

Efficiency–Performance Trade-offs in LoRA-family Fine-Tuning Methods for GPT-2

Stanford CS224N Default Project
Mentor: Sarah Chen

Christine Li
Electrical Engineering
Stanford University
cli27@stanford.edu

Zetian Li
ICME
Stanford University
zetianli@stanford.edu

Jason Yan
Computer Science
Stanford University
jasonyan@stanford.edu

Abstract

Low-Rank Adaptation (LoRA) is a leading parameter-efficient fine-tuning technique for adapting large language models to downstream tasks. However, the trade-off between performance and wall-clock efficiency across different LoRA variants remains underexplored. We conduct a systematic empirical study of LoRA-family methods, including LoRA, rsLoRA, DoRA, and AdaLoRA, across two tasks: paraphrase detection and sonnet generation. Our experiments evaluate these methods’ performance-efficiency trade-offs across different model scales, configurations, and dataset sizes. The results show that LoRA-based methods generally achieve performance comparable to full fine-tuning but do not consistently improve wall-clock training efficiency. Standard LoRA and rsLoRA provide the most favorable trade-offs, while more complex variants such as AdaLoRA and DoRA often introduce additional training overhead and instability. We further show that optimal adapter rank depends strongly on dataset scale, with lower ranks sufficient for small datasets and larger ranks benefiting from augmented data.

1 Introduction

Low-Rank Adaptation (LoRA) [1] is one of the most widely used parameter-efficient fine-tuning (PEFT) techniques for adapting large language models to downstream tasks without updating the full set of model weights. By freezing the pretrained model parameters and optimizing only low-rank decomposition matrices, LoRA significantly reduces the number of trainable parameters and GPU memory usage with performance comparable to full fine-tuning (Full FT) [1]. However, the efficiency of LoRA has rarely been rigorously evaluated in terms of wall-clock training time. In addition, the relationship between adapter rank and dataset size remains underexplored: it is unclear how aggressively the rank can be reduced before performance drops, especially in low-resource settings.

Beyond the original LoRA formulation, a large family of LoRA variants modifies the adaptation mechanism to improve different aspects of the method [2], which may lead to different trade-offs between adaptation efficiency and model performance on downstream tasks. However, comprehensive empirical comparisons across these LoRA variants remain limited. Moreover, their relative advantages may vary depending on model scale and dataset size, which further motivates systematic investigation.

To address these gaps, we fine-tune GPT-2 Small and GPT-2 Medium models [3] on two downstream tasks using several representative LoRA-family approaches, including LoRA, rank-stabilized LoRA (rsLoRA) [4], Weight-Decomposed Low-Rank Adaptation (DoRA) [5], and Adaptive Low-Rank Adaptation (AdaLoRA) [6]. The downstream tasks are paraphrase detection, which represents semantic understanding, and sonnet generation, which represents controlled text generation. Through extensive empirical experiments, our contributions are as follows:

- We evaluate multiple LoRA-family methods across model scales and tasks and explore their optimal configurations, especially which layers to apply, analyzing how their adaptation mechanisms affect downstream performance.
- We benchmark task performance and time efficiency of LoRA-based methods against Full FT, providing a systematic comparison of their performance–efficiency trade-offs.
- We augment the training dataset and study how LoRA adapter rank interacts with dataset size, quantifying how aggressively the rank can be reduced before performance deteriorates, especially in low-resource sonnet generation.

Our experiments reveal several practical insights. While LoRA-based approaches achieve performance comparable to full fine-tuning, their advantages in wall-clock training efficiency are less consistent than commonly assumed. Among the evaluated variants, standard LoRA and rsLoRA provide the most favorable performance–efficiency trade-offs. We further show that model scale, adapter placement, and the volume of training data significantly influence adaptation behavior, highlighting the importance of careful adapter design in parameter-efficient fine-tuning.

2 Related Work

Large pretrained language models are typically adapted to downstream tasks through fine-tuning, but updating all model parameters requires extensive computational resources and is inefficient for multiple task-specific deployments. This bottleneck led to a surge in the development of PEFT techniques [2], which aim to reduce the number of fine-tuning parameters. Prefix-Tuning [7] and Prompt Tuning [8] adapt models by learning small sets of task-specific continuous prompts while keeping the backbone model frozen. More recently, QLoRA [9] combines low-rank adaptation with quantized base models to further reduce memory requirements during fine-tuning. These methods are particularly attractive in environments with limited computational resources.

Low-Rank Adaptation (LoRA) [1] adopts a different strategy by learning a pair of low-rank matrices that approximate the weight update. The paper shows strong empirical results that low-rank updates approximate Full FT well across a range of models and tasks. Given its simplicity and effectiveness, LoRA provides a strong foundation to our experiments. However, existing evaluations primarily focus on parameter count and memory efficiency, and we are interested in the savings of wall-clock training time. In addition, LoRA rests on the assumption that pretrained models already encode substantial linguistic structure, and useful updates are low-rank. This motivates us to examine how the benefits of LoRA vary across model scales and to investigate the relationship between LoRA rank and dataset size, particularly in low-resource settings.

Following the success of LoRA, many LoRA variants have been proposed to improve its adaptation capacity and training dynamics. DoRA [5] decomposes pretrained weight matrices into magnitude and directional components and applies low-rank adaptation to the directional component. This formulation improves the representational capacity of the adaptation while preserving the efficiency benefits of LoRA. Another extension, AdaLoRA [6] dynamically allocates the rank budget during training by pruning less important rank components and redistributing capacity to more influential ones, allowing the model to automatically determine an effective parameter allocation strategy. These variants introduce different mechanisms to balance adaptation efficiency and performance. However, systematic empirical comparisons across these LoRA-family methods remain limited, particularly when considering different tasks, model scales, and dataset sizes.

3 Approach

We reimplemented all LoRA-based techniques from scratch to add more control to our experiments.

LoRA LoRA freezes pretrained matrix W_0 and only optimizes low-rank factors $\Delta W = BA$, producing an effective weight

$$W = W_0 + \frac{\alpha}{r} \Delta W, \quad \Delta W = BA, \quad (1)$$

where $A \in \mathbb{R}^{r \times d_{in}}$, $B \in \mathbb{R}^{d_{out} \times r}$ with rank $r \ll \min(d_{in}, d_{out})$, and α is a scaling hyperparameter. Building on this, we enable LoRA to be applied only to targeted modules (attention-only, MLP-only, and all modules), while keeping the rest frozen.

rsLoRA rsLoRA changes the scaling rule of LoRA from $\frac{\alpha}{r}$ to $\frac{\alpha}{\sqrt{r}}$, so the update magnitude does not shrink too aggressively when r increases [4]. This makes optimization and hyperparameter transfer across ranks more stable in our experiments.

DoRA. DoRA decomposes pretrained weights into magnitude and directional components, applying low-rank adaptation to the direction while learning the magnitude separately [5]:

$$W = g \cdot \frac{W_0 + \Delta W}{\|W_0 + \Delta W\|} \quad \Delta W = BA, \quad (2)$$

where g is a learned scale parameter. The normalization is applied row-wise so each output neuron has its own magnitude. The training then updates only A , B , and g . As in LoRA, we also enable module-selective DoRA.

AdaLoRA. Instead of fixing the same rank for all layers, AdaLoRA dynamically reallocates the LoRA rank based on the importance of different weights [6]. It parameterizes the weight update using an SVD-style decomposition

$$\Delta W = U \Sigma V^\top, \quad (3)$$

where $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r)$ contains learnable singular values controlling the contribution of each rank component. To measure the importance of each component, AdaLoRA estimates a sensitivity score derived from the first-order Taylor approximation of the loss with respect to σ_i :

$$\Delta \mathcal{L} \approx \frac{\partial \mathcal{L}}{\partial \sigma_i} \sigma_i. \quad (4)$$

Thus the importance of the i -th rank component is defined as

$$I_i = \left| \sigma_i \frac{\partial \mathcal{L}}{\partial \sigma_i} \right|, \quad (5)$$

which approximates the change in loss if that component were removed. Our implementation applies AdaLoRA to all linear layers. During training, AdaLoRA periodically evaluates each component’s importance and prunes less important ones, while redistributing the rank budget to more important layers, subject to a global rank budget. This adaptive rank allocation allows the model to concentrate parameter capacity on the most task-relevant weights while maintaining overall parameter efficiency.

FlashAttention We further adopted FlashAttention [10] in our attention module using PyTorch’s `scaled_dot_product_attention` (SDPA) API. This can leverage FlashAttention-style tiled, fused kernels when available, enabling higher throughput and often better GPU utilization to accelerate attention without changing model behavior.

4 Data

In addition to the default datasets, we include Hugging Face datasets that are reformatted to match the provided data loader. A detailed breakdown of the data augmentation is in Table A.1 in the Appendix.

Paraphrase Detection The dataset consists of pairs of questions with a binary label indicating whether they are paraphrases. We add about 380k samples from `en-wiki-paraphrased` [11] and `llm-paraphrases` [12], which contain large collections of paraphrase pairs generated from Wikipedia and LLMs. Specifically, because the original dataset contains only 36.8% positive pairs, we randomly sample from the additional datasets to rebalance the class distribution. To further improve classifier robustness, we construct challenging negative pairs using the following strategies:

- **Embedding-similar negatives.** Sentences that are close in embedding space but not paraphrases are paired together. These pairs often share overlapping vocabulary while differing in meaning, encouraging the model to learn sharper decision boundaries.

- **Topic-similar negatives.** Sentences are encoded and clustered with MiniBatchKMeans. For each sentence, we pair it with a nearest neighbor from the same cluster while filtering by a similarity band to avoid trivial negatives or likely paraphrases.
- **Near-miss semantic edits.** We generate adversarial-style negatives by making small semantic edits (e.g., polarity flips, entity substitutions, or numeric changes) and pairing them with the original sentences. Although these pairs remain lexically similar, their meanings differ.

Sonnet Generation Since the original dataset contains only 143 Shakespeare sonnets, we augment it with 920 LLM-generated sonnets from `shakespeare_sonnets_diffused` [13]. Given the first three lines of a sonnet, the model outputs a continuation that completes the poem.

5 Experiments

5.1 Evaluation Method

Model performance is evaluated using dev-set accuracy for paraphrase detection and chrF scores for sonnet generation. Time efficiency is measured by total wall-clock training time (in hours) and dev-set inference time (in seconds).

5.2 Baseline

We implement Full FT of GPT-2 on each task using pretrained GPT-2 weights as the primary baseline. This baseline allows us to evaluate whether LoRA-based fine-tuning methods can achieve comparable task performance while reducing training time.

5.3 Experimental Details

We fine-tune GPT-2 Small and GPT-2 Medium on paraphrase detection and sonnet generation. For each task, we evaluate Full FT, LoRA, rsLoRA, and AdaLoRA. Experiments are conducted using both the default datasets and the augmented datasets described earlier.

We perform grid search over key hyperparameters. The learning rate is selected from $\{1, 5, 8\}e-5$, $\{1, 5, 7\}e-4$, and $\{1, 5, 7\}e-3$. Because the two tasks differ significantly in dataset size, we adjust training epochs and batch sizes accordingly. For paraphrase detection, $\text{epochs} \in \{5, 16\}$ with batch size = 64, while for sonnet generation, $\text{epochs} \in \{20, 40, 55\}$ with batch size = 4. The hyperparameter search ranges for each fine-tuning approach are summarized in Table 6 in the Appendix. The best-performing configuration for each fine-tuning approach is listed in Table 7 and Table 8 in the Appendix.

Paraphrase detection experiments are trained using $8 \times \text{H100}$ GPUs with distributed training for efficient scaling. Sonnet generation experiments are conducted on a single H100 GPU due to the smaller dataset size. All experiments use the cosine learning rate scheduler with warm-up. The learning rate increases linearly during the first 5% of total training steps before decaying following a cosine curve for the remainder of training, which helps stabilize early training and smoothly reduces the learning rate to avoid overshooting. Gradient clipping with a max norm of 1.0 is also applied to prevent gradient explosion.

6 Results and Discussion

6.1 Paraphrase Detection

Based on Table 1, Full FT achieves the strongest overall performance on this task, obtaining the highest mean accuracy, competitive training time, and the fastest inference. All LoRA-based methods achieve comparable accuracy with less than 0.01 average drop. Among them, standard LoRA and rsLoRA provide the best trade-offs, achieving similar training time with competitive accuracy. In particular, rsLoRA achieves the highest best accuracy for GPT-2 Small, slightly surpassing Full FT.

However, the expected improvement in wall-clock training efficiency is not consistently observed. AdaLoRA and DoRA increase training time, which is likely due to their additional computation during

Table 1: Paraphrase detection dev-set results

Model	Method	Mean Accuracy ¹	Best Accuracy ²	GPU Hrs ¹	Infer Sec ¹
GPT-2	Full FT	0.8955	0.9001	1.97	9.7
GPT-2	LoRA	0.8837	0.8986	1.72	10.4
GPT-2	rsLoRA	0.8878	0.9002	1.94	11.4
GPT-2	AdaLoRA	0.8863	0.8911	2.25 ↑	12.1
GPT-2	DoRA	0.8875	0.8911	2.25 ↑	11.3
GPT-2 Medium	Full FT	0.9053	0.9107	2.90	21.1
GPT-2 Medium	LoRA	0.8983	0.9092	2.48	22.7
GPT-2 Medium	rsLoRA	0.8965	0.9095	2.98 ↑	25.5
GPT-2 Medium	AdaLoRA	0.9023	0.9048	3.56 ↑	26.5
GPT-2 Medium	DoRA	0.9001	0.9030	3.82 ↑	23.6

¹ Mean accuracy, training time (GPU Hrs), and inference time (Infer Sec) are averaged across all runs within each experimental setting. ² Best accuracy corresponds to the best-performing configuration reported in Table 7.

training. AdaLoRA periodically evaluates parameter importance and dynamically reallocates rank through pruning and redistribution, which introduces extra optimization overhead. Similarly, DoRA decomposes weight updates into magnitude and directional components, increasing the number of operations during each forward and backward pass. Notably, AdaLoRA also exhibits instability in our experiments: Under certain configurations, especially with large initial ranks and aggressive scaling, the model occasionally collapses during training. The dynamic rank pruning makes optimization harder and more sensitive to hyperparameter choices.

Additionally, all LoRA-based methods show slightly slower inference. After examining our implementation, this may be because the LoRA update path is computed separately from the base weights during inference rather than being merged into the base linear layer. This introduces additional matrix operations at runtime, and this overhead can be eliminated by merging the LoRA weights into the base model after training.

More broadly, LoRA-based approaches match Full FT performance on both GPT-2 Small and GPT-2 Medium, indicating that they can effectively preserve model performance across scales. However, applying LoRA methods to GPT-2 Medium leads to a larger efficiency cost relative to Full FT. While LoRA reduces the number of trainable parameters, the total computation required for forward and backward passes through the frozen base model remains largely unchanged. As model size increases, the cost of these base computations dominates the training time, reducing the relative benefit of parameter-efficient updates in terms of wall-clock efficiency.

6.2 Sonnet generation

Table 2: Sonnet generation dev-set results

Model	Method	Mean chrF ¹	Best chrF ²	GPU Hrs ¹	Infer Sec ¹
GPT-2	Full FT	44.2236	46.8388	0.18	9.6
GPT-2	LoRA	42.6331	45.4034	0.18	11.7
GPT-2	rsLoRA	43.9413	46.6770	0.21 ↑	12.7
GPT-2	AdaLoRA	41.9058	42.4160	0.27 ↑	13.0
GPT-2	DoRA	42.8238	44.3154	0.26 ↑	15.2
GPT-2 Medium	Full FT	44.3919	47.0809	0.28	13.0
GPT-2 Medium	LoRA	43.3734	46.7045	0.26	17.5
GPT-2 Medium	rsLoRA	44.4281	47.1849	0.28	17.8
GPT-2 Medium	AdaLoRA	42.3463	44.1076	0.41 ↑	21.9
GPT-2 Medium	DoRA	44.8469	47.8265	0.41 ↑	27.8

¹ Mean chrF, training time (GPU Hrs), and inference time (Infer Sec) are averaged across all runs within each experimental setting. ² Best chrF corresponds to the best-performing configuration reported in Table 8.

Compared with paraphrase detection, Table 2 shows larger variation in chrF scores across fine-tuning approaches. While not always dominant, Full FT remains competitive when considering both performance and efficiency.

For this task, LoRA-based methods generally perform worse than Full FT, with mean chrF scores around 2 points lower, and this performance gap is larger than that observed for paraphrase detection. This may be due to the much smaller training dataset for sonnet generation and the inherently more complex nature of stylistic text generation compared to classification tasks. When data is limited, restricting parameter updates through low-rank matrices may reduce the model’s capacity to capture fine-grained stylistic patterns required for poetry generation.

In terms of efficiency, the relative trends are similar to those observed in paraphrase detection. Standard LoRA achieves training time comparable to Full FT, while more complex variants such as AdaLoRA and DoRA incur additional overhead and lead to longer training times. Inference time is also consistently higher for LoRA-based methods due to the separate computation of the LoRA update path in our implementation.

Across model scales, GPT-2 Medium generally achieves higher chrF scores than GPT-2 Small under LoRA-based adaptation. In particular, DoRA achieves the best performance among all methods on GPT-2 Medium, with a best chrF of 47.83, surpassing Full FT. This suggests that the larger model capacity helps capture stylistic structure in poetry generation when combined with the weight-decomposition strategy of DoRA. However, this improvement comes at the cost of increased training and inference time compared to Full FT.

6.3 Adapter Placement Across Modules

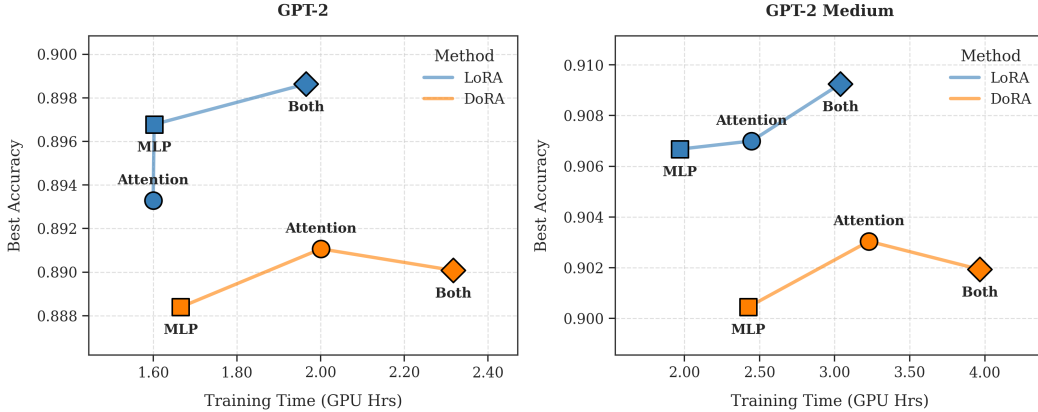


Figure 1: Module-level efficiency-performance trade-offs on paraphrase detection.

For LoRA and DoRA, whose configurations include targeted modules, we further analyze how adapter placement affects the trade-off between model performance and efficiency.

As shown in Figure 1 and Figure 2, applying adapters to all linear layers generally yields the best performance for both LoRA and DoRA across tasks. However, this configuration substantially increases training time. In contrast, restricting adapters to either attention or MLP modules achieves comparable performance with noticeably better efficiency.

More specifically, MLP-only adapters provide a better performance–efficiency trade-off for LoRA across both tasks, while attention-only adapters are more effective for DoRA. This difference may arise from how the two methods parameterize weight updates. LoRA directly learns low-rank updates to the target linear layers, and adapting the MLP layers may be especially effective. These layers contain many of the model’s parameters and govern feature transformation, allowing substantial task-specific adaptation with fewer modified modules. In contrast, DoRA decomposes weight updates into magnitude and directional components, which appear to interact more effectively with the attention projections responsible for token-to-token interactions, making attention layers a more effective adaptation target for this method.

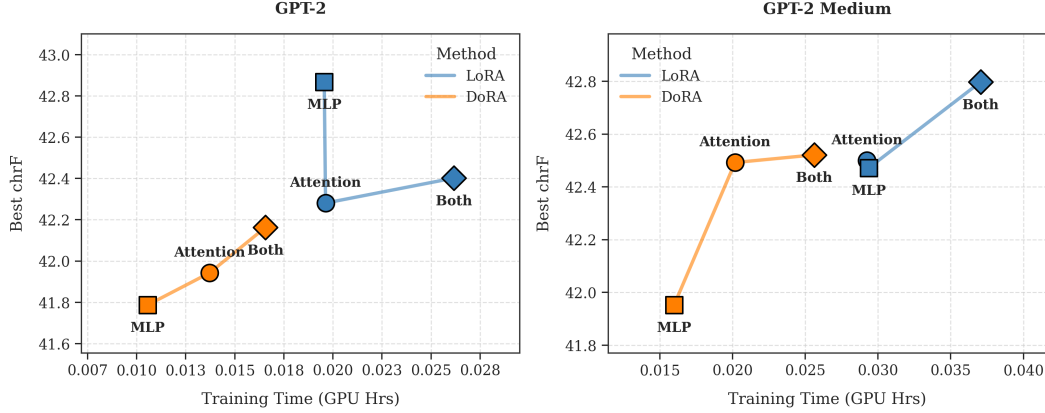


Figure 2: Module-level efficiency-performance trade-offs on sonnet generation.

6.4 Adapter Rank and Dataset Size Interaction

Table 3: Paraphrase detection test results

Accuracy	GPT-2	Method	Dataset
0.897	medium	rsLoRA	Augmented
0.877	medium	Full FT	Augmented
0.868	medium	Full FT	Default

Table 4: Sonnet Generation test results

chrF	GPT-2	Method	Dataset
44.99	medium	rsLoRA	Augmented
43.362	medium	Full FT	Augmented
43.364	medium	DoRA	Augmented

Data augmentation improves performance for both paraphrase detection and sonnet generation. For paraphrase detection, although Table 7 shows that the best-performing configurations on the development set always prefer the default dataset, Table 3 shows that training Full FT with the augmented dataset improves test accuracy from 0.868 to 0.877. This discrepancy between development and test performance may be due to differences in label distribution, as the augmented dataset is strictly balanced while the original dataset is not. For sonnet generation, all best-performing runs adopt the augmented dataset, further validating the effectiveness of data augmentation in low-resource settings.

Since Table 3 and Table 4 both show that rsLoRA with GPT-2 Medium achieves the best overall performance, we use this setting to further investigate how LoRA adapter rank interacts with dataset size in low-resource scenarios of sonnet generation.

With the default dataset, Figure 3 shows that performance varies only slightly across ranks, peaking around $r = 16$. Reducing the rank to $r = 8$ causes only a small drop in chrF, suggesting that the adapter rank can be reduced by roughly half without substantial performance loss. However, performance deteriorates when the rank becomes too small ($r \leq 4$), indicating insufficient adaptation capacity. Increasing the rank beyond $r = 16$ also slightly degrades performance, likely due to overfitting under limited data.

In contrast, with the augmented dataset, the model benefits from larger adapter ranks. The chrF score steadily improves as the rank increases beyond $r = 16$, indicating that additional adaptation capacity can be effectively utilized when more training data is available. These results suggest that the optimal adapter rank is closely tied to dataset scale: smaller datasets favor lower-rank adapters to avoid overfitting, whereas larger datasets support higher ranks that enable richer task-specific adaptation.

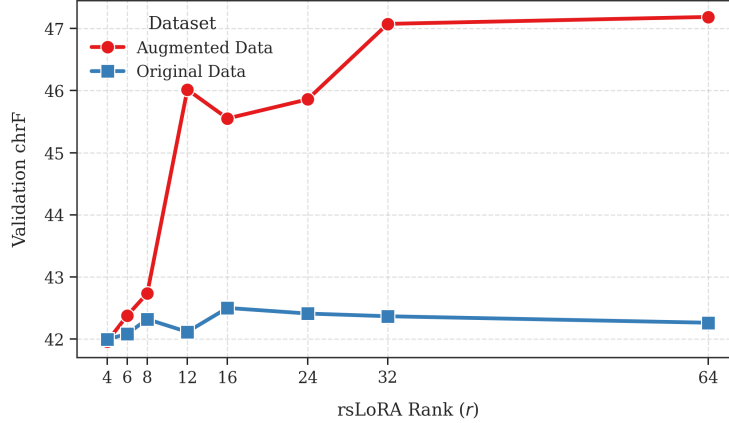


Figure 3: rsLoRA rank sweep for sonnet generation with GPT-2 Medium

7 Conclusion

We present a systematic empirical study of LoRA-family PEFT methods by fine-tuning GPT-2 Small and GPT-2 Medium on two tasks: paraphrase detection and sonnet generation. We compare several representative approaches, including LoRA, rsLoRA, AdaLoRA, and DoRA, and analyze their performance–efficiency trade-offs across tasks, model scales, adapter configurations, and training data scales.

Our results show that LoRA-based approaches generally match the performance of full fine-tuning, but their advantages in wall-clock efficiency are less consistent than commonly assumed. Among the evaluated methods, standard LoRA and rsLoRA provide the most favorable performance–efficiency trade-offs, while AdaLoRA and DoRA often introduce additional training overhead. Across both tasks, restricting adapters to specific modules can achieve performance close to full adapter placement while improving efficiency, with MLP adapters performing best for LoRA and attention adapters more effective for DoRA. We also find that dataset scale strongly interacts with adapter capacity. For small datasets, relatively low adapter ranks are sufficient and can be reduced substantially before performance deteriorates, while larger datasets benefit from higher ranks that enable richer task adaptation.

Our study has several limitations. First, we focus only on GPT-2 Small and Medium, so our conclusions may not directly transfer to larger modern LLMs. Second, our experiments are restricted to two tasks—one classification task and one generation task—which limits the breadth of generalization. Third, although we measure GPU hours and downstream performance, we do not isolate memory savings or inference-time trade-offs in detail. Future work could extend this analysis to larger transformer models, broader NLP tasks, and additional parameter-efficient fine-tuning strategies for a more comprehensive evaluation.

Team Contributions

Christine Li: Built the base GPT-2 model, ran sentiment classification, improved the LoRA pipeline, implemented AdaLoRA and DoRA, designed experiments, and completed the project write-ups.

Zetian Li: Built the base GPT-2 model and pipeline for extension tasks, preprocessed extra datasets, ran all experiments with gpu optimization and distributed training, organized experiments results, and prepared submission for leaderboard.

Jason Yan: Implemented FlashAttention, LoRA, layer-wise rank allocation, and rsLoRA, and conducted quantitative and qualitative analysis with graph visualizations.

References

- [1] 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*, 2022.
- [2] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023.
- [3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 2019.
- [4] Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora, 2023.
- [5] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation, 2024.
- [6] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023.
- [7] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.
- [8] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.
- [9] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.
- [10] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.
- [11] David Samuel, Lucas Charpentier, and Sondre Wold. En wiki paraphrased. <https://huggingface.co/datasets/lgt/en-wiki-paraphrased>, 2024.
- [12] Warris Gill and Redis. Llm generated paraphrases. <https://huggingface.co/datasets/redis/llm-paraphrases>, 2025.
- [13] Lambent. Diffused shakespeare sonnets. https://huggingface.co/datasets/Lambent/shakespeare_sonnets_diffused, 2024.

A Appendix

A.1 Augmented Datasets

Table 5: Breakdown of the augmented training data

Dataset / Source	Samples Added
<i>Paraphrase Detection</i>	
Default	283,003
Positives (to balance original dataset)	73,845
Positives (to balance augmented dataset)	160,000
Random negatives	10,000
Embedding-similar negatives	50,000
Topic-similar negatives	30,000
Near-miss semantic edits	60,000
<i>Sonnet Generation</i>	
Default	143
Additional sonnets	920

A.2 Hyperparameters Grid Search

Most hyperparameters share the same search space across tasks unless otherwise specified.

Table 6: Hyperparameter search space

Hyperparameter	Search Range
Dropout	0.1
LoRA / rsLoRA / DoRA targeted modules	MLP-only, attention-only, all layers
LoRA / rsLoRA / DoRA rank r	{8, 16, 32}
LoRA / DoRA α	{16, 32, 64}
rsLoRA α	{5.6, 11.2, 16, 22.4, 32, 64}
AdaLoRA α	{8, 16}
AdaLoRA initial rank r_{init}	{8, 16, 32}
AdaLoRA target average rank r_{target}	{4, 8, 16}
AdaLoRA warm-up ratio t_{init}	Task dependent ¹
AdaLoRA pruning stop ratio t_{final}	0.1
AdaLoRA pruning interval ΔT	{10, 50}
AdaLoRA exponential moving average factor β_1	0.85
AdaLoRA exponential moving average factor β_2	0.85
AdaLoRA orthogonal regularization coefficient $orth_{reg}$	0.1

¹ $t_{\text{init}} \in \{0.1, 0.2\}$ for paraphrase detection and $t_{\text{init}} \in \{0.2, 0.3\}$ for sonnet generation.

Because preliminary experiments showed promising performance for rsLoRA and we aim to study the relationship between adapter rank and dataset size in low-resource settings, we expand the rsLoRA rank search for sonnet generation to $r \in \{4, 6, 8, 12, 24, 32, 64\}$.

A.3 Best-performing Configurations

Table 7: Best-performing configurations for paraphrase detection

Hyperparameter	Full FT	LoRA	rsLoRA	DoRA	AdaLoRA
Training Data	Default Training Dataset (S/M)				
Learning rate	1e-4 (S) 8e-5 (M)	1e-3 (S/M)	5e-4 (S) 7e-4 (M)	1e-3 (S/M)	1e-3 (S/M)
Epoch			16 (S/M)		
Batch size			64 (S/M)		
Dropout			0.1 (S/M)		
Target module	-	All linear	All linear	Attention	-
Rank r	-	32 (S), 16 (M)	32 (S/M)	32 (S/M)	-
α	-	32	32 (S), 11.2 (M)	64	16 (S), 8 (M)
Initial rank r_{init}			-		16 (S/M)
Target rank r_{target}			-		8 (S/M)
t_{init}			-		0.2
t_{final}			-		0.1 (S/M)
Δt			-		10 (S), 50 (M)
β_1			-		0.85
β_2			-		0.85
orth_reg			-		0.1

Table 8: Best-performing configurations for sonnet generation

Hyperparameter	Full FT	LoRA	rsLoRA	DoRA	AdaLoRA
Training Data	Augmented Training Dataset (S/M)				
Learning rate	1e-3 (S/M)	1e-3 (S) 7e-4 (M)	5e-4 (S) 7e-4 (M)	7e-4 (S) 1e-3 (M)	1e-3 (S/M)
Epoch	40 (S/M)	40 (S/M)	40 (S/M)	40 (S), 55 (M)	20 (S), 40 (M)
Batch size				4	
Dropout				0.1	
Target module	-		All linear (S/M)		-
Rank r	-	32 (S/M)	64 (M)	32 (S/M)	-
α	-	64 (S/M)	64 (M)	64 (S/M)	8 (S/M)
Initial rank r_{init}			-		8 (S), 16 (M)
Target rank r_{target}			-		4 (S), 8 (M)
t_{init}			-		0.2
t_{final}			-		0.1 (S/M)
Δt			-		10 (S)
β_1			-		0.85
β_2			-		0.85
orth_reg			-		0.1