**Assignment: Fine-Tuning a Small Language Model for a Specific Task**

**Introduction**  
In recent years, large language models (LLMs) have revolutionized natural language processing (NLP), enabling high performance across a wide range of downstream tasks. However, fine-tuning such models can be resource-intensive, especially when working with models exceeding hundreds of millions of parameters. This assignment focuses on applying *Parameter-Efficient Fine-Tuning* (PEFT) methods to adapt a large pre-trained language model (~1 billion parameters) to a specific NLP task — sentiment analysis on product reviews — while minimizing computational costs. We leverage **Low-Rank Adaptation (LoRA)**, a PEFT technique that introduces minimal additional trainable parameters, making large-scale model adaptation feasible in limited-resource environments. Our goal is to assess how well such techniques allow for task-specific adaptation without full model retraining.

**Model and Dataset Selection**

To carry out this task, we selected the **amazon\_polarity** dataset from Hugging Face Datasets, a diverse collection of labeled Amazon reviews covering product categories such as books, electronics and clothing. This dataset consists of three variables of 3.6 million observations. The title and content variables are review text fields and the label variable specifies the sentiment of each review as (1 = Positive, 0 = Negative). To stay within the scope of the assignment and training constraints, we will work from a curated subset of ~10,000–20,000 examples. We selected this dataset as it offers a realistic challenge for generalization due to its variability in writing style, tone, and product domains. Furthermore, it is well-balanced and easy to parse for binary sentiment classification.

For our model, we chose **EleutherAI’s pythia-1b-deduped**, a 1-billion parameter autoregressive transformer model that aligns with PEFT research and supports LoRA-based fine-tuning. The following attributes of the model provide a justification for its selection:

* **Size**: ~1 billion parameters — aligns with PEFT goals for large-model adaptation.
* **Architecture**: Decoder-only autoregressive transformer (GPT-style).
* **License**: Apache 2.0 — open and suitable for academic use.
* **Compatibility**: Works with bitsandbytes for 4-bit quantization and LoRA-based fine-tuning.
* **Proven Utility**: Frequently used in instruction-tuning, causal language modeling, and downstream task adaptation.

Together, this setup provides a meaningful testbed for evaluating the efficiency and effectiveness of large model adaptation on sentiment classification.

**Methodology**

This project fine-tunes a pre-trained language model for instruction-style sentiment classification using parameter-efficient fine-tuning (PEFT) with Low-Rank Adaptation (LoRA). The key components of the methodology are as follows:

**Model and Tokenizer**

* **Base Model:**  
  We used the EleutherAI/pythia-1b-deduped model for its compatibility with causal language modeling and instruction-following tasks.
* **Tokenizer Configuration:**  
  Tokenizer was loaded with fallback for missing padding tokens (using the EOS token) and set to pad sequences on the right.

**Parameter-Efficient Fine-Tuning (PEFT)**

* **Quantization (optional):**The code optionally supports 4-bit quantized loading using the bitsandbytes library, although standard precision fallback was used in the current run due to package availability.
* **LoRA Configuration:**We used the peft library to apply Low-Rank Adaptation (LoRA), enabling fine-tuning by injecting trainable adapters into specific attention modules (query\_key\_value for Pythia). LoRA parameters included:
  + r = 16 (rank of adaptation matrices)
  + alpha = 32
  + dropout = 0.05
  + bias = "none"
* **Model Wrapping and Trainable Parameters:**The LoRA adapters were applied using get\_peft\_model, and only ~0.2% of the total parameters were made trainable, dramatically reducing resource needs.

**Training Strategy**

* **Manual Training Loop:**A custom training loop was implemented to support:
* Gradient accumulation (gradient\_accumulation\_steps = 4)
* Custom learning rate scheduler (get\_linear\_schedule\_with\_warmup)
* Optimizer: AdamW with weight decay of 0.01
* Logging and checkpointing
* Reproducibility using random.seed(42)
* **Hyperparameters**

|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| Epochs | 1 |
| Batch size | 4 (GPU) or 2 |
| Learning rate | 2e-4 |
| Warmup steps | 50 |
| Max sequence steps | 512 tokens |

**Hyperparameter Tuning**

After initial fine-tuning with a base configuration, we performed hyperparameter tuning to optimize model performance. We experimented with variations in learning rate, batch size, LoRA rank (r), and dropout rate. These experiments are described in detail in the Hyperparameter Tuning section of Results.

**Data Preparation**

We began by curating and preprocessing the raw dataset to ensure it was suitable for instruction-tuned causal language modeling. Key steps in the data preparation pipeline include:

* **Random Seeding and Subsampling:**  
  A random seed (random.seed(42)) was set for reproducibility. The dataset was then randomly shuffled using HuggingFace’s .shuffle(seed=42) and downsampled to 10,000 examples for faster experimentation.
* **Label Mapping:**  
  Numerical sentiment labels (0 and 1) were mapped to human-readable classes: “negative” and “positive”.
* **Instruction Prompt Formatting:**  
  Each data instance was reformatted into an instruction-following prompt format to align with instruction-tuned fine-tuning practices. This includes an Instruction, Input, and Response block concatenated into a single text field.
* **Tokenization and Label Creation:**  
  Using the tokenizer from EleutherAI/pythia-1b-deduped, we tokenized each prompt to a maximum length of 512 tokens, applying truncation and padding where necessary. The labels used for training were created by duplicating the input\_ids (standard for causal language modeling tasks).
* **Wrapping in PyTorch Dataset:**  
  A custom PyTorch dataset class (CausalLMDataset) was defined to convert the tokenized dataset into a format suitable for PyTorch DataLoader.

**Evaluation Strategy and Metrics**

To evaluate the downstream performance of our instruction-tuned model on sentiment classification, we employed both generative and classification-based evaluation metrics, taking into account the unique autoregressive nature of decoder-only transformer architectures.

**Generation Decoding and Postprocessing**

Since our model predicts sentiment in a text-to-text fashion (e.g., generating “positive” or “negative” as raw tokens), evaluation required decoding raw outputs and mapping them back into discrete sentiment classes. Generation was conducted using greedy decoding (num\_beams=1, do\_sample=False) to preserve deterministic output behavior and facilitate reproducibility. Only the text following the ### Response: delimiter in the generated output was extracted and normalized (lowercased, stripped of whitespace) before comparison.

Outputs were mapped to binary class labels as follows:

* “positive” → 1
* “negative” → 0

Outputs that did not match either expected label exactly were treated as invalid and excluded from accuracy and F1 score computations but included in coverage/error rate analysis.

**Evaluation Metrics**

We employed multiple evaluation metrics to assess the model’s performance from different perspectives:

* **Accuracy**: The proportion of correctly predicted sentiment labels among all valid outputs.
* **Macro-Averaged Precision, Recall, and F1 Score**: These metrics were computed to account for class imbalance and provide a class-agnostic measure of model quality.
* **Exact Match Rate (EMR)**: The proportion of predictions where the generated string exactly matched one of the expected sentiment labels (“positive” or “negative”). This acts as a proxy for generation precision in text-to-text tasks.
* **Invalid Output Rate**: We additionally tracked the proportion of model outputs that failed to resolve into valid classes (e.g., hallucinated or malformed responses). This metric is critical for assessing robustness and decoding reliability.

**Dataset Splits and Inference Pipeline**

The original dataset was split into training (80%), validation (10%), and test (10%) subsets using stratified sampling to preserve sentiment class proportions. Evaluation was conducted on the held-out test set using a separate inference pipeline that emulated a real-world deployment: prompting the model, generating output, and extracting the label post hoc from the generated sequence.

We used Hugging Face’s Trainer interface with a custom compute\_metrics function, enabling real-time reporting of all core metrics at each evaluation checkpoint. All metrics were computed using scikit-learn’s classification\_report utilities, ensuring consistency with standard evaluation protocols.

**Results**

***Base Model Performance: LoRA-injected Language Model***

**Model Setup:**

* AdamW optimizer
* get\_linear\_schedule\_with\_warmup learning rate scheduler
* Gradient accumulation (simulate larger batches with small real batch size)
* Logging every 10 steps (after accumulation)
* Saving checkpoints at the end of each epoch

**Training Settings**

|  |  |
| --- | --- |
| epochs | 1 |
| batch size | 4 |
| gradient accumulation steps | 4 |
| learning rate | 2e-4 |
| warm up steps | 50 |
| logging steps | 10 |

After fine-tuning the base model using LoRA, we evaluated its performance on a held-out set of 50 test examples. Each example followed an instruction-style prompting format, containing:

* An **instruction prompt**
* A **product review** (composed of title and content)
* A **labeled sentiment response** ("positive" or "negative")

During evaluation, the model was asked to generate the sentiment label based only on the instruction and review, without access to the ground truth. Predictions were considered correct if the model-generated response included the correct label.

**Training Loss Observations**

Throughout fine-tuning, the training loss showed a steady downward trend, indicating that the model was successfully learning from the provided examples. However, occasional plateaus and fluctuations were observed, suggesting that the model had difficulty generalizing to examples with subtle or ambiguous sentiment cues.

**Inference and Sample Output Observations**

Key insights from sample generations include:

* **Correct Predictions**:  
  The model correctly classified 15 out of 50 examples, achieving an initial **sample accuracy of 30%**.
* **Common Error Patterns**:
  + **Incomplete Outputs**: The model often repeated the prompt format (e.g., ### input:) without providing a clear sentiment label.
  + **Misclassifications**: There was a notable tendency to mislabel **negative** reviews as **positive**, indicating a potential bias in generation.
  + **Handling of Ambiguity**: Reviews with subtle or mixed tones were especially challenging for the model, leading to inconsistent or incorrect predictions.

**Metrics Summary**

* Sample Accuracy (first 50 examples): 30%
* Sample F1 Score (first 50 examples): 0.68

**Confusion Matrix**

A blue squares with white text

AI-generated content may be incorrect.

***Hyperparameter-Tuning Run 1***

In this run, we modify key LoRA and optimizer parameters. These values increase the adapter capacity and learning aggressiveness. The goal is to observe whether this leads to faster convergence and improved evaluation accuracy.

**Model Setup**

* AdamW optimizer
* get\_linear\_schedule\_with\_warmup learning rate scheduler
* Gradient accumulation (simulate larger batches with small real batch size)
* Logging every 10 steps (after accumulation)
* Saving checkpoints at the end of each epoch

**Training Settings**

|  |  |
| --- | --- |
| epochs | 1 |
| batch size | 4 |
| gradient accumulation steps | 4 |
| learning rate | 3e-4 |
| warm up steps | 50 |
| logging steps | 10 |
| LoRA Rank (r) | 32 |
| LoRA Alpha | 64 |
| LoRA Dropout | 0.1 |

**Training Loss Observations**

* Training started with high initial losses (~14–15).
* **Steady and significant decline across the epoch:**
  + Loss dropped below 10 after ~50 steps.
  + Continued smooth decrease to ~3.2 by Step 200.
* **Training Curve Behavior:** Smooth, no major instability spikes observed. Loss curve shows good downward trajectory, suggesting effective learning at this LR and LoRA capacity.

**Inference**

After being evaluated on 50 unseen test samples, the model shows significant improvement, particularly in capturing positive sentiment more reliably. However, it still struggled to correctly label negative examples, often overpredicting "positive". The relatively high F1 Score (0.75) compared to the raw accuracy suggests that when the model predicts correctly, its confidence and precision are reasonably strong. The aggressive learning rate combined with larger LoRA rank likely helped the model specialize faster, but with some residual bias toward positive outputs.

**Metrics Summary**

Sample Accuracy (first 50 samples): 62%

Sample F1 Score (first 50 samples): 0.75

Confusion Matrix

A blue squares with white text

AI-generated content may be incorrect.

***Hyperparameter-Tuning Run 2***

This run explores the effects of a lighter and more stable configuration. The goal is to test whether a leaner setup can still generalize well and improve or match the accuracy achieved in Run 1.

**Model Setup**:

* **Base Model**: **Base Model**: Pretrained causal language model (AutoModelForCausalLM)
* **Training Type**: Fine-tuning with LoRA (Low-Rank Adaptation) applied
* **LoRA Configuration**
  + Rank (r): 8
  + Alpha: 16
  + Dropout: 0.01

**Training Settings:**

|  |  |
| --- | --- |
| epochs | 1 |
| batch size | 4 |
| gradient accumulation steps | 4 |
| learning rate | 1e-4 |
| warm up steps | 50 |
| scheduler | linear decay with warmup |

**Training Loss Observations**

Starting Loss: Very high (~14.2 at Step 10-20).

Steady Decline:

* Loss gradually decreased from ~14 to ~6-7 by 30%-40% of the way through training.

Later Phase:

* Loss further decreased toward ~5.9 by Step 230.
* Smooth, consistent downward trend.

Training Behavior:

* No major instability or spikes.
* Loss reduction looks relatively gradual, confirming **good learning stability**.

**Inference**

Run 2 underfit the training data severely. It lacked both capacity (small LoRA rank, low alpha) and training signal (low learning rate + short training time). This highlights a tradeoff: extreme parameter efficiency may lead to unacceptable performance loss when not supported by longer training or stronger optimization.

***Hyperparameter-Tuning Run 3***

Run 3 builds directly on Run 1 (which achieved 70% accuracy) and attempts to improve generalization by slightly reducing regularization strength and slowing down the learning rate. The goal was to allow more stable learning without drastically altering model capacity. The model itself remains the same architecture — a causal language model fine-tuned with LoRA (Low-Rank Adaptation).

**Model Setup:**

* **Base Model**: **Base Model**: Pretrained causal language model (AutoModelForCausalLM)
* **Training Type**: Fine-tuning with LoRA (Low-Rank Adaptation) applied
* **LoRA Configuration**
  + Rank (r): 32
  + Alpha: 64
  + Dropout: 0.05 (reduced from 0.1 in Run 1)

**Training Settings**:

|  |  |
| --- | --- |
| epochs | 1 |
| batch size | 4 |
| gradient accumulation steps | 4 |
| learning rate | 2e-4 |
| warm up steps | 50 |
| scheduler | linear with warmup |

**Training Loss Observations**:

Training progressed very smoothly with a clear, steady downward trend in loss, which suggests that the lowered learning rate and dropout did indeed help stabilize training:

|  |  |
| --- | --- |
| Step | Loss |
| 10 | 14.3251 |
| 20 | 14.2433 |
| 30 | 13.9411 |
| 50 | 11.6407 |
| 70 | 8.7170 |
| 100 | 6.3916 |
| 150 | 4.5755 |
| 200 | 3.6815 |
| 250 | 3.1264 |

* **Pattern**: Rapid loss decline early on (first 1000 steps) followed by more gradual reductions — indicating learning stabilization rather than instability or catastrophic forgetting.
* **Average Loss at End of Epoch**: Continued downward trend (full final average wasn't listed, but based on Step 250 it was around 3.1).

**Inference**:

While Run 3 produced the lowest training loss, it fell short of Run 1’s 70% accuracy, possibly due to reduced regularization. This suggests that Run 1 achieved the best balance between learning and generalization.

**Metrics Summary**:

Final Loss: 1.6484

Accuracy (10 samples): 60%

***Hyperparameter-Tuning Run 4***

With Run 4, we extend Run 1’s configuration to 2 epochs. This will help the model converge more fully and potentially push accuracy above the current best (70%).

**Model Setup:**

* **Base Model**: EleutherAI/pythia-1b-deduped
* **Training Type**: Fine-tuning with LoRA (Low-Rank Adaptation) applied
* **LoRA Configuration**
  + Rank (r): 32
  + Alpha: 64
  + Dropout: 0.1

**Training Settings:**

|  |  |
| --- | --- |
| epochs | 2 |
| batch size | 4 |
| gradient accumulation steps | 4 |
| learning rate | 3e-4 |
| warm up steps | 50 |
| logging steps | 10 |
| scheduler | linear with warmup |

**Training Loss Observations**:As the training progresses, the loss steadily decreases, starting at a high value of approximately 14.31 at the first batch of Epoch 1, and continuously dropping throughout the epoch. By the time the model reaches the later stages of Epoch 1, the loss has significantly reduced to around 1.8.

Here’s a sample of the loss trend observed in Epoch 1:

|  |  |
| --- | --- |
| Step | Loss |
| 10 | 14.3110 |
| 50 | 8.6651 |
| 100 | 4.7348 |
| 150 | 3.3944 |
| 200 | 2.7303 |
| 250 | 2.3251 |
| 300 | 2.0127 |

This gradual reduction in training loss indicates that the model is learning effectively from the training data.

**Inference**

In comparing the results of Run 4 to Run 1, we found that although both runs achieved a sample accuracy of 62%, there were notable differences in performance dynamics. Run 4 incorporated changes in its training setup, including the use of 2 epochs (as opposed to 1 in Run 1) and targeting different LoRA modules (query\_key\_value, dense\_h\_to\_4h) instead of just q\_proj and v\_proj. Despite these modifications, the overall accuracy did not see a significant improvement beyond Run 1, indicating a performance plateau. However, the F1 score remained consistent at 0.75, suggesting that the model's ability to correctly detect positive reviews remained stable. One challenge that persisted, similar to Run 1, was the misclassification of negative reviews, where the model occasionally predicted positive sentiment for these instances, highlighting an area for improvement in handling negative sentiment.

**Summary Metrics:**

Sample Accuracy: 62%

Sample F1 Score: 0.75

**Confusion Matrix:**

A diagram of a blue box

AI-generated content may be incorrect.

**Computational Cost Analysis**

Reflecting on the varying training setups conducted for this assignment, one can note how the number of epochs tends to increase computational cost and time. In this case, we experimented in Run 4 by having 2 epochs instead of the usual 1 used in the other model builds. This should be considered when dealing with resource allocation. Furthermore, the changing targeted LoRA modules from q\_proj and v\_proj to query\_key\_value and dense\_h\_to\_4h modifies the model’s architecture, which can affect the computational load. Touching on training times, we made use of Google Colab’s GPUS, and experienced training times such as 24 minutes for Run 1 of Hyperparameter Tuning.

**Discussion: Challenges, Limitations, Ethical Considerations, and Potential Improvements**

The analysis of **Run 4** compared to **Run 1** provides valuable insights into the performance and challenges faced by the model. Despite using more epochs and targeting different LoRA modules, the accuracy did not see a significant improvement over the initial run. This suggests that the model may be encountering **overfitting** or a **performance plateau**, which is a common issue when training on small datasets. As noted, the model’s **F1 score remained stable at 0.75**, indicating that while the precision and recall for detecting positive sentiments were reasonably strong, the model struggled to correctly identify negative sentiment, especially in reviews where misclassification occurred.

One **key challenge** was the model's **overconfidence in positive predictions**. Even when reviews were negative, the model often predicted positive sentiment with high confidence. This might be attributed to longer training without sufficient **regularization**, which can cause the model to develop a bias toward positive sentiment detection. In future iterations, incorporating **early stopping**, **balanced sampling**, and **cross-validation** would help mitigate this issue, ensuring that the model generalizes better to unseen data and avoids overfitting.

The performance variability observed across different test cases highlights the **limitations of training on small datasets**. The complexity of text data, such as varying review phrasing, can introduce **evaluation noise**, leading to inconsistencies in predictions. This points to the necessity of incorporating larger and more diverse training datasets, as well as methods like **data augmentation**, which can better expose the model to varied examples and improve its robustness in identifying negative sentiment.

From an **ethical perspective**, models trained to detect sentiment in text, especially from user reviews, must be careful not to reinforce biases or propagate harmful stereotypes. If the model consistently misidentifies negative sentiment, it could potentially lead to **misleading conclusions**, affecting business decisions, such as product improvements or customer satisfaction assessments. To address these concerns, it would be crucial to test the model across a wider range of review contexts and ensure that it is trained to handle diverse expressions of both positive and negative sentiments fairly.

Finally, **potential improvements** for the model include experimenting with **LoRA configurations**, **instruction tuning strategies**, and the integration of **constrained decoding techniques** to improve the fidelity of sentiment extraction. By exploring these avenues, the model can be better equipped to handle nuanced sentiment detection tasks, ultimately leading to improved accuracy and robustness. Additionally, a more thorough evaluation using larger test sets could provide clearer insights into its strengths and weaknesses, guiding further refinement.

In summary, while Run 4 showed promising results, it also highlighted the challenges inherent in training sentiment detection models, including issues with overfitting, evaluation variability, and bias in predictions. Moving forward, addressing these challenges through strategic improvements in training methodology and dataset expansion will be key to enhancing model performance.

**Conclusion**

In conclusion, while **Run 4** demonstrated similar accuracy to **Run 1**, it also highlighted areas for improvement, particularly in handling negative sentiment detection and mitigating overfitting. The stable F1 score suggests that the model can reliably detect positive sentiment, but its tendency to misclassify negative reviews underlines the need for further training and optimization. Ethical considerations regarding bias and fairness remain a key concern, underscoring the importance of using diverse datasets and applying techniques to ensure balanced predictions. By implementing strategies like cross-validation, data augmentation, and exploring more sophisticated tuning methods, the model can be refined to better capture sentiment nuances and perform more reliably across various contexts. These improvements will not only enhance the model’s predictive power but also make it a more robust tool for sentiment analysis in real-world applications.

**Statement of Individual Contributions**

**Kathryn Burkhart**

In this project, I focused on analyzing the results of the machine learning models and compiling the findings into a comprehensive report. My primary responsibilities included evaluating the performance of different models, and comparing their outcomes based on key metrics such as accuracy and F1 score.

I performed a detailed analysis of the results, identifying patterns and trends that contributed to the overall understanding of the model's strengths and weaknesses. I also provided insights into areas where the models performed well, such as positive sentiment detection, and areas that required improvement, such as misclassification in negative sentiment.

In addition to evaluating model performance, I wrote the final report, synthesizing the analysis into a clear and structured narrative. The report highlighted key findings, including potential causes for mixed results, and offered recommendations for future improvements. I emphasized the importance of regularization techniques and the need for further optimization in future runs.

Through this analysis, I contributed to the overall success of the project by providing actionable insights and clear documentation that will inform future model improvements and guide the direction of the work moving forward.

**Prince Praveen**

In this project, my primary focus was on designing and running hyperparameter tuning experiments to explore how different training settings impact model performance.

Specifically, my contributions included:

- Fine-Tuning Experiments: I configured and executed key fine-tuning runs using the LoRA (Low-Rank Adaptation) method, adjusting parameters such as learning rate, LoRA rank (`r`), alpha scaling, and dropout rate.

- Evaluation and Analysis: I contributed to evaluating the models on held-out test samples, calculating metrics such as accuracy and F1 score, and visualizing the results with confusion matrices.

- Model Setup Assistance: I helped in setting up the fine-tuning pipeline, ensuring that models were correctly loaded, tokenizers prepared, and training loops handled gradient accumulation and learning rate scheduling.

By conducting multiple controlled experiments and systematically tracking the results, I contributed to identifying trends such as model sensitivity to learning rate and overfitting behaviors when training for additional epochs.

**Debjani Sarma**

In this project, my focus was on refining the fine-tuning process by exploring additional hyperparameter configurations and analyzing their impact on model generalization.

My contributions included:

* Extended Hyperparameter Tuning: Building upon the initial experiments, I designed and conducted additional fine-tuning runs with modified LoRA setups, including changes to the LoRA rank, alpha, and dropout values, to evaluate their effect on stability and performance.
* Training Optimization: I experimented with adjusting learning rates and the number of training epochs to mitigate overfitting and improve robustness, especially on challenging examples with ambiguous sentiment.
* Evaluation and Reporting: I analyzed the training curves and inference outputs from these new configurations, comparing them against previous runs to identify improvements or regressions. My work helped validate that different LoRA and training setups influence convergence speed and output reliability.

Through systematic experimentation and comparative analysis, I contributed to strengthening the model's fine-tuning methodology and providing actionable insights for balancing efficiency and performance in low-resource settings.