

GPT-2 Implementation and Speedup

Stanford CS224N Default Project

Qi Wu

Department of Computer Science
Stanford University
q1wu@stanford.edu

Yuening Huang

Department of Computer Science
Stanford University
yueningh@stanford.edu

Abstract

Adapting large language models for specific tasks via full fine-tuning is computationally expensive. We construct a GPT-2 architecture from scratch, optimizing its performance on cloze-style paraphrase detection and sonnet generation. While standard parameter-efficient methods reduce compute, they often still incur memory overhead[1]. To mitigate computational bottlenecks and enhance performance, we apply Representation Finetuning (LoREFT)—a parameter-efficient approach that adapts model representations while reducing the trainable parameter count[2]. Upon applying LoREFT, we observe notable improvements in memory utilization and training speed, showing its efficacy for resource-constrained model adaptation.

1 Key Information to include

- Mentor: Sarah Chen
- External Collaborators (if you have any): N/A
- Sharing project: N/A
- Late Days (if you are using any): N/A

2 Introduction

Large language models require fine-tuning to achieve optimal performance on specialized tasks. Traditionally, this involves full fine-tuning, a process that updates every model parameter and incurs massive compute and memory costs. While standard parameter-efficient fine-tuning (PEFT) methods reduce this burden, many still introduce memory overhead or latency during training by appending new weights or layers[1]. This project addresses these inefficiencies using LoREFT, a method that adapts a model's hidden representations instead of its weights. To evaluate this approach, we construct a GPT-2 architecture from the ground up and adapt it for two distinct tasks: cloze-style paraphrase detection and sonnet generation[3]. Because LoREFT intervenes directly on representations, it minimizes the trainable parameter count[2]. Our experiments demonstrate that this technique lowers memory overhead and shortens training time, proving its viability for environments with limited computational capacity.

3 Related Work

Parameter-efficient finetuning methods adapt large language models without updating all model weights. Popular approaches such as adapters and LoRA reduce training and storage cost by learning a small number of additional parameters while keeping most pretrained weights fixed[4, 5]. However, these methods still assume that adaptation should happen in weight space.

Representation Finetuning (ReFT) explores a different direction: instead of modifying weights, it learns interventions directly on hidden representations[2]. This idea is motivated by prior work in

interpretability and activation steering, which shows that hidden states contain meaningful semantic information and that editing activations can causally influence model behavior[6]. Building on this view, Wu et al. propose ReFT as a framework for adapting frozen language models by inserting learned interventions at selected layers and token positions during the forward pass[2]. The main ReFT variant

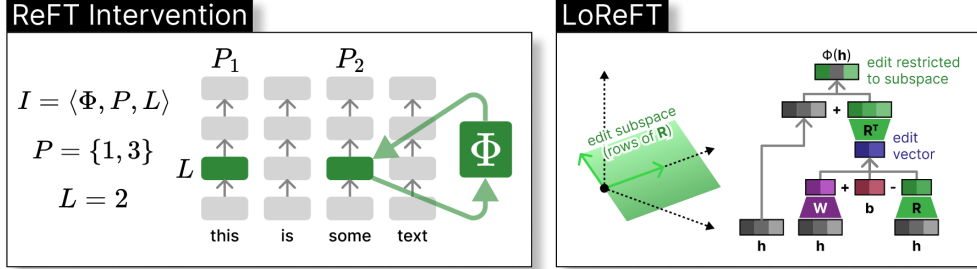


Figure 1: Illustration of ReFT. Left: an intervention Φ is applied to hidden representations at positions P in layer L . Right: LoReFT restricts the edit to a learned low-rank linear subspace spanned by the rows of R [2].

studied in this work is Low-rank Linear Subspace ReFT (LoReFT), which constrains representation edits to a learned low-rank subspace of the residual stream. Compared with weight-based PEFT methods such as LoRA, LoReFT reports a strong efficiency-performance tradeoff, using substantially fewer trainable parameters while remaining competitive on tasks including commonsense reasoning, instruction following, and GLUE[2]. This makes ReFT especially relevant to our project: it offers a natural alternative to standard PEFT for GPT-2, where hidden states are easily accessible and can be directly edited without changing the backbone weights.

4 Approach

4.1 GPT-2 & Adam Optimizer Implementation

GPT-2 Model. Our model is a 12-layer decoder-only transformer. Inputs are BPE-tokenized and represented by summing 768-dimensional token and positional embeddings, supporting a maximum context length of 1024. Each transformer layer uses masked multi-head self-attention to prevent future-token attendance during training, followed by layer normalization, skip connections, and a ReLU-activated multi-layer perceptron (MLP)[3]. The attention and MLP operations are defined as

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

For regularization, we apply dropout to the embeddings, attention layers, and MLPs.

Adam Optimizer. We implement the step function, which computes adaptive learning rates by estimating the first moment (m_t) and second moment (v_t) of the gradients with a bias correction.

Baselines. We validate correctness by comparing our model’s outputs to the official pre-trained model weights from HuggingFace. Our baseline is passing the provided `sanity_check.py`[3].

4.2 Sentiment Analysis with GPT-2

To verify the learning capabilities of our base GPT-2 implementation before introducing parameter-efficient optimizations, we evaluate it on sentiment analysis.

Model Formulation. We implement a GPT-2 sentiment classifier for SST (5-way) and CFIMDB (binary) using a pooled *last-token* sentence representation. Given tokenized input $x_{1:T}$ with attention mask m , GPT-2 produces hidden states $H \in \mathbb{R}^{T \times d}$. We take

$$t^* = \max\{t \mid m_t = 1\}$$

and use $h = H_{t^*}$ as the sentence embedding, then predict logits

$$z = W \text{ Dropout}(h) + b \in \mathbb{R}^C$$

(with $C \in \{5, 2\}$). We train with Adam to minimize cross-entropy, $\mathcal{L} = \frac{1}{B} \sum_{i=1}^B \text{CE}(z^{(i)}, y^{(i)})$, selecting checkpoints by dev accuracy.

Fine-tuning Regimes (baselines). (1) **Last-linear-layer:** freeze all GPT-2 parameters and train only the dropout+linear head. This provides a fast, stable baseline that tests the quality of pretrained representations. (2) **Full-model:** update all GPT-2 parameters and the classifier head jointly. The default project handout provides reference dev accuracies for these baselines (See Table 5) [3].

4.3 Computational Optimization: Low-rank Linear Subspace ReFT

Intervention Mechanism. To mitigate the computational bottlenecks observed during full-model fine-tuning, we integrate LoREFT for our primary downstream tasks. For a given hidden state $h \in \mathbb{R}^d$ at a targeted sequence position, we compute the intervened hidden state h' using the following equation:

$$h' = h + R^T(W h + b - R h)$$

where W and b are the learnable weight and bias parameters of the intervention, and R defines the low-rank projection matrix. During training, we freeze all original GPT-2 parameters and exclusively update W and b , avoiding the memory-intensive process of backpropagating gradients through the entire network architecture.

Hyperparameter Configuration. The capacity of our LoREFT implementation is controlled by three tunable parameters: **Rank** (the bottleneck dimension of the subspace projection), **LoREFT Layers** (the specific transformer layers targeted for intervention), and **Window Size** (K), which restricts the intervention to only the last K tokens of the input sequence to leverage the most comprehensive contextual information while strictly limiting activation caching.

Optimization & Regularization Strategy. To stabilize the randomly initialized LoREFT parameters (W and b), we apply several optimizations. We use the AdamW optimizer with grouped learning rates, assigning the LoREFT parameters a 10x higher learning rate than the pre-trained classification head, while explicitly setting their weight decay to 0.0 to prevent the L_2 penalty from slowing convergence. To manage these aggressive updates, we enforce global gradient clipping (max norm = 1.0) and utilize a learning rate schedule with a 15 percent linear warmup followed by a cosine decay.

4.4 Downstream Applications: Paraphrase Detection & Sonnet Generation

With the LoREFT architecture established, we adapt our model to two distinct generation-based tasks.

Cloze-Style Paraphrase Detection

We adapt our pre-trained GPT-2 for paraphrase detection on the Quora dataset by formulating the task as a cloze-style text generation problem[3]. For each question pair, we construct the prompt: "Is 's1' a paraphrase of 's2'?". We pass this sequence through the model, extract the final token's contextualized hidden state (applying our LoREFT interventions), and project it via weight-tied embeddings to generate a probability distribution. We optimize using **cross-entropy loss** to maximize the probability of the target word "yes" or "no". Given the large size of the Quora paraphrase dataset, full fine-tuning is slow and memory-intensive; applying LoREFT accelerates training iterations by eliminating the need for full-model activation caching. Our **baseline** is to fine-tune the ParaphraseGPT model to achieve a development set accuracy of at least 0.862.

Sonnet Generation

For the sonnet generation task, the model is given the first three lines of a held-out Shakespeare sonnet and must generate the remaining continuation. Because the sonnet dataset is relatively small, full fine-tuning easily leads to overfitting; LoREFT acts as a structural regularizer by restricting parameter updates to a low-rank subspace, preventing the model from simply memorizing the limited training data. We evaluate model outputs using **chrF**[7], following the default CS224N project setup[3]. Because sonnet generation depends strongly on local stylistic patterns, morphology, and surface form, chrF (*character n-gram F-score*) provides a robust character-level metric to measure how closely the generated continuation matches Shakespeare's original wording and style. Our **baseline** is provided by project handout with chrF score at least 41.414[3].

4.5 Originality & External Code

We utilize the `cs224n_gpt` scaffolding provided by the course staff[3]. Our implementations include the core GPT-2 architecture, the Adam Optimizer update logic, the LoREFT structure, and downstream adaptations (the sentiment classification head, the cloze-style paraphrase forward pass, and the weight-tied `hidden_state_to_token` projection).

5 Experiments

5.1 Sentiment Analysis with GPT-2

Data. For sentiment analysis, we evaluate on two datasets: the Stanford Sentiment Treebank (SST) and CFIMDB. SST contains 11,855 single sentences from movie reviews, yielding 215,154 unique phrases[8]. Each phrase is labeled on a 5-way scale (negative, somewhat negative, neutral, somewhat positive, or positive). We use the provided splits: train (8,544), dev (1,101), and test (2,210)[3]. CFIMDB consists of 2,434 highly polar movie reviews with binary sentiment labels (negative/positive), split into train (1,701), dev (245), and test (488)[3].

Evaluation method. For sentiment classification, we report **dev accuracy** as the primary metric.

Experimental Details We train for 10 epochs with Adam optimizer. We compare two baseline fine-tuning regimes: a last-linear-layer approach (freezing all GPT-2 parameters and training only the dropout+linear head) and a full-model approach (updating all GPT-2 parameters jointly). Checkpoints are selected based on the highest dev accuracy (see Appendix 4 for full hyperparameters).

Results. Relative to the baselines (SST: 0.462/0.513; CFIMDB: 0.861/0.976 for last-linear-layer/full-model) our best dev accuracies are: **SST** 0.467 (last-linear-layer) and 0.516 (full-model), and **CFIMDB** 0.869 (last-linear-layer) and 0.980 (full-model) (Table 5). See appendix for details A.

5.2 Cloze-Style Paraphrase Detection

Data. We use the Quora dataset provided by the CS224N course staff[3][9], consisting of question pairs with binary paraphrase labels. The splits are: train (141,506), dev (20,215), and test (40,431).

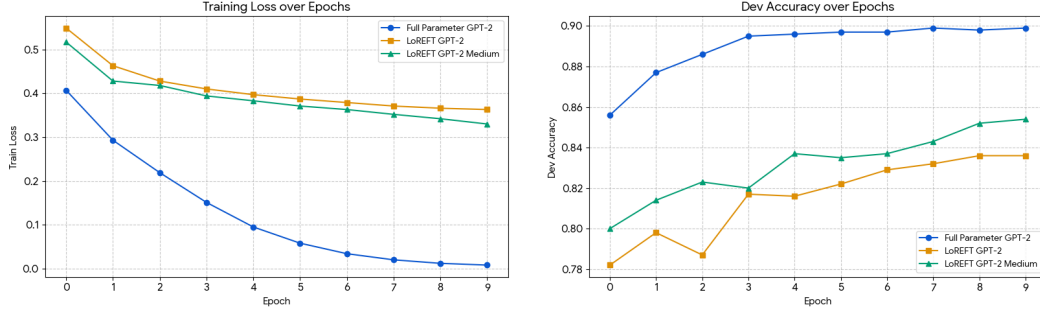
Evaluation method. Our primary evaluation metric is **accuracy** on the dev and test sets. We measure the model’s ability to correctly assign a higher probability to the correct target word (“yes” or “no”) based on the ground truth label. Test set accuracy is evaluated via the official course leaderboard.

Experimental Details We fine-tune our model on the Quora training split using a cloze-style prompt formulation (“Is ‘{s1}’ a paraphrase of ‘{s2}’?”). We optimize cross-entropy loss over the logits for the “yes” (BPE 8505) and “no” (BPE 3919) tokens. We evaluate the model under two distinct regimes: standard full-parameter fine-tuning, and our implemented LoREFT extension. (See Appendix A for full hyperparameters).

Results. Figure 2 shows dev acc and training loss across first 10 epochs. Table 1 compares our model’s dev accuracy against the target baseline (a dev set accuracy of 0.862). Our LoREFT fine-tuning achieved an dev accuracy of 0.863. Our full-parameter fine-tuning achieved a dev accuracy of 0.899 and a final test leaderboard score of 0.899.

Model	Mode	Target Baseline	Dev Acc.	Δ
GPT-2	Full fine-tuning	0.862	0.899	+0.037
GPT-2 Medium	LoREFT	0.862	0.863	+0.001

Table 1: Dev accuracy comparison against the target baseline for paraphrase detection on the Quora dataset.



(a) Training loss curves for different tuning approaches. (b) Development accuracy progression over 10 epochs.

Figure 2: Performance and convergence comparison of fine-tuning strategies for GPT-2 paraphrase detection. Panel (a) shows training loss for full parameter, standard LoREFT, and medium-model LoREFT. Panel (b) shows the corresponding development set accuracy for the same three regimes.

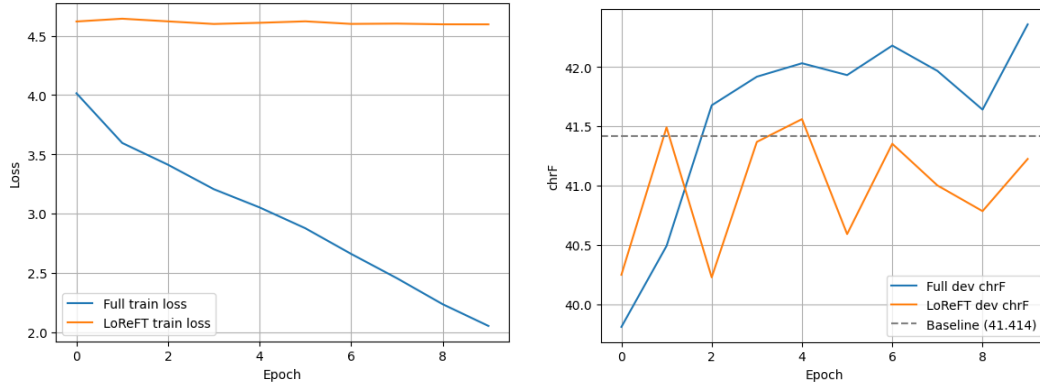
5.3 Sonnet Generation

Data. We use the Shakespeare sonnet dataset provided in the CS224N default final project[3]. The dataset contains 154 Shakespearean sonnets, where each example is a complete 14-line sonnet. Following the official setup, the task is conditional generation: the model is given the first three lines of a sonnet as context and must generate the remaining lines.

Evaluation method. Our primary evaluation metric is **chrF**, following the course handout[3]. chrF is a character-level n -gram F -score that measures overlap between a generated continuation and the gold sonnet continuation[7]. We report dev chrF during training and use the official course leaderboard for final test-set evaluation.

Experimental details. We fine-tune GPT-2 Large on the sonnet training data with a standard left-to-right language-modeling objective, i.e., token-level cross-entropy under teacher forcing. At evaluation time, we condition on the first three lines only and generate the remaining lines autoregressively. We compare two tuning regimes: (1) *full fine-tuning*, where all GPT-2 parameters are updated, and (2) *LoReFT*, where the GPT-2 backbone is frozen and only low-rank representation intervention parameters are trained. See appendix for detailed training configurations 7.

Results. Figure 3 shows dev chrF and training loss across epochs. Full fine-tuning performs best, reaching a peak dev chrF of about **42.3**. LoReFT achieves a best dev chrF of about **41.6**, which is competitive with the baseline chrF of 41.414 but remains below full fine-tuning. This result suggests a clear tradeoff: LoReFT is substantially more parameter-efficient and uses much less memory, but full-parameter adaptation still gives the strongest sonnet generation quality in our current setup.



(a) Sonnet training loss (full vs LoReFT).

(b) Sonnet chrF score (full vs LoReFT).

Figure 3: Training loss curves for sonnet generation.

6 Analysis

Model	Tuning Mode	Hardware	Batch	LR	Time / Epoch
GPT-2	Full fine-tuning	A100	128	1×10^{-4}	≈ 18 min
GPT-2	LoREFT	A100	128	1×10^{-4}	≈ 12 min
GPT-2 Medium	Full fine-tuning	H100	128	2×10^{-4}	≈ 24 min
GPT-2 Medium	LoREFT	H100	128	2×10^{-4}	≈ 15 min

Table 2: Comparison of training time per epoch between full-parameter fine-tuning and LoREFT interventions.

Method	Trainable Params	Checkpoint Size	GPU Memory	Best Dev chrF
Full fine-tuning	774,030,080	9.31 GB	~ 22.0 GB	42.3
LoReFT	122,976 (99.8% less)	3.12 GB	$\sim$ 11.6 GB	41.6

Table 3: Comparison of full fine-tuning and LoReFT on sonnet generation.

Throughput and Memory Improvements. Across both tasks, the primary advantage of LoREFT is its drastic reduction in computational and memory overhead. As highlighted by our sonnet results, LoREFT is highly parameter-efficient, utilizing only 122,976 trainable parameters—an approximately 99.8% reduction compared to full fine-tuning’s 774 million. This cuts the checkpoint size from 9.31 GB to 3.12 GB and peak GPU memory from ~ 22 GB to ~ 11.6 GB (Table 3).

This memory efficiency directly translates to faster training iterations, as gradients are computed only through the intervention modules rather than the full backbone. For the base GPT-2 model on paraphrase detection, LoREFT reduces training time from 18 minutes to 12 minutes per epoch (a 33% improvement). Similarly, for the GPT-2 Medium architecture, LoREFT completes an epoch in 15 minutes compared to the estimated 24 minutes for full fine-tuning (Table 2).

Training Dynamics: Cold Starts and Optimization Stability. While LoREFT yields significant throughput gains, freezing the backbone fundamentally alters the training dynamics, leading to a “cold start” and less stable optimization. In paraphrase detection, full-parameter training immediately leverages the pre-trained semantic space of GPT-2, achieving a dev accuracy of 0.856 at Epoch 0. In contrast, LoREFT starts much lower (0.782 for base, 0.800 for medium). This gap occurs because LoREFT introduces randomly initialized intervention matrices (W and b) into the hidden states. Before these weights converge, they effectively inject noise into the pre-trained representations, causing a temporary degradation in reasoning capabilities.

This instability is also evident in the sonnet generation task. While full fine-tuning steadily minimizes training loss and smoothly pushes dev chrF above the baseline, LoREFT’s training loss remains almost flat (around 4.5–4.6) across epochs, resulting in a more volatile dev chrF. This flat loss suggests that the targeted intervention modules receive only a weak or sparse gradient signal, making the cross-entropy objective harder to minimize than in full-model adaptation.

Performance Ceilings and Regularization. Despite these optimization hurdles, LoREFT successfully adapts to both tasks. In paraphrase detection, our targeted 10x learning rate multiplier and zero weight decay allow the low-rank matrices to rapidly recover, ultimately achieving a dev accuracy of 0.863 and crossing the 0.862 baseline. For sonnet generation, LoREFT reaches a best dev chrF of 41.6, surpassing the 41.414 baseline. Furthermore, on the small sonnet dataset, restricting updates to a low-rank subspace acts as a structural regularizer, effectively reducing the risk of the model simply memorizing the training set.

However, a clear performance ceiling remains. Full-parameter adaptation ultimately achieves the highest absolute metrics across the board (0.899 dev accuracy for paraphrase detection; 42.3 peak chrF for sonnets).

Summary. Ultimately, the tradeoff presented by LoREFT is highly favorable for resource-constrained environments. For tasks where maximum absolute performance (e.g., peak chrF or accuracy) is the sole objective, full-parameter adaptation remains preferable. However, LoREFT proves to be a robust

solution when compute or memory is limited, demonstrating that representation-space adaptation can preserve generation quality and beat established baselines while drastically reducing structural costs.

7 Conclusion

This project constructed a GPT-2 model from scratch and demonstrated that LoREFT is a highly efficient alternative to full fine-tuning. By restricting updates to a low-rank subspace, we reduced trainable parameters by 99.8% and halved peak memory usage. Despite these constraints, our LoREFT implementation successfully surpassed target baselines for both paraphrase detection (0.863 dev accuracy) and sonnet generation (41.6 chrF). We learned that representation-space adaptation is highly viable: it acts as a structural regularizer on small datasets and significantly accelerates training iterations on large ones.

However, this efficiency comes with a clear tradeoff. The primary limitation of LoREFT is a lower absolute performance ceiling compared to full-parameter updates, which achieved higher peak metrics (e.g., 0.899 dev accuracy for paraphrase detection). Additionally, freezing the backbone introduces optimization instability. The random initialization of intervention matrices creates a "cold start" effect that disrupts pre-trained representations during early epochs, resulting in flatter training loss curves and volatile optimization dynamics.

Future work offers clear paths to address these limitations. Exploring near-identity or data-driven initializations for the intervention matrices could mitigate the cold start effect and preserve the pre-trained semantic space early in training. Furthermore, dynamically allocating the intervention rank across different layers—rather than using a uniform rank—could optimize the balance between efficiency and expressivity. Finally, scaling this implementation to larger transformer architectures would further test the limits and necessity of representation-space adaptation.

Team Contributions

Yuening: Implemented the Adam optimizer and developed the core LoREFT mechanism. Yuening also managed the downstream adaptation and fine-tuning pipeline for the paraphrase detection task.

Qi: Constructed the foundational GPT-2 architecture and implemented the classification head for sentiment analysis. Qi also led the sonnet generation task and its associated full-parameter fine-tuning and LoREFT procedures.

References

- [1] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.
- [2] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Reft: Representation finetuning for language models, 2024.
- [3] Stanford University and CS 224N Course Staff. Cs 224n: Default final project: Build gpt-2. https://web.stanford.edu/class/cs224n/project/DFP_Instructions.pdf. Accessed: 2026-02-24.
- [4] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023.
- [5] 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.
- [6] Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. Finding alignments between interpretable causal variables and distributed neural representations, 2024.
- [7] Maja Popović. chrF: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, 2015.

- [8] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics, 2013.
- [9] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs. <https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs>, 2017.

A Appendix

A.1 GPT-2 sentiment classification training configuration

Setting	Dataset	Batch	LR	Dropout	Epoch time
Last-linear-layer	SST	64	1×10^{-3}	0.15	$\approx 11\text{s/epoch}$
Last-linear-layer	CFIMDB	64	1×10^{-3}	0.15	$\approx 22\text{s/epoch}$
Full-model	SST	64	3×10^{-5}	0.20	$\approx 34\text{s/epoch}$
Full-model	CFIMDB	64	3×10^{-5}	0.20	$\approx 62\text{s/epoch}$

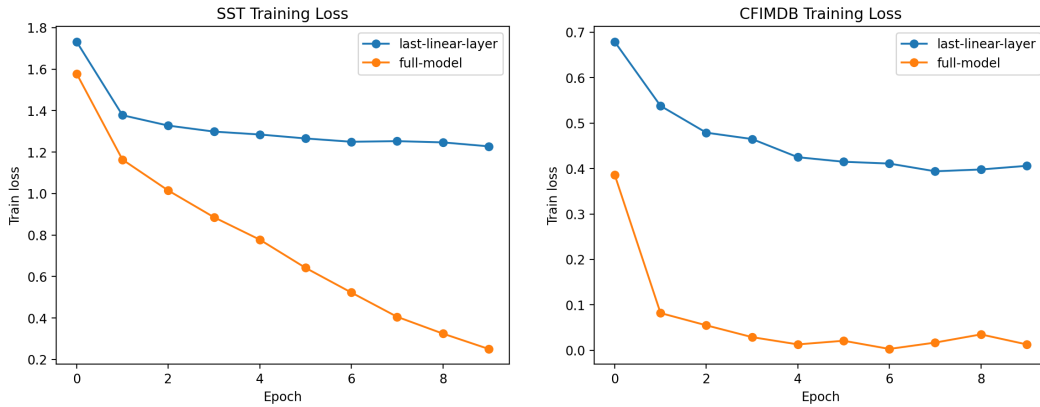
Table 4: Hyperparameters and observed training speed (GPU) for sentiment experiments. All runs use 10 epochs.

A.2 Results for GPT-2 sentiment classification

Dataset	Mode	Target Baseline	Ours	Δ
SST	Last Linear Layer	0.462	0.467	+0.005
SST	Full Model	0.513	0.516	+0.003
CFIMDB	Last Linear Layer	0.861	0.869	+0.008
CFIMDB	Full Model	0.976	0.980	+0.004

Table 5: Dev accuracy comparison against the target baselines. [3]

A.3 Loss curves for GPT-2 sentiment classification



(a) SST training loss (last-linear-layer vs full).

(b) CFIMDB training loss (last-linear-layer vs full).

Figure 4: Training loss curves for GPT-2 sentiment classification.

A.4 GPT-2 paraphrase detection training configuration

Configuration	Full fine-tuning	LoREFT (Small)	LoREFT (Medium)
Backbone model	GPT-2	GPT-2	GPT-2 Medium
Tuning mode	Full	LoREFT	LoREFT
Hardware (Modal)	A100	A100	H100
Batch size	128	128	128
Epochs	10	10	15
Learning rate	1×10^{-4}	1×10^{-4}	2×10^{-4}
LoREFT rank	–	32	32
LoREFT layers	–	{1, 4, 6, 8, 9, 10, 11}	{1, 4, 8, 12, 16, 18, 20, 21, 22, 23}
LoREFT window size	–	1	4
Time per epoch	≈ 18 min	≈ 12 min	≈ 15 min
Dev accuracy	0.899	0.836	0.863

Table 6: Training configurations, hardware setups, and results for paraphrase detection on the Quora dataset.

A.5 GPT-2 sonnet generation training configuration

Configuration	Full fine-tuning	LoReFT
Backbone model	GPT-2 Large	GPT-2 Large
Tuning mode	Full	LoReFT
Epochs	10	10
Learning rate	1×10^{-5}	1×10^{-5}
Temperature	1.2	1.0
Trainable parameters	774,030,080	122,976
LoReFT rank	–	8
LoReFT dropout	–	0.3
Gradient clipping	–	1.0

Table 7: Training configurations for sonnet generation under full fine-tuning and LoReFT.