

Efficient Alignment Is All You Need

Stanford CS224N {Default}

Lingbo Duan

Department of Electrical Engineering
Stanford University
lingbo@stanford.edu

Yufei Liu

Department of Computer Science
Stanford University
yf6@stanford.edu

Shatong Zhu

Department of Electrical Engineering
Stanford University
stzhu@stanford.edu

Abstract

Adapting large language models (LLMs) to stylistically constrained generation remains computationally expensive. We propose a GPT-2 framework combining **Sliding Window Attention**, **Low-Rank Adaptation (LoRA)**, and **Direct Preference Optimization (DPO)** to improve Shakespearean sonnet generation under limited compute. To keep evaluation feasible, we report component-wise evidence: Sliding Window and LoRA are evaluated in controlled classification settings, while DPO and decoding are evaluated directly on sonnet generation. Sliding Window Attention increases training throughput from 13.81 to 16.10 it/s (17%) without improving classification accuracy, indicating mainly computational benefits. On SST classification, LoRA substantially reduces trainable parameters and memory while improving over last-layer tuning, reaching accuracy close to full-model fine-tuning and slightly higher macro-F1 under our matched setting. For generation, supervised fine-tuning achieves 42.256 chrF with top- p sampling, improving to **43.153** with constrained beam search (best leaderboard). DPO reaches **43.003 chrF** under stochastic decoding but collapses to 41.914 chrF with beam search, highlighting decoding–alignment trade-offs.

1 Key Information to include

- TA mentor: Advit Deepak
- No External collaborators, No External mentor, No Sharing project
- Late Days (if you are using any):

Team member	Late days remaining
Lingbo Duan	6
Yufei Liu	5
Shatong Zhu	4
Total pooled	15
Team size	3
Late days applied for the report	1

2 Introduction

Adapting large language models (LLMs) to specialized domains remains computationally expensive, as full fine-tuning updates all parameters and requires substantial memory and compute. Stylistically

constrained tasks such as Shakespearean sonnet generation further challenge standard next-token training because models must capture rhyme, structure, and literary tone. Parameter-efficient fine-tuning (PEFT) methods exploit the observation that effective adaptation occurs in low-dimensional subspaces [1]. In particular, Low-Rank Adaptation (LoRA) inserts trainable low-rank matrices while freezing pretrained weights, reducing cost with minimal performance loss [2]. We combine LoRA with Direct Preference Optimization (DPO), which aligns outputs using pairwise preferences [3]. Because full end-to-end ablations are expensive, we evaluate Sliding Window and LoRA in controlled classification settings as efficiency proxies, and evaluate generation quality directly on the CS224N Sonnet Dev leaderboard using chrF. While alignment is a core objective, our study treats it as one pillar alongside compute efficiency and parameter efficiency.

3 Related Work

Recent work has explored **parameter-efficient fine-tuning (PEFT)** as an alternative to updating all parameters of large language models. **LoRA** introduces low-rank parameterizations of weight updates in transformer layers, allowing models to adapt to downstream tasks while keeping most pretrained parameters fixed [2]. As shown in Figure 1, this approach dramatically reduces training cost and storage requirements while maintaining performance comparable to full fine-tuning across a variety of NLP benchmarks. Related research such as **Representation Fine-Tuning (ReFT)** (Figure 2) further investigates how modifying intermediate representations rather than model weights can provide an efficient and interpretable mechanism for adapting pretrained language models [1]. These studies suggest that adaptation can often be achieved through small, structured modifications to pretrained models rather than full retraining.

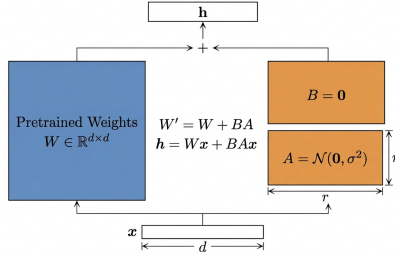


Figure 1: LoRA Structure Illustration[2]

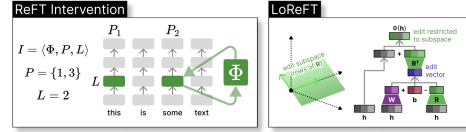


Figure 2: ReFT Structure Illustration[1]

Another line of work focuses on improving the efficiency of attention mechanisms for long-context sequence modeling. **FlashAttention** speed up the training by introducing an IO-aware exact attention algorithm that reduces GPU memory reads and writes through tiling strategies [4]. It can achieve both significant speedups and memory savings while preserving exact attention computation. However, naive windowed attention can degrade generation quality because transformers often rely on early tokens that accumulate disproportionately high attention weights, known as **attention sinks**. Prior work has shown that preserving a small number of these initial tokens in the attention cache while applying sliding-window attention to the remaining context can maintain generation stability and fluency [5].

Additionally, we focus on aligning language models with human preferences. **Direct Preference Optimization (DPO)** directly optimizes the model likelihood using pairwise preference comparisons, eliminating the need for an explicit reward model [3]. Empirical results show that DPO can match or outperform RLHF-based approaches on tasks such as summarization and dialogue while offering greater training stability.

Our work combines three complementary directions in recent language model research. Sliding-window attention targets compute efficiency, LoRA targets parameter-efficient adaptation, and DPO targets preference alignment quality. In this project, we integrate these ideas to study whether they can jointly improve the generation of stylistically faithful Shakespearean sonnets under limited data and compute resources.

4 Approach

4.1 Baseline

Our project builds on the GPT-2 baseline from the provided starter code. We implemented the following core components from scratch to build the baseline [6]:

- **Attention Layer:** We implemented masked multi-head self-attention. We applied an upper-triangular causal mask to the attention weights before softmax normalization to prevent attending to future positions.
- **GPT2Layer:** Using our masked self-attention, we constructed the core transformer layer. The forward pass contains the attention block, layer normalization, a multi-layer perceptron (MLP), and skip connections. The model stacks 12 of these layers.
- **GPT2 Model:** We implemented the embedding layer. This adds up trainable token embeddings (mapping byte pair encodings to 768-dimensional vectors) and positional embeddings (encoding up to 1024 tokens).
- **Optimizer:** We implemented the step() function of the Adam Optimizer that uses Decoupled Weight Decay Regularization and an efficient bias correction method.
- **Classifier:** We implemented the GPT2SentimentClassifier. This pipeline extracts the final hidden state of the last token, applies dropout, and projects it through a linear layer to predict sentiment classes.
- **Paraphrase Detection & Sonnet Generation:** The code that trains and tests for either ParaphraseGPT or SonnetGPT model.

“GPT-2 with Sliding Window Attention, LoRA, and DPO for Shakespearean Sonnet Generation”

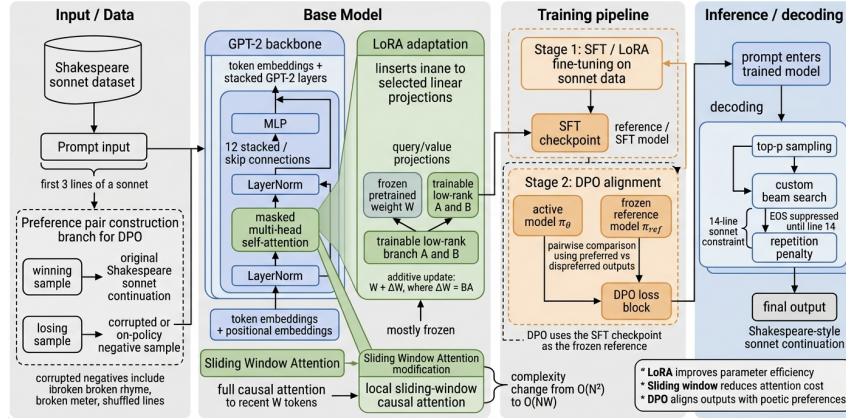


Figure 3: Improved Model Structure

4.2 Sliding Window Attention

We modified the standard GPT-2 self-attention mechanism to incorporate sliding window attention in order to reduce computational cost. In the original transformer architecture, each token attends to all previous tokens, resulting in quadratic computational complexity $O(N^2)$ with respect to sequence length. Sliding window attention restricts each token to attend only to a fixed number of recent tokens within a local context window, reducing the effective computation to approximately $O(NW)$, where W is the window size [5]. This approach follows the broader direction of improving attention efficiency in large language models through more memory- and compute-aware attention mechanisms [4]. In our implementation, we introduced a new configuration parameter `sliding_window_size` into the GPT-2 attention module. When enabled, the attention mechanism computes attention only within the local window instead of across the full sequence. If the parameter is not provided, the model falls back to the standard causal attention mechanism.

4.3 LoRA-based Parameter-Efficient Fine-Tuning

Following the PEFT motivation in Section 2, we study LoRA [2] as our parameter-efficient adaptation method. The core goal is simple: keep pretrained GPT-2 weights mostly fixed, and move task-specific learning into a small set of trainable low-rank modules. This gives a cleaner efficiency-performance tradeoff than full-model updates, which is important for our compute setting.

For each selected linear projection $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, LoRA keeps W frozen and learns an additive update $\Delta W = BA$, where $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$. The resulting transformation is

$$h = Wx + \frac{\alpha}{r}B(A(\text{Dropout}(x))).$$

Here, r is the adapter rank, α controls scaling, and dropout regularizes the adapter branch. We use standard LoRA initialization, with A randomly initialized and B initialized to zero, so the initial behavior stays close to the base model.

Our PEFT framework supports multiple adaptation modes, including a no-adapter baseline and LoRA-enabled training. In this subsection we focus on none versus lora; combined modes with other PEFT methods are part of the broader project but are not analyzed here. For LoRA itself, we vary four design axes: rank r , scaling α , adapter dropout, and target-module presets. The presets range from attention-only targets (e.g., query/value or query/key/value) to larger target sets that also include output and MLP projections.

Training follows a frozen-backbone regime by default: backbone parameters are frozen, while adapter parameters and task heads are optimized. We also include controlled unfreezing options for LayerNorm and bias terms as ablations. Although the framework supports broader targets and unfreezing settings, all final reported LoRA results in this report use the query/value target preset with a frozen backbone (no LayerNorm/bias unfreezing). In budget-sensitive phases we can enforce an explicit trainable-parameter budget B and reject settings with trainable parameter count above B . We report both the absolute trainable count and the trainable ratio

$$\rho = \frac{\#\text{trainable}}{\#\text{total}},$$

so each result can be interpreted jointly in terms of model quality and adaptation cost.

4.4 Beam Search Generation

To improve the decoding quality of our sonnet generation, we implemented a custom beam search algorithm. Instead of sampling from a truncated probability distribution, beam search deterministically explores the sequence space by maintaining the top b most probable sequences at each decoding step. To adapt this for Shakespearean poetry, we introduced structural constraints: the model heavily penalizes the end-of-sequence (EOS) token until 14 lines are generated, and forces the EOS token once the 14th line is completed. Additionally, to mitigate the repetitive loops common in deterministic decoding, we applied a repetition penalty of 1.2 to the logits of previously generated non-newline tokens.

4.5 Direct Preference Optimization (DPO)

To align the model’s output styles with the strict constraints of Shakespearean poetry (e.g., iambic pentameter and the ABAB CDCD rhyme scheme), we implemented Direct Preference Optimization (DPO). DPO replaces the complex and unstable reinforcement learning pipeline of standard RLHF by directly optimizing the language model policy using a binary cross-entropy objective over preference pairs [3]. Given a prompt x , a winning example y_w , and a losing example y_l , the model is optimized using the following loss function:

$$\mathcal{L}_{DPO} = -\mathbb{E}_{(x, y_w, y_l) \sim D} \left[\log \sigma \left(\beta \log \frac{\pi_{\theta}(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_{\theta}(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right]$$

where π_{θ} is the active model, π_{ref} is the frozen SFT reference model, and β controls the strength of the KL divergence penalty.

Because our sonnet dataset contains only positive examples, we designed two distinct strategies to construct the negative examples (y_l):

- **Heuristic Corruption (Offline):** We randomly chose a heuristic to corrupt the original poem. With probability $\frac{1}{2}$, we utilized NLTK and the CMU Pronouncing Dictionary to identify rhyming end-words and replaced them with non-rhyming WordNet synonyms, which breaks the original rhyme. With probability $\frac{1}{6}$, we injected random filler words such as "really" and "actually" to break the iambic pentameter. With probability $\frac{1}{3}$, we randomly shuffled some lines of the poem to break the overall structure.
- **On-Policy Generation:** We prompted our baseline SFT model using the first three lines of each sonnet and high-temperature sampling ($T = 0.9$) to generate model-driven hallucinations and structural errors as the negative samples.

5 Experiments

Given our compute budget, we evaluate the project in a component-wise manner rather than a full factorial end-to-end grid. Sliding Window and LoRA are benchmarked on classification pipelines to isolate compute and parameter efficiency effects under controlled conditions. DPO and decoding strategies are benchmarked directly on sonnet generation, where alignment quality is the primary objective.

5.1 Sliding Window Experiment

5.1.1 Data and Evaluation

The Experiment use classifier.py and SST to test performance. The average input sequence length during training was approximately 40 tokens. We evaluate the impact of sliding window attention by measuring training throughput, reported as iterations per second (it/s) during training.

5.1.2 Experimental details

We modified the attention mechanism by implementing sliding window attention inside the Causal-SelfAttention module. For our experiment, the sliding window size was set to $W = 1$, meaning each token attends only to its most recent local context. To evaluate the computational benefit of sliding window attention, we compare training throughput between the baseline and the proposed local sliding-window variant.

5.1.3 Results

Table 1: Training efficiency comparison between full attention and sliding window attention.

Setting	Avg. Throughput (it/s)	Relative Speedup
Full Attention	13.81	—
Sliding Window	16.10	+17%

The experimental results show that sliding window attention leads to a measurable improvement in training throughput. Table 1 presents the direct efficiency comparison at the same training stage. The sliding-window model achieves a training speed of **16.10 it/s**, compared to **13.81 it/s** for the full-attention baseline, around 17% improvement. The improvement occurs because the sliding window restricts the number of key–query interactions that must be computed within the attention layer. By reducing the number of attention comparisons per token, the model performs fewer computations during each forward pass. The throughput recorded in this section is implemented without LoRA, thus one should not compare the samples-per-second values with the SST LoRA section.

5.2 LoRA Experiments

5.2.1 Data

We evaluate LoRA on the module-level SST sentiment classification pipeline used in our implementation. This setup is used to isolate PEFT behavior under controlled conditions before full integration

with the other project modules. We keep tokenization and preprocessing fixed across all compared settings so that differences are attributable to adaptation strategy rather than input processing changes.

5.2.2 Evaluation method

We report both quality and efficiency metrics. For quality, we use Dev Acc and Dev Macro-F1. For efficiency and fairness, we report trainable parameter count, trainable ratio, training throughput, and peak GPU memory. The trainable ratio is the primary PEFT-efficiency indicator, while throughput and memory are system-level indicators. In this report, we present a matched single-seed comparison and a full hyperparameter sweep over $(r, \alpha, \text{dropout})$.

5.2.3 Experimental details

We compare a no-adaptor baseline and LoRA-based adaptation under matched training pipelines. The baseline side includes last-linear-layer and full-model settings, while the LoRA side uses a frozen backbone with trainable adapters and task heads. For LoRA, we sweep rank $r \in \{8, 16\}$, scaling $\alpha \in \{16, 32\}$, and adapter dropout $\in \{0.0, 0.05\}$. Unless noted otherwise, we use query/value targets for adapter insertion, consistent with the default LoRA grid setup.

To keep comparisons fair, we track trainable parameter counts for every run and optionally enforce a hard trainable-parameter budget in budget-sensitive phases. We run the same evaluation protocol across all compared settings, and we save all metrics into a unified CSV format. This makes later aggregation straightforward, especially when team members integrate non-LoRA modules.

5.2.4 Results

Table 2 summarizes the primary comparison on SST development metrics and efficiency indicators under matched training settings.

Method	Dev Acc	Dev Macro-F1	Trainable Params	Trainable Ratio	Throughput	Peak GPU Mem
Last-linear baseline	0.393	0.245	3,845	0.00003	237.30	477.48
Full-model baseline	0.500	0.412	125,034,245	1.00000	50.77	1492.78
LoRA (best: $r=8, \alpha=32, \text{drop} = 0.05$)	0.493	0.426	298,757	0.00238	94.05	480.44

Table 2: Main comparison for LoRA against non-PEFT baselines on SST dev (seed 11711, 3 epochs, full steps).

Compared with last-linear tuning, LoRA improves Dev Acc and Dev Macro-F1 while keeping trainable parameters and memory far below full-model fine-tuning. Throughput shows a mixed trade-off: LoRA is faster than full-model tuning (94.05 vs 50.77 samples/s) but slower than last-linear tuning (237.30 samples/s). All throughput values in this subsection are measured in samples/s under the SST pipeline and should be interpreted as within-task comparisons.

We ran a full LoRA hyperparameter sweep over $(r, \alpha, \text{dropout})$ and moved the complete 8-setting table to Appendix Table 5 for space. The same pattern holds there: higher rank increases trainable ratio, the best Dev Acc is achieved by $r=8, \alpha=32, \text{drop} = 0.05$, and the best Dev Macro-F1 is achieved by $r=16, \alpha=32, \text{drop} = 0.00$. Detailed sweeping results are showed in Appendix A.1

These LoRA results provide the finalized PEFT comparison used in our report-wide evaluation and serve as the report’s primary evidence for parameter-efficiency trade-offs.

5.3 DPO and Sonnet Generation Experiments

5.3.1 Data and Evaluation

We used the Sonnet Dataset, which consists of 154 sonnets written by Shakespeare. The model is evaluated on a held-out dev set of 12 sonnets using the chrF metric, which measures character-level n-gram overlap with the ground truth. During evaluation, the first three lines of a sonnet are given to the model for it to generate the remaining sequence.

5.3.2 Experimental Details

To prevent overfitting on the small dataset, we first experimented with different SFT training epochs and identified Epoch 20 as our optimal baseline. All subsequent DPO experiments were initialized

from this frozen SFT Epoch 20 checkpoint and trained for 5 epochs. We evaluated our two data construction methods (Offline Heuristic vs. On-Policy) using default hyperparameters ($\beta = 0.1, T = 1.2$). After identifying the better data strategy, we conducted a hyperparameter tuning over the DPO signal strength (β) and sampling temperature (T).

5.3.3 Results

Our key generation results are summarized in Table 3. All results are evaluated on the dev set. The SFT baseline at Epoch 20 achieved a chrF score of 42.256 using standard top- p sampling, and applying our custom beam search improved the score to 43.153.

We move the complete DPO ablation table to Appendix Table 7. In the full ablation, the offline heuristic corruption (42.553) slightly outperformed the on-policy generation (42.529) at default settings, and the best stochastic DPO setting reached 43.003.

Model Configuration	Decoding Strategy	Hyperparameters	chrF Score
SFT Baseline (Epoch 20)	Top- p Sampling	$T = 1.2$	42.256
SFT Baseline (Epoch 20)	Beam Search	$b = 5$	43.153
+ Offline DPO	Top- p Sampling	$\beta = 0.2, T = 1.2$	43.003
+ Offline DPO	Beam Search	$\beta = 0.2, b = 5$	41.914

Table 3: Key generation results in the main text. Full DPO ablations are provided in Appendix Table 7.

During our hyperparameter tuning of the winning offline DPO model, increasing the penalty anchor to $\beta = 0.2$ while maintaining a highly creative temperature ($T = 1.2$) achieved the best sampling score of 43.003. However, attempting to combine this optimal DPO model with beam search led to a severe performance collapse, and the score dropped to 41.914. Ultimately, the SFT Baseline with Beam Search was selected for our final leaderboard submission.

6 Test Leaderboards for paraphrase detection and sonnet generation

Paraphrase Dev	Paraphrase Test	Sonnet Dev	Sonnet Test
0.898	0.871	43.153	42.430

Table 4: Leaderboards for paraphrase detection and sonnet generation

On the test leaderboards, our final system achieved 0.871 accuracy on paraphrase detection and 42.430 chrF on sonnet generation, with corresponding dev scores of 0.898 and 43.153 chrF. These results were largely in line with expectations. Sliding Window Attention improved training throughput (17%) but did not improve accuracy, confirming its role as a computational optimization. LoRA provided a strong quality–efficiency trade-off: compared with last-layer adaptation, it significantly improved quality while using a tiny trainable ratio, and it stayed close to full-model accuracy in our SST proxy experiments. DPO improved stylistic fidelity under stochastic decoding but interacted poorly with beam search, reducing final leaderboard performance. The final leaderboard submission used the SFT+Beam configuration; LoRA and Sliding Window were reported as efficiency-focused module studies, and DPO was excluded from final decoding due to the beam-search collapse. Overall, the results highlight that efficiency extensions can be beneficial, but their impact depends on decoding strategy and task constraints.

7 Analysis

7.1 Sliding Window Attention

While sliding window attention can improve computational efficiency, it does not contribute to better training accuracy in this experiment. Limiting the attention window to such a small size restricts the model’s ability to capture dependencies across tokens in the sequence. As a result, the model loses access to broader contextual information that may be useful for understanding sentence relationships in the paraphrase classification task. In this experiment, the primary benefit of sliding window attention is computational efficiency rather than improved model performance.

7.2 LoRA

The main benefit of LoRA in this setting is the quality-cost tradeoff. In our SST proxy runs, LoRA improves over last-linear tuning (Dev Acc 0.393→0.493, Dev Macro-F1 0.245→0.426) while training only a small fraction of parameters (trainable ratio 0.00238). Compared with full-model tuning, it maintains substantially lower trainable ratio and lower memory pressure, while remaining close in development accuracy.

LoRA does not reduce all components of training cost equally. Even with a small trainable set, the frozen backbone still dominates a large part of the compute path. In our results, LoRA throughput sits between the two baselines (94.05 samples/s vs 50.77 for full-model and 237.30 for last-linear), so its advantage is primarily parameter/memory efficiency rather than absolute speed.

The sweep also shows clear hyperparameter sensitivity. Increasing rank or scaling can improve fit, and moderate adapter dropout helps stabilize performance in this SST setting.

Given this project scope, LoRA is a practical and cost-effective adaptation choice: it keeps implementation simple, substantially reduces trainable parameters, and improves quality over last-layer adaptation while staying close to full-model quality.

7.3 Sonnet Generation and Alignment Analysis

Our generation experiments show a consistent pattern. DPO improves stochastic decoding quality and structural faithfulness relative to the SFT sampling baseline, but the best DPO model collapses under beam search (41.914), while SFT+Beam remains best overall (43.153). This indicates a mismatch between preference-sharpened token distributions and deterministic decoding objectives in our setup. We move detailed qualitative case studies and long-form output examples to Appendix Section A.4.

8 Conclusion

This paper presents a compute-aware GPT-2 framework that integrates Sliding Window Attention for compute efficiency, LoRA for parameter efficiency, and DPO for preference alignment in Shakespearean sonnet generation. Empirically, Sliding Window yields a 17% throughput gain on paraphrase classification, LoRA delivers strong quality-to-cost trade-offs in SST proxy experiments, and on sonnet generation the best final configuration is SFT+Beam at 43.153 chrF while DPO improves stochastic decoding up to 43.003 chrF but fails under beam search (41.914). Because our evidence is component-wise under limited compute (proxy-task efficiency studies plus direct generation evaluations), the claims should be interpreted as modular rather than fully compositional, motivating future work on tighter end-to-end co-design of adaptation, alignment, and decoding.

Team contributions (Required for multi-person team)

Yufei Liu (DPO & Sonnet Generation): I implemented a custom beam search algorithm with length and repetition penalties for Sonnet generation. I also designed and implemented the Direct Preference Optimization (DPO) pipeline by creating both offline heuristics and on-policy dataset construction methods, and conducted the corresponding ablation studies and hyperparameter tuning. Wrote the DPO parts of Method, Experiments, and Analysis.

Lingbo Duan (Baseline Training & Sliding Window Attention): I implemented the baseline GPT-2 fine-tuning pipeline for both sentiment classification, paraphrase detection, and Shakespearean sonnet generation. I designed and integrated the Sliding Window Attention mechanism into the transformer self-attention modules to improve memory efficiency and scalability for long-context generation. I also ran baseline training experiments, performed debugging and optimization of training stability and generation quality, and wrote the baseline and Sliding Window Attention sections of the Method, Experiments, and Analysis.

Shatong Zhu (LoRA module) I implemented LoRA-based PEFT components and integrated them into the sentiment classification training pipelines, including trainable-parameter budget/fairness controls. I also ran LoRA baseline and hyperparameter-sweep experiments, and wrote the LoRA parts of Method, Experiments, and Analysis.

References

- [1] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Reft: Representation finetuning for language models. 2023.
- [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] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, 2023.
- [4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [5] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv*, 2023.
- [6] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

A Extended Experimental Results

In the main text, we presented a distilled set of experiments highlighting only the key results. To preserve completeness while staying within the page limit, we place the full LoRA hyperparameter sweep, comprehensive DPO ablations, and qualitative generation case studies in this appendix.

A.1 LoRA Hyperparameter Sweep

Table 5 reports the full 8-setting sweep over $(r, \alpha, \text{dropout})$ for SST dev under the same matched run setting used in the main text.

r	α	Dropout	Dev Acc	Dev Macro-F1	Trainable Ratio
8	16	0.00	0.470	0.411	0.00238
8	16	0.05	0.490	0.418	0.00238
8	32	0.00	0.469	0.410	0.00238
8	32	0.05	0.493	0.426	0.00238
16	16	0.00	0.480	0.418	0.00473
16	16	0.05	0.481	0.421	0.00473
16	32	0.00	0.481	0.432	0.00473
16	32	0.05	0.477	0.430	0.00473

Table 5: Full LoRA hyperparameter sweep on SST dev (seed 11711, 3 epochs, full steps).

The same trends reported in the main text hold in the full sweep: higher rank increases trainable ratio, the best Dev Acc is achieved by $r=8, \alpha=32, \text{drop} = 0.05$, and the best Dev Macro-F1 is achieved by $r=16, \alpha=32, \text{drop} = 0.00$.

A.2 Supervised Fine-Tuning (SFT) Epoch

To establish a strong baseline and identify the point of overfitting, we trained the base GPT-2 (Large) model for 20, 25, and 30 epochs. For each checkpoint, we evaluated both Top- p sampling ($T = 1.2$) and beam search ($b = 5$).

As shown in Table 6, while the Top- p sampling score continued to rise slightly at Epoch 30 (42.937), the beam search score peaked at Epoch 20 (43.153) and subsequently reduced at Epoch 30 (41.927). This divergence indicates that the model began overfitting to the training distribution after 20 epochs. Consequently, SFT Epoch 20 was selected as the foundational checkpoint for all subsequent DPO experiments.

Base Model	Training Epochs	Decoding Strategy	chrF Score
GPT-2 (Large)	SFT Epoch 20	Top- p ($T = 1.2$)	42.256
GPT-2 (Large)	SFT Epoch 25	Top- p ($T = 1.2$)	42.462
GPT-2 (Large)	SFT Epoch 30	Top- p ($T = 1.2$)	42.937
GPT-2 (Large)	SFT Epoch 20	Beam Search ($b = 5$)	43.153
GPT-2 (Large)	SFT Epoch 25	Beam Search ($b = 5$)	42.853
GPT-2 (Large)	SFT Epoch 30	Beam Search ($b = 5$)	41.927

Table 6: Different epochs for the Supervised Fine-Tuning (SFT) baseline

A.3 Comprehensive DPO Ablations

Table 7 includes the full hyperparameter tunings conducted to optimize the DPO alignment phase. We tested both Offline (heuristic corruption) and On-Policy dataset construction methods against different SFT initialization checkpoints.

The results demonstrate that DPO consistently improves the base stochastic generation across almost all configurations compared to the SFT equivalents. However, the data also clearly illustrates the severe generation collapse that occurs when applying beam search to a DPO-aligned model (dropping from 43.003 to 41.914), which confirms our hypothesis regarding probability traps and deterministic search algorithms.

SFT Checkpoint	DPO Method	DPO β	Temp (T)	Decoding	chrF Score
Epoch 20	Offline	0.1	1.2	Top- p	42.553
Epoch 25	Offline	0.1	1.2	Top- p	42.059
Epoch 20	On-Policy	0.1	1.2	Top- p	42.529
Epoch 25	On-Policy	0.1	1.2	Top- p	42.157
Epoch 20	Offline	0.1	0.8	Top- p	41.592
Epoch 20	Offline	0.2	1.2	Top- p	43.003
Epoch 20	Offline	0.2	1.2	Beam Search	41.914

Table 7: Ablation studies of DPO training and decoding hyperparameters. All DPO models were trained for 5 epochs using a batch size of 2.

A.4 Qualitative Generation Case Studies

Our experiments showed several critical behaviors in out-of-distribution generation and the complex interaction between preference optimization and deterministic search algorithms.

Out-of-Distribution Hallucinations: Qualitative inspection of the baseline SFT model outputs demonstrated severe domain drift. When generating sequences using high temperature, the SFT model frequently lost the Shakespearean context entirely and hallucinated modern phrases. For example, one SFT generation completely abandoned the sonnet format to output: *"...Unmoved by reputation or looks of men. / In good faith I swear it, and so begin / To tell of a night in which many died and were slain. / The first was Major Thomas Duval, of Georgetown County..."*. The implementation of DPO successfully mitigated this failure mode. By heavily penalizing these distributions, the DPO-aligned model remained strictly grounded in the target vocabulary.

Structural Accuracy: The offline DPO training also improved the model’s structural integrity during Top- p -sampling. While the SFT model would frequently drop the ABAB rhyme scheme or lose track of line lengths, the DPO model, which had been explicitly trained against the `break_rhyme_nltk` and `break_meter` heuristics, showed a tighter adherence to iambic pentameter and terminal rhymes.

DPO + Beam Search Collapse: The most striking finding was the severe degradation in chrF score (down to 41.914) when applying beam search to our best DPO-aligned model. Qualitative analysis revealed that DPO worsens the repetitive loops in beam search. For example, in Sonnet 1, the DPO + Beam Search model gives: *"...’Gainst myself alone do I chide, / And ’gainst myself together chide, /*

Since what I most love doth hate, / And 'gainst myself jointly chide...". This shows a fundamental mismatch between preference optimization and deterministic decoding. DPO naturally sharpens the probability distribution around a few highly rewarded stylistic tokens (e.g., "chide"). When we apply beam search, its objective is to maximize the sequence log-probability using the DPO-induced peaks, which leads to repetitions. Ultimately, while DPO clearly improves the baseline distribution for stochastic sampling, standard SFT with constrained beam search provides better structural stability with our highest chrF score of 43.153.