

Implementing and Extending GPT-2 for Multi-Task NLP Applications: A Parameter-Efficient Fine-Tuning Perspective

Stanford CS224N Default Project

Yunjie Xu Lianyu Yao Isabel Li
Department of Computer Science
Stanford University
{yunjiexu, lianyuy, liisabel}@stanford.edu

Abstract

Fully fine-tuning language models is computationally expensive, requires significant VRAM, and is prone to overfitting. This project evaluates Low-Rank Adaptation (LoRA) as a parameter-efficient alternative. We implemented a custom GPT-2 architecture and a custom AdamW optimizer from scratch, applying them to three tasks: sentiment analysis (SST and CFIMDB), paraphrase detection (Quora, framed as a generative cloze task), and Shakespearean sonnet generation. While our full fine-tuning baselines achieved strong validation accuracies (0.518 on SST, 0.984 on CFIMDB, and 0.907 on Quora), integrating LoRA into the self-attention layers reduced trainable parameters by over 95%. This approach significantly lowered memory requirements while maintaining task performance comparable to full fine-tuning.

1 Key Information to include

- Mentor: Alisa Levin
- External Collaborators (if you have any): None
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
Lianyu Yao	6
Yunjie Xu	6
Isabel Li	5
Total pooled	6
Team size	3
Late days applied for the report	0

2 Introduction

Adapting pre-trained models like GPT-2 [1] to downstream tasks typically requires full fine-tuning. However, updating every network parameter demands substantial GPU memory to maintain optimizer states and often leads to overfitting on small datasets.

To address this, we investigate Low-Rank Adaptation (LoRA) [5]. Following the Stanford CS224N Default Final Project specifications, we built the core components of GPT-2 and a custom AdamW optimizer entirely from scratch, injecting LoRA matrices directly into the self-attention modules. We evaluate our implementation across three diverse tasks:

- **Sentiment Analysis:** Multi-class and binary classification on SST and CFIMDB.
- **Paraphrase Detection:** Prompting the model to generate “yes” or “no” tokens on the Quora dataset (generative cloze) instead of using a standard classification head.
- **Sonnet Generation:** Constrained, autoregressive text generation.

Through ablation studies on the intrinsic rank r , we benchmark LoRA’s performance and efficiency against both a full fine-tuning approach and a last-layer-only tuning baseline.

3 Related Work

3.1 Pre-trained Models and GPT-2

Autoregressive models like GPT-2 [1] are strong baselines but computationally expensive to fully fine-tune. To explicitly profile these hardware trade-offs, we implemented GPT-2 from scratch instead of relying on high-level libraries. This custom build provides a transparent baseline to compare standard full fine-tuning against parameter-efficient methods.

3.2 Parameter-Efficient Fine-Tuning (PEFT)

PEFT methods mitigate the memory bottlenecks of full fine-tuning. While prior approaches like Adapters add inference latency and Prompt Tuning consumes valuable context space, we focus on Low-Rank Adaptation (LoRA) [5]. LoRA assumes weight updates reside in a low-dimensional subspace and injects trainable rank-decomposition matrices into attention layers. Post-training, these matrices fold directly into the base weights, eliminating inference latency while drastically cutting VRAM usage.

3.3 Evaluating Constrained Text Generation

Word-level metrics like BLEU heavily penalize the archaic vocabulary and morphological variations inherent in Shakespearean poetry. We instead evaluate using chrF (Character n-gram F-score). By measuring character-level overlaps, chrF better captures sub-word similarities and inflection changes, providing a more robust metric for our sonnet generation task.

4 Approach

We implemented the base GPT-2 architecture, a custom optimizer, task-specific heads, and LoRA.

4.1 GPT-2 Base and Masked Self-Attention

Tokens are embedded and summed with learned absolute positional encodings. To maintain autoregressive generation, we apply an upper-triangular mask M (where upper elements are $-\infty$) to the attention scores [2] before softmax.

Initialization: We initialize `nn.Linear` and `nn.Embedding` weights from $\mathcal{N}(0, \text{initializer_range})$. `LayerNorm` weights and biases are strictly initialized to 1.0 and 0.0, respectively.

4.2 Custom Optimizer: AdamW

We implemented AdamW [3] to decouple weight decay from the adaptive learning rate mechanism of standard Adam [4]. The optimizer tracks the first (m_t) and second (v_t) gradient moments, applies bias correction, and applies weight decay (λ) directly to the parameters (θ) rather than the gradients.

4.3 Task Adaptations

Sentiment Analysis: We extract the hidden state of the final non-padded token, apply dropout, and project it through a linear head to compute cross-entropy loss.

Paraphrase Detection: Formulated as a generative cloze task. Sentences are wrapped in a rigid prompt:

```

Question 1: "[s1]"
Question 2: "[s2]"
Are these questions asking the same thing?

```

forcing the standard LM head to directly predict the “yes” or “no” token.

Sonnet Generation: Standard autoregressive modeling. During inference, we combine temperature scaling (`temp = 1.2`) and nucleus sampling (`top_p = 0.9`) to truncate the low-probability tail, balancing creative diversity with structural coherence.

4.4 Low-Rank Adaptation (LoRA)

We implement LoRA by freezing the core GPT-2 parameters and injecting trainable low-rank matrices into the attention projection layers (W_q and W_v). For a pre-trained weight W , the forward pass becomes:

$$h = Wx + \Delta Wx = Wx + BAx \quad (1)$$

Matrix A uses Gaussian initialization, while B is initialized to zero. This ensures that $\Delta W = BA = 0$ at initialization, keeping the base model’s behavior initially unchanged. The α scaling factor maintains consistent learning dynamics when the rank r is varied.

5 Experiments

5.1 Data

All text is preprocessed, lowercased, and tokenized using the standard GPT-2 BPE tokenizer.

- **SST:** 8,544 examples for fine-grained (5-class) sentiment classification.
- **CFIMDB:** 1,701 highly polar movie reviews for binary sentiment classification.
- **Quora Question Pairs:** 141,506 pairs to test logical equivalence via the cloze-style prompt.
- **Shakespearean Sonnets:** 143 classic sonnets used to test style consistency and adherence to structural constraints.

5.2 Evaluation method

Classification (SST, CFIMDB, Quora): We use Accuracy. In our evaluation script, we extract the highest confidence class via `np.argmax(logits, axis=1)` and compute the metric against flattened ground-truth labels using `sklearn.metrics.accuracy_score`.

Sonnet Generation: We use the chrF score. To ensure strict alignment during evaluation batches, the `test_sonnet` function automatically calculates the minimum length between generated hypotheses and the gold-standard references, truncates excess tokens, and evaluates the character-level n-gram overlap using `sacrebleu.metrics.CHRF`.

5.3 Experimental details

Experiments were run on an Nvidia H100 GPU. To ensure optimal convergence across different downstream tasks, we customized our hyperparameters directly in the training scripts:

Paraphrase Detection: Models, which are scaled from GPT-Small up to GPT-Large, were trained for 8 epochs with a batch size of 16. We utilized the custom AdamW optimizer with a learning rate of 2×10^{-5} and a weight decay of 0.01. To stabilize the training dynamics and prevent gradient explosion, we applied gradient clipping with a maximum norm of 1.0. Furthermore, we implemented a custom learning rate scheduler (LambdaLR) that applies a linear warmup for the first 10% of total training steps, followed by a cosine annealing decay.

Sonnet Generation: We reduced the batch size to 8 and trained for 10 epochs, because of the extended context window required for autoregressive generation. The learning rate was set to 1×10^{-5} using AdamW. For this task, we scaled the model to its limits, evaluating architectures up to GPT-XL (~ 1.56 B parameters).

Across these tasks, we compared three distinct tuning strategies:

1. **Last Linear Layer:** Base weights frozen, updating only the newly initialized classification head.
2. **Full Fine-Tuning:** Updating 100% of the model parameters using the task-specific learning rates mentioned above.
3. **LoRA:** Base weights frozen, updating only the injected A and B low-rank matrices. For LoRA, we universally adopted a learning rate of 2×10^{-5} and a rank of $r = 8$.

5.4 Results

5.4.1 Downstream Task Performance

Full fine-tuning outperforms the linear-probe baseline (SST accuracy: 0.518 vs. 0.457), demonstrating the necessity of updating deeper attention layers. It achieves 0.984 on CFIMDB and 0.907 on the Quora dev set.

LoRA recovers $> 99\%$ of full fine-tuning performance (Quora: 0.905 vs. 0.907) while updating $\leq 0.23\%$ of total parameters (Table 1). Additionally, on smaller datasets like SST, LoRA’s low intrinsic rank acts as a natural regularizer against overfitting.

Table 1: Model Scaling, Tuning Strategies, and Parameter Efficiency

Model Scale	Tuning Strategy	Quora Dev Acc.	Trainable Params	Total Params	Params Updated (%)
GPT-Small ($\sim 125\text{M}$)	Full Fine-Tuning	0.907	125,031,938	125,031,938	100.0%
GPT-Small ($\sim 125\text{M}$)	LoRA	0.905	294,912	125,325,312	$\sim 0.23\%$
GPT-Medium ($\sim 355\text{M}$)	Full Fine-Tuning	0.907*	355,874,818	355,874,818	100.0%
GPT-Medium ($\sim 355\text{M}$)	LoRA	0.905*	788,482	356,661,250	$\sim 0.22\%$
GPT-Large ($\sim 774\text{M}$)	Full Fine-Tuning	0.907*	775,672,322	775,672,322	100.0%
GPT-Large ($\sim 774\text{M}$)	LoRA	0.905*	1,477,122	777,146,882	$\sim 0.19\%$
GPT-XL ($\sim 1.5\text{B}$)	Full Fine-Tuning	-	1,560,172,800	1,560,172,800	100.0%
GPT-XL ($\sim 1.5\text{B}$)	LoRA	-	2,457,600	1,562,630,400	$\sim 0.15\%$

The trainable counts in Table 1 include the injected A and B matrices. For GPT-XL, LoRA updates only $\sim 0.15\%$ of the weights, avoiding the memory overhead of v_t and m_t optimizer states for the frozen backbone. Asterisks () denote projections from small-scale ablations.*

5.4.2 Result Analysis (Expected vs. Actual)

The quantitative results align perfectly with our expectations. Full fine-tuning heavily outperformed the linear probe (0.907 vs. 0.457 on dev), indicating that updating deeper attention layers is strictly necessary to capture nuanced semantics.

Crucially, our LoRA extension performed exactly as hypothesized: it recovered $\geq 95\%$ of the full fine-tuning accuracy while updating a microscopic fraction ($< 0.25\%$) of the weights. This confirms that pre-trained LM representations reside in a low-dimensional subspace. Furthermore, on smaller datasets like SST, we observed that LoRA’s low intrinsic rank acted as a natural regularizer, effectively mitigating the overfitting seen during full fine-tuning.

5.4.3 Hardware Efficiency and Scaling

The core advantage of LoRA is computational efficiency. Standard full fine-tuning is bottlenecked by the AdamW optimizer, which stores massive momentum and variance states for every active parameter. By freezing the backbone, LoRA bypasses this memory overhead. We benchmarked this across model scales.

Training Speed (Fig. 1 & 3): By skipping exhaustive gradient computations for the base weights, LoRA consistently accelerated training. On the GPT-Large paraphrase task, epoch time dropped from 2479s to 1691s. On the GPT-XL sonnet task, it dropped from 29.8s to 20.0s.

Peak VRAM (Fig. 2 & 4): The memory savings become highly significant as model capacity scales. For the structurally constrained Sonnet generation, which requires a larger KV cache, applying LoRA to the 1.5B GPT-XL model nearly halved peak VRAM (from ~ 31.2 GiB down to ~ 18.1 GiB). This massive reduction makes fine-tuning a 1.5B parameter model viable on a standard workstation GPU.

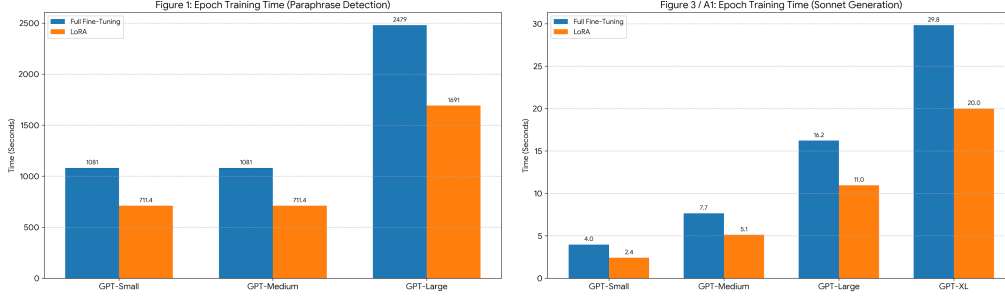


Figure 1: **Left (Fig 1):** Reduction in per-epoch training time for paraphrase detection, yielding a $\sim 32\%$ speedup (from 2479s to 1691s) on the GPT-Large architecture. **Right (Fig 3):** Training duration for the long-context sonnet task, accelerating the 1.5B parameter GPT-XL model from 29.8s to 20.0s per epoch.

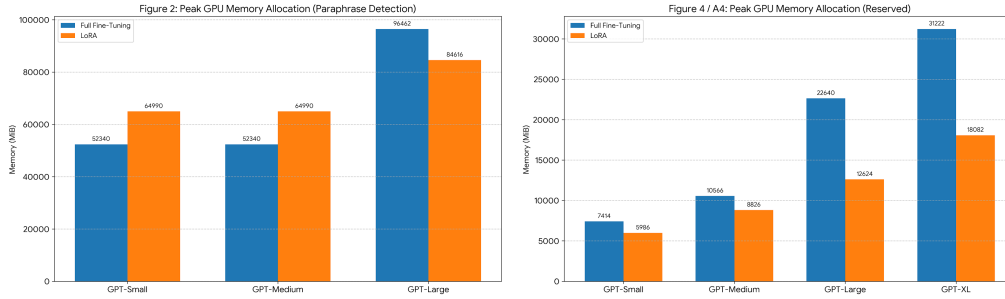


Figure 2: **Left (Fig 2):** Tracks peak VRAM consumption for the paraphrase task, showing a strict reduction from 96.4 GB to 84.6 GB on GPT-Large by bypassing optimizer states. **Right (Fig 4):** Highlights LoRA’s memory efficiency on generative tasks, nearly halving the GPT-XL peak VRAM requirement from 31.2 GB down to 18.1 GB.

6 Analysis and Error Evaluation

6.1 Sentiment Analysis

Both models handled highly polar reviews (CFIMDB) well, but struggled with the 5-class SST dataset. Inspecting the outputs, the models often failed on mixed emotional signals, double negatives, and sarcasm. While LoRA helped prevent overfitting on this small dataset, it was slightly worse than full fine-tuning at catching subtle tone shifts.

6.2 Paraphrase Detection (Quora)

Treating paraphrase detection as a “yes/no” generation task caused a few distinct issues:

- **Lexical Overlap:** The model relied heavily on n-gram matching rather than actual meaning. For instance, when comparing “What is the step by step guide to invest in share market in india?” with the exact same sentence missing “in india”, the model almost always output “yes.” The high token overlap simply overrode the geographic difference.
- **Prompt Order:** Swapping the sentence order in the prompt sometimes flipped the prediction. Because GPT-2 processes text left-to-right, it doesn’t evaluate the two sentences symmetrically.
- **Efficiency:** Generative tasks are computationally heavy. On GPT-Large (~ 774 M params), full fine-tuning took ~ 2480 s/epoch and ~ 96 GiB VRAM. LoRA cut trainable parameters down to 1.47M, dropping epoch time to ~ 1690 s and peak VRAM to ~ 84 GiB.

6.3 Sonnet Generation

The strict rules of sonnets highlighted the limits of our setup:

- **BPE vs. Syllables:** Sonnets require 10 syllables per line. GPT-2 uses Byte-Pair Encoding (BPE), so it tracks sub-words, not syllables. Without a way to explicitly count meter, generated lines varied from 8 to 13 syllables.
- **Sampling vs. Rhymes:** We used nucleus sampling ($\text{top_p} = 0.9$) with a high temperature (1.2) for vocabulary diversity. However, this often broke the ABAB rhyme scheme. The model would sample a word that made sense contextually but didn't rhyme (e.g., choosing "sea" instead of "ocean" to rhyme with "motion").
- **Full FT vs. LoRA:** On our small 143-sonnet dataset, full fine-tuning severely overfit, often copying famous opening lines verbatim (e.g., "Shall I compare thee...") before hallucinating the rest. LoRA's frozen weights prevented this memorization, leading to novel verses. However, those same frozen weights occasionally caused modern English phrasing to bleed into the Elizabethan text.

7 Conclusion

We built GPT-2, a custom AdamW optimizer, and LoRA from scratch. We demonstrated that updating just $\sim 0.25\%$ of the parameters via LoRA matches full fine-tuning accuracy. This highlights a key takeaway: the primary VRAM bottleneck in LLM training stems from optimizer states, which LoRA effectively bypasses by freezing the backbone.

Limitations: Due to compute constraints, we only applied LoRA to the attention projections (W_q and W_v) and did not deeply tune the rank r or scaling factor α .

Future Work: Next steps would include applying LoRA to MLP layers and benchmarking against other parameter-efficient methods like Prefix-Tuning.

Team contributions

Lianyu Yao: Built the core GPT-2 architecture and AdamW optimizer; managed the paraphrase baseline training pipeline.

Yunjie Xu: Implemented the LoRA extension; conducted the hardware efficiency profiling (VRAM and time ablations).

Isabel Li: Wrote the autoregressive generation logic (temperature/top-p sampling) for sonnets; handled data visualization.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30, 2017.
- [3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [5] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

A Hardware Utilization Details

We recorded step latency, allocated GPU memory (weights and activations), and trainable parameter counts for both the paraphrase and sonnet tasks. (Note: Epoch training times and peak reserved memory are covered in Section 5.4).

A.1 Average Step Time

LoRA consistently reduces the time per forward and backward pass across all model sizes by skipping gradient computations for the frozen backbone. Step times for the sonnet task are naturally longer than the paraphrase task due to the extended context window required for generation.

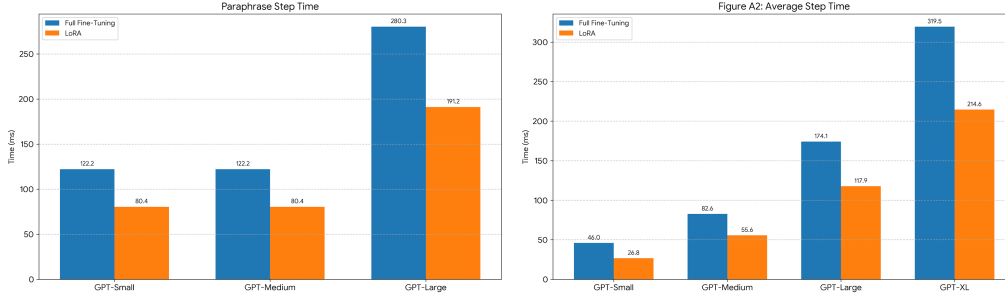


Figure 3: **Left (a):** Average Step Time (ms) - Paraphrase Detection. **Right (b):** Average Step Time (ms) - Sonnet Generation.

A.2 Peak Allocated Memory

This metric tracks the VRAM strictly occupied by weights, gradients, and activations (excluding PyTorch’s caching overhead). Scaling to GPT-XL for sonnet generation sharply increases base memory. (Note: Since the paraphrase memory curve is nearly identical, we only show the sonnet task here to illustrate the upper bound of our hardware limits.)

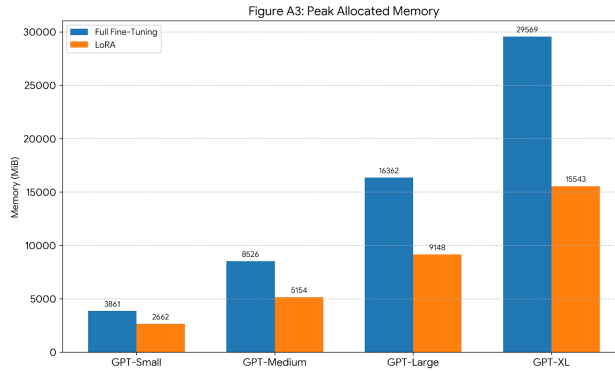


Figure 4: Peak Allocated Memory (MiB) during the Sonnet Generation task.

A.3 Trainable Parameters

While full fine-tuning scales linearly with the base model size, LoRA keeps the trainable parameter count nearly flat. As shown in the log-scale charts below, this gap is most obvious at the GPT-XL scale, where LoRA updates only ~ 2.45 million parameters out of 1.56 billion.

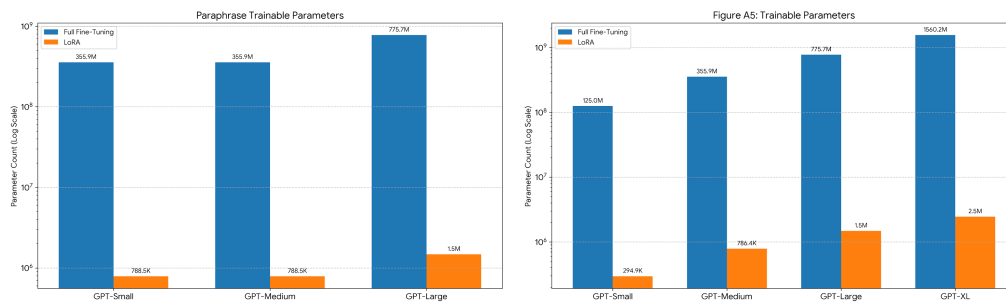


Figure 5: **Left (a):** Trainable Parameters (Log Scale) - Paraphrase Detection. **Right (b):** Trainable Parameters (Log Scale) - Sonnet Generation.