

# Low-Rank Adaptation for Efficient GPT-2 Fine-Tuning: Evaluation Across Classification and Generation Tasks

Stanford CS224N Default Project

**Vivek Tiwari**

Department of Computer Science  
Stanford University  
vivekt@stanford.edu

## Abstract

Fine-tuning large pretrained language models for downstream tasks requires updating all parameters, which is computationally expensive. We investigate Low-Rank Adaptation (LoRA) [1] applied to GPT-2 [2] (124M parameters) across four tasks: sentiment classification (SST and CFIMDB), paraphrase detection (Quora), and Shakespearean sonnet generation. We implement LoRA by injecting trainable low-rank matrices into frozen GPT-2 layers and conduct systematic ablation studies across rank ( $r \in \{2, 4, 8, 16\}$ ), target modules (query-value, QKV, attention, MLP, and all linear layers), and learning rates. Our results show that LoRA matches or approaches full fine-tuning performance across all four tasks while training less than 1% of model parameters. On SST, LoRA achieves 0.534 dev accuracy (vs. 0.516 for full fine-tuning); on CFIMDB, 0.988 (vs. 0.967). For sonnet generation, LoRA with tuned generation hyperparameters achieves 42.1 test chrF (vs. 41.6 for full fine-tuning). For paraphrase detection, LoRA achieves 0.878 test accuracy, competitive with full fine-tuning at 0.885. Per-layer gradient analysis reveals that all tasks exhibit higher gradient norms in early/mid layers than late layers. Classification tasks show a steep drop-off, while sonnet generation distributes gradients more uniformly, with the final layer also among the most active.

## 1 Key Information

- **Mentor:** Luke Bailey
- **External collaborators:** None
- **Sharing project:** No
- **AI tool usage:** Claude/Copilot assisted with writing, organization, brainstorming, LoRA implementation and evaluation scripts. All base code implementation, technical decisions and experiment design are my own.
- **Late days used:** NA

## 2 Introduction

Pretrained language models such as GPT-2 [2] achieve strong performance across a range of NLP tasks when fine-tuned on task-specific data. However, full fine-tuning requires updating all model parameters—124 million in the case of GPT-2—which is computationally expensive and requires storing a separate copy of the model for each task. As models continue to scale, this approach becomes increasingly impractical.

Parameter-efficient fine-tuning (PEFT) methods address this challenge by updating only a small subset of parameters while keeping the pretrained weights frozen. Among these, Low-Rank Adaptation

(LoRA) [1] has emerged as a particularly effective approach: it injects trainable low-rank matrices alongside frozen weights, adding no inference latency and requiring minimal additional parameters.

In this work, we implement LoRA for GPT-2 from scratch and evaluate it across four downstream tasks spanning both classification and generation: five-class sentiment classification on SST [3], binary sentiment classification on CFIMDB, cloze-style paraphrase detection on Quora, and Shakespearean sonnet generation. We conduct systematic ablation studies across LoRA rank, target module selection, and learning rate to understand how these design choices affect performance across task types.

Our main findings are:

- LoRA matches or approaches full fine-tuning performance on all four tasks while training less than 1% of parameters.
- Target module selection affects performance in a task-dependent manner, with applying LoRA to all linear layers generally performing best.
- For paraphrase detection, LoRA requires more training epochs to converge compared to other tasks, suggesting that the large dataset size and cloze-style formulation demand longer adaptation.
- Per-layer gradient analysis shows that all tasks exhibit higher gradient norms in early/mid layers, but classification tasks show a steep drop-off toward later layers, while sonnet generation is more uniform, with the final layer also among the most active.

### 3 Related Work

Parameter-efficient fine-tuning (PEFT) methods aim to adapt large pretrained models without updating all parameters. Adapter layers [4] insert bottleneck modules between transformer layers but add inference latency; prefix-tuning [5] and prompt tuning [6] prepend learnable vectors to transformer inputs but reduce usable sequence length. LoRA [1] avoids both limitations by decomposing weight updates into low-rank matrices that can be merged with the original weights at inference time, adding zero latency. The original LoRA evaluation focuses on GPT-3 (175B) with attention-only targets. Our work applies LoRA to the smaller GPT-2 (124M) and extends the ablation to include MLP layers alongside attention projections across both classification and generation tasks. More recently, ReFT [7] applies learned interventions to hidden representations rather than weights, building on similar intuitions about low-rank structure in fine-tuning—a complementary direction to weight-space approaches like LoRA.

## 4 Approach

### 4.1 Baseline: GPT-2 Fine-Tuning

We build on the provided GPT-2 [2] implementation, consisting of 12 transformer [8] layers with 768-dimensional hidden states and 12 attention heads (124M parameters). We use the AdamW optimizer [9, 10] with weight decay set to 0. Two baseline fine-tuning modes are evaluated: (1) **last-linear-layer**, where only the task-specific head is trained, and (2) **full fine-tuning**, where all GPT-2 parameters are updated.

For **sentiment analysis**, the last token’s hidden state is passed through a linear classifier. For **paraphrase detection**, we use a cloze-style formulation where the model generates “yes” or “no” in response to a natural language prompt asking whether two sentences are paraphrases. For **sonnet generation**, GPT-2 is fine-tuned with cross-entropy loss on Shakespeare’s sonnets and conditioned on the first three lines to generate the remaining eleven.

### 4.2 Low-Rank Adaptation (LoRA)

LoRA [1] hypothesizes that weight updates during fine-tuning have low intrinsic rank. For a pretrained weight matrix  $W_0 \in \mathbb{R}^{d \times k}$ , the update is constrained to a low-rank decomposition:

$$W = W_0 + \Delta W = W_0 + BA, \quad B \in \mathbb{R}^{d \times r}, \quad A \in \mathbb{R}^{r \times k}, \quad r \ll \min(d, k) \quad (1)$$

During training,  $W_0$  is frozen and only  $A$  and  $B$  are updated. The output includes a scaling factor  $\alpha/r$ ; following the common practice of setting  $\alpha = 2r$ , and since most of our ablation uses  $r=4$ , we fix  $\alpha = 8$  across all experiments to avoid introducing an additional hyperparameter. Matrix  $A$  is initialized with Kaiming uniform initialization and  $B$  with zeros, so that  $\Delta W = 0$  at the start of training.

We implemented LoRA from scratch (with Copilot AI assistance) rather than using an existing library. Our implementation wraps each targeted `nn.Linear` layer with a `LoRALinear` module that computes:

$$h = W_0x + \frac{\alpha}{r} \cdot BAx \quad (2)$$

All original model parameters are frozen; only the LoRA matrices  $A$ ,  $B$  and the task-specific head are trainable.

### 4.3 LoRA Target Configurations

We define five target configurations that determine which linear layers in each transformer block receive LoRA adapters:

Table 1: LoRA target configurations and trainable parameter counts at rank  $r=4$ . Percentages are relative to the full GPT-2 model.

| Target    | Adapted Layers                                       | LoRA Params | % of Model |
|-----------|------------------------------------------------------|-------------|------------|
| qv        | $W_Q, W_V$                                           | 147K        | 0.12%      |
| qkv       | $W_Q, W_K, W_V$                                      | 221K        | 0.18%      |
| attention | $W_Q, W_K, W_V, W_O$                                 | 295K        | 0.24%      |
| mlp       | $W_{\text{up}}, W_{\text{down}}$                     | 369K        | 0.30%      |
| all       | $W_Q, W_K, W_V, W_O, W_{\text{up}}, W_{\text{down}}$ | 664K        | 0.53%      |

The MLP layers ( $W_{\text{up}}: \mathbb{R}^{768 \times 3072}$ ,  $W_{\text{down}}: \mathbb{R}^{3072 \times 768}$ ) contribute more LoRA parameters per layer than the attention projections ( $\mathbb{R}^{768 \times 768}$ ) due to their larger dimensions. Even at the most expansive configuration (*all*), LoRA trains only 0.53% of the model’s parameters.

## 5 Experiments

### 5.1 Data

We use the datasets provided for the default project: **SST** [3]: 8,544 / 1,101 / 2,210 train/dev/test examples for five-class sentiment classification. **CFIMDB**: 1,701 / 245 / 488 train/dev/test examples for binary sentiment classification of longer movie reviews. **Quora**: 283,003 / 40,429 / 80,858 train/dev/test question pairs for binary paraphrase detection via cloze-style generation. **Sonnets**: Shakespeare’s sonnets split into training and held-out sets, with 12 test sonnets evaluated by conditioning on the first three lines.

### 5.2 Evaluation

Classification tasks (SST, CFIMDB, Quora) are evaluated with **accuracy**. Sonnet generation is evaluated with **chrF** [11], a character-level n-gram F-score measuring similarity to Shakespeare’s original sonnets.

### 5.3 Experimental Details

All experiments were run on a single NVIDIA A100 GPU via Google Colab. We use the AdamW optimizer with a random seed of 11711.

**Baselines.** For sentiment classification, we trained for 10 epochs with batch size 8, learning rate  $10^{-3}$  (last-linear-layer) or  $10^{-5}$  (full fine-tuning), and dropout 0.3. Paraphrase detection and sonnet generation baselines used learning rate  $10^{-5}$  for 10 epochs.

**LoRA experiments.** We conduct three types of ablation: (1) learning rate sweeps, (2) rank sweeps with  $r \in \{2, 4, 8, 16\}$  targeting all layers, and (3) target module sweeps across the five configurations in Table 1. All LoRA runs use  $\alpha = 8$ , dropout 0.0, weight decay 0, and batch size 64 for SST and Quora or 8 for CFIMDB and Sonnets. For classification tasks, we train for 10 epochs and select the best epoch by dev performance. For paraphrase detection, the initial sweep used 3 epochs (since paraphrase required most computation); we additionally train the best configuration for 10 epochs. For sonnet generation, we also tune generation temperature and top- $p$  sampling parameters for leaderboard submission.

We log per-layer gradient norms at each epoch to analyze how adaptation distributes across the network depth.

## 5.4 Results

Table 2: Results across all tasks. Dev set results for SST and CFIMDB; dev and test results for paraphrase detection and sonnet generation (as required for Gradescope leaderboard). <sup>†</sup>Sonnet LoRA results use temperature = 1.1 and top- $p = 0.9$ .

| Task           | Split | Last-Linear | Full FT      | LoRA Best               | LoRA Config                         |
|----------------|-------|-------------|--------------|-------------------------|-------------------------------------|
| SST (Acc.)     | Dev   | 0.473       | 0.516        | <b>0.534</b>            | $r=2$ , all, lr $3 \times 10^{-4}$  |
| CFIMDB (Acc.)  | Dev   | 0.865       | 0.967        | <b>0.988</b>            | $r=16$ , all, lr $3 \times 10^{-4}$ |
| Quora (Acc.)   | Dev   | —           | <b>0.896</b> | 0.892                   | $r=16$ , all, lr $3 \times 10^{-4}$ |
|                | Test  | —           | <b>0.885</b> | 0.878                   | 10 epochs                           |
| Sonnets (chrF) | Dev   | —           | 41.8         | <b>42.3<sup>†</sup></b> | $r=4$ , all, lr $3 \times 10^{-3}$  |
|                | Test  | —           | 41.6         | <b>42.1<sup>†</sup></b> | $t=1.1$ , $p=0.9$                   |

LoRA matches or approaches full fine-tuning on all tasks while training less than 1% of parameters.<sup>1</sup> On sentiment classification, LoRA slightly outperforms full fine-tuning on both SST (+0.018) and CFIMDB (+0.021). For sonnet generation, LoRA with tuned generation hyperparameters (temperature and top- $p$ ) achieves a modest improvement over full fine-tuning (+0.5 chrF on test). For paraphrase detection, full fine-tuning maintains a small advantage (+0.007 on test), though LoRA remains competitive with only 2.13% of trainable parameters. The paraphrase result uses 10 training epochs; with only 3 epochs used for the ablation sweeps, longer training was needed, as the 10-epoch run continued improving through epoch 9.

Our baseline results are comparable to the provided baselines from the project handout (SST: 0.462 / 0.513, CFIMDB: 0.861 / 0.976 for last-linear / full fine-tuning respectively), with our CFIMDB full fine-tuning result (0.967) slightly below the reference (0.976).

## 6 Analysis

We present hyperparameter ablation results for each task, examining sensitivity to learning rate, rank, and target module selection. Training dynamics and per-layer gradient heatmaps are provided in Appendix A.

### 6.1 Sentiment Analysis: SST

Figure 1 shows three observations. First, SST is sensitive to learning rate, with  $10^{-3}$  performing best—notably higher than the  $10^{-5}$  used for full fine-tuning. Second, the smallest rank ( $r=2$ ) achieves the best dev accuracy (0.534), and performance decreases with higher ranks. Third, targeting all layers performs best, though attention-only configurations are competitive. The best LoRA result (0.534) exceeds full fine-tuning (0.516) while training only 0.27% of model parameters at  $r=2$ .

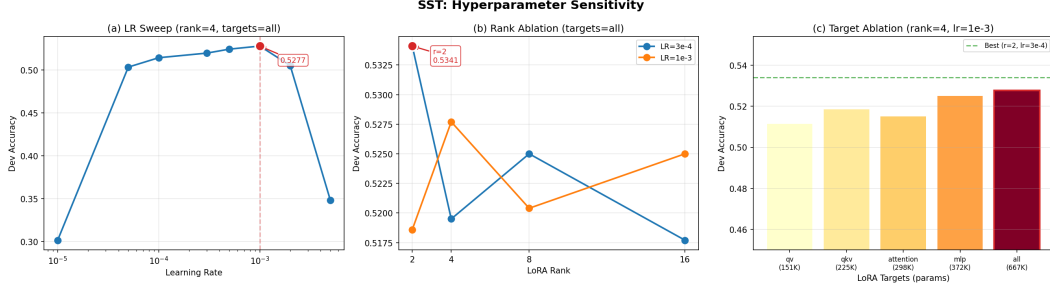


Figure 1: SST hyperparameter ablation. **Left:** Learning rate sweep ( $r=4$ , targets=all). **Center:** Rank ablation (targets=all, best LR). **Right:** Target module ablation (best rank and LR).

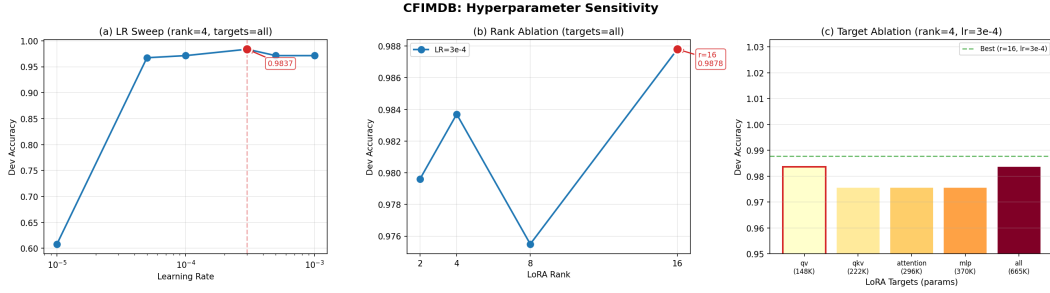


Figure 2: CFIMDB hyperparameter ablation. **Left:** Learning rate sweep. **Center:** Rank ablation. **Right:** Target module ablation.

## 6.2 Sentiment Analysis: CFIMDB

Figure 2 shows three observations. First, learning rate  $3 \times 10^{-4}$  performs best. Second, unlike SST, higher ranks improve performance, with  $r=16$  achieving 0.988 dev accuracy. Third, the *qv* configuration (147K parameters) achieves comparable performance to the *all* configuration (664K parameters), though *all* remains the best. The best LoRA result (0.988) exceeds full fine-tuning (0.967) while training 2.13% of parameters at  $r=16$ , though  $r=4$  (0.53%) already achieves 0.984.

## 6.3 Sonnet Generation

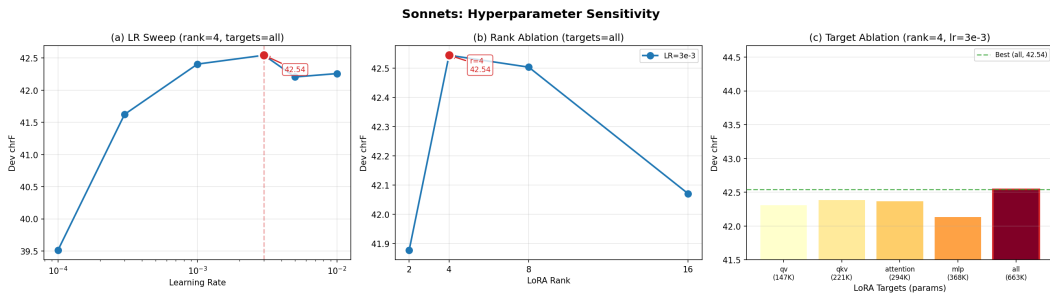


Figure 3: Sonnets hyperparameter ablation. **Left:** Learning rate sweep. **Center:** Rank ablation. **Right:** Target module ablation.

Figure 3 shows a relatively flat performance landscape, with chrF scores clustered in a narrow range regardless of rank or target choice. The best learning rate ( $3 \times 10^{-3}$ ) is higher than the other tasks,

<sup>1</sup>At  $r=4$  with all targets (0.53% of parameters), LoRA already achieves near-peak performance on every task: 0.528 on SST, 0.984 on CFIMDB, 42.5 chrF on Sonnets, and 0.865 on Quora. The best CFIMDB and Quora configurations use  $r=16$  (2.13%), though  $r=4$  is already competitive.

and the best configuration uses  $r=4$  with all targets (tied with  $r=8$  at 42.5 dev chrF; we prefer  $r=4$  for its lower parameter count). The *all* target configuration performs best, consistent with the other tasks.

For the submitted results, we additionally tuned generation hyperparameters (temperature and top- $p$ ), finding that temperature = 1.1 with top- $p$  = 0.9 yields the best chrF scores (42.1 test vs. 41.6 for full fine-tuning). We note that the baseline did not undergo the same generation hyperparameter tuning, so the comparison is not fully controlled.

## 6.4 Paraphrase Detection

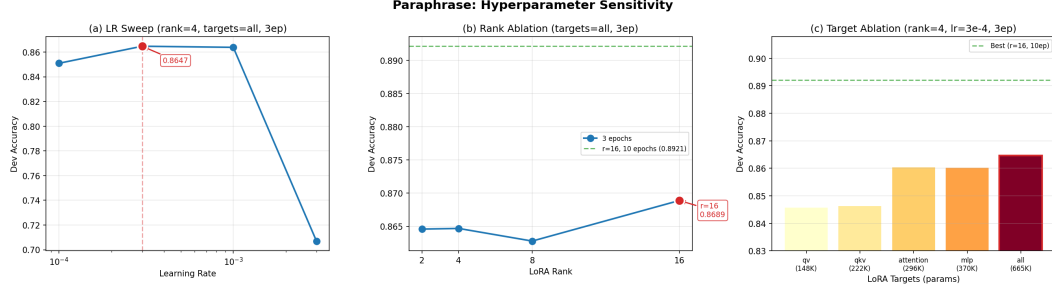


Figure 4: Paraphrase detection hyperparameter ablation. **Left:** Learning rate sweep. **Center:** Rank ablation. **Right:** Target module ablation.

Paraphrase detection uses the largest training set (283K examples) with a cloze-style formulation. Due to compute constraints, we ran the ablation sweeps for only 3 epochs each. Figure 4 shows that learning rate  $3 \times 10^{-4}$  with all targets performs best, consistent with the other tasks. The best rank is  $r=16$ , matching CFIMDB but contrasting with SST ( $r=2$ ) and sonnets ( $r=4$ ). We then trained the best configuration ( $r=16$ , all,  $lr = 3 \times 10^{-4}$ ) for 10 epochs, finding that performance continued to improve through epoch 9 (see Appendix A). If more compute time was available, we would have evaluated for longer epochs.

The final gap between LoRA (0.892 dev, 0.878 test) and full fine-tuning (0.896 dev, 0.885 test) is small, and longer training may further close it.

## 6.5 Cross-Task Observations

Several consistent patterns emerge across all four tasks:

**LoRA can use higher learning rates than full fine-tuning.** Full fine-tuning uses  $lr = 10^{-5}$ , while LoRA performs best at  $3 \times 10^{-4}$  (classification tasks) or  $3 \times 10^{-3}$  (sonnet generation)— $30\times$  to  $300\times$  higher. This is expected since LoRA updates far fewer parameters and starts from  $\Delta W = 0$ .

**Low-rank updates are competitive with full fine-tuning.** Across all four tasks, LoRA matches or approaches full fine-tuning while training less than 1% of parameters. On SST and CFIMDB, LoRA outperforms full fine-tuning; on sonnets and paraphrase detection, the gap is small.

**Applying LoRA to all linear layers consistently performs best.** While the original LoRA paper [1] recommends targeting only  $W_Q$  and  $W_V$ , the *all* configuration achieves the best results on every task in our experiments. The *qv* configuration remains competitive (especially on CFIMDB), but there is no task where a subset outperforms *all*.

**Sequential hyperparameter search is sufficient.** We sweep learning rate first, then rank, then target modules—rather than an exhaustive grid search (which could be computationally infeasible). This linear search strategy produces strong results across all tasks with substantially fewer runs than a full grid over all combinations. We note that in SST, different learning rates may work better for different ranks, so sequential search may not find the global optimum—but it consistently yields results competitive with full fine-tuning across all tasks.

**Early layers show higher gradient norms than later layers.** Per-layer gradient analysis (Appendix A) shows that across all tasks, early and middle layers (0–6) have higher gradient norms than

late layers (8–11). For classification tasks, the drop-off is steep ( $2\text{--}4\times$  max/min ratio), while sonnet generation is more uniform ( $1.4\times$  ratio) with layer 11 also among the most active.

## 7 Conclusion

We implemented LoRA for GPT-2 and evaluated it across four NLP tasks spanning classification and generation. Our results demonstrate that LoRA matches or approaches full fine-tuning performance while training less than 1% of model parameters, confirming the effectiveness of low-rank adaptation at the 124M parameter scale.

Our ablation studies reveal several practical insights: rank requirements are task-dependent, ranging from  $r=2$  for SST to  $r=16$  for CFIMDB and paraphrase detection; targeting all linear layers generally performs best but attention-only configurations remain competitive at lower parameter cost; and LoRA benefits from learning rates  $30\text{--}300\times$  higher than full fine-tuning.

**Limitations.** Our evaluation is limited to GPT-2 (124M parameters), and scaling behavior may differ for larger models. The sonnet generation comparison includes generation hyperparameter tuning (temperature, top- $p$ ) for LoRA but not for the baseline. The paraphrase ablation sweep used 3-epoch runs for most configurations, and exhaustive longer training across all configurations may yield different conclusions about optimal rank and target selection.

**Future work.** Promising directions include extending the evaluation to GPT-2 Medium (355M parameters), applying the same generation hyperparameter tuning to the full fine-tuning baseline for a controlled comparison on sonnets, and exploring layer-selective LoRA application informed by the gradient patterns observed in our analysis.

## References

- [1] Phillip Wallis Zeyuan Allen-Zhu Yuanzhi Li Shean Wang Lu Wang Weizhu Chen Edward J. Hu, Yelong Shen. Lora: Low-rank adaptation of large language models, 2021.
- [2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.
- [3] 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, 2013.
- [4] Stanislaw Jastrzebski Bruna Morrone Quentin de Laroussilhe Andrea Gesmundo Mona Attariyan Sylvain Gelly Neil Houlsby, Andrei Giurgiu. Parameter-efficient transfer learning for nlp, 2019.
- [5] Percy Liang Xiang Lisa Li. Prefix-tuning: Optimizing continuous prompts for generation, 2021.
- [6] Noah Constant Brian Lester, Rami Al-Rfou. The power of scale for parameter-efficient prompt tuning, 2021.
- [7] Zheng Wang Atticus Geiger Dan Jurafsky Christopher D. Manning Christopher Potts Zhengxuan Wu, Aryaman Arora. Reft: Representation finetuning for language models, 2024.
- [8] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attention is all you need, 2017.
- [9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- [10] Frank Hutter Ilya Loshchilov. Decoupled weight decay regularization, 2019.
- [11] Maja Popovic. chrF: character n-gram f-score for automatic mt evaluation, 2015.

## A Training Dynamics and Gradient Analysis

Figures 5–8 show training curves and per-layer gradient norm heatmaps for each task. The heatmaps visualize how gradient magnitudes distribute across the 12 transformer layers over training epochs, revealing which layers adapt most actively under LoRA.

**Heatmap construction.** After each `loss.backward()` call (on the first batch of each epoch), we record the L2 (Frobenius) norm  $\|\nabla_{\theta}\|_2$  of the gradient tensor for every LoRA matrix. For the *all* target configuration, each transformer layer contains 12 LoRA matrices ( $A$  and  $B$  for each of  $W_Q, W_K, W_V, W_O, W_{up}, W_{down}$ ). We average these 12 norms to obtain a single per-layer gradient norm  $g_{\ell}^{(e)}$ :

$$g_{\ell}^{(e)} = \frac{1}{|\mathcal{P}_{\ell}|} \sum_{\theta \in \mathcal{P}_{\ell}} \|\nabla_{\theta}\|_2 \quad (3)$$

where:

- $g_{\ell}^{(e)}$  is the average gradient norm for transformer layer  $\ell$  at epoch  $e$ ,
- $\mathcal{P}_{\ell}$  is the set of LoRA parameters in layer  $\ell$  ( $|\mathcal{P}_{\ell}| = 12$  for the *all* configuration),
- $\|\nabla_{\theta}\|_2$  is the L2 norm of the gradient of LoRA matrix  $\theta$ .

The resulting  $12 \times E$  matrix (layers  $\times$  epochs) is visualized as a heatmap, with brighter colors indicating higher gradient norms and thus more active adaptation.

Across all four tasks, early and middle layers (0–6) exhibit higher gradient norms than late layers (8–11), and layer 0 is almost always among the top three highest-gradient layers. However, the degree of this imbalance differs by task type. For classification tasks (SST, CFIMDB, Quora), the drop-off is steep, with max-to-min layer ratios of  $2.2\text{--}4.3\times$ . For sonnet generation, the distribution is much more uniform (max/min ratio of only  $1.4\times$ ). Additionally, layer 11 is among the top three most active layers for sonnets—a pattern not seen in any classification task, where layer 11 consistently has among the lowest gradients. Table 3 summarizes these per-task gradient statistics.

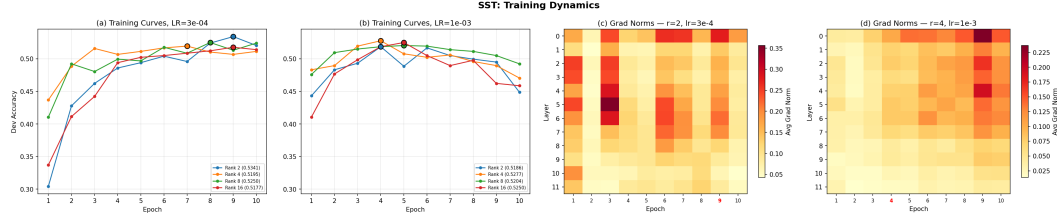


Figure 5: SST training dynamics: training curves and per-layer gradient norm heatmaps across epochs.

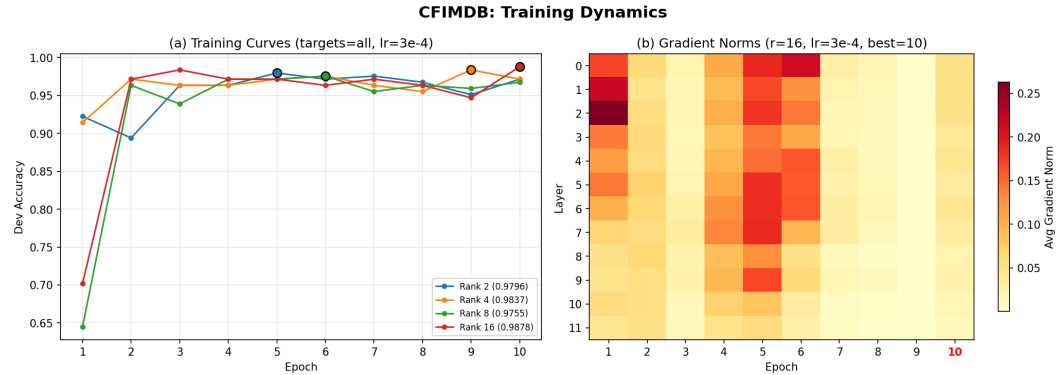


Figure 6: CFIMDB training dynamics.

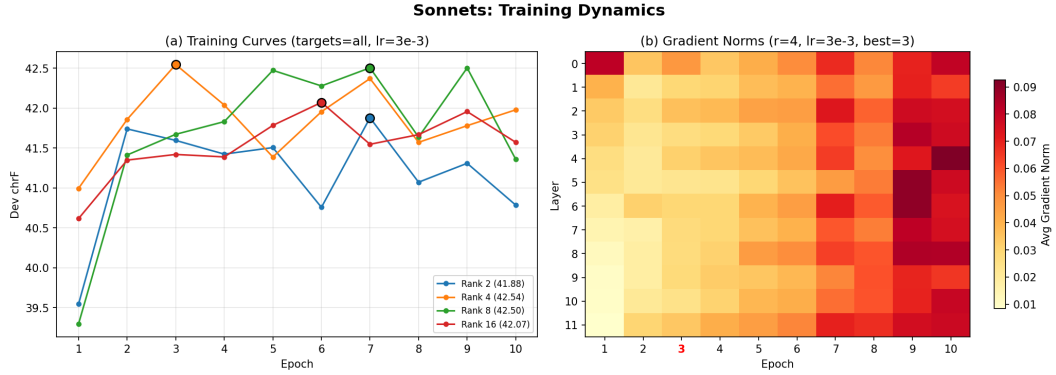


Figure 7: Sonnets training dynamics.

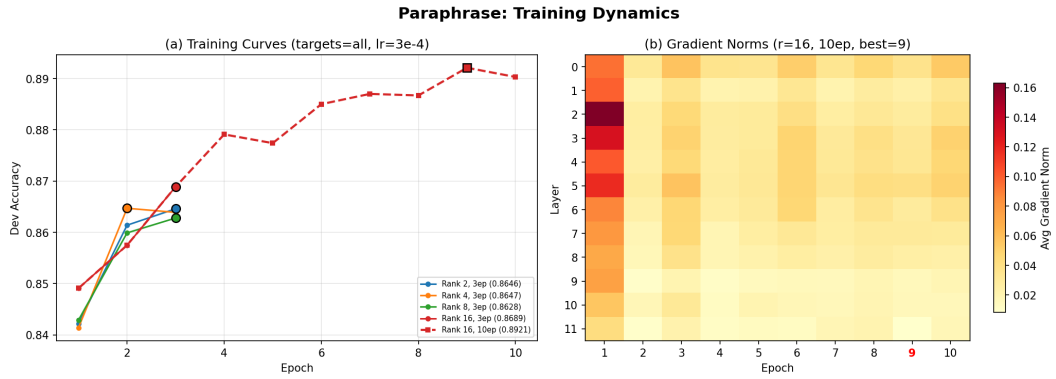


Figure 8: Paraphrase detection training dynamics.

Table 3: Per-layer gradient norm statistics across tasks. Max/min ratio is computed over average gradient norms across all layers and epochs.

| Task              | Max/Min Ratio | Top 3 Layers    | Bottom 3 Layers |
|-------------------|---------------|-----------------|-----------------|
| SST ( $r=2$ )     | $2.24\times$  | 0, 5, 6         | 10, 11, 9       |
| SST ( $r=4$ )     | $4.28\times$  | 0, 4, 2         | 11, 10, 8       |
| CFIMDB ( $r=16$ ) | $3.12\times$  | 0, 2, 1         | 11, 10, 8       |
| Quora ( $r=16$ )  | $3.20\times$  | 2, 3, 5         | 11, 9, 10       |
| Sonnets ( $r=4$ ) | $1.36\times$  | 0, 2, <b>11</b> | 9, 5, 10        |

## B Full Hyperparameter Sweep Results

Table 4 lists all 60 LoRA configurations evaluated across the four tasks. For each run, we report the best dev-set score (accuracy or chrF) and the epoch at which it was achieved. SST and CFIMDB runs train for 10 epochs; Quora ablation runs use 3 epochs due to compute constraints, with one extended 10-epoch run for the best configuration. All runs use  $\alpha = 8$  and dropout 0.0.

Table 4: Full hyperparameter sweep results across all tasks. Best epoch selected by dev performance. Bold indicates best result per task.

| Task    | Target    | Rank | LR                 | Best Dev     | Best Ep. |
|---------|-----------|------|--------------------|--------------|----------|
| SST     | all       | 2    | $3 \times 10^{-4}$ | <b>0.534</b> | 9        |
| SST     | all       | 2    | $10^{-3}$          | 0.519        | 4        |
| SST     | all       | 4    | $10^{-5}$          | 0.301        | 10       |
| SST     | all       | 4    | $5 \times 10^{-5}$ | 0.503        | 10       |
| SST     | all       | 4    | $10^{-4}$          | 0.514        | 7        |
| SST     | all       | 4    | $3 \times 10^{-4}$ | 0.519        | 7        |
| SST     | all       | 4    | $5 \times 10^{-4}$ | 0.524        | 5        |
| SST     | all       | 4    | $10^{-3}$          | 0.528        | 4        |
| SST     | all       | 4    | $2 \times 10^{-3}$ | 0.505        | 2        |
| SST     | all       | 4    | $5 \times 10^{-3}$ | 0.348        | 2        |
| SST     | all       | 8    | $3 \times 10^{-4}$ | 0.525        | 8        |
| SST     | all       | 8    | $10^{-3}$          | 0.520        | 5        |
| SST     | all       | 16   | $3 \times 10^{-4}$ | 0.518        | 9        |
| SST     | all       | 16   | $10^{-3}$          | 0.525        | 5        |
| SST     | qv        | 4    | $3 \times 10^{-4}$ | 0.505        | 10       |
| SST     | qv        | 4    | $10^{-3}$          | 0.511        | 7        |
| SST     | qkv       | 4    | $3 \times 10^{-4}$ | 0.506        | 10       |
| SST     | qkv       | 4    | $10^{-3}$          | 0.519        | 8        |
| SST     | attention | 4    | $3 \times 10^{-4}$ | 0.509        | 10       |
| SST     | attention | 4    | $10^{-3}$          | 0.515        | 10       |
| SST     | mlp       | 4    | $3 \times 10^{-4}$ | 0.518        | 8        |
| SST     | mlp       | 4    | $10^{-3}$          | 0.525        | 6        |
| CFIMDB  | all       | 2    | $3 \times 10^{-4}$ | 0.980        | 5        |
| CFIMDB  | all       | 4    | $10^{-5}$          | 0.608        | 10       |
| CFIMDB  | all       | 4    | $5 \times 10^{-5}$ | 0.967        | 6        |
| CFIMDB  | all       | 4    | $10^{-4}$          | 0.971        | 5        |
| CFIMDB  | all       | 4    | $3 \times 10^{-4}$ | 0.984        | 9        |
| CFIMDB  | all       | 4    | $5 \times 10^{-4}$ | 0.971        | 4        |
| CFIMDB  | all       | 4    | $10^{-3}$          | 0.971        | 9        |
| CFIMDB  | all       | 8    | $3 \times 10^{-4}$ | 0.976        | 6        |
| CFIMDB  | all       | 16   | $3 \times 10^{-4}$ | <b>0.988</b> | 10       |
| CFIMDB  | qv        | 4    | $3 \times 10^{-4}$ | 0.984        | 9        |
| CFIMDB  | qkv       | 4    | $3 \times 10^{-4}$ | 0.976        | 9        |
| CFIMDB  | attention | 4    | $3 \times 10^{-4}$ | 0.976        | 8        |
| CFIMDB  | mlp       | 4    | $3 \times 10^{-4}$ | 0.976        | 2        |
| Sonnets | all       | 2    | $3 \times 10^{-3}$ | 41.9         | 7        |
| Sonnets | all       | 4    | $10^{-4}$          | 39.5         | 10       |
| Sonnets | all       | 4    | $3 \times 10^{-4}$ | 41.6         | 6        |
| Sonnets | all       | 4    | $10^{-3}$          | 42.4         | 7        |
| Sonnets | all       | 4    | $3 \times 10^{-3}$ | <b>42.5</b>  | 3        |
| Sonnets | all       | 4    | $5 \times 10^{-3}$ | 42.2         | 8        |
| Sonnets | all       | 4    | $10^{-2}$          | 42.3         | 8        |
| Sonnets | all       | 8    | $3 \times 10^{-3}$ | <b>42.5</b>  | 7        |
| Sonnets | all       | 16   | $3 \times 10^{-3}$ | 42.1         | 6        |
| Sonnets | qv        | 4    | $3 \times 10^{-3}$ | 42.3         | 4        |
| Sonnets | qkv       | 4    | $3 \times 10^{-3}$ | 42.4         | 6        |
| Sonnets | attention | 4    | $3 \times 10^{-3}$ | 42.4         | 8        |
| Sonnets | mlp       | 4    | $3 \times 10^{-3}$ | 42.1         | 10       |
| Quora   | all       | 2    | $3 \times 10^{-4}$ | 0.865        | 3        |
| Quora   | all       | 4    | $10^{-4}$          | 0.851        | 3        |
| Quora   | all       | 4    | $3 \times 10^{-4}$ | 0.865        | 2        |
| Quora   | all       | 4    | $10^{-3}$          | 0.864        | 2        |
| Quora   | all       | 4    | $3 \times 10^{-3}$ | 0.707        | 1        |
| Quora   | all       | 8    | $3 \times 10^{-4}$ | 0.863        | 3        |
| Quora   | all       | 16   | $3 \times 10^{-4}$ | 0.869        | 3        |
| Quora*  | all       | 16   | $3 \times 10^{-4}$ | <b>0.892</b> | 9        |
| Quora   | qv        | 4    | $3 \times 10^{-4}$ | 0.846        | 3        |
| Quora   | qkv       | 4    | $3 \times 10^{-4}$ | 0.846        | 3        |
| Quora   | attention | 4    | $3 \times 10^{-4}$ | 0.860        | 3        |
| Quora   | mlp       | 4    | $3 \times 10^{-4}$ | 0.860        | 3        |

\*Extended to 10 epochs; all other

Quora runs use 3 epochs.

## C Parameter Count Derivations

### GPT-2 (Full Fine-Tuning)

Table 5 through Table 7 decompose the full GPT-2 parameter count.

Table 5: GPT-2 embedding parameters.

| Component          | Shape               | Params            |
|--------------------|---------------------|-------------------|
| Word embedding     | $50,257 \times 768$ | 38,597,376        |
| Position embedding | $1,024 \times 768$  | 786,432           |
| <b>Subtotal</b>    |                     | <b>39,383,808</b> |

Table 6: GPT-2 per-transformer-layer parameters ( $\times 12$  layers).

| Component                            | Shape                      | Params            |
|--------------------------------------|----------------------------|-------------------|
| $W_Q$ + bias                         | $768 \times 768 + 768$     | 590,592           |
| $W_K$ + bias                         | $768 \times 768 + 768$     | 590,592           |
| $W_V$ + bias                         | $768 \times 768 + 768$     | 590,592           |
| $W_O$ + bias                         | $768 \times 768 + 768$     | 590,592           |
| $W_{\text{up}}$ + bias               | $768 \times 3,072 + 3,072$ | 2,362,368         |
| $W_{\text{down}}$ + bias             | $3,072 \times 768 + 768$   | 2,360,064         |
| LayerNorm (attn)                     | $768 + 768$                | 1,536             |
| LayerNorm (output)                   | $768 + 768$                | 1,536             |
| <b>Per layer</b>                     |                            | <b>7,087,872</b>  |
| <b><math>\times 12</math> layers</b> |                            | <b>85,054,464</b> |

Table 7: GPT-2 model-level parameters.

| Component           | Shape                             | Params                                                |
|---------------------|-----------------------------------|-------------------------------------------------------|
| Final LayerNorm     | $768 + 768$                       | 1,536                                                 |
| <b>Subtotal</b>     |                                   | <b>1,536</b>                                          |
| <b>Grand total:</b> | $39,383,808 + 85,054,464 + 1,536$ | <b>124,439,808 (<math>\approx 124\text{M}</math>)</b> |

Note: the language modeling head (lm\_head) shares weights with the word embedding and adds no new parameters.

### LoRA Parameter Calculation

For each adapted weight  $W \in \mathbb{R}^{d \times k}$ , LoRA adds  $A \in \mathbb{R}^{r \times k}$  and  $B \in \mathbb{R}^{d \times r}$ , contributing  $r(d + k)$  parameters (no bias). Table 8 shows the per-layer breakdown.

Substituting each rank value yields the total LoRA trainable parameters (Table 9).

Table 8: LoRA parameters per transformer layer for the *all* target configuration.

| Component                                | <i>A</i> shape   | <i>B</i> shape   | Params                       |
|------------------------------------------|------------------|------------------|------------------------------|
| $W_Q$ ( $768 \times 768$ )               | $r \times 768$   | $768 \times r$   | $1,536r$                     |
| $W_K$ ( $768 \times 768$ )               | $r \times 768$   | $768 \times r$   | $1,536r$                     |
| $W_V$ ( $768 \times 768$ )               | $r \times 768$   | $768 \times r$   | $1,536r$                     |
| $W_O$ ( $768 \times 768$ )               | $r \times 768$   | $768 \times r$   | $1,536r$                     |
| $W_{\text{up}}$ ( $768 \times 3,072$ )   | $r \times 3,072$ | $768 \times r$   | $3,840r$                     |
| $W_{\text{down}}$ ( $3,072 \times 768$ ) | $r \times 768$   | $3,072 \times r$ | $3,840r$                     |
| <b>Per layer</b>                         |                  |                  | <b><math>13,824r</math></b>  |
| <b><math>\times 12</math> layers</b>     |                  |                  | <b><math>165,888r</math></b> |

Table 9: Total LoRA trainable parameters by rank (*all* target configuration). Percentages are relative to the full GPT-2 model (124.4M).

| Rank   | Calculation         | LoRA Params | % of GPT-2 |
|--------|---------------------|-------------|------------|
| $r=2$  | $165,888 \times 2$  | 331,776     | 0.27%      |
| $r=4$  | $165,888 \times 4$  | 663,552     | 0.53%      |
| $r=8$  | $165,888 \times 8$  | 1,327,104   | 1.07%      |
| $r=16$ | $165,888 \times 16$ | 2,654,208   | 2.13%      |

## D Dev Prediction Examples

Below are representative correct and incorrect predictions from the full fine-tuning baseline on each classification dev set, illustrating where the model succeeds and where it fails.

### SST (5-class Sentiment, Dev Accuracy = 0.516)

#### Correct predictions:

1. 55a7fd84 — Label: 0, Pred: 0  
*“Taylor appears to have blown his entire budget on soundtrack rights and had nothing left over for jokes .”*
2. 9ab97ed3 — Label: 4, Pred: 4  
*“A warm , funny , engaging film .”*
3. d8636a20 — Label: 2, Pred: 2  
*“This is as respectful a film as Byatt fans could hope for , though lovers of the book may wonder why it 's necessary .”*

#### Incorrect predictions (off by $\geq 2$ ):

1. 50c7312e — Label: 4, Pred: 2  
*“In a way , the film feels like a breath of fresh air , but only to those that allow it in .”*  
The hedging language (“in a way,” “only to those”) misleads the model toward neutral despite strongly positive intent.
2. 164e411b — Label: 1, Pred: 3  
*“You wo n’t like Roger , but you will quickly recognize him .”*  
The model misses the negative framing; “recognize” reads as endorsement rather than criticism.

#### Incorrect predictions (off by 1, subtle):

1. a7d575c9 — Label: 3, Pred: 4  
*“It ’s a lovely film with lovely performances by Buy and Accorsi .”*
2. 33cfa329 — Label: 2, Pred: 3  
*“No one goes unindicted here , which is probably for the best .”*

### CFIMDB (Binary Sentiment, Dev Accuracy = 0.967)

Only 8 of 245 dev examples are misclassified. The errors involve mixed-signal reviews where surface-level positive or negative words conflict with the overall sentiment.

#### Correct predictions:

1. 9844a320 — Label: 0, Pred: 0  
*“There ’s something awful about the fact that a movie made in 1934 can be head and shoulders below every Tarzan movie that followed it ... Not recommended .”* [excerpt]
2. 9dcf468a — Label: 1, Pred: 1  
*“This is one of those movies you see in the video store that you just HAVE to get because it just looks so horribly bad . But surprisingly , we love most of it .”* [excerpt]

#### Incorrect predictions:

1. 50de1fc4 — Label: 0, Pred: 1 (false positive)  
*“I absolutely love this show , but I saw the second episode first . After watching the first episode I could see why people were glued . The first episode ’s humor is the best ...”* [excerpt]  
Positive language (“love,” “the best”) dominates despite the review being labeled negative overall.

2. cb23bd5b — Label: 1, Pred: 0 (false negative)  
*“I sought out this film for one reason — Al Adamson . He is among the worst directors of all time — right up there with Ed Wood , Jr. . . .”* [excerpt]  
Strongly negative language about the director misleads the model, though the review is ultimately positive (appreciating the filmmaker ironically).

### Quora (Binary Paraphrase, Dev Accuracy = 0.896)

#### Correct predictions:

1. True positive — *“What is addiction?” / “What exactly is addiction?”* Label: 1, Pred: 1
2. True negative — *“If I am on private on Twitter and I favorite a tweet will it be seen/notified by the person I favorited?” / “How can I know how much favorites I have on twitter?”*  
Label: 0, Pred: 0

#### Incorrect predictions:

1. False positive — Label: 0, Pred: 1  
Q1: *“How do deaf-born people think?”*  
Q2: *“How do people born blind and deaf think?”*  
High lexical overlap (“born,” “deaf,” “think”) but different questions (deaf-only vs. blind-and-deaf).
2. False positive — Label: 0, Pred: 1  
Q1: *“What was life like before and after the Industrial Revolution?”*  
Q2: *“What was life like before the Industrial Revolution?”*  
Near-identical phrasing but different scope (before-and-after vs. before-only).
3. False negative — Label: 1, Pred: 0  
Q1: *“How do I find a job in a startup in India?”*  
Q2: *“How do I join a startup company in India?”*  
Semantically equivalent but uses different verbs (“find a job” vs. “join”).

## E Generated Sonnet Examples

Below are two representative LoRA-generated sonnets from the test set ( $r=4$ , all targets,  $\text{lr} = 3 \times 10^{-3}$ , temperature = 1.1, top- $p = 0.9$ ). The first three lines (shown in blue) are the input prompt; the remaining lines (in black) are generated by the model.

**Sonnet 153** (stronger example — maintains mythological imagery and couplet structure):

Cupid laid by his brand and fell asleep.  
A maid of Dian's this advantage found,  
And his love-kindling fire did quickly steep ten hearts,  
Mad princess alight long before him  
Into dream-places doth ensconce  
From devotion, and doth thence doth attend.  
Sainthood then doth seem, but hence will youth be—  
For his mark falls amid sumptuous strain,  
And like barren rib alder stealthily hides,  
And sooned barren bewitched flower survives hid—  
False mortal grief, haunted love dare alive dare dwell!  
And now false grown lamp, unable tenderly hid,  
Sick spring pricked his dress, yet caroused still with melancholy wild looks.

**Sonnet 148** (typical example — shows common failure modes: invented words, metric breakdown):

O me, what eyes hath love put in my head,  
Which have no correspondence with true sight!  
Or if they have, where is my judgment fled,  
Which where will hatred line mine mercy lie?  
But with myself nothing, it is left not a diseased face.  
Now thou art almost contumacious,  
And if thy lives, which have refracted,  
Suffangle like whirlstone, demand my love.  
The whole swift hath fleered away the soul,  
And God then wastes his sweet night with silver fever;  
Thus, fond memories melt from him right by,  
O Sweet Jute cherub.  
This profit cannot even translate to view.  
But, that which gives merit either, he what earth and mist.

Both sonnets attempt Shakespearean vocabulary and couplet endings, but exhibit common generation artifacts: line-length inconsistency, invented words (“suffangle,” “whirlstone”), and semantic drift from the prompt’s theme. The chrF metric captures these issues quantitatively.