

Preference-Tuning GPT-2 with LoRA and DPO for Classification and Poetry

Stanford CS224N Default Project

Yuliia Murakami, Hiromichi Murakami
Department of Computer Science
Stanford University
ym256@stanford.edu, hm256@stanford.edu

Abstract

We study parameter-efficient fine-tuning as an alternative to full-model fine-tuning for adapting GPT-2 to downstream tasks. Specifically, we evaluate LoRA and DPO on both classification and generation tasks. Our results show that LoRA improves training efficiency with only a small performance drop, while DPO can partially recover this gap when initialized from a strong SFT baseline. We also find that the impact of negative examples varies by task, suggesting that preference construction must be tuned empirically.

1 Key Information to include

- Mentor: Advit Deepak
- External Collaborators or Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
Yuliia	4
Hiromichi	2
Total pooled	6
Team size	2
Late days applied for the report	2

2 Introduction

Large pretrained language models have demonstrated strong performance across a wide range of natural language processing tasks. However, many practical applications still benefit from task-specific post-training, and fully fine-tuning all model parameters can be computationally expensive and memory-intensive. This challenge has motivated the development of parameter-efficient fine-tuning methods that adapt models while updating only a small subset of parameters.

One widely used approach is LoRA, which inserts trainable low-rank matrices into selected model layers while keeping most pretrained weights frozen. LoRA offers substantial savings in training cost and memory usage, but recent work suggests that these gains may come with a trade-off in adaptation quality. Another important post-training approach is Direct Preference Optimization (DPO), which directly trains a model to prefer chosen outputs over rejected ones using pairwise preference data. Although DPO has shown promising results, its performance depends strongly on the quality of the preference signal and the design of negative examples.

In this project, we study the interaction between LoRA and DPO using a GPT-2 baseline on two tasks: paraphrase detection and Shakespearean sonnet generation. These tasks let us compare the methods across both classification and generation settings. We focus on three main questions: the trade-off between performance and efficiency under LoRA, the effect of DPO when initialized from different supervised baselines, and the role of negative-example construction in preference learning across tasks.

3 Related Work

Pretrained language models such as GPT-2 demonstrate strong zero-shot transfer across a wide range of NLP tasks, suggesting that sufficiently large models can perform downstream tasks without task-specific parameter or architecture changes [1]. However, zero-shot transfer does not solve every task equally well, and post-training remains useful when higher task-specific performance, alignment, or adaptation to a particular domain is needed. Post-training can require substantial amounts of data and computational resources, making it expensive in practice. One influential approach for reducing this cost and enabling more efficient GPU sharing is **Low-Rank Adaptation (LoRA)** [2], which performs parameter-efficient fine-tuning by learning low-rank updates while freezing most pretrained weights. LoRA has been shown to achieve competitive performance across a range of tasks while substantially reducing memory and training costs relative to full fine-tuning [2]. Although parameter-efficient adaptation methods can in general face a trade-off between efficiency and model quality, the LoRA paper argues that low-rank updates can often avoid much of this gap [2]. However, recent work suggests that LoRA and full fine-tuning are not always equivalent: even when downstream performance is similar, LoRA can learn different spectral structure and may be less robust under continued adaptation, pointing to a possible trade-off between efficiency and expressivity [3]. These trade-offs motivate our analysis of not only task performance but also practical efficiency metrics, including VRAM usage and training time, in order to understand when LoRA provides a meaningful advantage in realistic task settings.

A complementary line of work addresses performance through preference-based alignment. Traditional reinforcement learning from human feedback (RLHF) trains a reward model from human preference data and then optimizes the policy with reinforcement learning [4, 5]. **Direct Preference Optimization (DPO)** [6] replaces this more complex pipeline with a simpler objective defined directly over preferred and rejected response pairs. Although DPO is attractive for its simplicity and strong empirical performance, its effectiveness depends critically on the availability of informative negative examples. Constructing such negatives is substantially more challenging in open-ended generation tasks, where poor responses are not naturally defined and the usefulness of a negative example is often ambiguous.

This challenge motivates our use of a **curriculum-style** negative construction strategy inspired by curriculum learning [7]. Instead of exposing the model to all rejected examples at once, we organize negatives from easy to hard, allowing it to first learn coarse structural violations and then progressively refine finer stylistic distinctions. Prior work on curriculum learning suggests that presenting training examples in a meaningful order can improve optimization and generalization by guiding the model toward better solutions [7]. In the DPO setting, this idea is especially relevant because informative negative examples are critical for effective preference learning, yet they are often difficult to design for structured generation tasks. Our work is therefore motivated not only by the goal of improving performance on the provided tasks, but also by the broader question of how negative-example design influences the effectiveness of DPO in practice.

4 Approach

We begin with the baseline model provided in the CS224N default project and extend it with two adaptation methods: parameter-efficient fine-tuning using LoRA and preference optimization using DPO. For sonnet generation, we compare three training settings: (1) **SFT with full fine-tuning**; (2) **SFT with layer unfreezing and LoRA**; and (3) **SFT with full fine-tuning followed by DPO with full fine-tuning**.

For paraphrase detection, we additionally evaluate a fourth configuration: (4) **SFT with full fine-tuning followed by DPO with LoRA adapters**. This comparison allows us to isolate the effects of parameter-efficient adaptation and preference-based optimization relative to the baseline supervised fine-tuning setup.

4.1 Baseline Model

Our baseline implementation follows the provided CS224N GPT-2 scaffold and adheres closely to the standard GPT-2 architecture introduced by Radford et al. [1]. We use the **GPT-2 small configuration** (12 layers, hidden size 768, 12 attention heads) to maintain comparability across experiments and tasks. We deliberately fix the model size throughout our study so that changes in performance reflect

the effect of our proposed architectural and training modifications, rather than the advantage of scaling to a larger pretrained model, which would likely yield stronger results.

All components of the baseline model were implemented using the project’s starter code without architectural modifications. The baseline training pipeline and model implementation are based on the provided repository and course infrastructure.

4.2 Sentiment Classification

For this baseline implementation, we follow the provided configuration and do not modify hyperparameters beyond those specified in the starter code.

4.3 Paraphrase Detection

Our baseline paraphrase detection model follows the GPT-2 scaffold provided in the CS224N starter code. The model receives a prompt of the form and predicts a binary label for the pair.

```
Question 1: "<s1>"
Question 2: "<s2>"
Are these questions asking the same thing?
```

We implement LoRA using the HuggingFace PEFT library. LoRA adapters are inserted into the query and value projection matrices of the GPT-2 attention layers.

In the adapter-only configuration, all pretrained GPT-2 backbone weights remain frozen and only the LoRA adapter parameters are updated. To study the trade-off between parameter efficiency and adaptation capacity, we additionally experiment with selectively unfreezing the final k transformer layers of the policy model. In the LoRA sweep, we restrict our comparison to the last-2, last-4, and last-6 layer configurations because evaluating a broader range of settings is computationally prohibitive on this dataset.

For preference optimization, we implement DPO using custom training code based on the standard DPO objective. The policy model is initialized from the best SFT checkpoint, while a frozen reference model remains fixed during optimization. Unlike our sonnet-generation experiments, the paraphrase DPO experiments do not use multi-stage curriculum training. Instead, we compare several alternative strategies for constructing rejected examples within the same DPO framework.

Negative pair construction for paraphrase DPO. We explored two strategies for constructing preference pairs. The first is a simple *label-flip* baseline, where the chosen response is the correct label and the rejected response is the opposite label. The second strategy uses *pairwise negative sampling*, a paraphrase-specific preference construction method motivated by TA feedback. Instead of generating the rejected response by simply flipping the label, we construct a rejected prompt by pairing the first question with a different non-paraphrase question sampled from the training set. This produces a chosen prompt (s_1, s_2^+) and a rejected prompt (s_1, s_2^-) , where s_2^- is a non-paraphrase question. We categorize these negatives into three difficulty levels: easy, medium, and hard, based on lexical overlap between questions. Specifically, for a candidate negative question s_2^- , we compute a token-overlap score:

$$\text{overlap}(s_1, s_2^-) = \frac{|\text{tokens}(s_1) \cap \text{tokens}(s_2^-)|}{|\text{tokens}(s_1) \cup \text{tokens}(s_2^-)|}$$

Easy negatives are randomly sampled non-paraphrase questions. Medium negatives are sampled from the middle of the overlap distribution, while hard negatives are sampled from the highest-overlap region, producing pairs that are lexically similar but semantically different.

We also explored an SFT-hard strategy in which an SFT-trained GPT-2 paraphrase classifier was used to select the most confusing negative candidate from a pool. In this strategy, several candidate non-paraphrase questions are sampled for each positive pair (s_1, s_2^+) . The SFT-trained classifier is then used to estimate the probability that each candidate would be predicted as a paraphrase. The candidate with the highest predicted paraphrase probability is selected as the rejected example. The motivation is to construct model-informed hard negatives that the current classifier would find particularly confusing.

Finally, we also evaluate a combined SFT + LoRA + DPO configuration. In this setting, the policy model is initialized from the best SFT checkpoint, LoRA adapters are attached to the attention projections, and the model is optimized using the DPO objective while the reference model remains frozen. In some runs, we additionally allow the final k transformer layers to update together with the LoRA adapters.

4.4 Sonnet Generation

For the baseline model, we train GPT-2 using standard supervised fine-tuning (SFT) with the cross-entropy next-token prediction objective, following the provided starter code.

To improve training efficiency and adaptation quality, we extend the baseline model with LoRA [2]. In our implementation, LoRA adapters are inserted into the attention projection matrices of GPT-2 using the `peft` library [8]. Most pretrained GPT-2 weights remain frozen, but we do not use a strictly adapter-only setup. Instead, we additionally unfreeze the final k transformer layers and the final layer normalization parameters, where k ranges from 2 to 12, and evaluate how this choice affects downstream performance. The LoRA configuration further includes the adapter rank r , scaling factor α , and dropout applied to adapter activations.

To improve generation quality, we apply DPO. [6]. In our implementation, we construct preference pairs for sonnet generation by treating the ground-truth sonnet continuation as the preferred output and corrupted or mismatched continuations as rejected outputs. We implement DPO training using the `trl` library’s `DPOTrainer`, while adapting our GPT-2 implementation to match the HuggingFace causal language model interface [9].

The reference model is the baseline SFT model and it remains fixed during training, while the policy model is updated to increase the probability of preferred continuations relative to rejected ones.

Negative pair construction for sonnet DPO. A key original part of our approach is how we construct rejected continuations for preference learning on the sonnet task. We organize the rejected examples by difficulty and use a curriculum-style strategy: the model is first trained on *easy* negatives that are clearly wrong, and then exposed to progressively harder negatives that are structurally closer to valid Shakespearean continuations. This design is intended to first teach coarse structural constraints and then refine stylistic and semantic preferences. We emphasize that these negatives are created automatically from existing sonnet data via augmentation and corruption; we do not manually author new rejected examples. All custom training scripts to create dataset of negative examples and model modifications were implemented by our team. Details of the negative-example construction procedure are provided in Appendix A.2.

5 Experiments

5.1 Data

We use the datasets provided in the CS224N default project starter code:

- **SST** (Stanford Sentiment Treebank): 5-class sentiment classification on single sentences.
- **CFIMDB**: binary sentiment classification on movie reviews.
- **Quora Question Pairs**: paraphrase detection on question pairs.
- **Sonnet Dataset**: 154 sonnets written by Shakespeare.

For the paraphrase DPO experiments, we constructed two types of preference datasets.

First, we used a simple label-flip strategy where the correct label is treated as the chosen response and the opposite label is treated as the rejected response. We constructed preference pairs directly from the original Quora Question Pairs training set. Rather than precomputing and storing a separate preference dataset, the pairs are generated dynamically during data loading from `data/quora-train.csv`.

Second, we construct a **pairwise negative sampling** dataset in which non-paraphrase questions are used as rejected examples. We categorized negatives into three difficulty levels:

- Easy negatives: randomly sampled non-paraphrase questions
- Medium negatives: sampled from the middle of a lexical-overlap distribution

- **Hard negatives:** sampled from the highest lexical-overlap region

We also experimented with an SFT-hard strategy in which an SFT-trained GPT-2 paraphrase classifier was used to select the most confusing negative candidate from a pool.

All preference pairs are generated dynamically during dataset loading rather than stored as separate files. This design simplifies experimentation and ensures that preference pairs remain consistent with the original Quora dataset.

The dynamic construction of preference pairs also allows us to easily compare multiple negative-construction strategies without modifying the underlying dataset.

For the sonnet preference optimization experiments, we additionally construct a derived preference dataset from the original sonnet corpus. We store the training data in three JSONL files—`train_easy.jsonl`, `train_medium.jsonl`, and `train_hard.jsonl`—corresponding to curriculum levels of increasing difficulty, and we store the development data in a single combined file, `dev.jsonl`.

We follow the provided preprocessing and dataloading pipeline without modification.

5.2 Evaluation method

We follow the official project evaluation setup for all tasks. For sentiment classification and paraphrase detection, we report **accuracy** on the dev and test splits as computed by the provided autograder. This metric is appropriate because both tasks require discrete label prediction.

For sonnet generation, we report **chrF**, a character-level n-gram similarity metric that compares generated sonnet continuations against the reference Shakespeare continuations. We observed that our internal implementation of the chrF metric produced slightly different absolute values from the provided evaluation pipeline; therefore, all reported results use the official evaluation script for consistency. Because automatic overlap metrics do not fully capture poetic quality, we also present representative generated sonnets for qualitative analysis of fluency, coherence, structural adherence, and Shakespearean style.

In addition to task performance, we evaluate **computational efficiency** and **training/total runtimes** for the LoRA experiments. We record approximate end-to-end runtime and peak GPU memory (VRAM) usage during training and evaluation. These measurements allow us to analyze the trade-off between model quality and resource efficiency, and to test whether parameter-efficient fine-tuning can reduce computational cost while preserving strong performance.

5.3 Experimental details

Environment: Experiments were run on a Google Cloud Tesla T4 (16GB) GPU VM.

Model: Pretrained GPT-2 baseline loaded via the provided scaffold.

Optimizer: AdamW (starter implementation).

Hyperparameters:

- **Sentiment (SST / CFIMDB):** We ran two fine-tuning modes:
 - **Last-linear-layer** (freeze GPT-2 and train only the classifier head):
batch_size=64, lr=1e-3, hidden_dropout_prob=0.1, epochs=5
 - **Full-model** (fine-tune all GPT-2 parameters):
batch_size=8, lr=1e-5, hidden_dropout_prob=0.1, epochs=3
- **Paraphrase:**
 - **SFT baseline:**
batch_size=8, lr=1e-5, epochs=3
 - **SFT + LoRA:**
batch_size=8, lr=1e-4, epochs=3, r=8, alpha=16, dropout=0.05
Adapters are applied to the query and value projections of the last 6 transformer layers.
 - **SFT + DPO:**
batch_size=8, lr=1e-5, epochs=3, beta=0.1
The last 6 transformer layers are updated.
 - **SFT + LoRA + DPO:**
Same DPO hyperparameters as above, combined with the same LoRA configuration.

- **Sonnet generation:**
 - **SFT baseline:**
batch_size=8, lr=1e-5, epochs=10
 - **SFT + LoRA:**
batch_size=8, lr=1e-5, epochs=10, r=8, alpha=16, dropout=0.05
 - **SFT + DPO:**
batch_size=8, lr=1e-5, epochs=10, beta=0.1

Additional details on the hyperparameter sweeps are provided in Appendix A.1.

5.4 Results

Baseline model. Table 1 reports the autograder accuracies for provided tasks on baseline model.

Task / Setting	Dev Acc	Test Acc
SST (Last-Linear)	0.438	0.451
SST (Full Model)	0.505	0.526
CFIMDB (Last-Linear)	0.857	—
CFIMDB (Full Model)	0.971	—

Table 1: Baseline model task results.

Paraphrase Detection. Table 2 summarizes the best development and test accuracy of each configuration.

Model	Dev Accuracy	Δ vs. SFT	Test Accuracy
SFT baseline	0.8940	+0.0000	0.8620
SFT + LoRA	0.8720	-0.0220	—
SFT + DPO	0.8965	+0.0025	0.8710
SFT + LoRA + DPO	0.8962	+0.0022	—

Table 2: Paraphrase detection results on the Quora dev and test sets.

The final 3-epoch SFT baseline already performs strongly, however for this task we were able to achieve the best overall result by SFT + DPO with 0.8965 dev accuracy. SFT + LoRA + DPO reaches a nearly identical 0.8962. In contrast, SFT + LoRA alone falls below the full SFT baseline.

SFT The SFT phase revealed that updating only a small fraction of the model is insufficient for strong paraphrase performance. Training only the classifier head reached 0.720 dev accuracy, while unfreezing only the last two transformer layers improved performance to 0.848. In contrast, full supervised fine-tuning for three epochs reached 0.894, showing that a strong SFT checkpoint is already highly competitive on this task. These preliminary ablations motivated our later focus on strong SFT initialization for all downstream DPO experiments.

LoRA While SFT + LoRA did not match the full SFT baseline in accuracy, it remained substantially more efficient. Per-epoch training time decreased from roughly 75.6 minutes for full SFT to about 34.5 minutes for LoRA, corresponding to an approximately 54% reduction in training time. The LoRA configuration also required only 147,456 trainable parameters, which is about 0.12% of the full model. In terms of GPU memory usage, full SFT reached approximately 4484 MiB peak allocated VRAM and 5228 MiB peak reserved VRAM during training, whereas LoRA used about 1965 MiB peak allocated VRAM and 3096 MiB peak reserved VRAM. Thus, LoRA reduced peak allocated VRAM by about 56% and peak reserved VRAM by about 41% relative to full SFT. These results show that LoRA provides a clear efficiency advantage in runtime, trainable parameters, and GPU memory usage, although it does not match the strongest full SFT baseline in dev accuracy.

DPO We also evaluated multiple strategies for constructing DPO preference pairs. The label-flip DPO baseline achieved the best performance among the negative-construction variants. Although we expected that harder negative examples would provide a stronger learning signal for DPO, the pairwise strategies consistently underperformed the simpler label-flip baseline: Pairwise Easy, Medium, Hard, and SFT-hard achieved dev accuracies of 0.7190, 0.7143, 0.7161, and 0.6593 respectively. These

results suggest that more complex negative construction does not necessarily lead to better preference learning signals for this task.

The quality of the SFT initialization proved especially important for DPO. In earlier experiments using a weaker SFT checkpoint, label-flip DPO reached 0.8718 dev accuracy and LoRA + DPO reached 0.8732. After rerunning both methods from the best 3-epoch SFT checkpoint (0.8940), their performance improved substantially to 0.8965 and 0.8962, respectively. This comparison shows that DPO is most effective when it starts from a strong supervised model.

Sonnet Generation.

SFT For sonnet generation, we first established a SFT baseline and performed a small hyperparameter search over the learning rate. Among the tested values, $1e-5$ produced the best dev chrF score (41.427), while higher or lower learning rates resulted in slightly worse generation quality. Interestingly, the configuration with the best chrF did not correspond to the lowest training loss, suggesting that next-token prediction loss does not perfectly correlate with the chrF evaluation metric used for this task.

We also evaluated the effect of model scale and observed that larger GPT-2 variants generally achieved slightly higher chrF scores, with GPT-2 XL reaching the best dev score of 41.755.

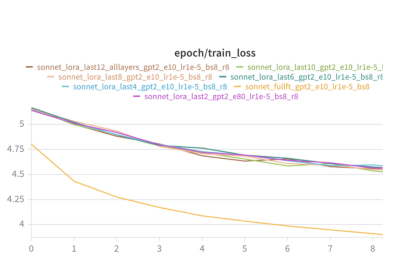


Figure 1: Loss for LoRA relative to full SFT.

Across all runs, LoRA achieved a large reduction in trainable parameters while maintaining reasonable performance relative to SFT. Full SFT updates 125.0M parameters, whereas LoRA updates only 4.9×10^4 – 2.95×10^5 parameters (0.039%–0.236% of SFT). Despite this drastic reduction, the final training loss remained relatively stable across LoRA depth: applying LoRA to the last 2 through all 12 layers worsened loss by a narrow band of approximately 16.0%–17.9% relative to SFT (Figure 1). We were surprised that increasing the number of LoRA-applied layers did not consistently close the gap to SFT; the results were largely flat across layer choices. In terms of efficiency, LoRA substantially reduced training time compared to SFT, with the last-2 setting yielding the largest speedup. Peak allocated VRAM increased with LoRA depth (consistent with greater active/gradient memory), while peak reserved VRAM was non-monotonic, reflecting allocator caching effects rather than purely model capacity. Overall, if the goal is maximum efficiency, applying LoRA to the last two layers provides the fastest and most memory-efficient configuration, at the cost of a $\sim 17.9\%$ increase in loss versus SFT. For full experimental details and metrics, please refer to Table 4 in the Appendix A.3.

DPO Our performance of the best DPO Hard model on test set is **41.025**. We evaluate a curriculum-style DPO pipeline initialized from the best SFT checkpoint and then apply DPO in three stages using progressively harder negative examples: easy, medium, and hard. The results are reflected in Table 3.

We initially expected a monotonic improvement across these stages, but were surprised that only the Hard model achieved a substantial gain over the SFT baseline. This suggests that negatives closest to valid sonnet continuations provide the most informative feedback. Easy and medium negatives mostly emphasize coarse errors that the SFT model may already avoid, whereas hard negatives better match the subtle stylistic and structural distinctions that matter at evaluation time.

Model	Dev chrF	Δ vs. SFT
SFT Baseline	41.427	+0.000
DPO Easy	41.637	+0.210
DPO Medium	41.159	-0.268
DPO Hard	42.233	+0.806

Table 3: Sonnet-generation results for curriculum-style DPO.

Qualitative Analysis Since automatic overlap metrics do not fully capture poetic quality, we also examine representative generated sonnets to assess fluency, coherence, structural adherence, and Shakespearean style in Appendix A.4. Overall, the generations show a progression in stylistic quality

from easy to hard, but all three settings reveal a common limitation: the model can produce words and phrases that resemble poetry without consistently producing poems that make poetic sense. This suggests that successful sonnet generation requires more than archaic vocabulary or local fluency.

6 Analysis

Across both tasks, our main finding is that LoRA and DPO serve different but complementary roles. LoRA is most valuable as an efficiency method: it makes adaptation substantially cheaper, but in our setting it does not fully recover the expressivity of full fine-tuning. DPO, in contrast, is most useful as a refinement method: it provides benefits only when the underlying supervised model is already strong. Taken together, these results suggest that parameter-efficient adaptation and preference optimization should not be viewed as interchangeable alternatives, but as tools that operate at different stages of the post-training pipeline.

A second broader lesson is that preference optimization is highly sensitive to the quality of the training signal. Our experiments show that simply making negative examples more complicated does not guarantee better learning. Instead, the usefulness of a negative example depends on how well it matches the structure of the task. For classification, overly constructed negatives can introduce noise and weaken the preference signal, while for structured generation, negatives that are closer to valid outputs appear to be more informative. This suggests that negative-example design is not a one-size-fits-all recipe, but an empirical problem that must be tailored to the task and the type of errors the model needs to correct.

Overall, our experiments point to a practical recipe for small-model post-training: first obtain the strongest possible SFT checkpoint, then use DPO only as a targeted refinement step, and use LoRA primarily when efficiency constraints matter. More generally, our findings suggest that the success of post-training depends less on adding more sophisticated methods in isolation and more on matching each method to the role it is best suited to play.

7 Conclusion

In this project, we studied LoRA and DPO for GPT-2 post-training on paraphrase detection and sonnet generation. Overall, our findings suggest that the two methods play complementary roles: LoRA is most useful for improving training efficiency, while DPO is most useful as a refinement step on top of a strong SFT checkpoint. We also find that the effectiveness of preference optimization depends strongly on task-specific negative-example design.

These observations point to several promising directions for future work. For paraphrase detection, a natural next step is to improve the quality of DPO negatives using stronger external LLMs to generate or rank more semantically precise hard negatives. For sonnet generation, it would be valuable to study the effect of each negative-example category in more detail and to evaluate alternative decoding strategies such as temperature tuning, different sampling methods, and beam search. On the adaptation side, our current LoRA setup only modifies attention projections; exploring broader adapter placement, as well as alternative methods such as QLoRA or TinyLoRA, may help reduce the remaining gap to full fine-tuning while preserving efficiency.

Team contributions

Hiromichi led the paraphrase detection experiments, including implementing and evaluating the SFT baseline as well as the LoRA, DPO, and LoRA + DPO configurations. This included designing and running experiments on different negative-example construction strategies and analyzing their impact on preference optimization.

Yuliia led the sonnet generation experiments. This included implementing LoRA and the curriculum-style DPO training pipeline, constructing negative examples of varying difficulty levels, and evaluating generation quality using the chrF metric and qualitative analysis.

Both team members collaborated closely on the implementation of the GPT-2 baseline and the sentiment classification experiments through pair programming. In addition, both authors jointly designed the experimental setup, analyzed the results, and contributed equally to writing and editing the final report.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI Technical Report, 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, 2021.
- [3] Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. Lora vs full fine-tuning: An illusion of equivalence, 2024.
- [4] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.
- [5] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback, 2020.
- [6] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2023.
- [7] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48, 2009.
- [8] Hugging Face. Pefit: Parameter-efficient fine-tuning of billion-scale models on low-resource hardware. <https://github.com/huggingface/peft>, 2026. Accessed 2026-03-09.
- [9] Hugging Face. Trl: Transformer reinforcement learning. <https://github.com/huggingface/trl>, 2026. Accessed 2026-03-09.

A Appendix

A.1 Hyperparameter Sweep Details

Paraphrase Detection. We conducted a sequence of small sweeps to identify stable configurations for SFT, LoRA, DPO, and LoRA + DPO. Our overall strategy was to first establish the strongest SFT checkpoint, and then use that checkpoint as the initialization for downstream preference-optimization experiments.

(1) SFT baseline sweep. We first explored how much of the GPT-2 backbone should be updated under supervised fine-tuning. In preliminary experiments, freezing the entire GPT-2 backbone and training only the classification head substantially hurt performance, reaching only 0.720 dev accuracy. Unfreezing only the last two transformer layers improved dev accuracy to 0.848, while full fine-tuning remained the strongest setting. In the earlier 1-epoch baseline run, full fine-tuning with learning rate 1×10^{-5} achieved 0.872 dev accuracy and required roughly 75 minutes per epoch. We then extended the full SFT baseline to 3 epochs with the same learning rate (1×10^{-5}), which produced the best checkpoint used in the final comparison. This 3-epoch full fine-tuning setup achieved 0.894 dev accuracy and became the reference SFT checkpoint for later DPO-based experiments.

(2) SFT + LoRA sweep. To isolate the effect of parameter-efficient fine-tuning, we next applied LoRA to the same paraphrase task using the same dataset and prompt format as the SFT baseline. We inserted LoRA adapters into the query and value projection matrices of GPT-2 attention layers and varied the number of adapted top transformer layers. In particular, we tested LoRA on the last 2, 4, and 6 transformer layers. We didn’t test any further since it is computationally expensive. In these early 1-epoch sweeps, last-2 LoRA reached 0.809 dev accuracy, last-4 reached 0.803, and last-6 reached 0.824, while also reducing training time and peak reserved VRAM relative to stronger LoRA settings.

(3) DPO sweep. For DPO, all experiments were initialized from an SFT checkpoint and used a fixed training setup of batch size 8, learning rate 1×10^{-5} , $\beta = 0.1$, and AdamW. We first swept the amount of trainable backbone capacity by varying `train_last_k`. In the label-flip DPO setting, updating only the last 2 transformer layers gave 0.8109 dev accuracy, while updating the last 6 layers improved this to 0.8376. Extending training to 3 epochs with `train_last_k=6` further improved dev accuracy to 0.8718, nearly matching the earlier 1-epoch SFT baseline.

(4) SFT + LoRA + DPO sweep. Finally, we evaluated a combined SFT + LoRA + DPO setup, following the TA suggestion to start from the same SFT checkpoint as full DPO and inject LoRA only during the DPO stage. In these experiments, the LoRA configuration was fixed to query/value adapters on the last 6 transformer layers with $r = 8$, $\alpha = 16$, dropout 0.05, batch size 8, learning rate 1×10^{-5} , and $\beta = 0.1$, while we swept the DPO-side backbone capacity by varying `train_last_k`. When starting from the weaker earlier SFT checkpoint, updating only the LoRA adapters (`train_last_k=0`) reached 0.7976 dev accuracy, adding the last two transformer layers improved this to 0.8462, and updating the last six transformer layers improved it further to 0.8732. These results showed that LoRA adapters alone were insufficient during DPO and that additional trainable backbone capacity was necessary.

Sonnet Generation. We performed a small hyperparameter sweep to select the final configurations used in the main experiments. For sonnet generation under SFT, we varied the learning rate and found that $1e-5$ achieved the best development chrF score, while both larger and smaller values gave slightly worse results. We also compared different GPT-2 model sizes and observed that larger models produced somewhat higher chrF scores, although we kept GPT-2 small as the main model for consistency across experiments.

For LoRA, we varied the number of adapted top transformer layers and compared efficiency and training loss across configurations. In particular, we evaluated LoRA applied to the last 2, 4, 6, 8, 10, and 12 transformer layers. These experiments showed that LoRA consistently reduced the number of trainable parameters, runtime, and GPU memory usage relative to full fine-tuning, while the final loss remained within a relatively narrow range across layer choices. The detailed results of this sweep are reported in Table 4.

We next performed a focused sweep over the DPO β parameter while fixing the best learning rate from the previous sweep (3×10^{-6}), along with `epochs=1` and `warmup_ratio=0.0`. We evaluated $\beta \in \{0.03, 0.1, 0.3\}$ and selected the best configuration using `dev_chrf_continuation` on held-out sonnet completions. The best performance was achieved with $\beta = 0.1$, yielding `dev_chrf_continuation=29.47` and `dev_chrf_full=44.29`. Both smaller (0.03) and larger (0.3) values resulted in lower chrF scores. Notably, although $\beta = 0.3$ produced a slightly lower evaluation loss, it led to worse generation quality, indicating that improvements in the DPO objective do not necessarily translate to better text generation performance.

A.2 Construction of Negative Examples for Sonnet Generation DPO

This section describes the procedure used to construct negative examples for preference learning in the sonnet generation task.

Each sonnet is first parsed from the dataset and filtered to ensure that it contains exactly 14 lines. For each sonnet, the first three lines are used as the prompt, while the remaining eleven lines form the ground-truth continuation, referred to as the chosen response. For each prompt–chosen pair, we generate three rejected continuations corresponding to different difficulty levels: *easy*, *medium*, and *hard*. These examples are stored as JSONL entries containing the fields `id`, `prompt`, `chosen`, `rejected`, `difficulty`, and `corruption_type`.

Easy negative examples. Our *easy* rejected examples include continuations with the wrong number of lines, severely truncated outputs (e.g., only 5–8 lines), shuffled continuation lines, or repeated copies of a single line. The goal is to provide strong negative signals for obvious structural violations of sonnet form.

Medium negative examples. The *medium* negatives are designed to appear superficially plausible but clearly worse than the correct continuation. These are created using operations such as replacing one line with a line sampled from another sonnet, swapping the order of two lines, repeating a single line once, or replacing the final couplet with lines from a different sonnet.

Hard negative examples. Finally, the *hard* negatives preserve most of the sonnet structure while introducing subtle defects. These include small lexical edits to a line (e.g., duplicating or removing a word), replacing a single line with a line from another sonnet, or introducing a small stylistic or rhyme-related disruption.

A.3 Efficiency Comparison of SFT and LoRA Unfreezing Levels for Sonnet Generation

Model	Trainable params (%)	Train-only total s (% Δ)	Wall total s (% Δ)	Peak reserved MiB (% Δ)	Peak alloc MiB (% Δ)	Final loss (% Δ)
SFT (full FT)	125,030,400 (100.000%)	62.08 (+0.0%)	153.20 (+0.0%)	7,800 (+0.0%)	3,859.8 (+0.0%)	3.8730 (+0.0%)
LoRA last2	49,152 (0.039%)	30.57 (-50.8%)	71.54 (-53.3%)	7,468 (-4.3%)	1,990.8 (-48.4%)	4.5670 (+17.9%)
LoRA last4	98,304 (0.079%)	34.53 (-44.4%)	76.37 (-50.2%)	5,402 (-30.7%)	2,154.9 (-44.2%)	4.5583 (+17.7%)
LoRA last6	147,456 (0.118%)	40.09 (-35.4%)	80.66 (-47.4%)	7,754 (-0.6%)	2,321.7 (-39.8%)	4.5487 (+17.4%)
LoRA last8	196,608 (0.157%)	44.18 (-28.8%)	85.44 (-44.2%)	5,238 (-32.8%)	2,488.0 (-35.5%)	4.5396 (+17.2%)
LoRA last10	245,760 (0.197%)	48.41 (-22.0%)	90.60 (-40.9%)	6,472 (-17.0%)	2,653.4 (-31.3%)	4.4939 (+16.0%)
LoRA all (last12)	294,912 (0.236%)	51.65 (-16.8%)	92.75 (-39.5%)	6,976 (-10.6%)	2,821.4 (-26.9%)	4.5362 (+17.1%)

Table 4: Comparison of SFT and LoRA configurations with different numbers of unfrozen top transformer layers for sonnet generation. Percent deltas (% Δ) are computed relative to SFT as $(x - x_{SFT})/x_{SFT} \times 100$. For final loss, delta is computed as $(loss - loss_{SFT})/loss_{SFT} \times 100$, so positive values indicate higher loss than SFT. Bold text indicates the best value in each efficiency-related column among the LoRA configurations. Red marks the largest loss gap relative to SFT, and green marks the smallest loss gap.

A.4 Sonnets Qualitative Analysis

Original Sonnet The original poem expresses the speaker’s suffering caused by the beloved’s disdain while praising the beloved’s appearance. Its central conceit personifies the beloved’s eyes as mourners dressed in black, so that sorrow itself becomes a form of beauty. The poem follows the Shakespearean sonnet form of three quatrains and a concluding couplet, and it develops its idea through clear rhetorical progression, elevated diction, and a memorable paradox in the final lines.

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be,
Looking with pretty ruth upon my pain.
And truly not the morning sun of heaven
Better becomes the gray cheeks of the east,
Nor that full star that ushers in the even
Doth half that glory to the sober west
As those two mourning eyes become thy face.
O, let it then as well beseem thy heart
To mourn for me, since mourning doth thee grace,
And suit thy pity like in every part.
Then will I swear beauty herself is black,
And all they foul that thy complexion lack.

Easy Generation The easy-level continuation quickly departs from the meaning and structure of the original poem. Although some words sound poetic in isolation, the lines do not form a coherent argument or image. In poetic terms, the continuation makes little sense: it reads less like a sonnet and more like a sequence of locally probable words and phrases. The central conceit disappears, the syntax is unstable, and the text fails to build toward a concluding couplet.

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be,
For when thy best blessed blood rushes
With thee, clean green glade kept from looking,
Yet burning hot ashes wax replete me;
In blindness patience longs thou in love more quickly
Than I’s black speech, waning days nor winters delivered,
The most strangely houring homage
To thy ugly vowel, whose limbs die under decay,
Despite me! Sire grant I contented yourself

With every one that would not wail over me,
Nor consolument hated me to that age
Himself refrained in his riot, and went out,
To beg their intercession

Medium Generation The medium-level generation is slightly more fluent on the surface, but it still lacks semantic coherence and poetic unity. Some phrases imitate elevated diction, yet the poem does not preserve the original conceit or develop a meaningful progression across lines. As in the easy setting, the result often feels like probability-driven word selection rather than an intentional poetic composition, and it does not achieve the structural closure expected of a sonnet.

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be,
For when thy outward tone reposes
Upon a bleak score lay eyes, full of wound,
It descends into living wit to greet thy loved say.
Even so am I adored, to whom the plea sings 'I love thee most blessed-form
Cause thee so captivated is my love,
Making all mine mysteries whether for thee alive or gone
Nurse's cup and vashes my beauty to creeds and leaves.
As thou was guilty of every herb I ate,
My beauty died from all cold walls, delighting morrow.
'Tis despair of thy living memory by dry

Hard Generation The hard-level generation comes closest to Shakespearean tone and vocabulary, with more serious diction and somewhat stronger local fluency. However, it still introduces unrelated themes and breaks the conceptual continuity of the original poem. While it sounds more literary than the easier variants, it remains weak as poetry because it does not sustain the sonnet's governing idea or arrive at a strong rhetorical conclusion. In this sense, it is a better stylistic imitation, but not yet a fully coherent poem.

Thine eyes I love, and they, as pitying me,
Knowing thy heart torment me with disdain,
Have put on black, and loving mourners be,
For sorrowing body becomes melancholy, and passion a hapless gift,
Sitting disgraced on body to hold;
Within themselves not grace but self unworthy sort
For thy body to mourn becomes with thee to mourn thy friend's death.
On self exposed swindler, kings and all other idols,
Your were my humility's head, on false gods mockers thy hurt;
So forbearance is a tyrant's delight
That if I doth write forgiveness, what whit I can use it.
The goddess forbids thee to write excuse,
That gushing desert wastes thy health;