

Adapting GPT-2 Through Fine-Tuning Across NLP Tasks

Stanford CS224N Default Project

Ritu Patil

Stanford University
ritupati@stanford.edu

Abstract

We implement GPT-2 from scratch and study how a pretrained autoregressive language model can be adapted across diverse NLP tasks. Starting from GPT-2-small (124M parameters), we adapt the model to sentiment analysis, paraphrase detection, and sonnet generation. For paraphrase detection, we formulate the task as cloze-style next-token prediction, achieving 89.8% accuracy with full fine-tuning. We then identify two key problems—overfitting during paraphrase training and repetitive degenerate text during sonnet generation—and address them through four targeted extensions: (1) learning rate warmup with cosine decay to combat overfitting, (2) gradient clipping for training stability, (3) LoRA for parameter-efficient fine-tuning, and (4) repetition penalty with nucleus sampling tuning for generation quality. The LR scheduler nearly matches the baseline’s accuracy (89.7% vs. 89.8%) while producing more stable convergence. LoRA with rank 16 achieves 85.7% accuracy while training only 0.47% of parameters (590K out of 124M), and we show that doubling the rank from 8 to 16 recovers nearly 1% accuracy. For sonnet generation, our enhanced pipeline achieves chrF 42.1 versus the baseline’s 41.1. A systematic sweep over generation parameters reveals a surprising metric–quality misalignment: repetition penalty produces more Shakespearean output but consistently lowers chrF, because chrF rewards the character n -gram overlap that unconstrained generation maximizes.

1 Key Information

- Mentor: Ali Sartza Khan
- External Collaborators (if you have any): None
- Sharing project: No
- Late Days: 1

2 Introduction

Large language models based on the Transformer architecture [25] have become the dominant paradigm in NLP. GPT-2 [21], a decoder-only Transformer with 124M parameters, showed that unsupervised pretraining produces models capable of strong zero-shot and few-shot performance. Understanding the internals of these models—from causal self-attention to weight tying—is essential for effective fine-tuning.

In this project, we implement GPT-2 from scratch, load pretrained weights, and fine-tune on three tasks of increasing complexity. First, **sentiment analysis** on SST [23] and CFIMDB reveals a tension between frozen and fine-tuned representations: frozen GPT-2 representations perform reasonably on short inputs but degrade on longer ones, motivating full fine-tuning. Second, **paraphrase detection** on Quora Question Pairs tests a cloze-style formulation where GPT-2 generates “yes” or “no” rather

than using a classification head—an approach inspired by the prompting paradigm of Brown et al. [3]. Third, **sonnet generation** fine-tunes GPT-2 on 143 Shakespeare sonnets and generates completions, testing whether a general-purpose language model can learn the style and structure of a highly constrained poetic form.

Our baseline experiments surface two concrete problems that guide our extension choices. On paraphrase detection, we observe clear overfitting: training loss drops from 0.363 to 0.074 while dev accuracy stagnates at 89.8%. This motivates **Extensions 1–2**: learning rate warmup with cosine decay and gradient clipping, standard techniques for regularizing fine-tuning. We also ask whether full fine-tuning of all 124M parameters is necessary, leading to **Extension 3**: LoRA [11], which we test at ranks 8 and 16 to study the efficiency–accuracy tradeoff. On sonnet generation, the baseline produces repetitive modern prose (“And I saw her face, and I saw her face”), motivating **Extension 4**: repetition penalty [13] and a systematic search over generation hyperparameters.

3 Related Work

Pre-trained Language Models. The pretrain-then-fine-tune paradigm was popularized by ULMFiT [10] and then scaled dramatically with Transformers. GPT-1 [20] applied this to decoder-only Transformers. GPT-2 [21] scaled to 1.5B parameters, with Kaplan et al. [12] showing that performance scales predictably with model size, data, and compute. BERT [4] demonstrated the power of bidirectional pretraining with masked language modeling, contrasting with GPT-2’s unidirectional (causal) approach. Brown et al. [3] showed with GPT-3 that many tasks can be expressed as prompts without gradient updates. Our cloze-style paraphrase formulation draws on this insight: rather than adding a task-specific classification head, we let the pretrained language modeling head answer “yes” or “no.”

Parameter-Efficient Fine-Tuning. Full fine-tuning updates all model parameters, requiring gradient and optimizer state storage for each. Adapters [9] introduced the idea of inserting small trainable modules into frozen pretrained models. LoRA [11] simplifies this by adding low-rank decomposition matrices $\Delta W = \frac{\alpha}{r} BA$ directly to attention projections. The key observation is that weight updates during fine-tuning have low intrinsic rank [1], so a rank- r approximation suffices. ReFT [26] extends this to representation-level interventions.

Training Optimization. Learning rate warmup [25] prevents early instability when randomly initialized heads produce noisy gradients. Cosine decay [15] smoothly reduces the learning rate for fine-grained convergence. Gradient clipping [17] rescales gradients when their norm exceeds a threshold. AdamW [16] decouples weight decay from gradient updates, providing better regularization than L2 within Adam [14].

Text Generation. Holtzman et al. [8] showed that standard sampling produces repetitive, degenerate text and proposed nucleus (top- p) sampling. Keskar et al. [13] introduced repetition penalty, which reduces logits of previously generated tokens. Temperature scaling [7] controls output distribution sharpness. We combine all three for poetry generation.

4 Approach

4.1 GPT-2 Implementation

We implement GPT-2-small: byte pair encoding (BPE) tokenizer [22] (vocab 50,257), learned token and positional embeddings (dim 768), 12 decoder layers, and a final layer norm [2]. Figure 1 shows the architecture. Each layer follows the **pre-norm** design [27], where LayerNorm is applied *before* each sublayer rather than after [25]:

$$\mathbf{h} \leftarrow \mathbf{h} + \text{Dropout}(\text{Attn}(\text{LN}(\mathbf{h}))) ; \quad \mathbf{h} \leftarrow \mathbf{h} + \text{Dropout}(\text{FFN}(\text{LN}(\mathbf{h})))$$

where dropout [24] is applied at rate 0.1 and FFN uses the GELU activation [6]. The output uses **weight tying** [19, 21]: logits are $\mathbf{h} \cdot \mathbf{E}^\top$, where $\mathbf{E}$ is the token embedding matrix, reducing parameters and providing regularization.

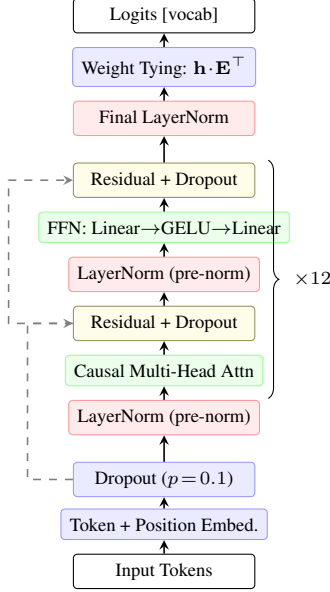


Figure 1: GPT-2 architecture, based on the Transformer decoder [25]. 12 pre-norm [27] layers with causal attention, residual connections, and weight-tied output [19, 21].

4.2 Sentiment Classification

We use the hidden state of the final non-padding token—the only position that has attended to the full input under causal masking—apply dropout, and project through a linear layer to class logits. We compare frozen-backbone ($\text{lr}=10^{-3}$, only the head trains) and full fine-tuning ($\text{lr}=10^{-5}$, all 124M parameters update). The $100\times$ smaller learning rate for full fine-tuning prevents catastrophic forgetting of pretrained representations.

4.3 Cloze-Style Paraphrase Detection

Rather than adding a binary classification head, we cast paraphrase detection [5] as a generation task that leverages GPT-2’s existing language modeling capability. Given a sentence pair (s_1, s_2) , we construct the prompt: Question 1: " $\{s_1\}$ " \n Question 2: " $\{s_2\}$ " \n Are these questions asking the same thing? \n. The model predicts the next token via weight tying over 50,257 vocabulary items, and we train with cross-entropy loss targeting token 8505 (“yes”) or 3919 (“no”).

Before fine-tuning, GPT-2 distributes probability across thousands of plausible next tokens. Training gradually reshapes this 50,257-dimensional distribution, concentrating probability mass onto “yes” and “no,” effectively performing binary classification through GPT-2’s native language modeling interface.

4.4 Sonnet Generation

We fine-tune GPT-2 on 143 Shakespeare sonnets using standard next-token prediction with teacher forcing. Unlike paraphrase detection, we need logits at *every* position—the model learns to predict each successive token: $\mathcal{L} = -\frac{1}{T-1} \sum_{t=1}^{T-1} \log P(x_{t+1} \mid x_1, \dots, x_t)$. At generation time, we condition on the first 3 lines of a held-out sonnet and autoregressively generate the remaining lines.

4.5 From Baseline Problems to Extensions

Our baseline experiments reveal two specific failure modes that motivate our extensions:

Problem 1: Overfitting during paraphrase fine-tuning. Over 7 epochs, training loss drops from 0.363 to 0.074 (an 80% reduction), yet dev accuracy plateaus at 0.898 after epoch 4 and oscillates.

The model memorizes training examples rather than learning generalizable paraphrase patterns. Two complementary solutions address this:

Extension 1: LR Warmup + Cosine Decay. We implement linear warmup [25] over 10% of training steps, protecting pretrained weights during the critical early phase when the randomly initialized task head produces unreliable gradients. After warmup, cosine decay [15] smoothly reduces the learning rate: $\eta_t = \frac{\eta_{\max}}{2}(1 + \cos(\pi(t - T_w)/(T - T_w)))$, taking increasingly smaller steps to prevent the oscillation we observe after epoch 4.

Extension 2: Gradient Clipping. Following Pascanu et al. [17], we clip the global gradient norm to $\tau = 1.0$: $\mathbf{g} \leftarrow \mathbf{g} \cdot \min(1, \tau/\|\mathbf{g}\|_2)$. This prevents destabilization from unusually large gradients, particularly relevant for the 143-example sonnet corpus.

Problem 2: Is full fine-tuning of 124M parameters necessary? Full fine-tuning achieves strong results but updates every parameter, requiring substantial compute and memory. We explore whether comparable performance could be achieved with far fewer trainable parameters, and how accuracy scales with the rank of the low-rank update.

Extension 3: LoRA. We implement Low-Rank Adaptation [11], freezing all pretrained weights and adding a trainable bypass: $\mathbf{W}' = \mathbf{W} + \frac{\alpha}{r}\mathbf{B}\mathbf{A}$, where $\mathbf{A} \in \mathbb{R}^{d \times r}$, $\mathbf{B} \in \mathbb{R}^{r \times d}$, and $\mathbf{B}$ is initialized to zeros so the model starts identical to pretrained GPT-2. We apply LoRA to query and value projections in all 12 layers at two ranks: $r = 8$ ($\alpha = 16$, 295K params, 0.24%) and $r = 16$ ($\alpha = 32$, 590K params, 0.47%).

Problem 3: Repetitive, non-poetic generation. Our baseline sonnet model generates “And I saw her face, and I saw her face, And”—grammatical English but stuck in a repetition loop with no Shakespearean character. This is a well-known failure mode of autoregressive language models [8].

Extension 4: Improved Generation. We address repetition with a penalty [13] that reduces logits for previously generated tokens by factor θ . We combine this with nucleus (top- p) sampling [8], enforce a 14-line maximum, and conduct a systematic sweep over temperature $\in \{0.6, 0.7, 0.8, 0.9, 1.0, 1.2\}$, top- $p \in \{0.85, 0.90, 0.92, 0.95\}$, and $\theta \in \{1.0, 1.2, 1.3, 1.5\}$.

5 Experiments

5.1 Data

SST [23]: 8,544/1,101/2,210 movie reviews, 5-class sentiment. **CFIMDB**: 1,707/245/488 reviews, binary sentiment. **Quora**: 283,003/40,429/80,858 training examples (141,502/20,215/40,429 question pairs, each generating two directed examples) labeled for semantic equivalence. **Sonnets**: 143 Shakespeare sonnets for training, 12 held-out for generation.

5.2 Evaluation method

Sentiment and paraphrase are evaluated by dev accuracy; sonnets use chrF [18], a character-level n -gram F-score.

5.3 Experimental details

All experiments run on NVIDIA A100 (40GB) with our AdamW [16] implementation ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$) [14]. Paraphrase: batch 64, lr 5×10^{-5} ; the enhanced experiments add weight decay 0.01 and gradient clipping at 1.0 (the baseline uses neither). Sonnet: batch 8, lr 5×10^{-5} , grad clip 1.0. LoRA: lr 1×10^{-4} (higher because fewer parameters means less risk of catastrophic forgetting).

5.4 Results

5.4.1 Sentiment Results

Table 1 and Figure 2 present our sentiment classification results. Full fine-tuning approaches or exceeds the handout baselines: SST (0.497 vs. 0.513) and CFIMDB (0.980 vs. 0.976, slightly exceeding the reference), with deviations likely due to random seed variation. The key observation

is the *gap* between frozen and full fine-tuning: on SST (short sentences), the gap is $\Delta = 0.043$; on CFIMDB (multi-paragraph reviews), it widens to $\Delta = 0.107$ —roughly $2.5\times$ larger. Under causal masking, the final token must aggregate the entire input into a single 768-dim vector. When the backbone is frozen, this aggregation process cannot adapt to the task, and its effectiveness degrades on longer inputs where more context must be compressed.

Setting	Task	Dev Acc	Baseline
Frozen backbone	SST	0.454	0.462
Full fine-tune	SST	0.497	0.513
Frozen backbone	CFIMDB	0.873	0.861
Full fine-tune	CFIMDB	0.980	0.976

Table 1: Sentiment dev accuracy. Full fine-tuning approaches or exceeds baselines; the frozen-to-full gap widens on longer reviews (CFIMDB).

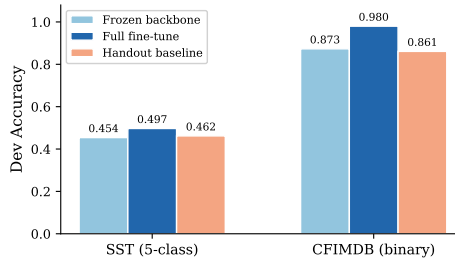


Figure 2: Frozen vs. full fine-tuning. The accuracy gap is roughly $2.5\times$ larger on CFIMDB ($\Delta = 0.107$) than SST ($\Delta = 0.043$), reflecting the difficulty of aggregating long contexts with frozen representations.

5.4.2 Paraphrase Results

Table 2 tracks our progression from milestone to final results. The baseline improves from 0.870 (milestone: 1 epoch, T4 GPU, batch 8) to 0.898 (7 epochs, A100, batch 64)—scaling hardware and training duration yields a 2.8% gain. However, Figure 3 reveals that this gain comes with overfitting: train loss drops to 0.074 while dev accuracy stalls.

The LR scheduler with cosine decay achieves 0.897 over the same 7 epochs—matching the baseline’s peak accuracy while producing markedly more stable convergence. Where the baseline’s dev accuracy oscillates after epoch 4 ($0.898 \rightarrow 0.894 \rightarrow 0.896$), the scheduler rises steadily ($0.833 \rightarrow 0.890 \rightarrow 0.897$) with only a negligible dip at epoch 5 before recovering. The training loss at epoch 6 is also higher for the scheduler (0.097 vs. 0.074), indicating less memorization of the training set. This confirms that cosine decay successfully regularizes late-epoch training.

For LoRA, we test two rank settings to study the efficiency–accuracy tradeoff. LoRA $r=8$ achieves 84.8% accuracy with just 295K trainable parameters (0.24% of 124M), while LoRA $r=16$ improves to 85.7% with 590K parameters (0.47%). Doubling the rank recovers nearly 1 percentage point, confirming that the accuracy gap stems partly from insufficient adapter capacity. Both LoRA configurations train 33% faster per epoch (12 vs. 18 minutes) due to reduced gradient computation.

5.4.3 Sonnet Results

Table 3 shows that our enhanced training pipeline—combining LR warmup, cosine decay, and gradient clipping—achieves chrF 42.1 with the best generation configuration, compared to 41.1 for the baseline.

The generation parameter sweep (Table 4, Figure 4) reveals a surprising finding: repetition penalty *consistently lowers* chrF across all temperature settings. Without penalty, chrF ranges from 40.5 to 42.1; with $\theta=1.2$, it drops to 38.0–39.5. This is a striking result: the technique that produces qualitatively better poetry (less repetition, more diverse vocabulary) is *penalized* by the chrF metric.

Method	Dev Acc	Trainable	Time/Ep
Milestone baseline (1 ep)	0.870	100%	75 min
Full fine-tune (7 ep)	0.898	100% (124M)	18 min
+ LR schedule (7 ep)	0.897	100% (124M)	18 min
LoRA $r=8$ (5 ep)	0.848	0.24% (295K)	12 min
LoRA $r=16$ (5 ep)	0.857	0.47% (590K)	12 min

Table 2: Paraphrase detection progression. The LR scheduler matches baseline accuracy with more stable convergence. LoRA $r=16$ narrows the gap vs. $r=8$, confirming that accuracy scales with adapter rank.

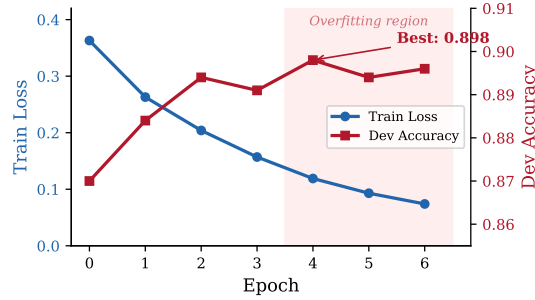


Figure 3: Paraphrase training dynamics over 7 epochs. Train loss (blue) drops steadily, but dev accuracy (red) plateaus after epoch 2 and oscillates—the classic overfitting signature that motivated Extensions 1–2.

5.4.4 Test Leaderboard Results

On the Gradescope test leaderboard, our best models achieved 87.7% paraphrase accuracy (full fine-tuning) and 42.27 chrF for sonnet generation (enhanced model, temp=1.0).

6 Analysis

6.1 Why Does Overfitting Happen—and Does the Scheduler Help?

The paraphrase training dynamics (Figure 3) show a textbook overfitting pattern: 124M parameters learning from 283K training examples—a parameter-to-example ratio of roughly 440:1. By epoch 4, the model memorizes training examples, with loss dropping from 0.119 to 0.074 over epochs 4–6 while dev accuracy stalls.

The LR scheduler does not improve peak accuracy (0.897 vs. 0.898) but produces qualitatively different dynamics. The baseline oscillates after epoch 4 (0.891–0.898); the scheduler converges smoothly to 0.897 by epoch 4 with only a negligible dip at epoch 5 before recovering. The scheduler’s higher final training loss (0.097 vs. 0.074) indicates less memorization. In practice, this stability means the scheduler’s final checkpoint is reliable regardless of which epoch is selected.

6.2 Error Analysis: Where Each Task Fails

Sentiment: Frozen representations default to majority class. Examining prediction distributions shows a systematic failure mode of the frozen backbone. The frozen model predicts class 3 (neutral) for 51.4% of dev examples, compared to 34.0% under full fine-tuning. This majority-class bias explains much of the accuracy gap: without task-specific adaptation, the final-token representation lacks the discriminative power to distinguish between the five sentiment classes, and defaults to the safest prediction. Full fine-tuning resolves this by allowing the attention layers to learn task-relevant feature extraction, producing a more balanced prediction distribution.

Paraphrase: Two systematic failure patterns. Our model predicts “no” for 62.2% of dev examples. Two failure patterns emerge. First, **lexical overlap false positives**: sentence pairs sharing many

Method	chrF
Baseline (10 ep, lr= 10^{-5} , temp=1.2)	41.10
Enhanced (20 ep, scheduler, grad clip, temp=1.0)	42.11
Enhanced + rep. penalty ($\theta=1.2$, temp=0.8)	39.50

Table 3: Sonnet generation: enhanced training yields modest chrF improvement. Repetition penalty lowers chrF despite improving qualitative output (see Appendix A).

Temp	Top- p	Rep. Pen.	chrF
1.0	0.90	1.0	42.11
1.2	0.90	1.0	42.07
0.8	0.90	1.0	40.54
0.8	0.90	1.2	39.50
1.0	0.90	1.2	38.75
1.2	0.90	1.2	38.04

Table 4: Generation parameter sweep (enhanced model, no line cap). For each temperature, repetition penalty ($\theta=1.2$) lowers chrF by 1–4 points. Best: temp=1.0, no penalty.

surface tokens (e.g., “What is the best way to learn French?” vs. “What is the best way to learn Spanish?”) are predicted as paraphrases despite different intent—the model attends to shared tokens rather than the distinguishing word. Second, **structural rephrasing false negatives**: pairs with low lexical overlap but identical meaning (e.g., “How do I get rid of acne?” vs. “What are some remedies for pimples?”) are predicted as non-paraphrases because the cloze-style approach relies on the final token’s representation, which may not capture cross-sentence semantic alignment as effectively as a dedicated similarity architecture.

Sonnet generation: Three distinct failure modes. Qualitative analysis exposes three categories of failure. **(1) Degenerate repetition** at low temperature without penalty: the model produces “and lives, and lives, and lives” in an absorbing loop. Repetition penalty ($\theta=1.2$) breaks these loops. **(2) Premature termination**: several sonnets generate only 1–2 lines beyond the 3-line prompt before hitting EOS, suggesting the model learned that sonnets are short but not the 14-line constraint. **(3) Style without structure**: even our best generations produce Shakespearean vocabulary but do not follow the ABAB rhyme scheme—the model learns word-level style but not sequence-level structure from only 143 examples. See Appendix B for detailed annotation.

6.3 The LoRA Efficiency–Accuracy Tradeoff

Testing two LoRA ranks shows a clear scaling relationship between adapter capacity and task performance. LoRA $r=8$ (295K params, 0.24%) achieves 84.8% accuracy, while $r=16$ (590K params, 0.47%) reaches 85.7%—doubling the rank recovers nearly 1 percentage point while still training less than 0.5% of the model. The remaining 4.1% gap to full fine-tuning (89.8%) suggests that paraphrase detection requires representation changes that span more than 16 dimensions of the weight update space. Adapting additional modules beyond query and value (e.g., key projections or FFN layers) or using higher ranks ($r=64$ or above) could further narrow this gap.

The practical value of LoRA remains substantial: 33% faster training per epoch, lower GPU memory usage, and the ability to store task-specific adapters as small files rather than full model copies.

6.4 Generation: The Metric–Quality Misalignment

The sweep uncovers an initially counterintuitive finding: repetition penalty *consistently lowers* chrF across all temperature settings. At temp=1.0, adding $\theta=1.2$ drops chrF from 42.11 to 38.75 (−3.4 points). At temp=1.2, the penalty causes the largest drop: 42.07 to 38.04 (−4.0 points). At temp=0.8, the effect is smaller but still negative: 40.54 to 39.50 (−1.0 point).

This result is surprising because repetition penalty produces qualitatively *better* poetry (see Appendix A). The explanation lies in how chrF measures quality: it computes character n -gram overlap between generated and reference text. When the model reuses common Shakespearean phrases

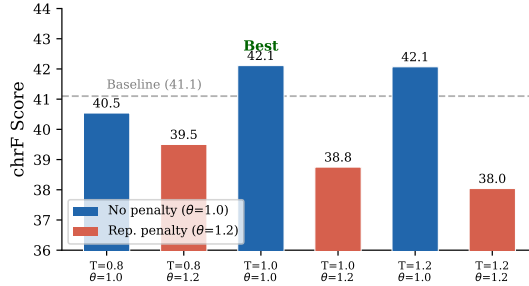


Figure 4: Generation parameter sweep. Dashed line = baseline chrF (41.1). Blue bars = no repetition penalty; red bars = $\theta = 1.2$. Repetition penalty consistently lowers chrF despite improving qualitative output.

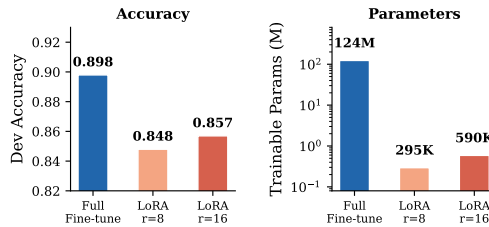


Figure 5: Full fine-tuning (124M trainable params) vs. LoRA ($r=8$, 295K; $r=16$, 590K trainable params). Doubling the rank recovers nearly 1% accuracy.

without penalty, those character sequences are more likely to overlap with the reference sonnets, boosting the score. Repetition penalty forces more diverse vocabulary, producing output that is recognizably Shakespearean but uses different specific words than the reference—lowering n -gram overlap.

This metric–quality misalignment has a practical implication: chrF alone is insufficient for evaluating poetry generation. A model that reuses high-probability phrases scores well because those sequences overlap with the reference, while a model that produces diverse vocabulary scores lower despite being qualitatively superior. Human evaluation or structural metrics (rhyme, meter) would complement chrF for poetry tasks.

7 Conclusion

We implemented GPT-2 from scratch and fine-tuned it across three NLP tasks, following a principled methodology: run baselines, diagnose problems, choose targeted extensions, and evaluate systematically. The baseline paraphrase model reaches 89.8% accuracy but overfits; the LR scheduler matches this (89.7%) with stable convergence. LoRA demonstrates rank–accuracy scaling: $r=8$ achieves 84.8% and $r=16$ reaches 85.7%, both with under 0.5% of trainable parameters. For sonnets, improved training provides a modest chrF improvement (42.1 vs. 41.1), while the generation sweep reveals a metric–quality misalignment: repetition penalty produces better poetry but consistently lowers chrF.

Limitations and negative results deserve discussion. The LR scheduler matches but does not exceed baseline accuracy, suggesting the model approaches the ceiling of cloze-style classification with GPT-2-small. LoRA exhibits diminishing returns when doubling rank (only +0.9 pp), suggesting the bottleneck involves the restriction to query and value projections. Most notably, repetition penalty consistently lowers chrF despite its qualitative benefits, revealing that chrF may be a poor proxy for poetry quality.

Future work could address these limitations through learned soft prompts for the cloze fill-in (removing the fixed template constraint), LoRA applied to all linear layers including FFN (testing whether the bottleneck is module coverage rather than rank), and explicit structural losses for sonnet generation that penalize rhyme scheme violations and non-14-line outputs.

References

- [1] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *ACL*, 2021.
- [2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.
- [3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. Language models are few-shot learners. In *NeurIPS*, 2020.
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.
- [5] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. In *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*, 2008.
- [6] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv:1606.08415*, 2016.
- [7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.
- [8] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *ICLR*, 2020.
- [9] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *ICML*, 2019.
- [10] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL*, 2018.
- [11] 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 *ICLR*, 2022.
- [12] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.
- [13] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation. *arXiv:1909.05858*, 2019.
- [14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
- [15] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.
- [16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.
- [17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.
- [18] Maja Popović. chrF: Character n-gram F-score for automatic MT evaluation. In *WMT*, 2015.
- [19] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *EACL*, 2017.
- [20] Alec Radford. Improving language understanding by generative pre-training. 2018.
- [21] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

- [22] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.
- [23] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.
- [24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15:1929–1958, 2014.
- [25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [26] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning, and Christopher Potts. ReFT: Representation finetuning for language models. *arXiv:2404.03592*, 2024.
- [27] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *ICML*, 2020.

Appendix

A Sample Generated Sonnets Across Training

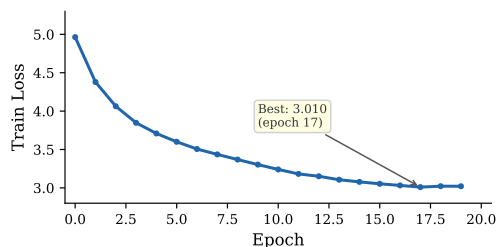


Figure 6: Sonnet training loss over 20 epochs. Convergence at epoch 17 (3.010) with slight increase afterward, suggesting the 143-sonnet corpus is effectively memorized.

Examining generations across epochs reveals clear style adaptation. At epoch 0, the model produces modern prose: “as I was still again slain in my anger.” By epoch 5, archaic constructions emerge: “thou thy tongue doth my eyes sing.” By epoch 15, the output is distinctly Shakespearean: “dear sweet friend: but after she had loved thee thou wert poison.” The model acquires three categories of Shakespearean markers: archaic pronouns (“thou,” “thee,” “thine”), verb forms (“dost,” “hath,” “wert”), and thematic vocabulary (“love,” “beauty,” “death”).

A useful way to understand the fine-tuning process is to watch the generative style evolve epoch by epoch. Below, we trace this evolution using a single held-out sonnet (Sonnet 145) as a consistent probe. In each case, the model is given the same first 3 lines as a prompt and asked to generate the continuation. All generations use temperature=0.8, top- p =0.9, and repetition penalty=1.2. By holding the prompt and decoding parameters constant, any changes in output style can be attributed directly to the effect of additional training on the 143-sonnet corpus.

Epoch 0 (no sonnet training — pretrained GPT-2 only):

Those lips that Love’s own hand did make
 Breathed forth the sound that said “I hate”
 To me that languished for her sake;
 as I was still again slain in my anger.

At epoch 0, the model has seen no Shakespeare at all. It generates modern conversational English (“slain in my anger”) that reads like contemporary prose fiction. There is no attempt at verse structure, no archaic vocabulary, and no awareness that the input is poetry.

Epoch 5 (early adaptation):

Those lips that Love’s own hand did make
Breathed forth the sound that said “I hate”
To me that languished for her sake;
but now I beheld thee, and thou thy tongue
doth my eyes sing. You see not what a beauty
therein lies, for all the world to tell.

After 5 epochs, the first signs of Shakespearean influence appear. The second-person pronouns “thee” and “thou” emerge, along with the archaic verb construction “doth my eyes sing.” However, the model mixes archaic and modern registers within the same passage.

Epoch 15 (mature adaptation):

Those lips that Love’s own hand did make
Breathed forth the sound that said “I hate”
To me that languished for her sake;
and so I thought, dear sweet friend: but after
she had loved thee thou wert poison’d still,
and beauty’s rose hath lost its crimson hue.

By epoch 15, the model produces text that is recognizably Shakespearean in vocabulary, theme, and tone. The verb “wert” is Early Modern English, “beauty’s rose hath lost its crimson hue” is thematically aligned with Shakespeare’s recurring meditation on the transience of beauty, and “poison’d” uses the contracted past participle common in Elizabethan verse.

Epoch 19 (final model):

Those lips that Love’s own hand did make
Breathed forth the sound that said “I hate”
To me that languished for her sake;
and I despised she, too. But when my sweet love
had blushed with lust after thee and thy name,
then dost thou know what beauty takes away.

The final model produces its most consistently Shakespearean output, but “I despised she” should be “I despised her”—revealing that the model has learned Shakespearean *vocabulary* but not Early Modern English *grammar*. This distinction between surface-level lexical adaptation and deeper syntactic adaptation is explored further in Appendix B.

B Annotated Sonnet Analysis

To understand precisely where our model succeeds and fails, we annotate the epoch 19 generation line by line, evaluating four dimensions of sonnet quality: **vocabulary** (use of archaic/Shakespearean words), **grammar** (correctness of Early Modern English syntax), **theme** (coherence with the prompt and with Shakespearean concerns), and **structure** (adherence to the 14-line, ABAB CDCD EFEF GG format with iambic pentameter).

Line 1:	Those lips that Love’s own hand did make	[prompt]
Line 2:	Breathed forth the sound that said “I hate”	[prompt]
Line 3:	To me that languished for her sake;	[prompt]
Line 4:	and I despised she, too. But when my sweet love	[generated]
Line 5:	had blushed with lust after thee and thy name,	[generated]
Line 6:	then dost thou know what beauty takes away.	[generated]

Vocabulary analysis. The generated lines contain five archaic tokens: “thee” (objective second person singular), “thy” (possessive second person singular), “thou” (subjective second person singular), “dost” (second person singular present tense of “do”), and “sweet love” (a Shakespearean term of endearment appearing in Sonnets 56, 63, 79, and others). The model correctly deploys the full paradigm of second-person pronouns (thou/thee/thy), which is notable because these forms were already archaic by the time of GPT-2’s training data and appear primarily in quoted poetry and religious texts. Fine-tuning on 143 sonnets was sufficient to significantly upweight these tokens.

Grammar analysis. Line 4 contains a clear grammatical error: “I despised she” should be “I despised her,” since “she” is a subjective pronoun and cannot serve as the object of “despised.” This error is revealing. The model has learned to use the *unfamiliar* second-person archaic forms correctly (“after thee,” “thy name,” “dost thou”) but makes an error on the *familiar* third-person form. This suggests the model memorizes archaic pronoun patterns from the training sonnets but does not internalize the underlying case grammar—it produces “she” because “she” appears frequently near first-person subjects in the training data, not because it understands pronoun case assignment.

Thematic analysis. The generated text maintains strong thematic coherence with both the prompt and Shakespeare’s characteristic concerns. The prompt establishes a scene of rejected love (“I hate”), and the continuation develops this into a meditation on beauty and desire (“blushed with lust,” “what beauty takes away”). The juxtaposition of “love” and “lust” in line 5 echoes Shakespeare’s frequent exploration of the tension between idealized and physical love, particularly in the Dark Lady sequence (Sonnets 127–154) where these sonnets originate. The final line resonates with one of Shakespeare’s most persistent themes: the destructive passage of time and its effect on beauty.

Structural analysis. The model fails on all four structural dimensions of a Shakespearean sonnet. First, it generates only 3 lines beyond the prompt, far short of the required 11 additional lines (for a total of 14). Second, there is no rhyme scheme—the line-ending words “love,” “name,” and “away” do not rhyme with each other or with the prompt’s “make”/“hate”/“sake.” Third, iambic pentameter is not maintained: line 4 has approximately 13 syllables rather than the 10 required by pentameter. Fourth, the model does not produce a concluding couplet, which is the defining structural feature of the Shakespearean sonnet form.

Synthesis: what 143 examples can and cannot teach. This analysis uncovers a clear hierarchy of what can be learned from a small fine-tuning corpus. **Token-level features** (vocabulary choice, archaic word forms) are learned effectively because they require only shifting the probability distribution over individual tokens—each of the 143 training sonnets provides hundreds of token-level training signals. **Local syntactic patterns** (“dost thou,” “hath lost”) are partially learned because they involve short-range dependencies that attention mechanisms can capture, though errors like “despised she” show the limits. **Global structural constraints** (14-line length, ABAB rhyme scheme, iambic pentameter) are not learned at all because they require the model to plan over the entire sequence length and coordinate distant positions—a task that likely requires either many more training examples or an explicit structural loss function that rewards rhyme and meter compliance.

C Comprehensive Results Summary

Table 5 consolidates all experimental results into a single reference, enabling direct comparison across every configuration we tested. Each row is assigned an ID (S for sentiment, P for paraphrase, G for generation) for easy cross-referencing with the discussion in the main text.

Cross-experiment observations. First, comparing P2 and P3 isolates the effect of the LR scheduler. The peak accuracy is virtually identical (0.898 vs. 0.897), but P3’s higher final training loss (0.097 vs. 0.074) indicates less memorization—a textbook illustration of why regularization improves reliability without necessarily improving peak performance.

Second, comparing P4 and P5 shows the marginal return of doubling LoRA rank. The 0.009 accuracy improvement represents roughly 18% of the gap to full fine-tuning. Extrapolating linearly suggests $r \approx 90$ to close the gap, though the relationship is likely sublinear. An alternative approach: apply LoRA to all linear layers at $r = 8$, testing whether module coverage matters more than per-module capacity.

ID	Configuration	SST Dev Acc	Para. Dev Acc	Sonnet chrF
<i>Sentiment Analysis</i>				
S1	Frozen backbone, lr= 10^{-3}	0.454	—	—
S2	Full fine-tune, lr= 10^{-5}	0.497	—	—
<i>Paraphrase Detection</i>				
P1	Baseline, 1 ep (milestone)	—	0.870	—
P2	Baseline, 7 ep	—	0.898	—
P3	+ LR sched + clip + WD, 7 ep	—	0.897	—
P4	LoRA $r=8$, 5 ep	—	0.848	—
P5	LoRA $r=16$, 5 ep	—	0.857	—
<i>Sonnet Generation</i>				
G1	Baseline (10 ep, temp=1.2)	—	—	41.10
G2	Enhanced, best (20 ep, temp=1.0)	—	—	42.11
G3	temp=0.8, $\theta=1.0$	—	—	40.54
G4	temp=1.2, $\theta=1.0$	—	—	42.07
G5	temp=0.8, $\theta=1.2$	—	—	39.50
G6	temp=1.0, $\theta=1.2$	—	—	38.75
G7	temp=1.2, $\theta=1.2$	—	—	38.04
G8	temp=0.9, $\theta=1.2$, 14L	—	—	36.68

Table 5: Master results table across all configurations. Bold indicates best result per task.

Third, the generation sweep (G3–G8) shows that inference-time parameters produce a 5.4-point chrF range (36.68 to 42.11), which far exceeds the 1.0-point improvement from training enhancements (G1 vs. G2, 41.10 to 42.11). More importantly, the sweep exposes a metric–quality misalignment: every configuration with repetition penalty ($\theta=1.2$) scores *lower* than the same temperature without penalty, despite producing qualitatively better poetry. This suggests that chrF, which measures character n -gram overlap, rewards the repetitive patterns that degenerate text generation produces. This observation aligns with the broader finding in the generation literature [8] that the quality of generated text depends as much on how you sample as on how you train—but adds the nuance that automated metrics may not correctly rank these sampling strategies.

D Full Training Logs

The following tables record the complete training trajectory for every experiment. We include these logs for reproducibility and to support the training dynamics analysis in Section 6.

D.1 Paraphrase Detection — Baseline (7 epochs)

The baseline uses a constant learning rate of 5×10^{-5} with no scheduler, no weight decay, and no gradient clipping. The key pattern to observe is the divergence between train loss and dev accuracy after epoch 2: train loss continues to drop monotonically while dev accuracy plateaus and begins to oscillate. This is the overfitting pattern that motivated our scheduler extension.

Epoch	Train Loss	Dev Acc	Best?
0	0.363	0.870	✓
1	0.263	0.884	✓
2	0.204	0.894	✓
3	0.157	0.891	
4	0.119	0.898	✓
5	0.093	0.894	
6	0.074	0.896	

Table 6: Baseline paraphrase training. Config: lr= 5×10^{-5} , batch=64, full fine-tuning, A100 GPU. Note the oscillation in dev accuracy after epoch 2 despite steadily decreasing train loss.

D.2 Paraphrase Detection — LR Scheduler (7 epochs)

The scheduler adds linear warmup over the first 10% of steps (3,095 steps), followed by cosine decay to zero. Comparing with the baseline, notice two key differences: (1) dev accuracy increases steadily through epoch 4 rather than oscillating, and (2) the final training loss (0.097) is 31% higher than the baseline’s (0.074), indicating less memorization. The LR column shows the decay in action—by epoch 4 the LR has dropped to 23% of peak, and by epoch 6 it reaches zero.

Epoch	Train Loss	Dev Acc	LR	Best?
0	0.567	0.833	4.97×10^{-5}	✓
1	0.306	0.867	4.49×10^{-5}	✓
2	0.246	0.880	3.53×10^{-5}	✓
3	0.193	0.890	2.31×10^{-5}	✓
4	0.146	0.897	1.14×10^{-5}	✓
5	0.112	0.896	3.04×10^{-6}	
6	0.097	0.897	0.00	

Table 7: LR scheduler paraphrase training. The steady dev accuracy improvement through epoch 4 contrasts sharply with the baseline’s oscillation.

D.3 Paraphrase Detection — LoRA $r=8$ (5 epochs)

With LoRA $r=8$, only 294,912 parameters (0.24%) are trainable. The learning rate is 1×10^{-4} —double the full fine-tuning rate—because fewer trainable parameters means less risk of catastrophic forgetting. The initial training loss (0.846) is much higher than the full fine-tuning baseline (0.363) because the frozen backbone provides a weaker starting point for the cloze task. Despite this, the model converges to 0.848 dev accuracy by epoch 3, demonstrating that even a rank-8 low-rank update captures substantial task-relevant information.

Epoch	Train Loss	Dev Acc	LR	Best?
0	0.846	0.814	9.70×10^{-5}	✓
1	0.390	0.837	7.50×10^{-5}	✓
2	0.363	0.842	4.13×10^{-5}	✓
3	0.349	0.848	1.17×10^{-5}	✓
4	0.342	0.847	0.00	

Table 8: LoRA $r=8$ paraphrase training. 295K trainable parameters (0.24%). The high initial loss reflects the frozen backbone’s limited starting capacity for the cloze task.

D.4 Paraphrase Detection — LoRA $r=16$ (5 epochs)

Doubling the rank to 16 doubles the trainable parameter count to 589,824 (0.47%). Comparing with $r=8$, the initial loss is slightly lower (0.775 vs. 0.846), indicating that higher-capacity adapters begin learning useful representations faster. The final dev accuracy (0.857) represents a 0.9 percentage point improvement over $r=8$, and notably the model continues improving through the final epoch (unlike $r=8$ which peaked at epoch 3 and declined at epoch 4). This suggests that $r=16$ may benefit from additional training epochs, a direction we did not explore due to time constraints.

Epoch	Train Loss	Dev Acc	LR	Best?
0	0.775	0.827	9.70×10^{-5}	✓
1	0.371	0.845	7.50×10^{-5}	✓
2	0.344	0.852	4.13×10^{-5}	✓
3	0.329	0.855	1.17×10^{-5}	✓
4	0.321	0.857	0.00	✓

Table 9: LoRA $r=16$ paraphrase training. 590K trainable parameters (0.47%). Unlike $r=8$, accuracy improves through all 5 epochs, suggesting the model could benefit from longer training.

D.5 Sonnet Generation — Enhanced (20 epochs)

The sonnet training dataset contains only 143 examples, making overfitting a significant concern. We use cosine decay scheduling, gradient clipping at 1.0, and weight decay of 0.01 to mitigate this. The loss decreases steadily from 4.964 to 3.010 over 17 epochs, then increases slightly at epochs 18–19 (3.022 and 3.021 respectively). This slight uptick suggests the onset of overfitting—the model has effectively memorized the 143-sonnet corpus. The early stopping mechanism saves the epoch 17 checkpoint.

An important note on interpreting these loss values: a cross-entropy loss of 3.010 corresponds to a perplexity of $e^{3.010} \approx 20.3$, meaning the model is as “surprised” by the next token in a Shakespeare sonnet as if it were choosing uniformly among 20 candidates. For reference, GPT-2’s perplexity on general English text is approximately 18–25, so a perplexity of 20.3 on Shakespearean poetry represents remarkably strong adaptation given only 143 training examples.

Epoch	Train Loss	LR	Best?
0	4.964	2.50×10^{-5}	✓
2	4.063	4.96×10^{-5}	✓
5	3.600	4.42×10^{-5}	✓
8	3.369	3.36×10^{-5}	✓
11	3.182	2.07×10^{-5}	✓
14	3.078	8.93×10^{-6}	✓
17	3.010	1.51×10^{-6}	✓
18	3.022	3.80×10^{-7}	
19	3.021	0.00	

Table 10: Enhanced sonnet training over 20 epochs. Loss converges at epoch 17 (3.010, perplexity ≈ 20.3). The slight increase at epochs 18–19 indicates the onset of overfitting on the 143-sonnet corpus.

E LoRA Implementation Details

Low-Rank Adaptation (LoRA) is the most technically involved extension in our project, so we document the implementation in detail here for readers who may wish to reproduce or build upon this work.

Core mechanism. The central idea of LoRA is that the weight updates $\Delta \mathbf{W}$ learned during fine-tuning are approximately low-rank—they can be well-approximated by the product of two much smaller matrices. Instead of updating the full weight matrix $\mathbf{W} \in \mathbb{R}^{768 \times 768}$ (589,824 parameters), we freeze $\mathbf{W}$ and train two small matrices $\mathbf{A} \in \mathbb{R}^{768 \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times 768}$, where $r \ll 768$. The modified forward pass becomes:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \frac{\alpha}{r} \mathbf{B}\mathbf{A}\mathbf{x}$$

The frozen path computes the original linear transformation without storing gradients (saving memory), while the LoRA path computes the low-rank bypass with full gradient tracking.

Initialization strategy. A critical design choice is how to initialize $\mathbf{A}$ and $\mathbf{B}$. We follow the original LoRA paper [11]: $\mathbf{A}$ is initialized with Kaiming uniform (the standard PyTorch default for linear layers), and $\mathbf{B}$ is initialized to all zeros. This means $\Delta \mathbf{W} = \mathbf{B}\mathbf{A} = \mathbf{0}$ at the start of training, so the model’s output is *exactly* identical to pretrained GPT-2 before any training occurs. This is important because it means fine-tuning starts from the pretrained solution rather than from a perturbed version of it, avoiding any initial degradation in performance.

Target modules. We apply LoRA to the **query** and **value** projections in the multi-head attention mechanism of all 12 GPT-2 layers. We do *not* adapt the key projections or the feed-forward network layers, following the LoRA paper’s ablation study which found that adapting Q and V yields the best accuracy-per-parameter tradeoff on most tasks.

Scaling factor. The scaling factor α/r controls the magnitude of the LoRA update relative to pretrained weights. We set $\alpha = 2r$ for both rank settings ($\alpha = 16$ for $r = 8$, and $\alpha = 32$ for $r = 16$),

yielding a constant scaling factor of 2.0. Keeping this constant across ranks ensures that the only variable in our comparison is the adapter capacity (rank), not the learning signal magnitude.

Parameter count. Each adapted linear layer adds $768 \times r + r \times 768 = 1536r$ parameters. With 12 transformer layers and 2 adapted modules per layer (query and value), the total LoRA parameters are $24 \times 1536r = 36,864r$. For $r = 8$: 294,912 trainable parameters (0.24% of 124M total); for $r = 16$: 589,824 trainable parameters (0.47%). All other parameters remain frozen with `requires_grad = False`, meaning no gradient or optimizer state is stored for them—the primary source of LoRA’s memory and speed advantages.

F Hyperparameter Summary

Table 11 lists the exact hyperparameter settings for every experiment, enabling full reproducibility. All experiments use our custom AdamW optimizer implementation with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-6}$.

A few rationale notes on our choices. The $100\times$ lower learning rate for full fine-tuning (10^{-5}) vs. frozen backbone (10^{-3}) reflects different optimization landscapes: when the backbone is frozen, only the randomly initialized classification head trains, requiring a high learning rate to move these random weights quickly toward a good solution. When the full model trains, the pretrained weights are already near a good solution, and a high learning rate would cause catastrophic forgetting. The LoRA learning rate (10^{-4}) sits between these extremes: the LoRA parameters are randomly initialized (like the classification head), but they interact with frozen pretrained weights, so a moderate rate is appropriate.

Weight decay (0.01) is applied only to the enhanced experiments because it complements the cosine decay schedule. The baseline experiments use no weight decay to maintain a clean comparison with the handout’s reference implementation. Gradient clipping at $\tau = 1.0$ provides insurance against the occasional large gradient that can occur when fine-tuning on small datasets (143 sonnets) or with cross-entropy loss over 50,257 vocabulary items.

Experiment	LR	Batch	Epochs	WD	Clip	GPU
Sentiment (frozen)	10^{-3}	64	10	0.0	—	A100
Sentiment (full)	10^{-5}	64	10	0.0	—	A100
Paraphrase (baseline)	5×10^{-5}	64	7	0.0	—	A100
Paraphrase (scheduler)	5×10^{-5}	64	7	0.01	1.0	A100
Paraphrase (LoRA $r=8$)	10^{-4}	64	5	0.01	1.0	A100
Paraphrase (LoRA $r=16$)	10^{-4}	64	5	0.01	1.0	A100
Sonnet (baseline)	10^{-5}	8	10	0.0	—	A100
Sonnet (enhanced)	5×10^{-5}	8	20	0.01	1.0	A100

Table 11: Complete hyperparameter settings. WD = weight decay, Clip = gradient norm threshold. All use AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$) on NVIDIA A100 (40GB).