

Evaluating LoRA for Efficient GPT-2 Fine-Tuning

Stanford CS224N Default Project

Raymond Llata

Department of Computer Science
Stanford University
rllata@stanford.edu

Vania Chow

Department of Computer Science
Stanford University
vaniac@stanford.edu

Abstract

Fine-tuning pretrained language models is effective but expensive, since updating all weights for each downstream task increases training cost and requires storing separate model copies. In this work, we evaluate Low-Rank Adaptation (LoRA) as a lightweight alternative for adapting GPT-2 on the CS224N default tasks spanning both classification and generation: sentiment classification (SST and CFIMDb), cloze-style paraphrase detection (QQP), and Shakespeare sonnet generation. We implement configurable LoRA modules inside GPT-2 linear projections and compare them against standard fine-tuning and parameter-efficient baselines under matched training settings. We study how performance depends on adapter rank and placement, with a focus on attention-only adaptations versus broader LoRA insertion. Overall, our results show that appropriately targeted low-rank updates can recover much of the benefit of heavier fine-tuning while training far fewer parameters, with the best configuration depending on the task.

1 Project Information

- Mentor: Simon Kim
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

Fine-tuning pretrained language models is effective, but updating all model weights for each downstream task is expensive in compute, memory, and storage. These costs grow as models scale and as practitioners maintain separate fine-tuned copies for multiple tasks. Parameter-efficient fine-tuning (PEFT) methods address this problem by adapting only a small subset of task-specific parameters while freezing the pretrained backbone.

In this work, we study Low-Rank Adaptation (LoRA) [1] as a parameter-efficient method for adapting GPT-2 on the CS224N default tasks: sentiment classification (SST and CFIMDb), cloze-style paraphrase detection (QQP), and Shakespeare sonnet generation. LoRA represents the task-specific update to a weight matrix as a low-rank decomposition, allowing the model to learn targeted changes without modifying the full set of pretrained parameters.

We focus on three questions: (1) whether LoRA can recover most of the performance of stronger fine-tuning baselines under matched training conditions, (2) how sensitive LoRA is to adapter rank and placement, especially within attention projections, and (3) whether these tradeoffs differ across classification and generation tasks.

We implement configurable LoRA modules in the CS224N GPT-2 codebase and compare them against full fine-tuning and several PEFT baselines under matched task-specific training settings. Overall, LoRA provides a strong accuracy-efficiency tradeoff: attention-focused, moderate-rank variants are especially effective on classification, while the best PEFT method on QQP and sonnet generation remains task-dependent.

3 Related Work

Transformer pretraining and adaptation. The Transformer architecture [2] enabled large-scale self-attention pretraining, and the GPT family showed that autoregressive language model pretraining can transfer effectively to a range of downstream tasks after adaptation [3, 4]. In this paradigm, a pretrained backbone is adapted to task-specific objectives such as classification or generation, traditionally through full fine-tuning. We follow this setup by adapting a pretrained GPT-2 model across both discriminative and generative tasks.

Parameter-efficient fine-tuning. A broad line of work seeks to reduce the cost of adaptation by updating only a small number of parameters while freezing the backbone. Adapter methods insert small trainable bottleneck modules into each layer and can approach full fine-tuning performance with substantially fewer updated parameters [5]. Prompt- and prefix-based methods instead learn continuous task-specific conditioning vectors. Prefix-tuning prepends learned key/value states to attention layers [6], while prompt tuning learns soft prompt embeddings that condition a frozen model without updating its backbone weights [7]. Other PEFT methods impose structured constraints on the update space: BitFit trains only bias terms [8], while (IA)³ applies learned rescaling to intermediate activations [9]. Together, these methods motivate comparing adaptation strategies not only by downstream performance but also by parameter budget and efficiency.

Low-rank adaptation. LoRA parameterizes task-specific weight updates as low-rank decompositions while freezing the pretrained weights, based on the idea that effective fine-tuning updates may lie in a low-dimensional subspace [1]. A key practical advantage is that the learned update can be merged into the original weight matrix at inference time, avoiding added latency. Subsequent work has extended this idea through adaptive rank allocation and quantization-aware variants such as AdaLoRA and QLoRA [10, 11]. Prior work shows that LoRA can be applied selectively to attention projections, and motivates studying how adapter placement affects the tradeoff between parameter budget and performance [1].

Why constrained updates can work. Several lines of work suggest that full fine-tuning may be unnecessarily high-dimensional for many downstream tasks. Intrinsic dimension analyses show that strong task performance can often be reached within a much smaller optimization subspace than the full parameter space [12]. This explains why methods such as LoRA, BitFit, and (IA)³ can remain competitive despite updating only a tiny fraction of model parameters. In our setting, it motivates analyzing how adaptation performance depends not just on parameter count, but on where limited adaptation capacity is allocated within the network.

4 Approach

We benchmark LoRA as a parameter-efficient alternative to fine-tuning a pretrained GPT-2 model across the CS224N default tasks: sentiment classification (SST and CFIMDb), paraphrase detection (QQP), and Shakespeare sonnet generation. Across methods, we hold the backbone checkpoint, tokenization, data splits, and evaluation pipeline fixed, and vary only which parameters are trainable.

4.1 Base Model

We use the pretrained GPT-2 backbone from the CS224N starter code and keep the tokenizer, architecture, data splits, and evaluation pipeline fixed across methods. For SST and CFIMDb, we use the final token representation with a linear classifier head. For QQP, we follow the default cloze-style formulation, predicting the next-token verbalizer yes or no. For sonnet generation, we train autoregressively on continuation tokens and generate from the provided prefix at test time.

4.2 Baselines

We compare LoRA against a range of fine-tuning and parameter-efficient adaptation baselines:

- **Last-layer fine-tuning (LL):** Freeze the GPT-2 backbone and update only the task-specific head (e.g., the classifier or paraphrase head).
- **Full-model fine-tuning (FT):** Update all GPT-2 parameters together with the task head.
- **Prefix-Tuning (KV-only):** We use a learned prefix of length $L_p = 10$ at every Transformer layer. The prefix consists of trainable key/value tensors prepended to the self-attention key and value sequences, and the attention mask is extended so tokens can attend to the prefix positions. All backbone weights remain frozen.

- **(IA)³**: We replace the attention key/value projections and the MLP intermediate projection (interm_dense) with IA3Linear wrappers that freeze the original weights and learn a per-output-channel scaling vector.
- **BitFit**: We train only parameters whose names end in .bias (excluding pooler_dense) together with the task head, while freezing all other backbone parameters.
- **Partial fine-tuning (generation baseline)**: For sonnet generation, update only the top Transformer blocks and the final layer normalization while freezing lower layers, providing a compute-lean alternative to full-model fine-tuning.

4.3 Low-Rank Adaptation (LoRA)

LoRA freezes the pretrained weights and learns a low-rank update to selected linear projections [1]. For a linear layer with pretrained weight matrix $W \in \mathbb{R}^{d \times k}$, LoRA parameterizes the update as

$$W' = W + \Delta W, \quad \Delta W = \frac{\alpha}{r} BA, \quad (1)$$

where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ are trainable, r is the rank, and α is a scaling hyperparameter. During the forward pass, for input x the adapted layer computes

$$W'x = Wx + \frac{\alpha}{r} B(Ax), \quad (2)$$

so only $\{A, B\}$ (and any task-specific head parameters) receive gradients.

Integration into GPT-2. We implement LoRA modules directly inside the starter-code GPT-2 model without external PEFT libraries. Our implementation supports configurable rank r , scale α , target selection, and trainable-parameter accounting. We validated the implementation with basic sanity checks, including gradient inspection and adapter-shape verification.

Targeting schemes. We evaluate two main targeting strategies:

- **LoRA-attn**: apply LoRA only to masked self-attention projections, evaluating three placements: query-only (W_q), value-only (W_v), and query+value ($W_q + W_v$).
- **LoRA-all**: apply LoRA to all nn.Linear layers in GPT-2 (including attention projections and MLP layers).

4.4 Ablations and Efficiency Instrumentation

Rank and placement sweeps. To study parameter-performance trade-offs, we sweep ranks $r \in \{4, 8, 16, 32\}$ (or a compute-feasible subset) and compare attention placements: W_q only, W_v only, and $W_q + W_v$.

Matched optimization. We keep optimization settings fixed within each task; full training details are provided in Section 5.3.

Efficiency metrics. In addition to task performance, we log the number of trainable parameters for each configuration. For the final report, we additionally measure system-level efficiency, including peak GPU memory and time-to-best-dev, to compare full fine-tuning and LoRA under realistic compute constraints.

5 Experiments

5.1 Data

We evaluate on the three CS224N default tasks: SST and CFIMDb for sentiment classification, QQP for cloze-style paraphrase detection, and Shakespeare sonnet generation for autoregressive text generation. We use the starter code data loaders and default splits throughout.

5.2 Evaluation

We follow the project evaluation setup, reporting accuracy for SST, CFIMDb, and QQP, and chrF for sonnet generation. Checkpoints are selected by best development-set performance for each task.

5.3 Experimental Details

All experiments use the CS224N starter codebase with a pretrained GPT-2 backbone and the provided tokenization and data preprocessing. Unless otherwise noted, we use fixed seeds for sweep runs. For classifier.py, we then rerun the top two configurations with two random seeds to estimate variability, while keeping the training setup otherwise identical within the task.

Method	Targets	r	SST Dev	CFIMDb Dev	#Trainable
LL	–	–	0.467	0.861	3.845×10^3
FT	all	–	0.514	0.988	1.250×10^8
LoRA-attn	$W_q + W_v$	4	0.501	0.967	1.31301×10^5
LoRA-attn	$W_q + W_v$	16	0.523	0.971	5.94×10^5
LoRA-attn	$W_q + W_v$	32	0.524	0.971 ± 0.000	1.183493×10^6
LoRA-all	all linear	16	0.503	0.984	2.66×10^6
Bitfit	–	–	0.510	0.971	1.05989×10^5
Prefix Tuning	–	–	0.430	0.580	2.91610×10^6
(IA) ³	–	–	0.483	0.890	5.6384×10^4

Table 1: Headline sentiment results illustrate (i) LoRA’s parameter efficiency relative to FT, (ii) strong attention-only placement ($W_q + W_v$), and (iii) rank scaling behavior ($r = 4, 16, 32$). Full sweep results are reported in Appendix 4.

LoRA configurations. In experiments, we evaluate two LoRA target families: all-linear and attention-only placements (W_q , W_v , and $W_q + W_v$), sweeping ranks $r \in \{4, 8, 16, 32\}$ where compute permits. Unless otherwise noted, we fix $\alpha = 16$ and log trainable parameter counts, peak memory, and time-to-best-dev for each configuration.

Config selection for QQP and sonnets. To balance coverage and compute, we evaluate a small set of LoRA configurations on the extension tasks that emphasize attention-only adaptation at higher rank. For QQP, we compare full fine-tuning against two attention-only LoRA variants: W_v at $r = 16$ and $W_q + W_v$ at $r = 32$. For sonnet generation, we compare the partial fine-tuning baseline against LoRA-all at $r = 16$ and attention-only $W_q + W_v$ at $r = 16$. These choices are motivated by our sentiment results showing that attention-only placements can be competitive while using substantially fewer trainable parameters and lower peak memory than broader LoRA placements.

Optimization and hyperparameters. Within each task, we use the same training setup across adaptation methods to ensure fair comparison. Unless otherwise noted, we train with AdamW, learning rate 1×10^{-4} , batch size 8, dropout 0.1, and 10 epochs, with linear warmup over the first 10% of steps. The maximum training horizon is $10 \cdot \lceil N_{\text{train}}/8 \rceil$ steps. For sonnet generation, we apply early stopping with patience 2 based on validation loss. We disable mode-specific hyperparameter adjustments so that differences reflect the adaptation method rather than task-specific training changes.

Generation settings (sonnets). For sonnet generation, we decode using temperature sampling with temperature $T = 1.2$ and nucleus sampling with $p = 0.9$ (top- p), matching the starter script defaults. We evaluate generated continuations with chrF on the development set.

Implementation and reproducibility. We run all experiments via the provided CLI entry points: `classifier.py` (SST/CFIMDb), `paraphrase_detection.py` (QQP), and `sonnet_generation.py` (sonnets). To ensure reproducibility, we keep data splits and tokenization unchanged and use fixed seeds for all sweeps. For `classifier.py`, we additionally rerun the top two configurations using two random seeds.

5.4 Results

5.4.1 Sentiment, Paraphrase, and Sonnet Results

The complete sentiment sweep table is provided in Appendix 4; in the main text we summarize the key comparisons that support our conclusions.

5.4.2 Final Test Submissions

Following model selection on the development set, we submitted our best-performing configurations for the extension tasks to the test leaderboard. On QQP paraphrase detection, the strongest final test result came from **full-model fine-tuning**, which achieved a test accuracy of **0.893**. This is consistent with our dev-set results, where full fine-tuning had the strongest configuration on QQP, suggesting that the task benefits from the additional adaptation capacity of updating the full backbone.

For sonnet generation, the strongest final test result came from (IA)³, which achieved a test chrF of **41.525**. Although partial fine-tuning and LoRA variants were competitive on the development

Method	LoRA Placement	r	Dev Acc	#Trainable	Peak Mem (MB)	Time-to-Best (s)
FT (full-model)	–	–	0.897	1.250×10^8	12975.88	14613.64
LoRA-attn	W_v	16	0.873	2.96450×10^5	2732.97	12978.8
LoRA-attn	$W_q + W_v$	32	0.879	1.183493×10^6	2820.60	14484.5
BitFit	–	–	0.845	1.037×10^5	2715.00	14140.0
(IA) ³	default	–	0.832	5.683×10^4	3330.59	14552.4

Table 2: QQP paraphrase detection results (development accuracy) with efficiency metrics.

Method	LoRA Placement	r	Dev chrF	#Trainable	Peak Mem (MB)	Time-to-Best (s)
Partial FT (top layers)	–	–	42.364	5.67045×10^7	2925.45	273.33
LoRA-all	all	16	41.078	2.654208×10^6	2955.95	352.55
LoRA-attn	$W_q + W_v$	16	40.967	5.93669×10^5	2581.50	244.4
BitFit	–	–	40.537	1.021×10^5	2530.36	203.3
(IA) ³	default	–	41.281	5.530×10^4	2839.61	214.8

Table 3: Sonnet generation results (development chrF) with efficiency metrics.

set, this result suggests that lightweight activation-scaling can generalize especially well for stylistic generation in this setting. More broadly, the final test submissions reinforce our main conclusion that the best parameter-efficient adaptation strategy is task-dependent: full fine-tuning remained strongest for paraphrase detection, while a highly parameter-efficient method, (IA)³, achieved the best held-out generation performance.

5.4.3 Efficiency and Optimization Stability

Peak Memory vs. Accuracy. LoRA on W_q , W_v , or $W_q + W_v$ consistently reduces peak memory relative to LoRA-all while achieving comparable dev accuracy in many settings. This indicates a favorable memory–accuracy tradeoff for attention-only LoRA placement.

Time-to-Best-Dev. We report the wall-clock time and optimization step at which best dev performance is reached. Some lower-rank and attention-only LoRA variants reach their best dev point earlier than higher-cost configurations, though this trend is not uniform across placements and tasks.

Parameter Efficiency. Trainable parameter counts span from $\sim 7.8 \times 10^4$ (e.g., single-target low-rank LoRA) to $\sim 5.3 \times 10^6$ (LoRA-all, $r=32$), while full fine-tuning updates $\sim 1.25 \times 10^8$ parameters. This confirms substantial parameter savings from LoRA.

6 Analysis

6.1 Parameter–Performance Scaling

We analyze how development accuracy varies with trainable parameter count across LoRA ranks and placements (Appendix Table 4; Table 1 shows representative configurations). Three patterns are consistent across the sentiment tasks.

LoRA approaches full fine-tuning with far fewer parameters. Full fine-tuning updates 1.25×10^8 parameters, while strong LoRA configurations update only 10^5 – 10^6 . For example, LoRA-attn $W_q + W_v$ with $r=16$ reaches 0.523 SST development accuracy versus 0.514 for full fine-tuning, using only 5.94×10^5 trainable parameters (Table 1). On CFIMDb, the same configuration reaches 0.971 versus 0.988 for full fine-tuning. These results show that much of the benefit of full adaptation can be recovered with a low-rank update.

Performance saturates at moderate ranks. Increasing rank improves accuracy up to a point, then yields diminishing returns. For $W_q + W_v$, SST rises from 0.501 at $r=4$ to 0.523 at $r=16$ (Table 1), with little additional gain at higher ranks in the full sweep (Appendix Table 4). This is consistent with the view that many downstream updates lie in a relatively low-dimensional subspace [1].

Attention-only LoRA is more parameter-efficient than LoRA-all. On SST, $W_q + W_v$ at $r=16$ achieves 0.523 accuracy with 5.94×10^5 trainable parameters, while LoRA-all at the same rank

reaches 0.503 with 2.66×10^6 parameters (Table 1). This suggests that attention projections carry much of the task-specific adaptation signal and supports attention-focused placement as an efficient default under constrained budgets.

6.2 Scaling Trends and Task Sensitivity

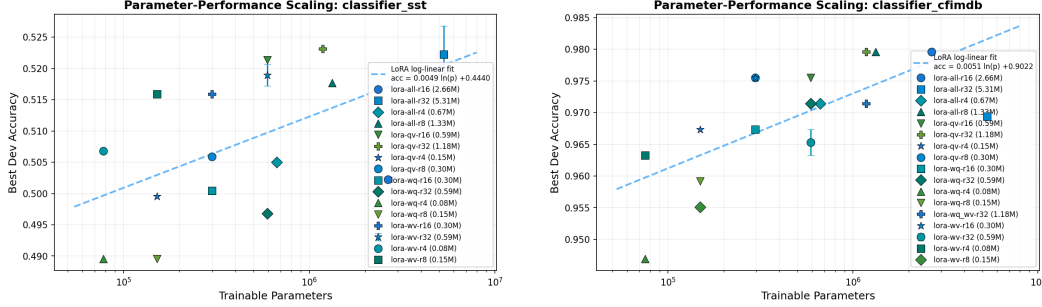


Figure 1: Parameter–performance scaling for sentiment tasks. **Left:** SST. **Right:** CFIMDb.

Figure 1 plots best development accuracy against trainable parameter count on a log scale for SST and CFIMDb. In both tasks, accuracy improves as adaptation capacity increases, but the gains diminish at higher parameter counts. This produces an approximately log-linear trend in which moderate-rank LoRA captures most of the benefit of larger updates. The pattern is tighter on CFIMDb, where many low-rank settings already achieve very strong accuracy, while SST shows more dispersion across placements and ranks.

This difference suggests that SST is more sensitive to how adaptation capacity is allocated. On CFIMDb, many LoRA configurations exceed 0.95 development accuracy, indicating that the task is comparatively forgiving and that several low-rank updates are sufficient to recover strong performance. On SST, by contrast, performance depends more strongly on both rank and placement, with attention-focused configurations such as $W_q + W_v$ benefiting more clearly from moderate rank increases. Full fine-tuning remains strongest or near-strongest overall, but its gains over the best LoRA settings are modest relative to the large increase in trainable parameters. Taken together, these results support a practical takeaway: moderate ranks such as 8–16 often provide the best tradeoff, and careful placement matters more on harder classification settings.

6.3 QQP and Sonnet Analysis

QQP: LoRA closes part of the gap to full fine-tuning. On QQP, full fine-tuning achieves the strongest development accuracy (0.897), but attention-only LoRA remains competitive while training dramatically fewer parameters. In particular, LoRA-attn with $W_q + W_v$ at $r=32$ reaches 0.879 using 1.18×10^6 trainable parameters, and LoRA-attn with W_v at $r=16$ reaches 0.873 using only 2.96×10^5 parameters (Table 2). Relative to full fine-tuning’s 1.25×10^8 trainable parameters, this indicates that much of the paraphrase-detection signal can be recovered through a small number of targeted low-rank updates.

Attention-only adaptation is effective but does not fully match FT on QQP. The remaining gap between LoRA and full fine-tuning is larger on QQP than on SST, suggesting that paraphrase detection requires richer adaptation than binary sentiment classification. This is plausible given the task formulation: the model must compare two input questions, represent semantic similarity rather than local sentiment cues, and map that comparison into a cloze-style yes/no decision. Under this setup, adapting only attention projections captures much of the needed cross-sequence interaction, but not all of the benefit of updating the full backbone.

Other PEFT baselines on QQP. BitFit and (IA)³ are even more parameter-efficient than LoRA, but they underperform the stronger LoRA configurations on QQP. BitFit reaches 0.845 development accuracy and (IA)³ reaches 0.832, compared with 0.873–0.879 for the best LoRA-attention settings (Table 2). This suggests that, for paraphrase detection, bias-only or activation-scaling updates are less expressive than structured low-rank changes to the attention projections.

Sonnets: LoRA remains competitive, but broader adaptation helps generation. For sonnet generation, the partial fine-tuning baseline remains best at 42.364 chrF, but both LoRA and other PEFT methods remain close while updating far fewer parameters (Table 3). LoRA-all at $r=16$ reaches 41.078 chrF with 2.65×10^6 trainable parameters, while attention-only LoRA with W_q+W_v at $r=16$ reaches 40.967 with only 5.94×10^5 trainable parameters. The gap to the partial fine-tuning baseline is therefore modest relative to the reduction in trainable parameters, indicating that low-rank adaptation remains effective even for autoregressive generation.

Generation benefits somewhat more from broader capacity. Compared with sentiment classification, the sonnet results suggest that generation is less forgiving to tightly constrained adaptation. LoRA-all slightly outperforms attention-only LoRA, which is consistent with the intuition that poetic generation depends not only on attention routing but also on broader transformations in the MLP layers and token-level style modeling. Unlike classification, where a targeted attention update can often capture the relevant decision boundary, sonnet generation requires sustained control over lexical choice, local fluency, and stylistic continuation across many decoding steps.

(IA)³ is surprisingly strong on sonnets. A notable result is that (IA)³ reaches 41.281 chrF, outperforming both LoRA configurations reported in Table 3 while using only 5.53×10^4 trainable parameters. This suggests that multiplicative activation scaling may be especially effective for style-sensitive generation when the goal is to preserve pretrained fluency while gently steering internal representations. At the same time, the absolute differences among the top sonnet methods are fairly small, so this result should be interpreted cautiously without broader seed coverage and qualitative analysis of generated outputs.

Efficiency across the extension tasks. The extension tasks reinforce the broader efficiency patterns observed on sentiment. Attention-only LoRA substantially reduces trainable parameter count relative to full fine-tuning, and typically lowers peak memory relative to broader LoRA placement while remaining competitive in task performance. Lower-capacity PEFT methods such as BitFit and (IA)³ offer even stronger parameter savings, but their effectiveness is more task-dependent. Overall, these results suggest that the most attractive adaptation method depends on the target task and resource constraint: full fine-tuning remains strongest when maximum task performance is the priority, while LoRA offers a strong accuracy–efficiency compromise, especially when careful adapter placement is used.

6.4 Error Analysis

QQP. QQP errors reveal two common failure modes. First, the model sometimes produces false positives when two questions share strong lexical overlap and a similar syntactic frame but differ in intent or meaning. For example, it predicts a paraphrase for “Why do Quorans answer questions that are already answered?” and “Why do Quorans downvote questions they cannot answer?”, likely over-weighting shared words such as “Why do Quorans” and “questions” while missing the difference in action and intent. Second, the model can miss small but important semantic modifiers, as in “How do deaf-born people think?” versus “How do people born blind and deaf think?”, where the added condition “blind” changes the meaning substantially. We also observe some ambiguity in the dataset labels themselves: for instance, “What is Shaoxing vinegar?” and “What is Shaoxing vinegar used for?” are labeled as duplicates even though one asks for a definition and the other for usage, suggesting that some QQP errors may reflect label noise or loose notions of paraphrase rather than purely model failure.

Sonnets. Qualitative comparison suggests that stronger models better sustain Shakespearean diction and thematic continuity, even when automatic metric differences are small. Weaker generations often produce a convincing opening quatrain but then drift into rambling prose-like continuation, malformed syntax, or truncated outputs. For instance, one weak sample begins well with “Be wise as thou art cruel; do not press / My tongue-tied patience with too much disdain,” but soon loses coherence, while another produces the ungrammatical phrase “Thou, then are thy debtor.” In contrast, a stronger sample extends “So, now I have confessed that he is thine / And I myself am mortgaged to thy will” with a more consistent thread of surrender and obligation. These examples suggest that stronger models improve local fluency and stylistic consistency more than long-range poetic structure.

6.5 Design Implications and Limitations

Our results support three conclusions for GPT-2 adaptation on these tasks: (i) competitive performance is achievable with well under 1% of parameters updated, (ii) allocating rank to attention projections,

especially W_q+W_v , is an efficient default for classification, and (iii) moderate ranks provide most of the benefit of larger updates, with diminishing returns at higher ranks.

Limitations. This study has several limitations. All experiments are conducted on a single pre-trained backbone, GPT-2, so the conclusions may not fully generalize to larger or more modern language models. In addition, while we rerun top configurations with additional seeds, seed coverage remains limited relative to the full sweep space. The non-LoRA PEFT baselines, including BitFit, Prefix-Tuning, and (IA)³, are also evaluated with fixed implementation settings rather than exhaustive hyperparameter tuning. Finally, for sonnet generation, chrF is only a partial proxy for quality and does not capture poetic properties such as rhyme, meter, or long-range coherence.

7 Conclusion

We evaluated Low-Rank Adaptation (LoRA) as a parameter-efficient alternative to adapting GPT-2 on the CS224N default tasks spanning sentiment classification, paraphrase detection, and sonnet generation. Across tasks, our results show that LoRA can recover much of the performance of heavier fine-tuning baselines while updating dramatically fewer parameters. In particular, attention-only LoRA is often a strong default: on sentiment classification, adapting W_q+W_v at moderate rank achieved performance comparable to, and in some cases slightly better than, full fine-tuning, while using only a small fraction of the trainable parameters. More broadly, we find that rank increases improve performance up to a point, after which returns diminish, indicating that effective task adaptation often lies in a relatively low-dimensional subspace.

Our comparisons across tasks also show that the best adaptation strategy depends on the task. For classification, targeted attention updates are especially parameter-efficient and competitive. For generation, LoRA remains competitive with the partial fine-tuning baseline, but the gap is smaller and broader adaptation can still help, suggesting that generative style transfer may benefit from capacity beyond a narrow subset of projections. The additional PEFT baselines further contextualize these results: BitFit and (IA)³ can offer attractive efficiency, but LoRA provides a stronger and more consistent accuracy-efficiency tradeoff across the tasks we study.

Overall, these findings support LoRA as a practical method for adapting pretrained language models under constrained compute and storage budgets. Our experiments suggest three main takeaways: updating well under 1% of model parameters can be sufficient for strong downstream performance, allocating capacity to attention projections is often an effective default, and moderate ranks capture most of the benefit of larger low-rank updates. Important limitations remain, including evaluation on only a single backbone scale, incomplete multi-seed coverage for all settings, and the use of chrF as only a partial proxy for generation quality. Extending this study to larger models, broader hyperparameter tuning, and richer qualitative evaluation would provide a fuller picture of when low-rank adaptation is most effective.

Team contributions (Required for multi-person team)

Vania Chow: Implemented LoRA/PEFT integration in the GPT-2 codebase, ran the main sentiment and QQP experiments, and led the quantitative analysis and results tables.

Raymond Llata: Implemented and validated baseline methods, ran sonnet-generation experiments and qualitative analysis, and co-wrote and edited the final report.

References

- [1] 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. In *International Conference on Learning Representations (ICLR)*, 2022.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [3] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI Technical Report, 2018.

- [4] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI Technical Report, 2019.
- [5] 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 *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.
- [6] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.
- [7] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.
- [8] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2022.
- [9] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [10] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2023.
- [11] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [12] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

A Appendix

Method	LoRA Targets	r	SST Dev	CFIMDb Dev	#Trainable	Peak Mem (MB)	Time-to-Best (s)
LL	–	–	0.467	0.861	3.845×10^3	–	–
FT	all	–	0.514	0.988	1.250×10^8	–	–
LoRA-attn	W_q	4	0.490	0.947	7.7573×10^4	853.15 / 4586.29	324.55 / 62.03
LoRA-attn	W_q	8	0.490	0.959	1.48994×10^5	854.11 / 4582.22	296.00 / 107.54
LoRA-attn	W_q	16	0.500	0.967	2.96450×10^5	856.02 / 4585.96	322.63 / 89.83
LoRA-attn	W_q	32	0.497	0.971	5.91362×10^5	859.84 / 4592.30	97.92 / 60.98
LoRA-attn	W_v	4	0.507	0.963	7.7573×10^4	849.5 / 4538.7	321.6 / 45.7
LoRA-attn	W_v	8	0.516	0.955	1.48994×10^5	850.5 / 4535.5	228.3 / 76.2
LoRA-attn	W_v	16	0.516	0.976	2.96450×10^5	852.4 / 4538.7	301.1 / 118.0
LoRA-attn	W_v	32	0.519 ± 0.002	0.965 ± 0.003	5.91362×10^5	856.2 / 4545.0	230.7 / 91.1
LoRA-attn	$W_q + W_v$	4	0.501	0.967	1.31301×10^5	855.1 / 4676.2	122.3 / 32.4
LoRA-attn	$W_q + W_v$	8	0.505	0.971	2.98757×10^5	857.0 / 4680.0	289.6 / 79.3
LoRA-attn	$W_q + W_v$	16	0.523	0.971	5.93669×10^5	860.8 / 4686.3	356.0 / 124.8
LoRA-attn	$W_q + W_v$	32	0.524	0.971 ± 0.000	1.183493×10^6	868.4 / 4712.5	322.8 / 140.4
LoRA-all	all linear	4	0.491	0.967	6.67397×10^5	992.2 / 5562.1	496.5 / 40.7
LoRA-all	all linear	8	0.515 ± 0.004	0.973 ± 0.009	1.330949×10^6	1000.5 / 5569.6	313.5 / 176.7
LoRA-all	all linear	16	0.503	0.984	2.658053×10^6	1017.2 / 5595.0	614.8 / 37.9
LoRA-all	all linear	32	0.522 ± 0.006	0.969 ± 0.003	5.312261×10^6	1050.4 / 5626.7	244.0 / 96.7

Table 4: Sentiment dev results with efficiency metrics (values shown as SST / CFIMDb for Peak Mem and Time-to-Best).