

Task-Driven Fine-Tuning and Efficient Attention for GPT-2

Stanford CS224N Default Project

Susan Li Yuzhou Bian Sixian Du

Department of Electrical Engineering

Stanford University

{shuixli, yuzhoub, dusixian}@stanford.edu

Abstract

Pretrained language models are commonly adapted to downstream NLP tasks through fine-tuning, but designing efficient and task-aware adaptation strategies remains challenging. In this project, we investigate task-driven fine-tuning strategies for GPT-2 across paraphrase detection and sonnet generation. For paraphrase detection, we introduce symmetric data augmentation to enforce sentence-order invariance, while for sonnet generation we adopt a two-stage fine-tuning strategy that first adapts model to a larger Shakespeare corpus before specializing on the sonnet dataset. We further explore parameter-efficient training using LoRA and efficiency improvements through FlashAttention. Experiments show that injecting LoRA across Transformer layers achieves paraphrase performance comparable to full fine-tuning while training far fewer parameters, and symmetric augmentation improves prediction consistency. Two-stage fine-tuning substantially improves sonnet generation quality, and FlashAttention significantly reduces GPU memory usage and enables larger training batches under limited hardware constraints. Our final models achieve 0.892 test accuracy on paraphrase detection and 43.52 test chrF on sonnet generation.

1 Key Information to include

- Mentor: Ali Sartaz Khan
- External Collaborators (if you have any): No
- Sharing project: No
- Late Days (if you are using any): Not using late days.

2 Introduction

Modern NLP systems are rarely trained from scratch. Instead, large pretrained language models such as BERT [1] and GPT-2 [2], are adapted to downstream tasks through task-specific fine-tuning, where task-specific fine-tuning is used to adapt the model to the target problem. As pretrained models grow larger, designing effective and efficient fine-tuning strategies becomes increasingly important.

In this project, we study how different fine-tuning strategies affect both performance and efficiency on two representative tasks: paraphrase detection and Shakespearean sonnet generation. These tasks highlight different challenges of adapting large pretrained models. In paraphrase detection, the task has a symmetric semantic structure: the prediction for (s_1, s_2) should ideally match that for (s_2, s_1) . However, standard fine-tuning does not explicitly enforce this property, which can lead to inconsistent predictions. While sonnet generation faces a data scarcity challenge, as Shakespeare wrote only a limited number of sonnets, making it difficult for large models to learn stylistic patterns.

Another practical challenge arises from the scale of modern language models. While full-parameter fine-tuning can achieve strong performance, it is often computationally expensive and may exceed

memory limits under realistic resource constraints [3]. This motivates the exploration of parameter-efficient training methods and architectural optimizations.

To address these challenges, we investigate several strategies for improving task-specific fine-tuning. For paraphrase detection, we utilize task structure through symmetric data augmentation. For sonnet generation, we adopt a two-stage fine-tuning strategy to better adapt GPT-2 to Shakespearean language. In addition, we explore efficiency-oriented techniques such as LoRA [3] and FlashAttention [4]. We further analyze how different LoRA injection locations affect performance, providing insight into effective parameter-efficient adaptation. Our experiments show that these strategies improve model performance across both tasks while also providing practical benefits in training efficiency.

3 Related Work

Recent work has explored parameter-efficient alternatives to full-parameter fine-tuning. Parameter-efficient fine-tuning (PEFT) methods adapt large language models by updating only a small subset of parameters, reducing memory usage and training cost compared to full-model fine-tuning. LoRA [3, 5, 6] is a widely used PEFT method that injects low-rank trainable matrices into Transformer weight updates while keeping the original model parameters frozen. This enables efficient adaptation with minimal additional parameters, motivating the use of LoRA for task-specific GPT-2 fine-tuning.

Standard Transformer attention requires constructing an $N \times N$ attention matrix, leading to quadratic memory usage and heavy memory bandwidth overhead. FlashAttention [4] improves efficiency without changing the attention formulation. It introduces an IO-aware tiled algorithm that computes attention in blocks that fit in on-chip SRAM, avoiding materialization of the full attention matrix and reducing memory transfers. This enables exact softmax attention with linear memory usage and substantial speedups in Transformer models. These improvements motivate investigating how such efficiency gains interact with task-specific fine-tuning strategies in practical GPT-2 applications.

For niche downstream tasks where data is limited and the language distribution differs from general pretraining corpora, domain-adaptive pretraining (DAPT) has been proposed as an effective approach. Gururangan et al. [7] systematically study DAPT, where a pretrained language model is further trained on a large unlabeled corpus from the target domain before task-specific fine-tuning. The experiments demonstrate that such intermediate pretraining consistently improves downstream performance across multiple domains and tasks. This setting is analogous to our sonnet generation task, where the available fine-tuning data is limited and the model must adapt to a specialized literary style.

4 Approach

4.1 Symmetric Data Augmentation for Paraphrase Detection

For paraphrase detection, we adopt the GPT-2 baseline from the CS224N default project, which formulates classification as a cloze-style generation task where the model predicts “yes” or “no” for a sentence pair (s_1, s_2) .

Notice that paraphrase detection is a symmetric task. If s_1 is a paraphrase of s_2 , then s_2 is also a paraphrase of s_1 . Ideally, the model should therefore satisfy: $P_\theta(y \mid s_1, s_2) = P_\theta(y \mid s_2, s_1)$. However, GPT-2 processes tokens autoregressively, and the prompt places s_1 before s_2 , which may introduce an unintended positional bias.

To address this issue, we apply symmetric data augmentation. For each training example (s_1, s_2, y) , we add an additional training instance (s_2, s_1, y) with the same label. This doubles the training dataset and acts as a data-level regularization enforcing the symmetry of the paraphrase relation[8]. This augmentation is implemented in the dataset preprocessing pipeline by appending the reversed sentence pair during data loading.

4.2 Parameter-Efficient Fine-Tuning (LoRA)

In addition to full-parameter fine-tuning, we explore parameter-efficient adaptation using Low-Rank Adaptation (LoRA) [3]. LoRA freezes the pretrained model weights and learns low-rank updates for selected linear layers, enabling efficient adaptation while keeping most model parameters fixed.

In our implementation, we insert LoRA adapters into several linear layers of GPT-2, including query-key-value attention projections and feed-forward network layers. This allows us to study how different LoRA injection locations affect downstream task performance while significantly reducing the number of trainable parameters.

4.3 Two-Stage Fine-Tuning for Sonnet Generation

The Shakespeare Sonnet dataset used for training is relatively small, which provides limited exposure to the broader linguistic patterns of Shakespearean writing. To address this limitation, we adopt a two-stage fine-tuning strategy inspired by domain-adaptive pretraining (DAPT) [7]. In addition to the sonnet dataset, we leverage a larger corpus of Shakespeare’s other works, which is approximately $80\times$ larger than the sonnet dataset.

We first adapt GPT-2 to this broader Shakespeare corpus using causal language modeling (first-stage fine-tuning), and then fine-tune the model on the Shakespeare Sonnet dataset to specialize it for sonnet generation (second-stage fine-tuning). This two-stage procedure allows the model to capture broader Shakespearean language patterns before adapting to the specific structure and style of sonnets. For baseline, we train models using only sonnet-level fine-tuning on the sonnet dataset.

4.4 FlashAttention for Efficient Training

To improve training efficiency, we integrate FlashAttention [4] into the GPT-2 architecture. FlashAttention provides a memory-efficient implementation of the standard attention operation while preserving the exact mathematical formulation.

Given query Q , key K , and value V , the attention operation is defined as $\text{Attention}(Q, K, V) = \sum_i \text{softmax}\left(\frac{QK_i^T}{\sqrt{d_k}}\right) V_i$, where K_i and V_i denote blocks of the key and value matrices.

FlashAttention computes this same operation using a memory-efficient kernel without materializing the full attention matrix. In our implementation, we replace the original attention module in GPT-2 with the FlashAttention kernel from the official library, while keeping the model architecture and training objective unchanged.

5 Experiments

5.1 Data

For paraphrase detection, we use the Quora Question Pairs dataset with standard train/dev/test splits. For sonnet generation, in addition to the Shakespeare sonnet dataset used for the final training stage, we construct a larger Shakespeare corpus for first-stage pretraining. This corpus is obtained from the Complete Works of William Shakespeare available on Project Gutenberg [9].

For the preprocessing of the Shakespeare corpus dataset, to avoid train–test leakage, we remove all sonnets from the corpus. We further verify that the remaining text does not contain any held-out sonnets (both dev and test) by matching distinctive first-line signatures from the evaluation set against the corpus. During this process, we discovered that *The Passionate Pilgrim* contains text overlapping with Sonnet 138, which we manually removed from the training corpus. Finally, we apply a lightweight preprocessing pipeline to clean the Shakespeare corpus by removing structural metadata such as stage directions, speaker labels, scene headers, and line numbers. This produces a cleaner text corpus for first-stage pretraining. For more details about the preprocessing procedures, please see Appendix A.

5.2 Evaluation method

We evaluate our models using both task-specific performance metrics and system efficiency metrics.

For the paraphrase detection task, we measure classification performance using **accuracy** and **F1 score**. For the sonnet generation task, we evaluate generation quality using the **chrF** metric.

In addition to task performance, we evaluate the efficiency impact of our proposed methods. For experiments involving LoRA, we additionally report the number and percentage of **trainable pa-**

Dataset	Model	Dev Acc	Test Acc
SST	Last-Linear	0.4696	0.4751
	Full Model	0.5104	0.5267
CFIMDB	Last-Linear	0.8694	–
	Full Model	0.9837	–

Table 1: Sentiment analysis results on the SST and CFIMDB datasets. We test last-layer linear probing and full-model fine-tuning.

Task	Dev Score	Test Score
Paraphrase Detection	Acc = 0.914	Acc = 0.892
Sonnet Generation	chrF = 44.308	chrF = 43.521

Table 2: Leaderboard performance for the paraphrase detection and sonnet generation tasks.

rameters to quantify parameter efficiency relative to full-model fine-tuning. Also, when comparing models with and without FlashAttention, we record system-level metrics including **average epoch training time** and **peak GPU memory usage**. These metrics allow us to quantify the computational benefits of the proposed techniques while maintaining comparable model performance.

5.3 Experimental details

For **paraphrase detection**, we evaluate both full-parameter fine-tuning and parameter-efficient adaptation with LoRA. We first study the effect of symmetric data augmentation by training GPT-2 Medium with and without the augmentation strategy. For parameter-efficient adaptation, we insert LoRA adapters into the transformer layers of GPT-2 Medium and GPT-2 Large. We evaluate several injection configurations including (1) query and value projections, (2) query-key-value projections, and (3) all linear layers in both attention and feed-forward modules. For all LoRA experiments, we use rank $r = 16$ and scaling factor $\alpha = 32$. The learning rate is fixed at 1×10^{-5} and all other optimization settings follow the default project configuration.

For the **sonnet generation** task, GPT-2 models are fine-tuned for 10 epochs on the Shakespeare sonnet dataset. In the two-stage training setting, we first perform first-stage pretraining on the larger Shakespeare corpus for 2 epochs, followed by task-specific second-stage fine-tuning on the sonnet dataset. The batch size is set to 8 for GPT-2 Medium and 4 for GPT-2 Large, with a learning rate of 1×10^{-5} . During generation, we use the decoding strategy provided in the starter code. In addition, we enable top- k sampling ($k = 40$) and increase the maximum generation length to 180 tokens in the final configuration.

To evaluate the efficiency impact of **FlashAttention**, we replace the standard scaled dot-product attention in GPT-2 with the FlashAttention kernel while keeping all other training configurations identical. This allows us to directly compare training time, memory usage, and task performance between the baseline and FlashAttention models. We report the average epoch training time and peak GPU memory usage for both paraphrase detection and sonnet generation experiments.

5.4 Results

5.4.1 Base Score and Leaderboard

The result of sentiment analysis is listed in table 1.

The leaderboard score for paraphrase detection and sonnet generation is listed in 2.

5.4.2 Effect of Symmetric Data Augmentation for Paraphrase Detection

We evaluate the effect of symmetric data augmentation. Table 3 compares the GPT-2 Medium model trained with FlashAttention under two settings: with and without symmetric augmentation.

Adding symmetric augmentation improves the development accuracy and the F1 score. This improvement is consistent with the structural property of the paraphrase detection task, where the prediction

Model	Dev Acc	Dev F1	Label Consistency $\uparrow$	Score Symmetry $\downarrow$
GPT-2 Medium	0.9092	0.9033	0.9293	0.0722
+ Augmentation	0.9140	0.9084	0.9654	0.0353

Table 3: Effect of symmetric data augmentation on paraphrase detection with FlashAttention. Higher LC and lower SS indicate stronger invariance to sentence order.

Model / LoRA Target Layers	Trainable Params (M)	Trainable %	Dev Acc	Dev F1
GPT-2 Medium (Full FT)	355.00	100%	0.9092	0.9033
GPT-2 Medium + LoRA				
Query + Value	1.57	0.44%	0.8770	0.8698
Query + Key + Value	2.36	0.66%	0.8803 (+0.38%)	0.8730 (+0.37%)
All Linear Layers (QKV + Attention + FFN)	7.08	1.95%	0.8972 (+2.30%)	0.8907 (+2.40%)
GPT-2 Large + LoRA				
All Linear Layers (QKV + Attention + FFN)	13.27	1.68%	0.9017 (+2.82%)	0.8959 (+2.99%)

Table 4: Effect of LoRA injection locations for paraphrase detection.

should remain invariant to the order of the sentence pair. Finally, symmetric data augmentation provides the best performance for paraphrase detection task, improving the dev accuracy to 0.9140 and the dev F1 score to 0.9084.

To further examine order invariance, we compare model predictions for (s_1, s_2) and (s_2, s_1) . We measure two metrics: *Label Consistency (LC)*, the fraction of examples whose predicted labels remain unchanged after swapping sentence order, and *Score Symmetry (SS)*, the average difference between prediction confidence scores $|p(y = 1|s_1, s_2) - p(y = 1|s_2, s_1)|$.

As we can see in the table, symmetric augmentation substantially improves order invariance: Label Consistency increases from 0.9293 to 0.9654, while Score Symmetry decreases from 0.0722 to 0.0353. These results indicate that augmenting the training data with reversed sentence pairs helps the model better capture the inherent symmetry of the paraphrase relation.

5.4.3 Effect of LoRA Injection Locations

Table 4 compares different LoRA injection configurations for paraphrase detection. We use the Query+Value setting as the LoRA baseline, since it is the lightest adaptation with only 1.57M trainable parameters. Extending LoRA to additional layers consistently improves performance: adding key projections yields a small gain, while adapting all linear layers provides a much larger improvement.

The best GPT-2 Medium LoRA configuration applies adapters to all linear layers, achieving a dev accuracy of 0.8972 and dev F1 of 0.8907 while training only 7.08M parameters. Notably, this performance is close to full fine-tuning of GPT-2 Medium (0.9092 dev accuracy and 0.9033 F1), despite using only a small fraction of the trainable parameters.

We further apply LoRA to GPT-2 Large, which improves performance to 0.9017 dev accuracy and 0.8959 dev F1 while training only 13.27M parameters (1.68% of the full model). Due to time and computational constraints, we did not explore even larger models, but these results suggest that LoRA is particularly advantageous for scaling adaptation to larger models, where full fine-tuning would be significantly more expensive.

5.4.4 Effect of Two-Stage Fine-Tuning in Sonnet Generation

Table 5 reports the development chrF scores for different model sizes and training configurations. The baseline models are fine-tuned only on the sonnet dataset. In contrast, the two-stage models first undergo first-stage pretraining on a larger Shakespeare corpus before the final sonnet fine-tuning stage.

Across both GPT-2 medium and GPT-2 large models, two-stage training consistently improves performance compared to sonnet-only fine-tuning. This result indicates that adapting the model to the broader Shakespearean language distribution before task-specific training leads to better sonnet generation quality. In addition, a simple preprocessing step provides a small further improvement, suggesting that removing structural metadata from the corpus can slightly benefit domain adaptation.

Model Configuration	Dev chrF
GPT-2 Medium	40.69
+ Two-stage fine-tuning	42.39 (+4.18%)
GPT-2 Large	41.787
+ Two-stage fine-tuning	42.57 (+1.86%)
+ Two-stage + preprocessing	42.64 (+2.05%)

Table 5: Effect of two-stage fine-tuning and corpus preprocessing on sonnet generation. Improvements are reported relative to the baseline model of the same size.

Model	Avg Epoch Time (s)	Peak Mem (MB)	Dev Acc	Dev F1
GPT-2-Medium (bs=4)	8766.24	8954	0.9111	0.9053
+FlashAttention (bs=4)	9026.60 (+3.0%)	7718 (-13.8%)	0.9107 (-0.04%)	0.9049 (-0.04%)
GPT-2-Medium (bs=8)	OOM on RTX 4070 (12GB) without FlashAttention			
+FlashAttention (bs=8)	5905.07 (-32.6%)	9894 (+10.5%)	0.9092 (-0.2%)	0.9033 (-0.2%)

Table 6: Effect of FlashAttention on training efficiency and performance for paraphrase detection.

This improvement is consistent with our expectations, as adapting the model to the broader distribution of Shakespearean language before task-specific fine-tuning should provide better stylistic and lexical grounding for sonnet generation.

Due to GPU memory limitations, full-parameter fine-tuning of the GPT-2 Large model encountered out-of-memory issues after 8 epochs. To further improve performance, we continued training using LoRA adapters for an additional 10 epochs, applied to the self-attention and feed-forward projection layers. This continuation training slightly improves the dev chrF score to 42.6511. Finally, we experimented with an improved generation strategy by adding top- k sampling in addition to temperature scaling and nucleus sampling. This modification results in our best-performing configuration with a dev chrF score of 44.3084.

5.4.5 Effect of FlashAttention

Table 6 summarizes the impact of FlashAttention on the **paraphrase detection** task. With the same batch size (bs=4), FlashAttention reduces peak GPU memory usage by 13.8% while maintaining nearly identical development accuracy and F1 scores. The average epoch training time increases slightly (about 3.0%), but overall model performance remains unchanged. More importantly, FlashAttention enables larger batch sizes under limited GPU memory. On an RTX 4070 GPU with 12GB memory, the baseline model without FlashAttention cannot run with batch size 8 due to CUDA out-of-memory errors. In contrast, FlashAttention supports stable training with bs=8, reducing the average epoch time by 32.6% compared to the baseline bs=4 setting while maintaining comparable accuracy and F1 scores.

Table 7 reports efficiency and generation performance for different GPT-2 model sizes on the **sonnet generation** task. Across GPT-2, GPT-2 medium, and GPT-2 large models, FlashAttention consistently reduces training time and GPU memory usage. The average epoch time decreases by up to 15% for GPT-2 and by about 6–7% for larger models, while peak memory usage drops by roughly 2–3%. These efficiency improvements come with minimal impact on generation quality. The resulting chrF scores remain comparable to the baseline models, indicating that FlashAttention substantially improves training efficiency while preserving generation performance.

6 Analysis

6.1 Error Analysis for Paraphrase Detection

Experimental Setting We conduct error analysis on our best-performing paraphrase detection model, which is a GPT-2 medium model fine-tuned with full parameter updates and symmetric data augmentation. The model is trained for 10 epochs with a learning rate of 1×10^{-5} . For analysis, we run the model on the Quora dev set and record the predicted label and confidence score for each

Model	Avg Epoch Time (s)	Peak Mem (MB)	Dev chrF
GPT-2	4.06	7612	42.998
+FlashAttention	3.45 (-15.0%)	7476 (-1.8%)	42.944 (-0.1%)
GPT-2-Medium	8.92	10802	43.151
+FlashAttention	8.31 (-6.9%)	10472 (-3.1%)	43.367 (+0.5%)
GPT-2-Large	18.94	22016	43.953
+FlashAttention	17.73 (-6.4%)	21548 (-2.1%)	43.617 (-0.8%)

Table 7: Effect of FlashAttention on training efficiency and generation quality for the sonnet generation task across GPT-2 model sizes. Percentages indicate the relative change compared to the baseline model.

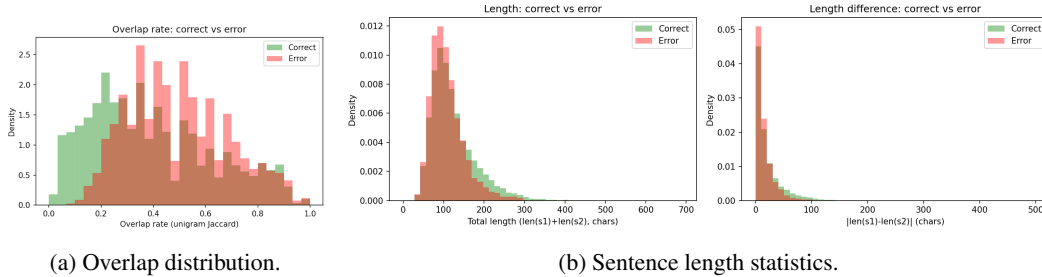


Figure 1: Error analysis statistics comparing correct predictions and errors.

sentence pair. The confidence score is defined as the predicted probability of the paraphrase class, $P(y = 1 \mid s_1, s_2)$. We first analyze statistical properties of correct predictions and model errors on the dev set. In particular, we study lexical overlap and sentence length characteristics to understand what types of inputs tend to cause mistakes.

Lexical overlap. To quantify surface similarity between two questions, we compute the unigram Jaccard similarity between their token sets: $\text{Jaccard}(s_1, s_2) = \frac{|T(s_1) \cap T(s_2)|}{|T(s_1) \cup T(s_2)|}$, where $T(s)$ denotes the set of lowercased tokens in sentence s .

We observe that lexical overlap is substantially higher for erroneous predictions than for correct ones. The average Jaccard similarity is 0.389 for correct predictions, compared to 0.4936 for false positives and 0.4812 for false negatives. Figure 1a further shows that the distribution of overlap values for error cases shifts toward higher similarity. This trend suggests that the model tends to rely heavily on surface lexical similarity when predicting paraphrase relationships. As a result, sentence pairs that share many words but differ semantically are more likely to be misclassified.

Sentence length effects. We further examine sentence length statistics. For each pair (s_1, s_2) , we compute the total character length $\text{len}(s_1) + \text{len}(s_2)$ and the absolute difference $|\text{len}(s_1) - \text{len}(s_2)|$.

Errors tend to occur on shorter inputs overall: the mean total length is 123.6 characters for correct predictions and 109.4 for errors. In addition, sentence pairs with smaller length differences are slightly more prone to errors, with mean $|\text{len}(s_1) - \text{len}(s_2)|$ of 21.0 for correct predictions and 15.1 for incorrect ones (Figure 1b). These observations suggest that shorter questions with similar lengths may encourage the model to rely more on superficial lexical cues rather than deeper semantic relationships.

6.2 Qualitative Comparison of Sonnet Generation

To illustrate the effect of two-stage fine-tuning, we qualitatively compare outputs from our best-performing configuration within 10-epoch fine-tuning constraint (GPT-2 large with full-parameter two-stage fine-tuning). We generate outputs on the dev set using three checkpoints: the pretrained baseline, the model after first-stage pretraining, and the final model after second-stage fine-tuning.

Baseline (No FT)	Stage 1 FT	Stage 2 FT	Ground Truth
These fierce feet, reddened by wandering pleasures! Attack for God! Love does hate, for He hath overcome it. Salute from it natural but thou know'st not what love.	For lukewarm and barren ewes cool, Or honey cold whose gleeth their chimney often runneth; Or a sweet hurt want a sweetness to swell—	Fair in manner, just in mo- tives, And consider what boun- teous act of thy mind Thou dost approve of oth- ers' baseness;	And thou shalt find it mer- its not reproving. Or if it do, not from those lips of thine, That have profaned their scarlet ornaments

Table 8: Qualitative comparison of generated sonnet continuations (first three lines shown).

For comparison, we select Sonnet 142 from the dev set and use its first three lines as the shared prompt. Table 8 shows the generated continuations along with the ground truth. Only the first three generated lines are shown; full outputs are provided in Appendix B.

We can notice that the baseline quickly drifts into modern, prose-like expressions and fails to maintain Shakespearean style. After first-stage pretraining, the text exhibits more Shakespearean vocabulary and imagery but remains semantically unstable. The final two-stage model further improves stylistic consistency and thematic coherence, capturing themes such as love, sin, and moral judgment that also appear in the ground truth. This example suggests that first-stage pretraining improves stylistic alignment, while the second-stage further enhances semantic coherence and rhetorical structure.

7 Conclusion

In this project, we explore several strategies for improving GPT-2 on two tasks: paraphrase detection and Shakespearean sonnet generation. For paraphrase detection, symmetric data augmentation improves robustness by enforcing invariance to sentence order, while error analysis shows that many model mistakes are associated with high lexical overlap. We further investigate parameter-efficient adaptation using LoRA, showing that inserting LoRA adapters into Transformer layers achieves performance comparable to full fine-tuning while training only a small fraction of model parameters.

For sonnet generation, a two-stage fine-tuning strategy inspired by domain-adaptive pretraining substantially improves generation quality by first adapting the model to a broader Shakespeare corpus. Additional preprocessing and decoding strategies provide further gains.

We also study efficiency-oriented techniques for large Transformer models. FlashAttention reduces GPU memory usage and enables training with larger batch sizes under limited hardware resources. Across both tasks, it consistently improves training efficiency while maintaining comparable model performance.

However, our experiments are constrained by limited computational resources, which prevents exhaustive exploration of model scales and hyperparameter settings. Future work could investigate larger models, more comprehensive ablations of parameter-efficient fine-tuning methods, and more advanced decoding strategies to further improve generation quality.

Team contributions (Required for multi-person team)

Sixian Du: Implemented the two-stage fine-tuning pipeline and generation method for sonnet generation, conducted experiments leading to the best-performing sonnet model, and performed qualitative and error analysis for both tasks (the analysis section).

Susan Li: Finished code blocks for paraphrase detection and sonnet generation, integrated FlashAttention into the GPT-2 model, and developed the infrastructure for logging and analyzing training efficiency metrics (runtime and GPU memory usage).

Yuzhou Bian: Implemented and evaluated LoRA-based parameter-efficient fine-tuning with experiments on different injection layers. Implemented symmetric data augmentation for paraphrase detection and training optimizations including learning rate scheduling and gradient accumulation.

References

- [1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [2] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
- [3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.
- [4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.
- [5] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.
- [6] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.
- [7] Suchin Gururangan, Ana Marasovic, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don’t stop pretraining: Adapt language models to domains and tasks. *CoRR*, abs/2004.10964, 2020.
- [8] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks, 2019.
- [9] William Shakespeare. *The Complete Works of William Shakespeare*. Project Gutenberg, 2000. Project Gutenberg eBook #100. Accessed: 2026-02-25.

A Shakespeare Corpus Preprocessing

A.1 Cleaning Pipeline

To construct the Shakespeare corpus used for the first stage of two-stage fine-tuning, we apply a cleaning pipeline to the Complete Works text obtained from Project Gutenberg. The goal is to remove structural metadata while preserving the underlying language used in dialogue and verse.

The preprocessing script removes several categories of non-linguistic content, including:

- stage directions such as *Enter*, *Exit*, and bracketed directions,
- scene and act headers (e.g., “ACT I”, “SCENE I”),
- speaker labels written in all caps (e.g., “COUNTESS.”, “BERTRAM.”),
- character list entries and work titles,
- trailing line numbers appearing in long poems,
- redundant blank lines.

After these steps, the corpus contains only the spoken dialogue and poetic text from Shakespeare’s works, providing a cleaner language distribution for first-stage pretraining. The following example illustrates the effect of the preprocessing pipeline.

Original text

ACT I
SCENE I. Rossillon. A room in the Countess’s palace.
Enter Bertram, the Countess of Rossillon, Helena, and Lafew, all in black.
COUNTESS.
In delivering my son from me, I bury a second husband.

BERTRAM.

And I in going, madam, weep o'er my father's death anew; but I must attend his majesty's command, to whom I am now in ward, evermore in subjection.

LAFEW.

You shall find of the king a husband, madam; you, sir, a father. He that so generally is at all times good, must of necessity hold his virtue to you, whose worthiness would stir it up where it wanted, rather than lack it where there is such abundance.

After preprocessing

In delivering my son from me, I bury a second husband.

And I in going, madam, weep o'er my father's death anew; but I must attend his majesty's command, to whom I am now in ward, evermore in subjection.

You shall find of the king a husband, madam; you, sir, a father. He that so generally is at all times good, must of necessity hold his virtue to you, whose worthiness would stir it up where it wanted, rather than lack it where there is such abundance.

A.2 Sonnet Leakage Verification

To ensure that the first-stage Shakespeare corpus does not contain evaluation data, we perform an automatic leakage verification step. We detect potential overlap by extracting a distinctive signature for each held-out sonnet (in dev and test datasets). Specifically, we use the first line of each sonnet as a signature and search for these strings within the Shakespeare corpus. Because the opening line of a sonnet is typically unique, this provides a reliable way to detect whether the sonnet text appears in the corpus.

If a match is found, the corresponding section of the corpus is flagged as overlapping with the evaluation set and removed. During this process, we identified that the section *The Passionate Pilgrim* in the Gutenberg corpus contains text corresponding to Sonnet 138. We therefore removed this section from the training corpus. This verification step prevents the model from observing evaluation or test sonnets during the first-stage training process.

B Full Sonnet Generation Examples

Table 9 shows the full generated continuations for the qualitative example discussed in Section 6.2.

Baseline (No FT)	Stage 1 FT	Stage 2 FT	Ground Truth
These fierce feet, redden by wandering pleasures! Attack for God! Love does hate, for He hath overcome it. Salute from it natural but thou know'st not what love. Divine Plan definitely alone; none follow but those son'rible! For now not even money but thy vows: None haste but devoted devotion. (6) Drive back sensual lust also, weigh results, wrong; Roath thy father passion with wound, obeie him angrily, In wrath trample the hero, give sacrifice; Don't suffer him to thank thee above with tears.	For lukewarm and barren ewes cool, Or honey cold whose gleeth their chimney often runneth; Or a sweet hurt want a sweetness to swell— Or venom foul, danger- ous stuff provoke, Prove a deadly flint to sides touched. Where delight lies, sum- mer grows, ease bathes, Spring strives, when man hopes on snow. False smells deceive great-skins with straw. Beauty comes naked to make fools delight; Art faint hast fear'd a lick and blush'd a blush. Bring me wine, officer! Let some	Fair in manner, just in mo- tives, And consider what boun- teous act of thy mind Thou dost approve of oth- ers' baseness; How many follies, many sins, add up in thee, Adding up to one, hate my sin more. How easy it is for hearts full of hate To accomplish some per- fect line, lacking grace, Although hearts them- selves are choked with evil. Then may I dare to say I love thee, yet doubt me not That love makes bold ac- tion so. For thy own part, Even now my conscience tells me I do not.	And thou shalt find it mer- its not reproving. Or if it do, not from those lips of thine, That have profaned their scarlet ornaments And sealed false bonds of love as oft as mine, Robbed others' beds' rev- enues of their rents. Be it lawful I love thee as thou lov'st those Whom thine eyes woo as mine importune thee; Root pity in thy heart, that, when it grows, Thy pity may deserve to pitied be. If thou dost seek to have what thou dost hide, By self-example mayst thou be denied.

Table 9: Full generated continuations for the qualitative example in Table 8.