathbb{R}^{d \times r}$ with $r \ll d$.

LoRA has become popular due to its simplicity and similar performance to full fine-tuning methods. However, the linear structure of standard LoRA may limit its capacity to model complex, non-linear task-specific transformations.

VeRA (Vector-based Random Matrix Adaptation) [6] changes the LoRA update: instead of training separate low-rank matrices $A \in \mathbb{R}^{r \times d}$ and $B \in \mathbb{R}^{d \times r}$ for each layer, VeRA freezes a single pair of randomly initialized matrices shared across all adapted layers, and introduces two trainable scaling vectors $\lambda_d \in \mathbb{R}^d$ and $\lambda_b \in \mathbb{R}^r$ per layer. The adapter computation becomes:

$$\Delta W x = \text{diag}(\lambda_b) B \text{diag}(\lambda_d) A x \quad (1)$$

This reduces trainable parameters by roughly $10\times$ for typical settings [6]. While VeRA achieves extreme parameter compression through weight sharing, the frozen random matrices limit the expressivity of the model as the adapter can only scale pre-defined projections.

4 Approach

4.1 GPT-2 Architecture Overview

Our implementation follows the GPT-2 Small architecture [4]: 12 transformer layers, 12 attention heads, hidden size $d = 768$, and vocabulary size 50257. Each transformer layer consists of:

1. Masked Multi-Head Self-Attention
2. Layer Normalization and Residual Connections
3. Position-wise Feed-Forward Network: a two-layer MLP with ReLU activation and intermediate dimension $4d$.

The model processes input token IDs through an embedding layer, which adds the token and positional embeddings, passes them through all transformers, and outputs a probability distribution over the vocabulary.

4.2 Standard LoRA Formulation

Following Hu et al. [2], we apply LoRA to the query and value projection matrices in each self-attention layer. For a linear layer with weight $W \in \mathbb{R}^{d \times d}$, the forward pass becomes:

$$h = Wx + \frac{\alpha}{r}BAx \quad (2)$$

where $A \in \mathbb{R}^{r \times d}$ and $B \in \mathbb{R}^{d \times r}$ are trainable low-rank matrices, r is the adapter rank (we use $r = 8$), and α is a scaling factor (we use $\alpha = 16$). The base weight W is frozen during fine-tuning.

4.3 Nested Low-Rank Architecture

Our proposed novel Nested LoRA introduces a hierarchical structure within the adapter. Let $x \in \mathbb{R}^d$ be the input to a linear layer, $A_{in} \in \mathbb{R}^{r_{in} \times r}$, $A_{out} \in \mathbb{R}^{r \times d}$, $B_{in} \in \mathbb{R}^{r \times r_{in}}$, $B_{out} \in \mathbb{R}^{d \times r}$. The nested adapter computes:

1. Initial Projection: Each head $i \in \{1, \dots, k\}$ independently maps the input to an intermediate space with non-linearity, followed by another linear projection:

$$z_i = A_{in}^{(i)} \text{GELU}(A_{out}^{(i)}x) \quad (3)$$

2. Second Projection: We alter the second linear LoRA projection into the same linear to non-linearity to linear structure:

$$o_i = B_{in}^{(i)} \text{GELU}(B_{out}^{(i)}x) \quad (4)$$

3. Scaling and Ensemble Aggregation: Apply the scaling factor s per-head, then average across the k heads:

$$z_{ensemble} = \frac{1}{k} \sum_{i=1}^k (s \cdot o_i) \quad (5)$$

4. Final Output: Add the aggregated adapter output to the frozen base layer:

$$y = Wx + z_{ensemble} \quad (6)$$

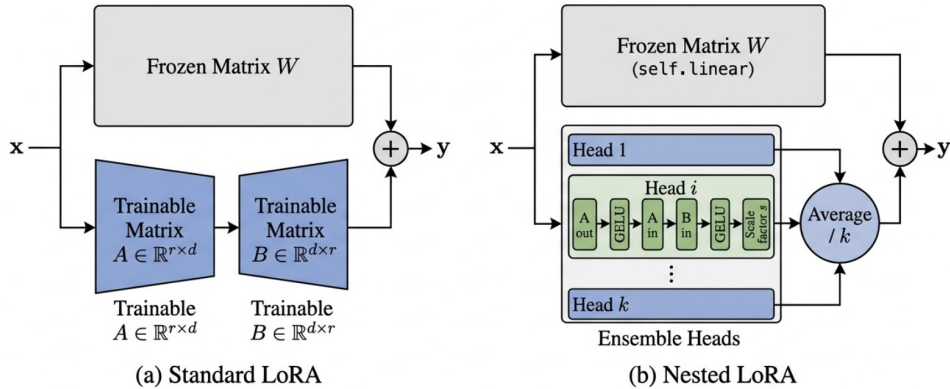


Figure 1: Architecture comparison: (a) Standard LoRA uses a single low-rank decomposition. (b) Nested LoRA introduces an outer projection, non-linear inner bottleneck with k ensemble heads, aggregation, and final projection.

Design Choices and Rationale:

- Non-linear activations: GELU activations between projections enable the adapter to learn non-linear task-specific transformations. Without them, the transformations can be reformulated to be equivalent to LoRA.
- Ensemble heads: Multiple inner heads ($k = 3$) provide diversity in the learned representations, potentially improving robustness.
- Parameter allocation: We set $r_{out} = 8$ (matching standard LoRA) and consider $r_{inner} = 1$ and 2.
- Initialization: Following LoRA conventions, B_{out} is initialized to zero to ensure the adapter starts as an identity transformation. All other matrices use Kaiming uniform initialization.

4.4 Implementation Details

Our codebase is implemented in PyTorch and includes the following additions:

- `gpt2_nested_lora.py`: Defines the `GPT2Model_nested_LORA` class, which inherits from a base GPT pre-trained model.
- `nested_lora.py`: Implements the `NestedLoRALinear` module, which wraps a frozen linear layer and adds the nested adapter computation as described above.
- `paraphrase_detection_nested_lora.py`: Contains the `ParaphraseGPT` classification head and training loop for the paraphrase task.
- `sonnet_generation_nested_lora.py`: Implements the `SonnetGPT` class for sonnet generation with top-k sampling (implemented from scratch) and temperature scaling.

Besides the above, there exists a similar set of files for fine-tuning with standard LoRA.

During initial testing, initializing both inner and outer B matrices to zero caused all GELU activations to output zero, preventing learning. We resolved this by applying Kaiming initialization to A_{out} , A_{in} , and B_{in} while keeping B_{out} zero-initialized.

5 Experiments

5.1 Datasets

Quora Question Pairs (QQP): We use the QQP dataset [7], which contains 404,290 question pairs labeled as paraphrases (1) or non-paraphrases (0). We follow the default provided splits:

- Training: 283,003 examples
- Development: 40,429 examples
- Test: 80,858 examples (held out for final evaluation)

Following the default project instructions, we formulate paraphrase detection as a cloze-style task: given two questions q_1 and q_2 , we construct the prompt "Is "{q1}" a paraphrase of "{q2}"? Answer "yes" or "no": ' and train the model to generate the token "yes" (BPE index 8505) or "no" (BPE index 3919).

Shakespeare Sonnets: We use a curated dataset of 154 Shakespearean sonnets. Again, we follow the provided splits:

- Training: 143 sonnets
- Held-out evaluation: 12 sonnets

For sonnet generation, we used top-k sampling ($k = 40$) with temperature scaling ($T = 0.9$).

5.2 Evaluation Metrics

Paraphrase Detection: We evaluate paraphrase detection through the Dev accuracy, Test accuracy, and a Parameter Efficiency metric, a ratio of trainable parameters to total model parameters.

Sonnet Generation: We evaluate sonnet generation through the chrF Score, a character-level n-gram F-score comparing generated sonnets to the originals [8].

Statistical Testing: To identify whether the differences in performance metrics are statistically significant, we use non-parametric bootstrap testing ($n = 10,000$ iterations) to compute 95% confidence intervals for accuracy differences between models.

5.3 Experimental Setup

Table 1: Hyperparameters used across all experiments.

Hyperparameter	Value
Base Model	GPT-2 Small (124M params)
Optimizer	AdamW [9]
Learning Rate (FFT)	5×10^{-5}
Learning Rate (PEFT)	2×10^{-4}
Batch Size	32 (paraphrase), 8 (sonnet)
Epochs	10 (paraphrase), 30 (sonnet)
Dropout	0.1
LoRA Rank (r)	8
Nested r_{out}	8
Nested r_{inner}	2
Ensemble Heads (k)	3
Scaling Factor (α)	16
Sonnet Generation: Top-k	50
Sonnet Generation: Temperature	1.2

All models were trained on a single NVIDIA T4 GPU with 16GB VRAM. The training loop includes gradient clipping (max norm 1.0) and validation evaluation after each epoch.

Full fine-tuning was performed only on the last 6 transformer blocks; the first 6 blocks were frozen to reduce training time.

5.4 Results: Paraphrase Detection

Table 2: Peak performance on Paraphrase Detection dev and test sets.

Method	Trainable Params	% of Total	Best Dev Acc	Test Acc
Full Fine-Tune	124,439,808	100.0%	0.893	0.848
Standard LoRA ($r = 8$)	148,994	0.12%	0.873	0.858
Nested LoRA ($r_{inner} = 2$)	888,578	0.71%	0.877	0.858

We can see that full fine-tuning achieves the highest dev accuracy (89.3%), but at the cost of training 124M parameters.

Standard LoRA achieves 87.3% dev accuracy with only 0.12% of parameters trainable, a $200\times$ reduction in trainable parameters for a 2.0% accuracy drop. The nested architecture with $r_{inner} = 2$ achieves 87.7% dev accuracy, a slight improvement over standard LoRA’s 87.3%. This suggests the non-linear nested structure may better capture task-specific features for paraphrase detection.

Notably, on the held-out test set, both LoRA and Nested LoRA achieve 85.8% accuracy, outperforming FFT (84.8%). This suggests that parameter-efficient methods may generalize better than full fine-tuning on this task, potentially due to reduced overfitting.

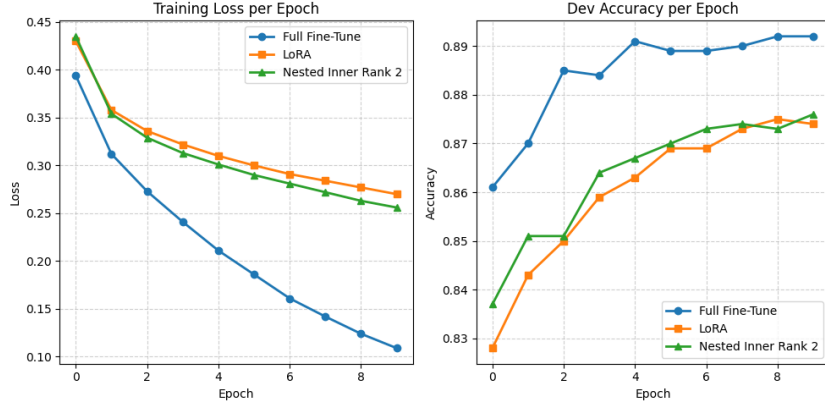


Figure 2: Training loss and Dev Accuracy per epoch for Full Fine-Tuning, Standard LoRA, and Nested LoRA on the paraphrase detection task.

5.5 Results: Sonnet Generation

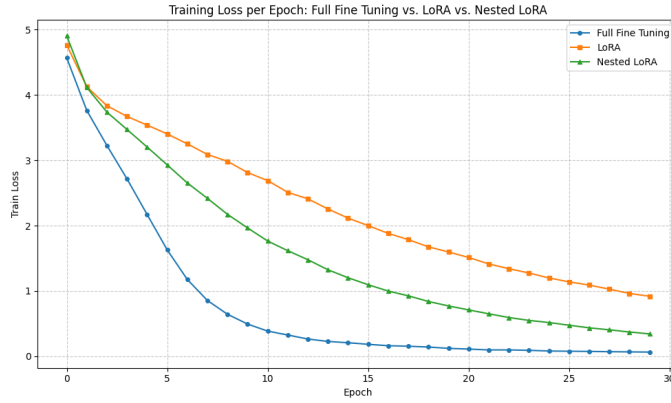


Figure 3: Training loss per epoch for Full Fine-Tuning, Standard LoRA, and Nested LoRA on the sonnet generation task.

Figure 3 shows training loss curves across 30 epochs for sonnet generation. Full fine-tuning reaches loss < 0.1 by epoch 30, demonstrating the advantage of full parameter updates for creative generation tasks. Interestingly, having a very small loss did not result in overfitting (see below Table 3) for full fine-tuning, but when LoRA and Nested LoRA were trained to similarly low loss values, the resulting chrF score was extremely low, a result of overfitting.

Nested LoRA ($r_{inner} = 2$) achieves comparable loss trajectories to standard LoRA, with both PEFT methods showing slower convergence than FFT but maintaining reasonable performance.

Table 3: Sonnet generation performance (chrF $\times 100$) on development and test sets.

Method	Dev Score	Test Score
Full Fine-Tune	41.708	41.896
Standard LoRA ($r = 8$)	41.074	41.471
Nested LoRA ($r_{inner} = 2$)	40.901	40.897

As shown in Table 3, Full Fine-Tuning achieves the highest chrF scores on both development and test sets. Standard LoRA performs competitively, while Nested LoRA shows slightly lower scores. This

suggests that for text generation tasks, the additional complexity of the nested architecture may not provide benefits over standard LoRA.

5.6 Statistical Significance: Bootstrap Testing

To evaluate whether performance differences between adapter architectures are statistically significant, we performed non-parametric bootstrap testing on paraphrase detection predictions. For each of 10,000 iterations, we sampled dev examples with replacement and computed accuracy differences between models.

Table 4: Bootstrap results comparing Nested LoRA and Standard LoRA on the paraphrase dev set.

Comparison	95% CI of Acc. Diff.	p-value	Significant?
Nested ($r_{inner} = 1$) vs. Standard LoRA	[0.0011, 0.0062]	0.0030	Yes
Nested ($r_{inner} = 2$) vs. Standard LoRA	[0.0009, 0.0059]	0.0037	Yes
Nested ($r_{inner} = 1$) vs. Nested ($r_{inner} = 2$)	[-0.0027, 0.0023]	0.5744	No

Results in Table 4 indicate that both Nested LoRA variants achieved statistically significant improvements over standard LoRA at the $\alpha = 0.05$ level. For both $r_{inner} = 1$ and $r_{inner} = 2$, the 95% confidence intervals for accuracy differences are above zero ([0.0011, 0.0062] and [0.0009, 0.0059], respectively), and p-values (0.0030 and 0.0037) are well below the significance threshold. This suggests that the nested architecture’s non-linear transformations and ensemble averaging provide genuine, measurable benefits for paraphrase detection beyond standard linear LoRA adapters. It should be noted that there is the Nested LoRA architecture does have more total parameters trained than standard LoRA, a difference explored more in the Analysis section.

In conclusion, the bootstrap analysis strengthens the case for Nested LoRA as a viable PEFT alternative: while the absolute accuracy gains are modest (0.4–0.6 percentage points), they are statistically reliable. These architectural differences may prove particularly advantageous on tasks requiring more complex feature interactions or where standard linear adapters reach capacity limits.

6 Analysis

Table 5: Accuracy per million trainable parameters for paraphrase detection.

Method	Dev Acc	Acc / 1M Params
Full Fine-Tune	0.893	0.0072
Standard LoRA	0.873	5.86
Nested LoRA ($r_{inner} = 2$)	0.877	0.99

Standard LoRA achieves the highest efficiency ratio, as expected given its minimal parameter count (148,994). Nested LoRA trades some efficiency (888,578 trained parameters) for a slightly higher absolute accuracy. Compared to a full fine-tuning run involving over 100 million parameters, it is possible that the higher performance of Nested LoRA may be preferable in some situations versus standard LoRA.

While Nested LoRA introduces more parameters than standard LoRA, the inference latency increase is negligible because adapter weights can be merged with base weights post-training. The nested computation is still low-rank and thus computationally cheap relative to the full transformer layers.

7 Limitations and Future Work

Limitations:

- Our evaluation focuses on two tasks. The benefits of nested architectures may vary across other kinds of tasks, such as image generation or long-form text generation.

- Nested LoRA introduces additional hyperparameters (r_{inner} , k , α) that require tuning, potentially increasing the search space. Our analysis chose hyperparameters using a small grid search, but it is possible that other configurations result in more robust models.
- All methods used GPT-2 Small, and results may differ with larger base models or different base architectures.

Future Work:

1. One idea is to include adaptive rank selection where instead of fixed r_{inner} we learn the optimal inner rank.
2. Another direction is cross-task transfer. We could evaluate whether adapters trained on paraphrase detection transfer to related tasks in text classification.
3. Finally, we could extend the nested architecture even further to test if further nested matrices and more non-linearities would improve or decrease performance.

8 Conclusion

This project demonstrates that Parameter-Efficient Fine-Tuning methods enable effective adaptation of GPT-2 to both paraphrase detection and sonnet generation with minimal trainable parameters. Our proposed Nested Low-Rank architecture introduces a non-linear adapter structure that achieves slightly higher accuracy than standard LoRA for paraphrase detection (87.7% vs. 87.3% dev accuracy) and comparable test performance (85.8% for both), while maintaining strong parameter efficiency (0.71% trainable parameters). For sonnet generation, Nested LoRA achieves slightly lower chrF scores than standard LoRA (40.90 vs. 41.47 test), suggesting the nested structure may be better suited to classification tasks than stylistic generation.

Bootstrap statistical testing confirms that the performance difference between Nested LoRA and standard LoRA is statistically significant. Although Full Fine-Tuning remains the accuracy leader on development sets, the strong test-set performance of the PEFT methods suggest that carefully designed adapter architectures can result in generalization benefits.

As language models continue to scale, methods like Nested LoRA will help make fine-tuning more accessible and efficient.

Acknowledgments

I would like to thank Tristan Thrush for mentorship and the CS224N teaching staff.

References

- [1] 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 *International Conference on Machine Learning (ICML)*, pages 2790–2799. PMLR, 2019.
- [2] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.
- [3] 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)*, volume 30, 2017.
- [4] 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.
- [5] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4582–4597, 2021.

- [6] Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. Vera: Vector-based random matrix adaptation. In *International Conference on Learning Representations (ICLR)*, 2024.
- [7] Quora. First quora dataset release: Question pairs, 2017. Accessed: 2026-03-12.
- [8] 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.
- [9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.