

Efficiency and Inference: A Comparative Study of PEFT and Full Fine-Tuning

Stanford CS224N Default Project

Name

Department of Computer Science
Stanford University
sanyamg@stanford.edu

Abstract

Modern large language models (LLMs) are predominantly based on decoder-only Transformer architectures, achieving state-of-the-art performance across various NLP tasks. This project reproduces GPT-2 (1) using the partial implementation provided in the CS224N course repository (2). The completed model, initialized with pretrained weights from Hugging Face, is fine-tuned on three distinct NLP tasks. Because each task requires interpreting model outputs differently, this process provides practical insight into LLM mechanics during both training and inference. After establishing baseline performance through full fine-tuning, Low-Rank Adaptation (LoRA) (3) is implemented to optimize the training process. The LoRA-augmented model achieves near-baseline scores while requiring significantly fewer computational resources. Throughout the study, various metrics are tracked to analyze training dynamics. Finally, the report concludes with an error analysis of the paraphrase detection task, highlighting the limitations of the current approach.

1 Key Information to include

- Mentor: Wei Liu
- External Collaborators (if you have any): N/A
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
Sanyam Gupta	3
Total pooled	3
Team size	1
Late days applied for the report	3

2 Introduction

Frontier LLMs exhibit strong zero-shot performance and can generalize across tasks. They are based on decoder-only Transformer architectures. This trend was pioneered in GPT-2 by Radford et al. (1), where the model achieved state-of-the-art performance across a range of NLP tasks. This work aims to understand GPT-2 components by building on top of the base implementation provided in the CS224N course codebase (2). Existing work explains high-level approaches and results, but often does not cover low-level implementation details. This project develops that core understanding and extends it by fine-tuning GPT-2 for different downstream tasks with: (1) the base version and (2) LoRA. It then compares training dynamics and performance across tasks.

Key Transformer components are attention layers, and optimization uses AdamW (4), as in GPT-2. Both are implemented from scratch. With the base implementation complete, pretrained weights

are downloaded from Hugging Face, and the model is fine-tuned for three downstream tasks: (1) sentiment classification, (2) cloze-style paraphrase detection, and (3) Shakespeare-style sonnet generation, as explained in Section 4. Performance across these tasks demonstrates how pretrained LLMs can be adapted to different downstream settings. This is Phase 1 of the project.

Phase 1 also establishes baseline model performance. Phase 2 aims to optimize fine-tuning using the parameter-efficient method LoRA (3). LoRA is implemented as an independent PyTorch module. Following the original paper, the query and value matrices in attention blocks are augmented with LoRA modules. Then, as in Phase 1, the model is fine-tuned for downstream tasks. In this phase, only paraphrase detection (Task 2) and sonnet generation (Task 3) are run. The model reaches performance close to the baseline while training faster per epoch, as described in Section 5, which also discusses LoRA training dynamics.

3 Related Work

Vaswani et al. (5) introduced the Transformer architecture with both encoder and decoder blocks. Radford et al. (1) later used decoder-only models and showed strong performance across many NLP tasks. Between these works, parameter counts increased substantially: from 65 million to 117 million for smaller models, and from 213 million to 1,542 million for larger models. This led to the view that increasing parameter count improves model performance. Kaplan et al. (6) formalized this with scaling laws, showing that increasing model size, dataset size, and compute reduces cross-entropy loss. Later work scaled models even further; for example, Llama 3 includes a 405B-parameter model (7).

Training frontier LLMs can be broadly divided into pretraining followed by fine-tuning. Pretraining captures broad language knowledge, while fine-tuning adapts models to specific tasks; this is reinforced by large autoregressive models such as GPT-3 (8). Aghajanyan et al. (9) showed that fine-tuning updates have low intrinsic dimensionality, suggesting that parameter-efficient methods are especially suitable for task adaptation. This is the core idea behind LoRA (3), which adds low-rank matrices to attention blocks while freezing pretrained parameters. As a result, only a small set of parameters is updated during fine-tuning. These models train faster and often retain strong performance. This improves on alternatives such as adapters (10), which may increase inference latency or reduce model accuracy.

This project is motivated by a practical gap between the high-level claims in prior literature and the low-level engineering choices required to reproduce those claims in a working system. In particular, while scaling-law and PEFT papers report strong aggregate results, they provide limited visibility into training dynamics across heterogeneous downstream objectives under a shared implementation setup. Our work serves as a useful replication and extension: we first reproduce a GPT-2 training/fine-tuning pipeline from partially implemented course code, then evaluate full fine-tuning versus LoRA under different task settings (classification, cloze-style reasoning, and generation). This provides a clear next step for prior work by pairing outcome metrics with process-level analysis (loss, gradient norm, train/dev behavior, and score dynamics), allowing us to study not only *whether* PEFT remains competitive, but also *how* it achieves this.

4 Approach

For Phase 1, we followed the CS224N DFP instructions (11) to build a working implementation of GPT-2 and the AdamW optimizer (4). We then fine-tuned GPT-2 on the three NLP tasks described below. Using this base model and its baseline results, Phase 2 focused on improving training efficiency while maintaining accuracy. The following sections describe each phase in more detail.

4.1 Baseline model

The provided course codebase (2) required updates to key modules of GPT-2, for implementing CausalAttention and the optimizer step. These are described in Sections 5.2 and 5.3 of the project handout (11). After implementing these core components, pretrained GPT-2 weights were used for three downstream tasks: (1) sentiment classification, (2) paraphrase detection, and (3) sonnet generation.

4.2 Downstream tasks

Autoregressive models output one token at a time. Since the downstream tasks differ, output tokens are interpreted differently. This section describes the model setup; performance is discussed in Section 5.3.

4.2.1 Sentiment classification

A new head layer was implemented. The last output token from the model was passed to this linear head, with an aim to map output token vector $\mathbf{V} \in \mathbb{R}^{d_{\text{hidden}}}$ to $\mathbb{R}^{\text{num_sentiments}}$, where $\text{hidden} \gg \text{num_sentiments}$.

This task was evaluated on two datasets, described in Section 5.1. The value of `num_sentiments` differs across datasets, resulting in different `fan_out` for the head.

1. For SST dataset - `num_sentiments` = 5. Project handout (11) reported baseline scores for setup where fine tuning was performed to update weights of 1) full model and, 2) only the last linear head.
2. For CFIMDB dataset - `num_sentiments` = 2. Baseline scores from the handout (11) were used, for setup similar to one described above.

4.2.2 Paraphrase detection

Rather than treating this as a binary classification task, it was posed as a cloze-style task, as described in Section 7.1 of the project handout (11). The final token representation is projected to the input word-embedding space to produce a Yes/No token. The full model was fine-tuned for this task.

4.2.3 Sonnet generation

The model generates sonnets autoregressively from the first three prompt lines. Top-p sampling (12) is used for generation. Baselines were not reported in the handout. The full model was fine-tuned for this task as well.

4.3 Improving training loop

Phase 1 mostly followed the project handout (11) and built on top of the provided course codebase (2). The newly implemented code behaved as expected for the base model, optimizer, and downstream tasks. This section describes updates Phase 1. All work discussed here is **original**, i.e., not partially implemented in the course codebase or described in the handout.

1. **Train-time metrics** - We implemented metrics to plot training loss, gradient norm, train accuracy, and dev accuracy over epochs. These were used for all three tasks to ensure effective learning and to monitor underfitting/overfitting. An example plot from sentiment analysis is shown below. Since the number of steps across 30 epochs is large, the training-loss and gradient-norm curves contain many points. To improve readability, values were averaged over 20 steps to produce cumulative-smoothed trends.
2. **Learning rate scheduler** - Pytorch’s learning rate scheduler was used to adjust learning rate during training. The Scheduler type was Linear, with warmup steps as 10% of total number of steps.
3. **Weight decay on specific model parameters** - Originally in the Phase 1 implementation, weight decay was not used. To prevent overfitting—especially with larger models like GPT-2 Medium and Large—weight decay was introduced. However, to maintain training stability, weight decay was explicitly disabled for bias terms and LayerNorm parameters.
4. **chrF score for sonnet generation** - The default codebase did not compute chrF. This made it difficult to track train-time chrF scores. The SacreBLEU library was used to compute chrF.
5. **Held-out dataset for train split** - The provided data did not include a held-out split for train-time sonnet evaluation. To compute chrF for train split, the top three lines were

truncated and written to another text file. This file was passed to the model in evaluation mode to generate sonnets and compute chrF.

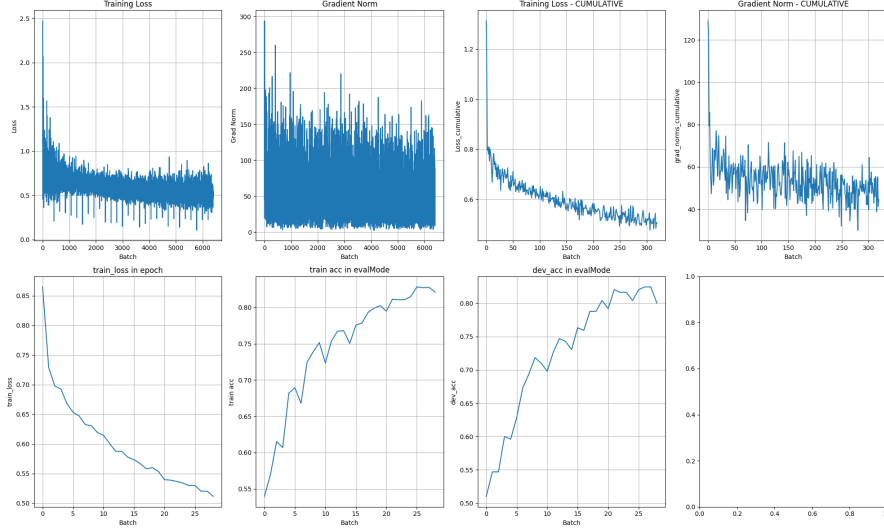


Figure 1: Train-time metrics for IMDB fine-tuning (batch size 32, learning rate 4×10^{-5} , 30 epochs).

4.4 PEFT with LoRA

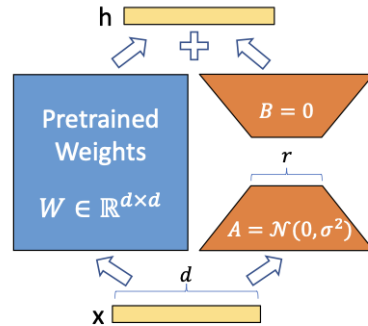
Parameter Efficient Fine Tuning was used by implementing LoRA as described in original paper by Hu et al (3).

4.4.1 Background

The adapted hidden representation is written as:

$$\mathbf{h} = W_0\mathbf{x} + \Delta W\mathbf{x} = W_0\mathbf{x} + B A \mathbf{x}$$

where $\mathbf{x}$ is the input token representation, W_0 is the frozen pre-trained projection matrix, and $\mathbf{h}$ is the output hidden representation after linear projection. The trainable update is ΔW , which is parameterized as low-rank factors B and A . Here, A projects $\mathbf{x}$ to a low-rank space and B projects it back to the original hidden dimension. This keeps most pre-trained parameters fixed and only learns a small number of LoRA parameters during fine-tuning.



4.4.2 Our implementation

In our work, the LoRA module is implemented as a PyTorch module. It is attached to the query and value projections in the CausalSelfAttention module of the default project repository, mimicking the original work by Hu et al. This simplifies integration with the existing code-base. When pretrained weights are loaded, the logic is augmented to load only the weights and biases for the original query and value matrices, not the LoRA modules.

Figure 2: LoRA reparameterization. Only low-rank matrices A and B are trained.

5 Experiments

5.1 Dataset

For each subtask defined in Section 4.2, the following datasets were used, as provided in the course repository (2).

1. Sentiment classification

- (a) **Stanford Sentiment Treebank (SST)** (13) consists of 11,855 movie reviews with labels: negative, somewhat negative, neutral, somewhat positive, and positive. These sentiments are encoded as integers from 0 to 4, in the same order. Splits: train (8,544), dev (1,101), test (2,210).
 - (b) **CFIMDB** consists of 2,434 highly polar movie reviews with binary labels, where 0 is negative and 1 is positive. Splits: train (1,701), dev (245), test (488).
2. **Paraphrase detection** The Quora dataset (14) contains question pairs labeled as paraphrases or non-paraphrases. Each example has two sentences and an `is_duplicate` binary label indicating whether the two sentences are paraphrases. A subset of the original dataset was used with splits: train (141,506), dev (20,215), test (40,431).
3. **Sonnet generation** The Sonnet dataset consists of 154 sonnets written by Shakespeare. Each sonnet has 14 lines following the rhyme scheme ABAB CDCD EFEF GG, where lines with the same letter rhyme. Splits: train (143), dev (12), test (12).

5.2 Evaluation metric

Accuracy was used for both sentiment classification subtasks and paraphrase detection task. For sonnet generation, chrF (15) was used to measure similarity to Shakespeare sonnets.

5.3 Experimental details

5.3.1 Sentiment Classification

For Phase 1 of the project, sentiment classification was performed on 2 datasets. For each dataset model was trained in 2 setups, 1) where full model was trained, 2) where only head layer was trained, with rest of the model frozen. The table below lists the best accuracy scores achieved for both setups, with relevant hyperparameters.

Dataset	Mode	Dropout Probability	BatchSize	LR	Epochs	DevAcc	Baseline Acc
CFIMDB	Full Model	0.4	64	4e-5	3	0.963	0.976
CFIMDB	Head only	0.3	32	4e-5	30	0.837	0.861
SST	Full Model	0.3	64	4e-5	5	0.515	0.513
SST	Head only	0.3	256	3e-4	30	0.450	0.462

Table 1: Training setup and development accuracy for sentiment classification.

Different hyperparameters for classification task in Head-Only mode

In order to understand convergence, batch size and learning rate were varied for different training runs. In all these experiments, other hyperparameters (epochs and dropout probability) were fixed to isolate their effect. The key observation is that when batch size is increased, using the same or lower learning rate, it hurts convergence. While a higher learning rate recovers performance. Intuitively, larger batches produce lower-variance gradients, so the optimizer can take larger stable steps.

Data to support the observation -

1. **Experiment 2 vs 3 (CFIMDB)** - with larger batch size and lower learning rate (Exp 3), both train and dev curves are weaker than Exp 2; see Appendix Figure 3.
2. **Experiment 6 vs 7 (SST)** - with the same learning rate and larger batch size (Exp 7), dev accuracy drops versus Exp 6; see Appendix Figure 4.

3. **Large batch with higher learning rate** - for SST at batch size 256, increasing LR from $1e-5$ (Exp 8) to $3e-4$ (Exp 10) improves dev accuracy from 0.297 to 0.450 (Table 2).

Experiment number	Dataset	Dropout Probability	BatchSize	LR	Epochs	DevAcc
1	CFIMDB	0.3	8	$1e-5$	30	0.673
2	CFIMDB	0.3	32	$4e-5$	30	0.837
3	CFIMDB	0.3	64	$1e-5$	30	0.686
4	CFIMDB	0.3	64	$4e-4$	30	0.935
5	SST	0.3	8	$1e-5$	30	0.414
6	SST	0.3	32	$3e-5$	30	0.423
7	SST	0.3	128	$3e-5$	30	0.392
8	SST	0.3	256	$1e-5$	30	0.297
9	SST	0.3	256	$1e-4$	30	0.422
10	SST	0.3	256	$3e-4$	30	0.450

Table 2: Different Batch size and Learning rate for sentiment classification.

5.3.2 Paraphrase detection and Sonnet generation

The following scores were achieved while fine-tuning the full model (i.e., without LoRA). The model size was GPT-2 (small). These results form the baseline scores for the two tasks.

Task	BatchSize	LR	Epochs	Scores
Paraphrase Detection	200	$1e-5$	6	0.853 (accuracy)
Sonnet Generation	16	$1e-5$	250	41.55 (chrF)

Table 3: Scores for Task 2 and 3.

5.3.3 Paraphrase detection and Sonnet generation with LoRA

LoRA was implemented as described in Section 4.4.2. The model was trained by freezing the base model and updating only LoRA matrices. Observations are listed below:

1. Across both tasks, increasing model size leads to better scores.
2. Final scores with LoRA are slightly lower than full fine-tuning, but still comparable.
3. In LoRA, matrix B is initialized to zeros, so the LoRA path contributes very little at the beginning of training. As optimization progresses, A and B move away from initialization and the LoRA update starts contributing more. This is why scores often dip in early epochs and then recover in later epochs (see Appendix Figure 5).

Task	BatchSize	LR	Epochs	Model Size	Scores
Paraphrase Detection	100	$1e-5$	6	GPT2	0.825 (accuracy)
Paraphrase Detection	100	$1e-5$	6	GPT2-medium	0.830 (accuracy)
Sonnet Generation	16	$1e-5$	250	GPT2	39.20 (chrF)
Sonnet Generation	16	$1e-5$	300	GPT-medium	39.31 (chrF)
Sonnet Generation	16	$1e-5$	300	GPT2-large	40.64 (chrF)

Table 4: Scores with LoRA

6 Error analysis for paraphrase detection

Following analysis were performed for the model configuration reported in Section 5.3.2, with score of 0.853. There were in total 3325 false positive and 2117 false negatives. Identified reasons for false positives -

6.1 Unable to distinguish scope

Example 1 -

- Sentence 1: "What is vaccination?"
- Sentence 2: "What is in a vaccine?"
- True Label: 0
- Predicted Label: 1

Example 2 -

- Sentence 1: "How do deaf-born people think?"
- Sentence 2: "How do people born blind and deaf think?"
- True Label: 0
- Predicted Label: 1

In both examples, there is high lexical overlap between sentences. However, the last few words are different enough to change the meaning. In Example 1, vaccination is a broad term, whereas vaccine is more specific. BPE tokenization for "vaccination" is "vacc", "ination", and for "vaccine" is "vacc", "ine" (see <https://tiktokenizer.vercel.app/?model=gpt2>). The shared token "vacc" may be overemphasized by the model. Similarly, in Example 2, Sentence 2 is broader than Sentence 1. The model fails to distinguish subset/superset relations, and it confuses a general query with a significantly more specific variation.

6.2 Unable to distinguish directionality

Example 1 -

- Sentence 1: "What advantages are there to being a female relative to being a male?"
- Sentence 2: "What advantages are there to being a male relative to being a female?"
- True Label: 0
- Predicted Label: 1

Example 2 -

- Sentence 1: "Why would a high school senior choose MIT over Caltech?"
- Sentence 2: "Why would a high school senior choose Caltech over MIT?"
- True Label: 0
- Predicted Label: 1

In these instances, only female/male and MIT/Caltech switch positions, but that change reverses the meaning of each sentence pair. The bag of words remains nearly identical, and here the causal model behaves similarly to a bag-of-words model. Due to the high overlap of tokens between the two sentences, the model fails to classify correctly.

7 Conclusion

Modern LLMs can perform a variety of NLP tasks. Three such tasks were evaluated in this study with GPT-2 as the base model. For each task, model outputs were interpreted differently: with a

head layer for sentiment analysis, with embedding-space mapping for paraphrase detection, and autoregressively with sampling for sonnet generation. This provided insight into how modern LLMs support different tasks. These experiments also helped build intuition about training dynamics and parameter-efficient fine-tuning. The effects of varying batch size and learning rate helped clarify key aspects of training deep neural networks, including LLMs. Since LLM training is resource-intensive, PEFT methods are important; LoRA is one such approach, and implementing it from scratch provided deeper appreciation of the original work by Hu et al.

Modern LLMs are typically not trained separately for each task; a single model is adapted across many tasks. Improvements on individual tasks can therefore provide broader gains, leading to stronger generalization and more capable models. This is a primary limitation of our work, since the model was fine-tuned separately for each task. Another limitation is how LoRA modules were applied. A large fraction of LLM parameters are in MLP layers (i.e., non-attention blocks), and post-LoRA work has shown that including LoRA modules in MLP layers can improve performance further. This work did not explore that direction. Both are promising areas for future work.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. OpenAI technical report.
- [2] StanfordNLP. Cs224n gpt assignment repository. https://github.com/stanfordnlp/cs224n_gpt, 2026.
- [3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL <https://arxiv.org/abs/2106.09685>.
- [4] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.
- [5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.
- [6] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [7] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmed Al-Dahle, Aiesha Letman, Anish Mathur, Angelica Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.
- [9] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning, December 2020. URL <http://arxiv.org/abs/2012.13255>.
- [10] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019. URL <https://arxiv.org/abs/1902.00751>.
- [11] Stanford CS224N. Cs224n dfp instructions. https://web.stanford.edu/class/cs224n/project/DFP_Instructions.pdf, 2026.

- [12] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2019.
- [13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013.
- [14] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. In *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*, pages 45–52, 2008.
- [15] Maja Popović. chrF: Character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

A Appendix

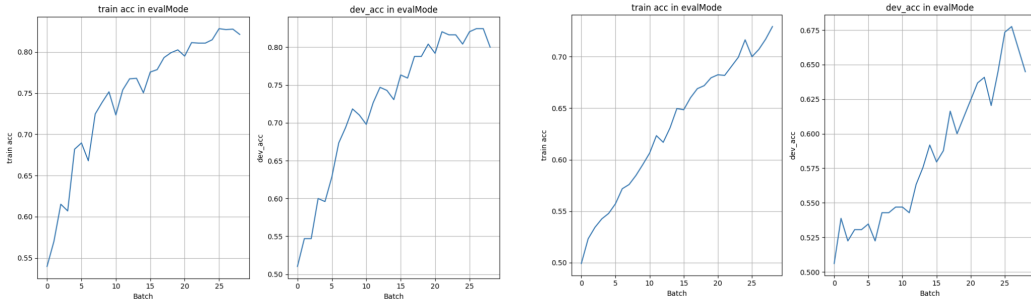


Figure 3: Head-only CFIMDB comparison: Experiment 2 (left) vs Experiment 3 (right). Larger batch with lower LR shows weaker train/dev behavior.

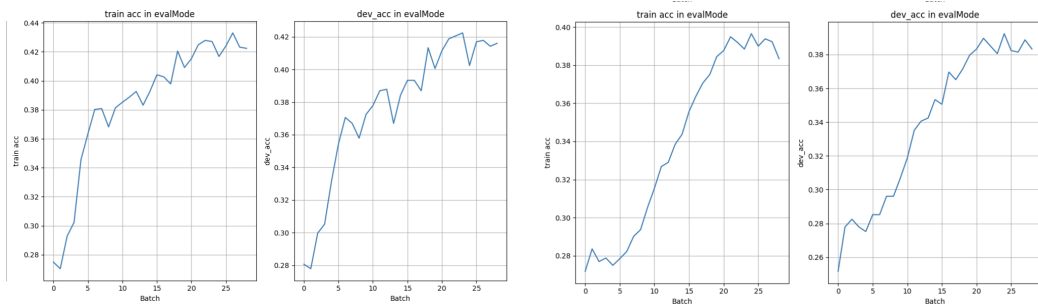


Figure 4: Head-only SST comparison: Experiment 6 (left) vs Experiment 7 (right). Increasing batch size at fixed LR reduces dev accuracy.

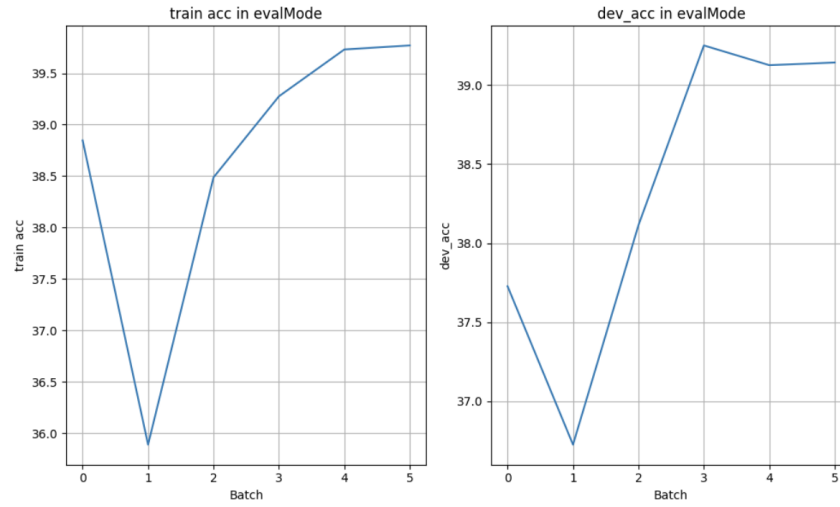


Figure 5: Score dynamics with LoRA across training epochs. Initial dip followed by recovery as LoRA updates become effective.