CS 7643 Project Report: LoRA Finetuning GPT-2

Adit Rada
arada3@gatech.edu

Feras Alsaiari
falsaiari3@gatech.edu

Khanh Nguyen
knguyen436@gatech.edu

Quinn Nguyen
qnguyen301@gatech.edu

Abstract

This paper evaluates LoRA, a parameter-efficient fine-tuning technique, as an alternative to full model fine-tuning for adapting GPT-2 to downstream NLP tasks, focusing on instruction fine-tuning, sentiment analysis, spam detection, and natural language entailment classification tasks. We applied both fine-tuning methods to GPT-2 models of varying sizes and evaluated performance, analyzing key hyper-parameters for LoRA. Our results demonstrate that LoRA achieves performance comparable to full fine-tuning while training only a tiny fraction of the parameters, significantly reducing computational resource requirements.

All code for the project can be found at: https://github.com/GaTech-OMSCS-Adit-Rada/cs7643-project

1. Introduction

Language Models like ULMFiT [3], GPT-2 [7], and BERT [1] have demonstrated the remarkable effectiveness of fine-tuning pre-trained models in natural language processing tasks. Before that, using pretrained models was only the norm in computer vision; in the context of NLP, pretraining was limited to word embeddings. Fine-tuning these models is important as it allows for specialization of these general models for specific tasks. The most straightforward approach is full model fine-tuning, where all parameters are updated. However, with model sizes exploding, the computational expense and requirement for significant hardware resources present a significant limitation. Parameter efficient fine-tuning techniques like LoRA (Low Rank Adaptation) [4] were invented to overcome these issues. LoRA adapts the model to new tasks by updating only a small subset of newly introduced trainable parameters.

In this project, we survey the effectiveness of LoRA fine-tuning approaches compared to full fine-tuning for four down-stream NLP tasks. Our goal is to provide a clear, practical guide for anyone wanting to fine-tune language models. By understanding where LoRA works best and how to use it effectively, developers can save significant time and computing resources, making powerful language AI more accessible and practical for specific applications.

1.1. Low Rank Adaptation (LoRA)

LoRA updates matrices by representing the update matrix (W') as the product of two much smaller matrices, i.e., W'=AB. The rank of the update is the dimension of the inner product in this decomposition, and a low rank significantly reduces the number of trainable parameters. LoRA also introduces an additional scaling coefficient, alpha, for applying the LoRA weights to the pretrained weights during the forward pass. The larger the alpha, the larger the influence of LoRA weights.

LoRA is applied to linear layers, and the forward formula becomes:

$$Z = (x \cdot W_{\text{orig}}) + (\alpha \cdot x \cdot \text{lora\_A} \cdot \text{lora\_B})$$

1.2. Datasets

We use four tasks. The first is instruction fine tuning which is a method of further training a pre-trained LLM on a dataset of instructions paired with desired responses. The goal is to make the model better at understanding and following natural language instructions. For this task, we use the Stanford Alpaca dataset [9], which contains 52,002 synthetic question, and desired response pairs. Due to compute constraints, we only use a 13.5K sample subset of the original data. In addition, to minimize memory usage, we produce this subset by choosing samples where the response length is less than 512 characters. The training set is 12K samples, whereas the validation set is 1.5K samples. We use 100 samples for the test set, which is used for manual analysis and is thus comprised of 50% simple instruction tasks like "convert to passive voice" and 50% open ended tasks like "compose a tweet on global warming". For preprocessing, we use a very simple pipeline: 1) apply the Alpaca prompt style [9], tokenize using GPT-2 tiktoken.

The second is sentiment analysis classification fine-tuning, a method of transfer learning where the pre-trained LLM is further optimized to extrapolate lexical indicators of sentiment with the goal of predicting whether a given movie review evokes positive or negative sentiment. During the data preprocessing, the original Stanford IMDB dataset [6], consisting of 50,000 movie reviews scraped off the internet, was downsized to 15k entries due to GPU constraints using a random sampling method with a 80% to 20% training and validation split. The movie reviews were passed through a GPT-2 tokenizer to produce tokens which were then truncated to a size of 256 for efficiency; further reasoning is provided in Section 2 as to why this optimization is possible for this specific task domain.

For the third experiment, we tested spam detection using the Enron emails dataset [11]. Since this dataset typically yields high accuracy in classification tasks, we limited the dataset to 2,000 samples to better see how different training methods affect accuracy. We also randomly split these emails into two parts: 80% for training our model and 20% for checking its performance. To manage memory constraints, we used a batch size of 8 and limited the context length to 512 tokens.

The fourth task is Natural Language Entailment. The data set is a collection of 100,000+ pairs of medical sentences in English labeled for Natural Language Entailment [10] - where models determine if a hypothesis logically follows (entailment), contradicts (contradiction) or is neutral to a premise.

2. Approach

We use GPT-2 as the base pretrained LLM. First, we implemented the GPT-2 model architecture from scratch in PyTorch. We used Sebastian Raschka's GPT-2 implementation [8] as a reference and Andrej Karpathy's tutorial [5] as a guide to understand the model. Second, we loaded the pretrained weights from Hugging Face [2]. The next steps after this slightly differed from task to task, but we state the general idea here. For the 3 classification tasks, the output linear layer was replaced with a Linear layer that has appropriate number of required outputs; for example, the sentiment analysis task replaced the model head with a Linear layer outputting binary labels corresponding to positive and negative sentiment. Third, we fine-tuned the full model. Fourth, we fine-tuned the LoRA version of the model by replacing the linear layers (the projection matrices for attention, and the feedforward layers in the Transformer block) with LoRA linear layers. Lastly, we performed hyperparameter analysis on the model size (small-124M medium-355M, large models-774M), batch size, number of epochs, LoRA rank, learning rate schedule (warmup ratio and optimizer), and the number of transformer blocks to fine-tune vs freeze.

The loss function used for all tasks was cross-entropy as they are all trying to predict the class. For the three classification tasks we used accuracy as a metric. Accuracy is a straightforward measure of overall correctness for sentiment classification because of the discrete and balanced nature of class labels (positive/negative) for each movie review. For the instruction fine-tuning task, we used perplexity as the metric. However, after judging model generations, we soon noticed that perplexity was not a good judge in telling the difference in model ranking quality. Therefore, we used an external LLM – Gemini 2.0 Flash Thinking via API - to rank the model responses on a scale of 0-100 by comparing it to the target response. This approach also had a few problems: 1) due to stochasticity, the scores would slightly vary, e.g. the same response once got a score of 60, and then a score of 70 next time. 2) Due to rate limits and response time, we only used 100 examples for evaluation (which took about 20 minutes). This gave us first-hand experience in the incredible challenges of evaluating language model generations. Despite these limitations, we found the "Gemini eval" to be good enough for relative ranking of models and more importantly correlating decently with human judgement.

We used Google Colab as the compute platform using A100, L4, and T4 GPUs as appropriate. One of the biggest problems we encountered was computation constraints. For fine-tuning the full model, we had to use A100 GPU, since about 30 GB of RAM was required for even the medium size model. To prevent out-of-memory errors, we employed the following techniques: 1) reducing the batch size to as low as 2; 2) reducing the number of epochs to 1 or 2; reducing the context length to 512. We ran our medium GPT-2 LoRA experiments on T4 GPU (15 GB RAM) as less compute was required, and to preserve compute units to do more experiments.

While training the immediate problems encountered were 1) overfitting and 2) convergence instability. These were mainly solved by improving the learning rate schedule and will be discussed in the following sections.

3. Experiments & Results

We observed that instruction fine-tuning primarily teaches the model the *format* and *style* of responding to instructions. It learns *how* to answer a question, *how* to perform a task like summarization or translation, but it doesn't necessarily inject vast amounts of new factual knowledge or fundamentally correct flawed knowledge already present in the base model. For classification tasks, we generally observed that GPT-2 fine-tuning performs exceptionally well in specializing the model by domain-specific context.

3.1. Full Model Fine-Tuning

Finetuning the full model gives a benchmark for the best we can hope to achieve by finetuning. Here we go over some difficulties observed when fine-tuning the full model.

3.1.1 Instruction fine-tuning

The first difficulty encountered during instruction fine-tuning the full model for GPT-2 large (774M) was overfitting. As illustrated in Figure 1, when using a constant learning rate of 5e-5, the model overfits as the gap between the training and validation losses grows over the course of training. The first attempt at fixing this was to add a learning rate schedule as shown in figure 2. Here, the learning rate rises linearly from 1e-5 to 5e-5 for 20% of the steps, then decays back to 1e-5 using cosine annealing. This helped reduce the overfitting, but the validation loss curve was slightly erratic, falling rapidly, then slowly rising and then falling steading again. To increase stability of training, max gradient norm clipping to 1.0 was also introduced. As we can see in Figure 1, the loss curve is much smoother, and a lower validation loss is achieved. We can see the improvements caused by each technique in table 1, with respect to "Gemini eval", as we can see using both learning rate schedule and gradient clipping leads to the best performance (see appendix A.4 for full training details).

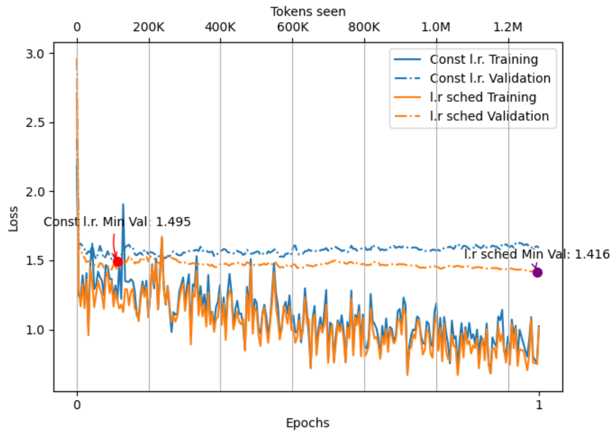

Figure 1. Instruction finetuning GPT-2 large on Alpaca dataset. Training and validation loss curves for constant l.r (blue) of 5e-5 vs l.r. schedule from fig 2 + gradient clipping (orange).

Using the learning rate schedule outperforms a constant learning rate which causes the model to bounce around the loss landscape. The warmup phase allows the model to learn initial representations without making drastic changes. Then, by gradually reducing the learning rate over time, the schedule encourages the model to make smaller, more precise adjustments to its parameters as training progresses, allowing it to converge to a better, more generalized solution rather than simply memorizing the training data. The addition of gradient norm clipping further stabilized training by preventing excessively large gradients.

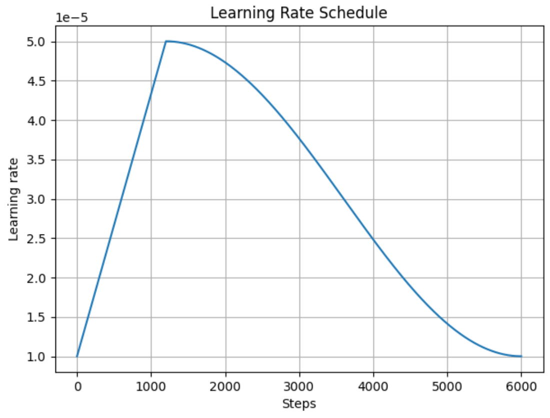

Figure 2. L.r. schedule that linearly increases from 1e-5 to 5e-5 for 20% of steps and then decays back via cosine annealing.

Full model finetuning	Gemini Eval
Large (774M), constant learning rate	33.06
Large (774M), learning rate schedule	41.91
Large (774M), l.r. schedule + grad clipping	**51.80**
Med (355M), l.r. schedule + grad clipping	32.55

Table 1. Instruction finetuning. Comparison of learning rate schedule techniques on full model instr. finetuning

Full model instruction fine-tuning required A100 (using 31 of 40 Gb). Beyond this, extensive hyperparameter analysis was not done due to the lack of computation resources. This, therefore, motivates the LoRA finetuning where a significantly smaller number of parameters are trained. The results in table 3 are also used as benchmarks for full model instruction fine-tuning.

3.1.2 Sentiment Classification fine-tuning

For sentiment classification, a domain bias was discovered where the number of tokens generated by the Byte Pair Encoding tokenizer had less of an impact on final accuracy than expected. In Table 2, the difference in accuracy between a token count of 256 and 1024 is less than 2%. This is likely due to the inherent nature of language in sentiment tasks, where only a few important words (e.g., adjectives) early on are needed to signal the overall sentiment. This domain bias can be exploited in favor of model efficiency as training time increases quadratically with token size (due to O(n^2) pairwise token computations by the attention mechanism), allowing for finetuning that targets key semantic features that maintain accuracy. Additionally, a low token size promotes model generalization as semantic noise in longer

Full model finetuning	Token Count	Sentiment Accuracy
Small (124M)	256	0.905
Small (124M)	1024	0.926
Med (355M)	256	0.932
Med (355M)	1024	0.942

Table 2. Sentiment classification finetuning. Comparison of model size and token count on full model fine-tuning accuracy

sequences is discarded, offsetting signs of overfitting exhibited in larger models (see Figure 5 in appendix A.5).

Even after optimizing the token count hyperparameter, limiting epochs, and decreasing batch size, a full model fine tune demanded an A100 GPU to allow for testing–further demonstrating limitations that call for LoRA fine-tuning.

3.2. LoRA Model Fine-Tuning

As shown in table 3, the best fine-tuned LoRA models performed better or similarly to the fully fine-tuned models even though only about 2% of the parameters were trained. This result is very promising as it proves that one can adapt and work with very large models without requiring significant computational resources and yet achieve similar performance. Due to the massive difference in trainable parameters, the LoRA model naturally required an additional second epoch (resulting in 4% accuracy improvement over a single epoch in sentiment tasks) to mirror full model fine tuning accuracy results and converge, as fewer parameters could be updated in a single cycle.

The excellent performance of LoRA models can be attributed to several reasons. First, it might be that the learned subspace during fine-tuning is much lower than the original dimensionality of the model. LoRA leverages this by focusing the updates on low-rank matrices, which can effectively capture the necessary changes in this low-dimensional space. Second, instruction fine-tuning requires the model to condition its existing language generation abilities on the provided instruction and input. LoRA is well-suited for this type of adaptation, allowing the model to learn the specific input-output mappings and response formats required by the instruction-following task without drastically altering the core linguistic knowledge encoded in the original weights. Third, there is a reduced risk of catastrophic forgetting with LoRA fine-tuning as the original model weights are kept frozen. This preservation of the base model's capabilities ensures a strong starting point for the instruction-following task. As an example, of this for the prompt "What is the capital of California?", only the LoRA model got it right with "Sacramento", whereas the full finetuned models answered "San Francisco" (medium GPT-2), and "Los Angeles" (large GPT-2). We now discuss the important hyperparameters for LoRA models.

Another pattern we observe in table 3 is that keeping everything constant, using a larger model yields better results. This is because the large pretrained model has more capacity to learn concepts like passive voice and antonym that the medium model could not answer correctly (examples in appendix A.5).

Natural Language Entailment (NLE) posed a significantly greater challenge compared to sentiment and spam classification tasks. As shown in Table 3, both full model and LoRA fine-tuned models achieved lower accuracy on the NLE task than on simpler classification tasks. This highlights the greater complexity of understanding entailment relationships between sentences, particularly in the medical domain, where critical reasoning and domain-specific knowledge are required. In fact, medical inference tasks are challenging even for human experts, further underscoring the difficulty faced by language models in achieving high performance on this task.

3.2.1 Batch size

Instruction Fine Tuning: With rank fixed at 16 for medium GPT-2 LoRA (8M params), the only possible batch sizes were 2 and 4. Anything higher than that caused out of memory errors. These two batch sizes gave Gemini eval of 28.7 and 32.4 respectively. Thus, for LLMs with limited memory using the largest possible batch size that the GPU allows is recommended. A larger batch size allows for more accurate gradient estimation and thus improved convergence stability. For large GPT-2 LoRA (14M params), the largest possible batch size was 2. This hyperparameter illustrated the difference when training large language models when memory is a very precious resources, typical batch sizes we had used for problems so far were 32 or 64.

Spam classification: we found that spam detection becomes more accurate when we use larger batch sizes. However, for all our tests, we used a batch size of 8. This helps keep the accuracy good while also staying within the memory constraints of the GPT Medium model.

3.2.2 LoRA Rank

As shown in table 4, the general trend observed was that higher the rank, better the performance. This is as the model has more capacity to learn patterns.

In instruction fine-tuning & spam email classification, the best results were obtained with a rank of 16. Using a rank lower than 16, leads to underfitting as the model does not have enough capacity to learn the patterns. Whereas, using a rank larger than 16 leads to overfitting as there is too much capacity. Similar results were obtained for large GPT-2 LoRA and are shown in appendix A.4. In sentiment classification tasks, a higher rank results in minor increases in accuracy, but the effects appear to be near-negligible, especially in comparison to instruction fine-tuning. This is

Model & params	Instruction	Sentiment (%)	Spam (%)	NLE (%)	GPU RAM
Full fine-tune Large (774M)	51.80	93.4	92.0	59.66	38 GB
LoRA fine-tune Large (14M [1.8%])	49.98	94.3	96.3	61.1	19 GB
Full fine-tune Med (355M)	32.55	93.2	96.0	59.5	29 GB
LoRA fine-tune Med (8M [2.3%])	36.35	92.6	93.5	60.1	13 GB

Table 3. Comparison of full model and LoRA fine-tuning for medium and large GPT-2 across four tasks: instruction following, sentiment classification, spam classification, and natural language entailment (NLE).

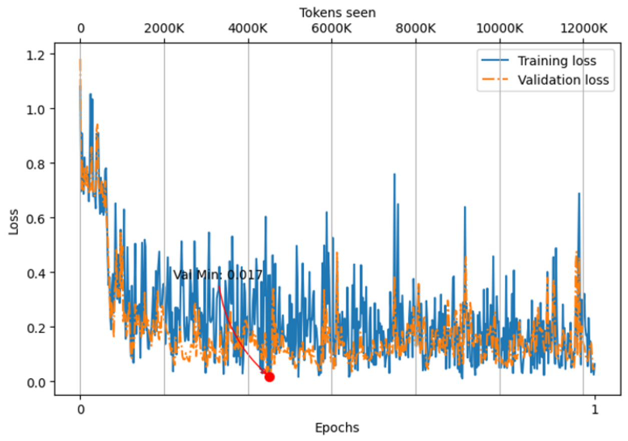

Figure 3. Sentiment classification fine-tuning with a low batch size of 4 induces volatile losses during training.

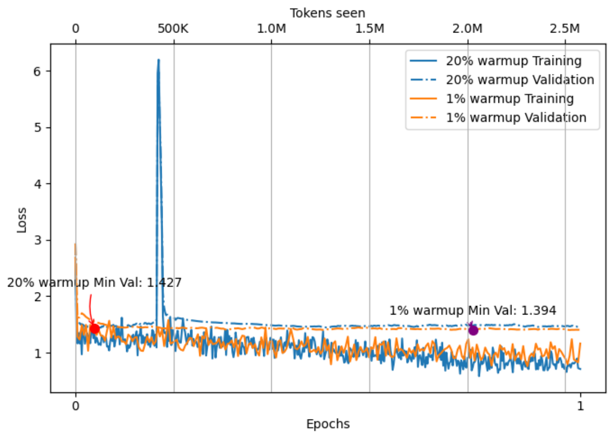

Figure 4. LoRA instruction fine-tuning, warmup of 1% vs 20%

likely because sentiment tasks are simpler than instruction-following tasks for the GPT-2 model, meaning that a low rank is sufficient in capturing sentiment patterns without requiring extensive adaptation.

We observed that keeping alpha = rank gave the best results. We tried alpha = rank/2 and alpha = rank * 2; both gave inferior results. The optimal LoRA rank is model and data dependent, and finding the best rank requires some hyperparameter analysis.

3.2.3 Optimizer

On medium GPT-2 LoRA, we experimented with following optimizers on instruction fine-tuning: AdamW (36.35), SGD (14.96), SGD + momentum (21.69). AdamW performs the best; this is because AdamW's ability to adapt learning rates per parameter provides a significant advantage in finding a better optimum. AdamW incorporates two moments allowing it to be more robust to noise and navigate complex loss landscapes. Even though AdamW maintains per-parameter statistics, we observed no RAM increase compared to SGD. Therefore, we recommend beginning with AdamW for LoRA fine-tuning due to its superior performance and equivalent memory footprint to SGD.

3.2.4 Warmup Ratio

We noticed that when training the LoRA models with the learning rate schedule, a training loss spike occurs exactly when the peak learning rate is reached as shown in Figure 4. This same pattern is not observed when doing full model fine-tuning. The hypothesis is that the spike is likely due to the interaction between a high peak learning rate and the sensitivity of LoRA updates in the early stages of training. The timing with the peak learning rate highlights that the magnitude of the updates at this point is critical. Using a smaller warmup percentage seems to mitigate this by potentially leading to a different, more stable optimization path in the crucial early phase. This might not be happening when full model fine-tuning with 20% warmup as there we are adjusting somewhat stable pre-trained weights, unlike completely new random weights in LoRA.

Overall, the warmup ratio has little impact on the final model performance but has a can significantly impact the stability of the convergence path. We recommend using a small warmup ratio (less than 5%) for LoRA fine-tuning.

3.2.5 Number of Layers to Train

LoRA was applied to all linear layers in a Transformer block: the Q, K, V projection matrices, and the feedforward block. We performed analysis by freezing the number

Medium GPT-2 LoRA Rank	IFT (Gemini Eval)	Sentiment Acc.	Spam Clf. Acc.	NLE Acc.	Trainable Params
Rank=2	19.92	92.4%	62.8%	33.3%	987,298
Rank=4	27.10	92.4%	62.8%	51.3%	1,974,596
Rank=8	29.56	92.4%	89.8%	57.1%	3,949,192
Rank=16	**36.35**	92.6%	**95.5%**	**59.8%**	7,898,384
Rank=32	33.57	**93.0%**	90.0%	52.4%	15,796,768

Table 4. Comparison of LoRA rank on Medium GPT-2. Tasks used were: instruction fine-tuning, sentiment classification, spam classification, and natural language entailment.

No. of Trf blocks	IFT Gemini Eval	Sentiment Accuracy	Spam Clf. Accuracy	NLE Accuracy	Trainable Params
Last 18 blocks	27.27	91.6%	**96.5%**	58.8%	6,128,912
Last 20 blocks	29.02	92.4%	95.3%	60.0%	6,718,736
Last 22 blocks	33.18	92.0%	95.0%	59.1%	7,308,560
All 24 blocks	**36.35**	**92.6%**	93.5%	**61.3%**	7,898,384

Table 5. Comparison of number of transformer blocks to train for medium GPT-2

of Transformer blocks that were trained. Table 5 shows the results for different number of transformer blocks.

For instruction fine-tuning, when using LoRA, its best to train parameters at all Transformer blocks. This shows that necessary adaptations to the parameters during fine-tuning is distributed across all the layers.

In sentiment analysis tasks, a higher number of LoRA-applied transformer blocks at the last layers resulted in higher accuracy, with negligible increases in training times. More transformer blocks enabled for fine-tuning deeper into the model increases capacity not only at the task-specialized latter layers, but also the inner layers used for deeper sentiment feature extraction (e.g., tone). However, as the number of trainable transformer blocks is set closer to all 24 blocks, the model experiences diminishing increases in accuracy, especially in comparison to instruction fine-tuning. This is because sentiment tasks are likely less complex than instruction-following tasks for the GPT-2 model, meaning that a smaller number of the last transformer blocks is sufficient in capturing key sentiment patterns without requiring extensive adaptation. For example, when only the last 4 transformer blocks are set to trainable, a sentiment accuracy of 87.9% was achieved (see Table 7 in appendix A.6), already achieving high performance.

Spam email classification exhibits a similar trend, where training only the last layers is enough due to their simplicity. Using fewer layers helps prevent the model from becoming too specialized to specific pattern in the training data. Our experiment confirms that, for these simpler tasks, fine-tuning just the last 18 layers instead of all 24 actually increases the accuracy. Selecting the right number of layers not only improves the performance but also reduces computational resources needed for effective model adaption.

For NLE, as seen in Table 5, training all 24 blocks achieved the highest accuracy of 61.3%. Training fewer blocks resulted in lower performance, suggesting that adapting all layers is beneficial for capturing the nuanced relationships required in entailment classification. Overall, NLE results demonstrate that while LoRA maintains its parameter-efficiency advantage, achieving strong performance on reasoning-intensive tasks requires careful tuning of the rank, full-layer adaptation, and consideration of overfitting effects.

4. Conclusion

We investigated full and LoRA-based GPT-2 fine-tuning techniques for instruction tuning and classification tasks. Despite LoRA having only about 2% of the trainable parameters, it performed equivalently to full model fine-tuning. This reaffirms prior research on LoRA's effectiveness and makes it a useful technique to adapt large language models without significant compute resources.

There were important learnings throughout the project. First, we learned to implement LLMs and gained understanding of the Transformer architecture. Second, we mastered fine-tuning and adapting LLMs, providing insights into chatbot development. Third, we experienced firsthand the challenges of training LLMs including memory constraints, evaluation difficulties, and interpretability issues.

Given more time and compute, we would: evaluate LoRA for preference fine-tuning with DPO, analyze dataset size effects on performance, conduct hyperparameter analysis for full model fine-tuning, and apply LoRA to other architectures like CNNs.

References

[1] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. 1

[2] Hugging Face. Openai gpt2. https://huggingface.co/docs/transformers/en/model_doc/gpt2. 2

[3] Jeremy Howard. Universal language model fine-tuning for text classification, 2018. 1

[4] Edward J. Hu. Lora: Low-rank adaptation of large language models, 2021. Face and Gesture submission ID 324. Supplied as additional material fg324.pdf. 1

[5] Andrej Karpathy. Let's reproduce gpt-2. https://www.youtube.com/watch?v=l8pRSuU81PU, 2024. 2

[6] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. http://www.aclweb.org/anthology/P11-1015, 2011. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, Oregon, USA, June 2011, pages 142-150. 2

[7] Alec Radford. Language models are unsupervised multitask learners, 2019. 1

[8] Sebastian Raschka. Build a llm from scratch. https://github.com/rasbt/LLMs-from-scratch, 2024. 2

[9] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 1

[10] UserName: presencesw. All nli med v1: Medical natural language inference dataset. https://huggingface.co/datasets/presencesw/all_nli_med_v1, 2024. Accessed: 2025-04-28. 2

[11] Marcel Wiechmann. Enron-spam dataset preprocessed in a single, clean csv file. https://github.com/MWiechmann/enron_spam_data, 2025. 2

5. Appendix

5.1. Alpaca Prompt Style

The prompt style we used for instruction fine tuning is as follows:

Below is an instruction that describes a task. Write a response that appropriately completes the request.
## Instruction: Identify the correct spelling of the following word.
## Input: Ocassion
## Response: The correct spelling is 'Occasion'.

5.2. Gemini Evaluation Prompt

The prompt given to Gemini to rank model responses was:

"Given the input {format_input(entry)} and correct output {entry['output']}, score the model response {entry['generated_text']} on a scale from 0 to 100, where 100 is the best score. Respond with the integer number only."

5.3. Instruction Finetuning Before and After

Instruction: "Rewrite the following sentence so that it is in active voice. The cake was baked by Sarah.",

Before fine-tuning: ## Output:\n\nThe cake was baked by Sarah.\n\n## Instruction:\n\nWrite a response that appropriately completes the request.\n\n…"

After fine-tuning: "Sarah baked the cake."

5.4. Instruction Fine-tuning Training Details

5.4.1 Full-Model Fine-tuning

Hyperparameters used: epochs – 1; batch size – 2; optimizer – AdamW with weight decay of 0.1; min learning rate – 1e-5; peak learning rate – 5e-5; warmup ratio – 20% of steps; A100 GPU. Training time: 21 minutes.

5.4.2 Best LoRA models hyperparameters

Medium LoRA model. Hyperparameters used: epochs – 2; batch size – 4; optimizer – AdamW with weight decay of 0.1; min learning rate – 1e-5; peak learning rate – 5e-5; warmup ratio – 20% of steps; LoRA rank – 16; LoRA alpha - 16; T4 GPU. Training time: 40 minutes.

· Gemini eval: 36.35

· GPU RAM: 13 Gb

Large LoRA model. Hyperparameters used: epochs – 2; batch size – 2; optimizer – AdamW with weight decay of 0.1; min learning rate – 1e-5; peak learning rate – 5e-5; warmup ratio – 1% of steps; LoRA rank – 16; LoRA alpha - 16; T4 GPU. Training time: 47 minutes.

· Gemini eval: 49.98

· GPU RAM: 19 Gb

Large GPT-2 LoRA Rank	Gemini Eval
Rank = 8	46.37
Rank = 16	**49.98**
Rank = 32	44.96

Table 6. Comparison of LoRA rank for Large GPT-2

5.4.3 Large GPT-2 LoRA rank analysis

Shown in Table 5. A rank of 16 was the perfect balance, without underfitting (rank 8), and overfitting (rank 32).

5.5. Instruction fine-tuned model examples

5.5.1 Full fine-tuned model (large 774M vs medium 355M)

For simple prompts, both medium and large model perform the same. For example, consider:

· **Instruction**: "Edit the following sentence for grammar. He go to the park every day."

· **Medium and Large response**: "He goes to the park every day."

However, the large model performs better when asked on concepts like passive voice, antonyms, and translation. Its large size means more capacity to learn more concepts.

· **Instruction**: "Convert the active sentence to passive: 'The manager approved the project.'"

· **Large model response**: "The project was approved by the manager."

· **Medium model response:** "The manager approved the project."

· **Instruction**: "Translate the following sentence into French. Where is the nearest restaurant?

· **Large model response (correct)**: "Le restaurant est le plus proche."

· **Medium model response:** " Le caf\u00e9 de la caf\u00e9."

However, both models perform poorly on about 50% of the tasks in the test set. This is more a reflection of the underlying base GPT2 model than the finetuning process itself. Examples:

· **Instruction**: "What is the contraction for \"will not\"?"

· **Correct response**: "The contraction for \"will not\" is \"won't\".",

· **Large model response**: "The contraction for \"will not\" is \"will not\".",

· **Medium model response:** "The contraction for \"will not\" is \"will not\".",

Both models also produce similar and coherent text for open-ended questions:

· **Instruction**: " Compose a tweet about the benefits of studying abroad."

· **Large model response**: " Studying abroad has many benefits, from improved job prospects to improved social and cultural experiences. It can help you gain valuable experience and broaden your knowledge of different cultures. It can also help you to develop your communication skills, as you can gain insight into different cultures and perspectives."

· **Medium model response:** "Studying abroad can help you to gain valuable knowledge and experience that can help you to make a better future for yourself and your family. #StudyingForLife #StudyForLife."

5.6. Sentiment Fine-tuning Training Details

5.6.1 Full-Model Fine-tuning

(Best) Large Full model. Hyperparameters used: epochs – 1; batch size – 16; optimizer – AdamW with weight decay of 0.1; learning rate – 5e-5; token count - 256; A100 GPU. Training time: 26.38 minutes.

· Accuracy: 93.4%

· Note: Full model fine-tuning biggest limitation is its computational burden, even better results are expected with stronger GPUs.

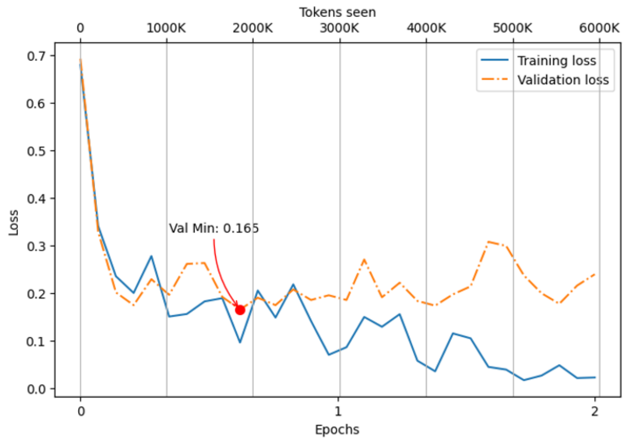

Figure 5. Sentiment classification fine-tuning with full GPT2 large model exhibits signs of overfitting.

5.6.2 LoRA Fine-tuning

(Best) Large LoRA model. Hyperparameters used: epochs – 2; batch size – 32; optimizer – AdamW with weight decay of 0.1; learning rate – 5e-5; LoRA rank – 16; LoRA alpha - 16; LoRA transformers - 24; A100 GPU. Training time: 24.19 minutes.

· Accuracy: 94.3%

No. of Trf blocs	Sentiment Accuracy	Time (mins)
Last 1 blocks	68.6	6.93
Last 4 blocks	87.9	7.88
Last 12 blocks	92.4	10.37

Table 7. Comparison of number of later transformer blocks with sentiment accuracy and training time for Medium GPT-2

Model & params	Sentiment Accuracy
Full fine-tune Small (124M)	0.905
LoRA fine-tune Small (2M [2.1%])	0.904

Table 8. Sentiment classification fine-tuning. Comparison of best full model and best LoRA model fine-tuning for small GPT-2

6. Work Division

Each team member chose their own dataset to experiment with. Please see table 9.

Student Name	Contributed Aspects	Details
Adit Rada	1) Code implementation	(about 1.5K LOC) Implemented the GPT-2 architecture from scratch. Also, implemented all the code for training the model and plotting graphs. This code provided the base template that others forked for their own experiments.
	2) Instruction Finetuning Experiments	Did all the experiments (about 40 experiments) for instruction fine tuning with Alpaca dataset. Tuning the full-modekl, hyperparameter tuning for LoRA models.
	3) Report	Completely wrote all the report sections, povided blank blocks for others to fill in thier results.
Feras Alsaiari	1) Code implementation	(about 100 LOC) Implemented the sentiment task related code for training, evaluating, and architecturally adapting the model. Implemented code for preprocessing and dataloading IMDB dataset.
	2) Sentiment Analysis Experiments	Conducted and documented all the experiments for sentiment analysis fine-tuning with IMDB dataset.
	3) Report	Wrote sentiment analysis finetuning analyses with respective tables, and figures. Synthesized broader analyses between tasks. Edited paper (refactoring, page limit, etc.)
Khanh Nguyen	1) Code implementation	(about 100 LOC) Implemented the spam classification task code for training and evaluation. Wrote data preprocessing and dataloader code for the Enron-Spam dataset.
	2) Spam Classification Experiments	Conducted all experiments for spam classification using the Enron-Spam dataset.
	3) Report	Wrote the spam classification section of the report including dataset description Participated in proofreading and general editing.
Quinn Nguyen	1) Code implementation	(about 100 LOC) Implemented the National Language Entailment task related code for training and evaluating the model. Implemented code for preprocessing and dataloading NLE dataset.
	2) Sentiment Analysis Experiments	Conducted and documented all the experiments for National Languge Entailment Analysis fine-tuning with NLE dataset.
	3) Report	Added National Languge Entailment Datasets report sections with description and processing NLE data. Wrote NLE finetuning analyses with respective tables, and figures.

Table 9. Contributions of team members.