## **Technical Report: LLM Fine-Tuning for Sentiment Analysis**

**Project Title:** Fine-Tuning a Large Language Model for Sentiment Analysis on the Stanford Sentiment Treebank (SST-2) Dataset

### **1. Introduction**

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of natural language processing (NLP) tasks. While pre-trained on vast amounts of text data, their general-purpose nature often limits their optimal performance on specialized tasks or domain-specific data. Fine-tuning addresses this by adapting a pre-trained LLM to a new dataset and task, enabling it to learn task-specific nuances and improve performance significantly.

This technical report details a project focused on fine-tuning a Large Language Model for binary sentiment analysis. The objective is to enhance the model's ability to classify text as either positive or negative sentiment, specifically utilizing the Stanford Sentiment Treebank (SST-2) dataset. We explore the entire fine-tuning pipeline, from data preparation and model selection to hyperparameter optimization, comprehensive evaluation, and in-depth error analysis. The insights gained from this project aim to not only improve model performance but also to understand the challenges and potential future directions in fine-tuning LLMs for sentiment classification.

### **2. Methodology and Approach**

Our fine-tuning project follows a structured methodology encompassing dataset preparation, model selection, fine-tuning setup, and hyperparameter optimization.

#### **2.1. Dataset Preparation**

The Stanford Sentiment Treebank (SST-2) dataset was chosen for this project. SST-2 is a widely-used benchmark for sentiment analysis, consisting of single sentences extracted from movie reviews, each labeled with either a positive (1) or negative (0) sentiment.

**Data Acquisition and Initial Splits:** The SST-2 dataset was loaded using the Hugging Face datasets library directly, simplifying the data ingestion process. The original dataset provides distinct splits:

* **Train Set:** 67,349 examples
* **Validation Set:** 872 examples
* **Test Set:** 1,821 examples

A critical observation during data exploration was that the test split of SST-2, as provided in the GLUE benchmark, does not contain labels. For a supervised evaluation in this project, the original validation set was designated as our test\_dataset for final evaluation. This ensures we have ground truth labels to assess the model's performance rigorously.

**Data Cleaning and Preprocessing:** While the SST-2 dataset is generally clean, a clean\_text function was implemented for initial preprocessing. This function primarily focused on:

* Removing extraneous whitespace from sentences.
* Filtering out very short sentences (less than 3 words), which often lack sufficient context for meaningful sentiment classification.

After this cleaning step, the training dataset was reduced from 67,349 to 55,838 examples, and the validation (now test) set from 872 to 870 examples, indicating a minimal loss of valid data.

**Custom Train/Validation Split:** To create a dedicated validation set for monitoring training progress and hyperparameter tuning, the cleaned original training data (train\_dataset\_full) was further split. A 85/15 ratio was used, resulting in:

* **New Training Set:** 47,462 examples
* **New Validation Set:** 8,376 examples
* **Final Test Set (original validation):** 870 examples

Class distributions across these splits were verified to ensure a balanced representation of positive and negative sentiment, minimizing potential bias during training and evaluation.

**Tokenization and Formatting:** The distilbert-base-uncased tokenizer was loaded to prepare the text data for the model. A tokenize\_function was defined to:

* Apply tokenization to the sentence field of each example.
* Truncate sequences to a max\_length of 128 tokens, chosen as SST-2 sentences are generally concise.
* Pad sequences to max\_length to ensure uniform input size for batch processing.
* Return PyTorch tensors (return\_tensors="pt") for compatibility with the training framework.

Finally, all datasets (training, validation, and test) were set to the "torch" format, specifying input\_ids, attention\_mask, and label columns, which are required by the Hugging Face Trainer.

#### **2.2. Model Selection**

**Pre-trained Model Choice:** The distilbert-base-uncased model was selected as the base LLM for fine-tuning. This model is a distilled version of BERT, offering a compelling balance between performance and computational efficiency.

**Justification for Model Selection:** Our choice of distilbert-base-uncased was driven by a careful consideration of the task requirements and available resources:

* **Task Requirements:** Binary sentiment classification demands strong text understanding for sequence classification.
* **Efficiency:** DistilBERT is significantly smaller (66M parameters vs. BERT's 110M) and faster (60%) than its full BERT counterpart, making it ideal for rapid experimentation and training within limited computational environments like Google Colab. Despite its smaller size, it retains approximately 95% of BERT's performance.
* **Domain Suitability:** Pre-trained on a large corpus of general English text, DistilBERT is well-suited for understanding the nuances of SST-2's movie review sentences. The "uncased" version handles variations in capitalization, which is appropriate for unformatted text.
* **GLUE Benchmark Performance:** Given that SST-2 is part of the GLUE benchmark, DistilBERT's strong performance on general GLUE tasks bodes well for its fine-tuned performance on SST-2.

**Model Architecture Setup for Fine-Tuning:** The AutoModelForSequenceClassification class from Hugging Face Transformers was used to load the pre-trained DistilBERT model. This class automatically adds a classification head on top of the pre-trained transformer encoder, making it suitable for classification tasks. The model was configured with num\_labels=2 to align with the binary sentiment classes (positive/negative), and id2label/label2id mappings were provided for interpretability.

**Parameter-Efficient Fine-Tuning (PEFT) with LoRA:** To further enhance efficiency and mitigate the risk of overfitting on a relatively smaller task-specific dataset, Low-Rank Adaptation (LoRA) was applied. LoRA freezes the original pre-trained model weights and injects small, trainable rank-decomposition matrices into selected layers (specifically q\_lin and v\_lin for attention mechanisms). This drastically reduces the number of trainable parameters while preserving most of the model's performance.

The LoraConfig was set with:

* task\_type=TaskType.SEQ\_CLS (sequence classification).
* r=8 (rank of the update matrices).
* lora\_alpha=32 (scaling factor for LoRA layers).
* lora\_dropout=0.1 (dropout rate for LoRA layers).

After applying LoRA, the trainable parameters were reduced from the original 66,955,010 to only 739,586, representing approximately 1.09% of the total parameters. This significant reduction allows for faster training and lower memory consumption.

#### **2.3. Fine-Tuning Setup**

**Training Environment Configuration:** The training environment was carefully configured for optimal performance and reproducibility:

* **Device Allocation:** PyTorch was configured to utilize the available GPU (Tesla T4) for accelerated training. GPU memory was confirmed at 14.7 GB.
* **Reproducibility:** Random seeds were set for torch and numpy to ensure that results are reproducible across runs.
* **Mixed Precision:** While not explicitly enabled in TrainingArguments in the provided notebook, typically fp16=True would be set in TrainingArguments to enable mixed-precision training, which further speeds up training on compatible GPUs by utilizing lower-precision formats.
* **Output Directories:** Dedicated directories (./results, ./models, ./logs) were created to store training checkpoints, logs, and final models, maintaining an organized project structure.

**Training Loop Implementation with Hugging Face Trainer:** The fine-tuning process was orchestrated using the Hugging Face Trainer API, which streamlines the training and evaluation process. The Trainer abstracts away much of the boilerplate code, handling:

* Optimization schedules.
* Gradient accumulation.
* Logging.
* Evaluation during training.
* Checkpointing.

**Training Arguments Configuration:** TrainingArguments were configured to define the training hyperparameters and behaviors:

* output\_dir: Specifies where model checkpoints and logs are saved.
* num\_train\_epochs: The total number of training epochs.
* per\_device\_train\_batch\_size and per\_device\_eval\_batch\_size: Batch sizes for training and evaluation.
* learning\_rate: The initial learning rate for the optimizer.
* weight\_decay: Regularization parameter to prevent overfitting.
* logging\_steps: Frequency of logging training metrics.
* save\_steps and eval\_steps: Frequency of saving checkpoints and performing evaluation on the validation set.
* save\_total\_limit: Limits the total number of checkpoints saved.
* report\_to=None: Prevents automatic reporting to external services like Weights & Biases or TensorBoard.
* push\_to\_hub=False: Prevents model uploads to the Hugging Face Hub.

#### **2.4. Hyperparameter Optimization**

A well-defined hyperparameter optimization strategy is crucial for achieving optimal model performance. We employed a targeted grid search approach by testing three distinct hyperparameter configurations.

**Search Space Definition:** The search space for hyperparameter optimization focused on:

* **Learning Rate (learning\_rate):** Explored values of 2e-5, 3e-5, and 5e-5. These are common ranges for fine-tuning BERT-based models.
* **Number of Training Epochs (num\_train\_epochs):** Tested 2 and 3 epochs, aiming to find a balance between underfitting and overfitting.
* **Batch Size (per\_device\_train\_batch\_size):** Fixed at 16 due to memory constraints and to ensure consistency across experiments.
* **Weight Decay (weight\_decay):** Fixed at 0.01, a standard regularization value.
* **Warmup Steps:** Fixed at 100, a common practice for initial learning rate scheduling.

**Selected Configurations:** Three specific configurations were tested:

1. **config\_1\_conservative**: Learning Rate = 2e-5, Epochs = 3. This represents a conservative approach.
2. **config\_2\_balanced**: Learning Rate = 3e-5, Epochs = 3. This is a balanced, typical setting.
3. **config\_3\_aggressive**: Learning Rate = 5e-5, Epochs = 2. This is a more aggressive approach with a higher learning rate and fewer epochs, aiming for quicker convergence.

**Evaluation Criteria:** The primary criterion for selecting the best configuration was the **F1 score** on the validation set, as it provides a balanced measure of precision and recall, crucial for classification tasks with potential class imbalance. Validation accuracy was a secondary consideration.

### **3. Results and Analysis**

This section presents the results of our hyperparameter optimization and the comprehensive evaluation of the best fine-tuned model against a pre-trained baseline.

#### **3.1. Hyperparameter Optimization Results**

The three defined configurations were trained, and their performance on the validation set was recorded.

| Configuration Name | Learning Rate | Epochs | Validation Accuracy | Validation F1 Score (Macro) | Training Time |
| --- | --- | --- | --- | --- | --- |
| config\_1\_conservative | 2e-05 | 3 | 0.8926 | 0.9018 | 0:18:08.295770 |
| config\_2\_balanced | 3e-05 | 3 | 0.9060 | 0.9144 | 0:18:05.469888 |
| config\_3\_aggressive | 5e-05 | 2 | 0.9126 | 0.9202 | 0:12:00.763983 |

Export to Sheets

*Note: The F1 score values might represent the eval\_f1 directly reported by the Trainer, which might be weighted or macro depending on how the evaluate library calculates it by default for binary classification.*

**Analysis of Hyperparameter Experiments:** From the results, config\_3\_aggressive (Learning Rate: 5e-05, Epochs: 2) yielded the highest validation F1 score of 0.9202 and accuracy of 0.9126. This suggests that for the SST-2 dataset and DistilBERT with LoRA, a slightly higher learning rate and fewer epochs allowed for faster convergence to a better local optimum, potentially due to the nature of the loss landscape or the simplicity of the fine-tuning task. It also demonstrated the fastest training time due to fewer epochs. This configuration was selected as the "best model" for further evaluation on the independent test set.

*(Include the "hyperparameter\_comparison.png" chart here as a figure, clearly labeled, comparing accuracy and F1 scores across configurations.)*

#### **3.2. Comprehensive Evaluation on Test Set**

The best-performing model (config\_3\_aggressive) was then evaluated on the dedicated test\_dataset (the original validation set). For a meaningful comparison, a baseline\_model (the pre-trained distilbert-base-uncased without any fine-tuning) was also evaluated on the same test set.

**Evaluation Metrics:** The evaluation was comprehensive, including:

* **Accuracy:** Overall correctness of predictions.
* **Precision (Macro & Weighted):** The proportion of positive identifications that were actually correct.
* **Recall (Macro & Weighted):** The proportion of actual positives that were correctly identified.
* **F1 Score (Macro & Weighted):** The harmonic mean of precision and recall, providing a balanced measure.
* **AUC-ROC:** Area Under the Receiver Operating Characteristic curve, useful for assessing the classifier's ability to distinguish between classes across various thresholds.
* **Per-Class Metrics:** Detailed precision, recall, and F1 for "negative" and "positive" classes.
* **Confusion Matrix:** A visual representation of true positives, true negatives, false positives, and false negatives.

**Test Set Performance - Fine-tuned Model:**

* **Accuracy:** 0.8851
* **F1 Score (Macro):** 0.8850
* **AUC-ROC:** 0.9578

**Test Set Performance - Baseline Model:**

* **Accuracy:** 0.4747
* **F1 Score (Macro):** 0.4531
* **AUC-ROC:** 0.4359

#### **3.3. Detailed Comparison and Statistical Significance**

The fine-tuned model demonstrates a dramatic improvement over the baseline model across all key metrics.

| Metric | Baseline | Fine-tuned | Absolute Improvement | Percentage Improvement |
| --- | --- | --- | --- | --- |
| Accuracy | 0.4747 | 0.8851 | +0.4103 | +86.4% |
| F1 Score (Macro) | 0.4531 | 0.8850 | +0.4319 | +95.3% |
| F1 Score (Weighted) | 0.4551 | 0.8850 | +0.4299 | +94.5% |
| Precision (Macro) | 0.4664 | 0.8852 | +0.4189 | +89.8% |
| Recall (Macro) | 0.4712 | 0.8848 | +0.4136 | +87.8% |
| AUC-ROC | 0.4359 | 0.9578 | +0.5219 | +119.7% |

Export to Sheets

**Analysis:** The fine-tuning process resulted in substantial gains across all performance indicators. The AUC-ROC score, in particular, saw a remarkable increase of almost 120%, indicating the fine-tuned model's vastly superior ability to discriminate between positive and negative sentiment compared to the untrained baseline. This highlights the effectiveness of fine-tuning pre-trained LLMs for specific downstream tasks.

*(Include the "confusion\_matrix.png" chart here as a figure, clearly labeled, showing the performance of the fine-tuned model.)*

**Statistical Significance Analysis (McNemar's Test):** To confirm that the observed improvement of the fine-tuned model over the baseline is statistically significant, McNemar's test was performed. This test compares the error rates of two classifiers on the same set of data.

* **Both Correct:** 356 examples
* **Baseline Only Correct:** 57 examples
* **Fine-tuned Only Correct:** 414 examples
* **Both Incorrect:** 43 examples

The McNemar's test resulted in a p-value of 0.0000. Since this p-value is significantly less than the conventional alpha level of 0.05, we can conclude that the performance difference between the fine-tuned model and the baseline model is **statistically significant**. The fine-tuned model's ability to correctly classify examples where the baseline failed (414 instances) far outweighs the cases where the baseline was correct and the fine-tuned was not (57 instances), reinforcing its superiority.

### **4. Limitations and Future Improvements**

While the fine-tuned model demonstrates strong performance, understanding its limitations and identifying areas for future improvement is crucial for developing more robust and practical sentiment analysis systems.

#### **4.1. Error Analysis**

A detailed error analysis was conducted to pinpoint specific types of errors the fine-tuned model makes. Out of 870 test examples, the model made 100 errors, resulting in an error rate of 11.5%. These errors were categorized as:

* **False Positives (54 cases / 6.2% of total examples):** Instances where the model incorrectly predicted a positive sentiment when the true label was negative.
* **False Negatives (46 cases / 5.3% of total examples):** Instances where the model incorrectly predicted a negative sentiment when the true label was positive.

Furthermore, **High-Confidence Errors (>0.9 confidence)** were a notable concern, comprising 44% of all errors. This indicates that the model is sometimes overconfident in its incorrect predictions, which can be problematic in real-world applications where confidence is used for decision-making.

Specific patterns identified in the errors include:

* **Subtle Negative Words:** The model struggles with nuanced negative phrasing or implied negativity (e.g., "This one is definitely one to skip").
* **Mixed Sentiment:** Sentences containing contrasting positive and negative elements are difficult for the model to correctly resolve (e.g., "It had some good moments, but ultimately disappointed").
* **Contextual Positives:** Positive words used ironically or in a negative context are often misclassified (e.g., "hilariously inept and ridiculous" classified as negative when true label is positive).
* **Sarcasm/Irony:** Direct instances of sarcasm were challenging for the model to detect, leading to misinterpretations of true sentiment.
* **Short Sentences:** Very short sentences with subtle sentiment can be ambiguous.

These findings suggest that the model, despite being fine-tuned, still faces challenges with linguistic complexities beyond direct positive/negative word associations, particularly those requiring deeper contextual understanding, common sense reasoning, or detection of rhetorical devices.

#### **4.2. Limitations of Current Approach**

* **Dataset Specificity:** While SST-2 is a good benchmark, it consists solely of movie reviews. The fine-tuned model may not generalize well to other domains (e.g., product reviews, social media) without further fine-tuning or domain adaptation.
* **Binary Classification:** The project focuses on binary sentiment. Many real-world applications require more granular sentiment (e.g., neutral, mixed) or aspect-based sentiment analysis.
* **Limited Hyperparameter Search:** Although three configurations were tested, a more extensive hyperparameter search (e.g., using advanced search algorithms like Bayesian Optimization) could potentially yield further performance gains.
* **No Custom Callbacks/Advanced Logging:** While Trainer is robust, a custom TrainerCallback for specific logging or more sophisticated early stopping could have provided deeper insights during training. Integrating with tools like Weights & Biases would significantly enhance logging and experiment tracking for a professional project.
* **Inference Efficiency (Batching):** While an inference pipeline was created, the batch\_predict method in the provided code iterates over single predictions rather than truly processing a batch of texts in parallel. This impacts real-world deployment efficiency for large volumes of text.

#### **4.3. Future Improvements**

Based on the error analysis and identified limitations, the following improvements are recommended for future iterations of this project:

**1. Data-Centric Improvements:**

* **Targeted Data Augmentation:**
  + **Sarcasm/Irony Augmentation:** Create or acquire datasets specifically labeled for sarcasm and irony, or augment existing data by paraphrasing sentences to introduce ironic expressions.
  + **Mixed Sentiment Augmentation:** Synthesize examples that explicitly combine positive and negative aspects within a single sentence to train the model on handling such complexities.
  + **Negation Handling:** Implement techniques like "not\_good" tokenization or generate negated versions of existing sentences (e.g., "This movie was great" -> "This movie was not great") to explicitly teach negation.
* **Domain Adaptation:** For deployment in new domains, collect small domain-specific datasets and perform further fine-tuning or transfer learning.
* **Multi-label/Aspect-Based Sentiment:** Expand the task to include more granular sentiment labels (e.g., 5-point scale) or aspect-based sentiment analysis, where sentiment is identified for specific entities or features within a text.

**2. Model-Centric Improvements:**

* **Explore Larger/Different Architectures:** While DistilBERT is efficient, experimenting with slightly larger models like RoBERTa or ELECTRA could yield higher performance, especially if computational resources allow.
* **Attention Mechanism Analysis for Interpretability:** Implement methods to visualize and analyze the attention weights of the model, particularly for misclassified examples. This can provide insights into which words or phrases the model focused on incorrectly.
* **Confidence Calibration:** Implement post-training calibration techniques (e.g., temperature scaling) to make the model's confidence scores more reliable, especially for high-confidence errors.
* **Ensemble Methods:** Combine multiple fine-tuned models (e.g., models with different random seeds or from different hyperparameter configurations) to leverage their diverse strengths and improve overall robustness.

**3. Training & Optimization Enhancements:**

* **Advanced Hyperparameter Tuning:** Utilize automated hyperparameter optimization frameworks like Ray Tune, Optuna, or Hyperopt to explore a wider range of hyperparameters (e.g., lora\_r, lora\_alpha, learning rate schedules) more systematically.
* **Custom Loss Functions:** Investigate the use of focal loss or other loss functions designed to address class imbalance or to focus training on hard-to-classify examples (e.g., those the model is overconfident about).
* **Curriculum Learning:** Implement strategies where the model first trains on "easier" examples and gradually moves to "harder" ones.
* **Adversarial Training:** Incorporate adversarial training techniques to make the model more robust to small perturbations in input, potentially improving its handling of ambiguous or challenging phrases.

**4. Deployment and Scalability:**

* **True Batched Inference:** Implement the batch\_predict functionality to genuinely process multiple inputs simultaneously through the model for improved throughput in production.
* **Quantization and Distillation:** For deployment on resource-constrained devices, explore post-training quantization (e.g., 8-bit, 4-bit) or model distillation (training a smaller "student" model to mimic the fine-tuned LLM's behavior) to reduce model size and inference latency.

By addressing these limitations and implementing the suggested improvements, the sentiment analysis model can be made more accurate, robust, and suitable for diverse real-world applications.

### **5. References**

* Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186.
* Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
* Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*.
* Dodge, J., et al. (2021). Measuring and improving the reproducibility of NLP models. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 5316–5331.
* Hugging Face Transformers Library documentation. (Accessed: July 7, 2025).<https://huggingface.co/docs/transformers/index>
* Hugging Face PEFT Library documentation. (Accessed: July 7, 2025).<https://huggingface.co/docs/peft/index>
* Hugging Face Datasets Library documentation. (Accessed: July 7, 2025).<https://huggingface.co/docs/datasets/index>
* Wang, A., et al. (2019). GLUE: A multi-task benchmark for natural language understanding. *arXiv preprint arXiv:1804.07461*.