

AdamW The Last LLM-Bender: The Legend of LoRA

Stanford CS224N Default Project

Ari Barbella Blaha
Department of Computer Science
Stanford University
aribarb@stanford.edu

Kieran Barrett
Department of Computer Science
Stanford University
kjavi@stanford.edu

Abstract

While large-scale pretrained language models like GPT-2 exhibit impressive zero-shot capabilities, their performance on specific, complex downstream tasks often lags behind supervised baselines. Addressing this limitation is challenging. Simply scaling up model capacity is computationally prohibitive and traditional full fine-tuning requires updating millions to billions of parameters, which is inefficient and prone to overfitting on smaller datasets.

In this work, we implement the GPT-2 architecture to evaluate parameter-efficient fine-tuning using Low-Rank Adaptation (LoRA) across three NLP tasks: sentiment classification, paraphrase detection, and sonnet generation. We investigate the impact of LoRA adapter placement within different transformer components and compare the efficacy of three optimization strategies (AdamW, Lion, and NAdam) under both full and parameter-efficient fine-tuning regimes.

Our experiments yield three main findings. First, applying LoRA to both attention projections and feed-forward networks (QKV + MLP) matches or exceeds the performance of full fine-tuning while updating less than 1% of the model parameters. Second, we observe that restricting trainable parameters via LoRA may act as an implicit regularizer, yielding better generalization and higher accuracy on held-out test sets (e.g., in paraphrase detection). Finally, we reveal task-dependent optimization dynamics: AdamW provides the greatest stability and performance for discriminative classification tasks, whereas NAdam achieves superior convergence and quality for generative sequence modeling.

1 Key Information to include

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

Team member	Late days remaining
Ari Barbella Blaha	1
Kieran Barrett	1
Total pooled	2
Team size	2
Late days applied for the report	1

2 Introduction

The success of large-scale pretrained language models has transformed natural language processing by enabling models to learn general linguistic representations from large corpora and adapt them to a variety of downstream tasks. Architectures such as GPT-2 demonstrate that scaling model capacity and training data significantly improves language understanding and generation capabilities [1]. In this project, we implement the GPT-2 architecture and evaluate its effectiveness across multiple NLP tasks.

Our implementation includes core components of the GPT-2 transformer architecture, including masked multi-head self-attention and the AdamW optimizer. We then fine-tune the model on three downstream tasks: sentiment classification (SST and CFIMDB), paraphrase detection (Quora Question Pairs), and sonnet generation using Shakespearean sonnets. These tasks allow us to evaluate the model across both discriminative and generative settings.

To improve adaptation efficiency, we extend the baseline with parameter-efficient fine-tuning using Low-Rank Adaptation (LoRA) [2]. We further investigate how optimization strategies affect both full fine-tuning and parameter-efficient adaptation by comparing AdamW, Lion, and NAdam [3]. Our implementation allows LoRA adapters to be inserted into different transformer components, including attention projections and feed-forward layers. Across tasks, LoRA achieves performance comparable to full fine-tuning while updating fewer than 1% of model parameters, and in some cases improves generalization on held-out test data.

3 Related Work

To contextualize GPT-2’s architecture and our fine-tuning methodology, we must trace sequence modeling’s shift from complex, bottleneck-prone recurrent networks to the highly parallelizable Transformer architecture [4]. Building on this scalable, self-attentive foundation, researchers found that training massive language models on millions of webpages naturally enables unsupervised multitask learning. As a result, these models can perform downstream tasks like question answering and translation without explicit supervision, with their zero-shot capabilities improving log-linearly as parameter size increases [1].

Despite this progress, massive unsupervised language models face limitations that motivate our current research. Their zero-shot performance on complex tasks like abstractive summarization often remains rudimentary and less practical than supervised baselines. Furthermore, billion-parameter models still underfit vast datasets like WebText, indicating that simply scaling capacity cannot resolve all inefficiencies. Building directly upon these foundational baselines, our project aims to improve their broad zero-shot generalization on specific downstream tasks [1].

4 Approach

4.1 Base architecture

Our base model is GPT-2 [1], implemented using pretrained weights and tokenization from the Hugging Face `transformers` library [5]. The embedding layer combines learned word and position representations: $\mathbf{H}_0 = \text{Dropout}(\mathbf{E}_w + \mathbf{E}_p)$. The sequence passes through L pre-norm residual layers, each comprising causal multi-head self-attention and a two-layer GELU feed-forward network. The final representation is $\mathbf{H}_{\text{out}} = \text{LayerNorm}(\mathbf{H}_L)$, from which we extract the last non-padding token for downstream classification.

Causal self-attention uses a scaled dot-product: $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right) \mathbf{V}$, where $\mathbf{M}$ combines a lower-triangular causal mask and an additive padding mask (mapping padding tokens to -10000). Multi-head reshaping is handled via `einops`. For language modeling, we employ weight tying, computing next-token logits as $\mathbf{h}\mathbf{E}_w^\top$. Optimization is performed using AdamW [6] with decoupled weight decay.

Our implementation follows the standard GPT-2 architecture [1] and AdamW optimization procedure [6]. The codebase is built on the provided final project starter code and uses the Hugging Face `transformers` library. Within this framework, we implemented the core transformer compo-

nents—including causal self-attention, GPT-2 blocks, and the forward pass—as well as the AdamW optimizer and the task-specific training and generation scripts for sentiment classification, paraphrase detection, and sonnet generation. Dataset loading, evaluation loops, and pretrained weights rely on the provided project infrastructure and the Hugging Face library.

4.2 Extensions to base architecture

Beyond implementing a full forward pass for sonnet generation, we extend this downstream task through enhancing the model’s training regime by adding weight decay (0.01), a linear learning-rate schedule with warmup (10%), gradient clipping (1.0), label smoothing (0.05), and early stopping (patience of 4 epochs) to save the best checkpoint.

For sonnet generation, the baseline relied solely on temperature, top- p sampling, and a fixed max length, stopping at the first EOS. Our extended approach introduces top- k filtering, a post-prompt repetition penalty, and an increased max length (e.g., 256). To strictly enforce the 14-line sonnet form given a 3-line prompt, we delay EOS until 11 newlines are generated, apply newline logit biases, and constrain line lengths between 8 and 14 tokens. Furthermore, we suppress nuisance tokens, discourage leading spaces on the first twelve lines, and encourage a double-space indent for the final couplet. All added hyperparameter controls and formatting constraints are fully exposed as command-line arguments.

To improve adaptation efficiency, we extend GPT-2 with Low-Rank Adaptation (LoRA) [2]. LoRA introduces trainable low-rank updates to selected weight matrices while freezing the pretrained model parameters. In our implementation, LoRA adapters can be applied to (1) the query and value projections (QV), (2) the query, key, and value projections (QKV), (3) the feed-forward network layers (MLP), or (4) both attention and feed-forward layers (QKV + MLP). This flexible design allows us to experimentally study how adapter placement affects downstream performance.

For a pretrained weight matrix $W_0 \in \mathbb{R}^{d \times d}$, LoRA parameterizes the update as

$$W = W_0 + BA,$$

where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times d}$, and $r \ll d$. During fine-tuning, only the low-rank matrices A and B and task-specific heads are updated, while the original weight W_0 remains fixed. This substantially reduces the number of trainable parameters while preserving model capacity.

LoRA updates are applied conditionally during the forward pass. When LoRA is disabled, the model computes projections using the original pretrained weight matrices. When LoRA is enabled, the low-rank update BA is added to the relevant projection matrices, allowing the adapters to modify the transformer computations while leaving the pretrained parameters unchanged.

Additionally, we minimize a loss $\mathcal{L}(\theta)$ over parameters θ using first-order updates $\theta_{t+1} = \theta_t - \alpha_t u_t$, where α_t is the learning rate and u_t is the update vector. This report assesses whether Lion and NAdam improve convergence speed and final performance over our baseline, AdamW, across sentiment analysis, paraphrase detection, and sonnet generation tasks. AdamW computes u_t using bias-corrected first ($\hat{m}_t$) and second ($\hat{v}_t$) gradient moments, applying decoupled weight decay λ after the gradient update:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} - \alpha \lambda \theta_{t-1}$$

To explore potential efficiency and performance gains, we evaluate two alternative optimizers. Lion reduces memory overhead by omitting the second moment entirely. It utilizes the sign of an exponentially moving average blending past momentum and the current gradient (c_t) to ensure uniform step magnitudes across coordinates:

$$\theta_t = \theta_{t-1} - \alpha (\text{sign}(c_t) + \lambda \theta_{t-1})$$

Because of this uniform scaling, Lion typically requires a substantially smaller learning rate than AdamW. Conversely, NAdam integrates Nesterov acceleration into the Adam framework. It applies a time-decaying momentum schedule (μ_t) to compute a lookahead first-moment estimate:

$$\hat{m}_t = \frac{\mu_{t+1} m_t}{1 - \prod_{i=1}^{t+1} \mu_i} + \frac{(1 - \mu_t) g_t}{1 - \prod_{i=1}^t \mu_i}$$

This Nesterov-corrected $\hat{m}_t$ is then used in place of the standard first moment in the AdamW update equation, which can accelerate convergence in certain topographies.

5 Experiments

5.1 Data

We utilize four datasets to evaluate our models across sentiment classification and text generation tasks. The first dataset is the Stanford Sentiment Treebank (SST) [7], which consists of 11,855 single sentences extracted from movie reviews. The dataset was parsed by the Stanford parser, resulting in a total of 215,154 unique phrases. Each phrase was annotated by three human judges into one of five sentiment classes: negative, somewhat negative, neutral, somewhat positive, or positive. The associated task is multi-class sentiment classification. Specifically, the exact input is a text string (a phrase or single sentence) passed through the model, and the output is a 5-way sentiment label predicted using GPT-2’s last token embedding. The SST dataset is partitioned into 8,544 training examples, 1,101 development examples, and 2,210 test examples.

The second dataset is the CFIMDB dataset [8], which comprises 2,434 highly polar movie reviews. Unlike the inputs of the SST dataset, many of the reviews in CFIMDB span multiple sentences. The associated task is binary sentiment classification. The model takes a full, multi-sentence movie review text as its input and utilizes GPT-2’s last token embeddings to output a binary classification label of either positive or negative. The CFIMDB dataset is split into 1,701 training examples, 245 development examples, and 488 test examples.

Third, we employ the Sonnet Dataset, which contains 154 sonnets authored by William Shakespeare. Each sonnet serves as a structured text input consisting of 14 lines that strictly adhere to an ABAB CDCD EFEF GG rhyme scheme, where lines assigned the same letter ID rhyme with one another. The input consists of tokenized sequences of poetic text, and the target output is the generated text that conforms to the established rhyme scheme and sonnet structure. This dataset is divided into a training set of 143 sonnets and a test set of 12 sonnets [9].

The fourth dataset is the Quora Question Pairs dataset [10], which contains pairs of user-submitted questions from the Quora platform labeled according to whether they represent the same underlying query. The task is paraphrase detection, where the model determines whether two questions are semantically equivalent. Each input consists of two question strings, which we format into a cloze-style prompt and pass through the GPT-2 model. The output is a binary label indicating whether the questions are duplicates (paraphrases) or not. The dataset contains over 400,000 question pairs with human-annotated labels, though the labels may contain some noise due to the large-scale annotation process. For our experiments, we use the provided train, development, and test splits to train and evaluate the model.

5.2 Evaluation method

Our evaluation methodology assesses model performance across three distinct tasks: paraphrase detection, sonnet generation, and sentiment classification. Across these tasks, we rely heavily on automatic and quantitative metrics to provide standardized benchmarks for model performance, with one qualitative data-cleaning choice available for text generation.

For the Quora paraphrase detection task, the model evaluates whether two questions are paraphrases of one another (a binary classification) based on cloze-style inputs. We utilize two primary metrics: accuracy, which measures overall correctness, and Macro F1, which evaluates discriminative performance while treating positive and negative classes equally to ensure the minority paraphrase class is not overlooked. Dev accuracy is used to guide early stopping and checkpoint selection. During testing, `model_test_paraphrase` generates predictions and writes them alongside sentence IDs to required CSV submission files using explicit token IDs (e.g., 8505 for "yes" and 3919 for "no") rather than binary 0/1 values.

In the sonnet generation task, the model is trained on Shakespearean sonnets and autoregressively generates text based on a prompt. The outputs are compared to held-out gold sonnets using the CHRF (Character n-gram F-score) metric [11]. CHRF is an automatic, reproducible scalar that measures surface-form similarity by evaluating character n-gram overlaps. It serves as a proxy for how closely the generated character sequences, spelling, and punctuation match the reference text. Evaluation is executed via the `test_sonnet` function, which parses generated and true sonnets by splitting numbering patterns, pairs them by index, trims them to a minimum length, and computes a single corpus-level CHRF score, which is part of the starter code.

Finally, for sentiment classification across the SST and CFIMDB datasets, we use the same GPT-2-based classifier to predict sentiment labels. The model outputs logits over the labels, and the predicted class is derived via the `argmax`. As with paraphrase detection, we measure Accuracy for overall correctness and Macro F1 to ensure balanced performance across all sentiment classes.

To evaluate the effectiveness of Low-Rank Adaptation (LoRA) for parameter-efficient fine-tuning, we first compare several LoRA configurations against full-model fine-tuning. Our primary evaluation metric is development set accuracy for the Sentiment Classification task on both SST and CFIMDB. We also measure the percentage of trainable parameters used during training to quantify the parameter efficiency of each method. We evaluate the following setups:

- **Full fine-tuning:** All model parameters are updated during training.
- **LoRA (QV):** LoRA adapters are applied to the query and value projection matrices in the attention layers.
- **LoRA (QKV):** LoRA adapters are applied to the query, key, and value projection matrices.
- **LoRA (MLP):** LoRA adapters are applied to the feed-forward network layers.
- **LoRA (QKV + MLP):** LoRA adapters are applied to both the attention projections and the feed-forward layers.

This experiment allows us to analyze the trade-off between model performance and parameter efficiency, and to understand which components of the transformer architecture benefit most from LoRA adaptation.

After identifying the best-performing LoRA configuration from the previous experiment, we train this configuration using three different optimizers (AdamW, NAdam, and Lion). For comparison, we also train the full fine-tuning model using the same set of optimizers as a control. This setup allows us to examine how the choice of optimizer influences convergence behavior and final task performance when only a small subset of parameters is updated.

For each optimizer setting, all other training hyperparameters (batch size, learning rate schedule, number of epochs, and dataset splits) are kept constant to ensure a controlled comparison. Model performance is evaluated using the same development set metrics described above. This experiment helps determine whether LoRA-based adaptation benefits from particular optimization dynamics compared to full fine-tuning under identical training conditions.

5.3 Experimental Details

All experiments use a pretrained GPT-2 model implemented with the Hugging Face `transformers` [5]. For sentiment classification, input text is tokenized using the GPT-2 tokenizer and passed through the transformer. The representation of the final non-padding token is used as input to a classification layer that predicts binary sentiment.

Across all sentiment classification experiments, models are trained for 10 epochs with a batch size of 64 and hidden dropout probability of 0.1. For full fine-tuning, all GPT-2 parameters are updated using a learning rate of 1×10^{-5} . For LoRA experiments, the pretrained GPT-2 weights are frozen and only the inserted low-rank adapter parameters and the final classification layer are trained, using a learning rate of 5×10^{-4} .

We evaluate several LoRA configurations that apply adapters to different components of the transformer architecture (QV, QKV, MLP, and QKV+MLP). All other training settings are kept identical across configurations to ensure fair comparison. Model performance is evaluated on the SST and CFIMDB development sets after training, and we report development accuracy. The percentage of trainable parameters is computed relative to the total number of parameters in the base GPT-2 model.

For the sonnet generation experiments, we fine-tune a pretrained `gpt2-medium` model on the Sonnet Dataset of Shakespearean sonnets and evaluate generation quality on a held-out development set. Training is performed with a batch size of 8 on GPU. We compare both full-model fine-tuning and LoRA-based parameter-efficient fine-tuning, across three optimizers: AdamW, Lion, and NAdam. For the LoRA configurations, adapters are applied either to the attention query/value projections (QV) or to the attention projections and feed-forward layers (QKV + MLP), since those were the highest performing for the classifier task, while the pretrained GPT-2 weights remain frozen.

All other training settings are kept constant across runs to ensure a controlled comparison. During generation, we use nucleus sampling with $p = 0.82$, a temperature of 0.55, and a repetition penalty of 1.30 to improve diversity while discouraging repetitive outputs.

For the paraphrase detection experiments, we fine-tune the GPT-2 classifier on the Quora Question Pairs dataset, where the model predicts whether two questions are paraphrases of one another. Inputs are formatted using the cloze-style prompt structure described in the task specification and tokenized with the GPT-2 tokenizer. The final hidden representation of the sequence is passed to a classification head that predicts a binary label.

Training is performed with a batch size of 64 and a hidden dropout probability of 0.1 for 10 epochs. We compare full-model fine-tuning with LoRA-based parameter-efficient fine-tuning using the LoRA configuration that performed best in the sentiment classification experiments (QKV + MLP). For full fine-tuning, all GPT-2 parameters are updated with a learning rate of 1×10^{-5} , while for LoRA experiments the pretrained model weights are frozen and only the LoRA adapter weights and task-specific classification head are trained with a learning rate of 5×10^{-4} . All other training settings are kept constant across runs to ensure a controlled comparison. Model performance is evaluated on the development set using accuracy and Macro F1 as described in the evaluation section.

5.4 Results

Table 1 reports development set accuracy for sentiment classification across different LoRA configurations. Overall, LoRA achieves comparable or better performance than full-model fine-tuning while updating only a small fraction of the parameters. In particular, the LoRA (QKV + MLP) configuration achieves the highest SST development accuracy while training under 1% of the model parameters.

Model	SST Dev Acc.	CFIMDB Dev Acc.	% Params Trained
Full Fine-tuning	0.509	0.980	100%
LoRA (QV)	0.510	0.984	0.2367%
LoRA (QKV)	0.510	0.984	0.3535%
LoRA (MLP)	0.518	0.971	0.5872%
LoRA (QKV + MLP)	0.527	0.971	0.9377%

Table 1: Sentiment classification results for different LoRA configurations using the AdamW optimizer. LoRA achieves competitive or superior performance while updating fewer than 1% of model parameters.

We next compare different optimizers while holding the model configuration fixed. Table 2 shows results for both full-model fine-tuning and our best-performing LoRA configuration (QKV + MLP). AdamW consistently achieves the best performance across both datasets, while Lion performs worse in this small-data fine-tuning setting.

Model	Optimizer	SST Dev Acc.	CFIMDB Dev Acc.
Full Fine-tuning	AdamW	0.509	0.980
Full Fine-tuning	Nadam	0.498	0.980
Full Fine-tuning	Lion	0.470	0.951
LoRA (QKV + MLP)	AdamW	0.527	0.971
LoRA (QKV + MLP)	Nadam	0.523	0.967
LoRA (QKV + MLP)	Lion	0.468	0.963

Table 2: Optimizer comparison for sentiment classification. AdamW provides the most stable and highest-performing optimization across both full fine-tuning and LoRA settings.

We additionally compare the performance of different model-optimizer configurations at the task of sonnet generation in Table 3. With the exception of the baseline model, all models were run with the extensions to the sonnet generation base model that are described in "Extensions to base architecture". We note that the NAdam optimizer performs the best across full fine-tuning and all LoRA configurations, with full fine-tuning using the NAdam optimizer performing the best overall. We also note that LoRA (QV) was the worst performing model, and AdamW was the worst optimizer among LoRA configurations.

Model	Optimizer	CHRF Score	% Params Trained
Full Fine-tuning	AdamW	43.013	100%
Full Fine-tuning	Lion	43.761	100%
Full Fine-tuning	NAdam	43.800	100%
LoRA (QKV + MLP)	AdamW	43.007	0.876%
LoRA (QKV + MLP)	Lion	43.418	0.876%
LoRA (QKV + MLP)	NAdam	43.561	0.876%
LoRA (QV)	AdamW	42.177	0.219%
LoRA (QV)	Lion	42.692	0.219%
LoRA (QV)	NAdam	42.457	0.219%

Table 3: Sonnet generation performance measured using CHRF. All models were trained using GPT-2 medium with identical decoding hyperparameters (temperature 0.55, top-p 0.82, repetition penalty 1.30). Full fine-tuning with NAdam achieves the highest score, though LoRA (QKV + MLP) remains competitive while updating significantly fewer parameters.

Based on the development results, we select the full fine-tuning model with the NAdam optimizer as the final model for evaluation on the held-out test set. This model achieves a CHRF score of 43.327 on the test set, compared to 43.035 for Lion and 43.209 for AdamW under full fine-tuning.

We additionally evaluate whether the best-performing LoRA configuration from the sentiment classification experiments transfers effectively to paraphrase detection with the AdamW optimizer. Table 4 compares full fine-tuning against LoRA (QKV + MLP) on the Quora paraphrase task. The full fine-tuning model achieves the highest development accuracy at 0.890, while LoRA (QKV + MLP) achieves a competitive development accuracy of 0.880. Although full fine-tuning performs slightly better, the LoRA model remains close in performance while updating only a small fraction of the model parameters, suggesting that parameter-efficient adaptation can transfer reasonably well to paraphrase detection.

Model	Dev Accuracy	% Params Trained
Full Fine-tuning	0.890	100%
LoRA (QKV + MLP)	0.880	0.9359%

Table 4: Paraphrase detection development accuracy for full fine-tuning and the best-performing LoRA configuration from the sentiment classification experiments with AdamW optimizer. LoRA achieves comparable performance while training under 1% of the model parameters.

On the held-out test set, we observe a different pattern. While full fine-tuning achieved higher development accuracy, the LoRA (QKV + MLP) model achieves a higher test accuracy of 0.839 compared to 0.816 for full fine-tuning. This suggests that the parameter-efficient LoRA model may generalize slightly better on this task despite training significantly fewer parameters.

6 Analysis

Our experiments provide several insights into the behavior of Low-Rank Adaptation (LoRA) across different NLP tasks.

First, the placement of LoRA adapters within the transformer architecture has a noticeable impact on downstream performance. In the sentiment classification experiments, the LoRA (QKV + MLP) configuration consistently performed best among the parameter-efficient variants, achieving the highest SST development accuracy while training under 1% of the model parameters. This suggests that allowing LoRA to modify both attention projections and feed-forward layers provides greater representational flexibility than restricting adaptation to attention projections alone. In contrast, the LoRA (QV) configuration generally performed worse, indicating that adapting only the query and value projections may limit the model’s ability to fully adjust its internal representations.

Second, our optimizer experiments reveal that optimization strategy can meaningfully affect performance in small-scale fine-tuning settings. For sentiment classification, AdamW consistently produced the strongest results for both full fine-tuning and LoRA configurations. This likely reflects AdamW’s stability and regularization properties for transformer training. In contrast, Lion performed noticeably worse across experiments, suggesting that its more aggressive update dynamics may be less suitable

when fine-tuning on relatively small datasets. Interestingly, the optimizer behavior differed slightly for the sonnet generation task. In this setting, NAdam achieved the highest CHRF scores across both full fine-tuning and LoRA models. This may reflect differences between discriminative and generative tasks. Text generation often benefits from optimizers that incorporate momentum-like updates, which may help stabilize training dynamics when modeling long-range sequence structure.

Third, our results highlight the parameter efficiency of LoRA. Across sentiment classification and paraphrase detection, LoRA models were able to achieve performance comparable to full fine-tuning while updating less than 1% of the model parameters. For example, in sentiment classification the best LoRA configuration slightly outperformed full fine-tuning on SST while using only 0.94% of the parameters. This demonstrates that many downstream adaptations can be captured through low-rank updates to pretrained transformer weights.

Finally, the paraphrase detection results suggest that LoRA may sometimes generalize better than full fine-tuning. Although full fine-tuning achieved higher development accuracy, the LoRA model achieved higher accuracy on the held-out test set (0.839 vs. 0.816). One possible explanation is that restricting training to a small set of adapter parameters acts as an implicit form of regularization, reducing overfitting to the development data.

Overall, our experiments demonstrate that LoRA provides a compelling trade-off between performance and efficiency. By carefully selecting where adapters are inserted and using stable optimization strategies, parameter-efficient fine-tuning can match or even surpass full fine-tuning on several tasks while dramatically reducing the number of trainable parameters.

7 Conclusion

In this work, we explored parameter-efficient fine-tuning of GPT-2 using Low-Rank Adaptation (LoRA) across three tasks: sentiment classification, paraphrase detection, and sonnet generation. Our experiments demonstrate that LoRA can achieve performance comparable to full fine-tuning while updating only a small fraction of the model parameters. In particular, the LoRA (QKV + MLP) configuration consistently performed best among the LoRA variants, indicating that adapting both attention projections and feed-forward layers provides greater flexibility than modifying attention projections alone.

We also investigated the role of optimization strategies in both full fine-tuning and LoRA-based adaptation. For classification tasks, AdamW produced the most stable and highest-performing results, while Lion performed noticeably worse in this small-data fine-tuning setting. In contrast, the sonnet generation experiments showed that NAdam produced the strongest results for generative modeling. These findings suggest that optimizer choice may interact differently with discriminative versus generative tasks.

Overall, our results highlight the effectiveness of LoRA as a practical alternative to full fine-tuning for adapting large language models. By carefully selecting adapter placement and optimization strategies, parameter-efficient methods can achieve strong performance across both classification and generation tasks while significantly reducing training cost.

Team contributions

Ari implemented the baseline GPT2 architecture, including the attention and MLP layers. He also implemented the AdamW optimizer and the LoRA extensions. He then ran the classifier and sonnet generation experiments with different LoRA configurations.

Kieran implemented the sentiment classifier along with the baseline for paraphrase detection and sonnet generation and fine tuned both tasks. He also implemented the NAdam and Lion optimizers and wrote a test suite to verify their correctness.

Ari and Kieran worked equally on the final report.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, Luan David, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. In *OpenAI*, 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] Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. In *arXiv*, 2025.
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 2017.
- [5] T Wolf. Transformers: State-of-the-art natural language processing. 2020.
- [6] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *arXiv*, 2019.
- [7] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Manning Christopher, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013.
- [8] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. In *Proceedings of the 11th annual research colloquium of the UK special interest group for computational linguistics*, 2008.
- [9] Cs 224n: Default final project: Build gpt-2. 2026.
- [10] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs. In <https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs>, 2017.
- [11] Maja Popović. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, 2015.