Applied Deep Learning (WS 2025)

Project Report (NLP – Sentiment Analysis)

Name: Aman Bhardwaj
Student ID: 12333472

Introduction

In this project, I implemented a binary sentiment classifier for IMDB movie reviews using deep learning and natural language processing (NLP) techniques. The goal is to classify each review as positive or negative.

I used a Bring-Your-Own-Method for this project, which focuses on re-implementing and fine - tuning a neural network architecture that operates on an existing, publicly available dataset (IMDB dataset). allowing me to fine-tune a custom solution. The total estimated time for the project was approximately 52 hours, and I completed it in around 45 hours, including data preprocessing, model training, evaluation, and web application development. Training the fine-tuned RoBERTa model took approximately 16 hours.

Project Summary and Approach

Project Description: The project builds a binary sentiment classifier that predicts whether a movie review is positive or negative using deep learning. It involved the following steps:

1) Data Preprocessing:

Clean text, remove special characters and other things. Convert text to lowercase

2) Tokenization:

Using tokenizer (RoBERTa byte-level BPE tokenizer) to convert text into input IDs, attention masks, and segment IDs.

3) Model Building:

Re-Implementing the RoBERTa model, improving this state-of-the-art method. Optimizing and Fine-tuning on dataset for binary sentiment classification.

4) Training & Hyperparameter Tuning:

Experiment with learning rate, batch size, and number of epochs. Implementing early stopping to prevent overfitting.

5) Evaluation:

Metrics: Accuracy, Precision, Recall, F1-score, F1 being the primary metric I used.

Visualization: Confusion matrix, loss and accuracy plots etc.

6) Application Development:

A small application using Streamlit to run model and input custom reviews and get sentiment predictions, provide deliverables and to ship it.

Key components used in this project are in the table as under:

Component	Description
Dataset	IMDB Large Movie Review Dataset (50,000 reviews, 25k train / 25k test). Binary labels: Positive / Negative. Balanced and suitable for sentiment classification.
Method	Fine-tuning a transformer-based model (RoBERTa) for NLP sentiment classification
Model	**RoBERTa-base transformer**: - Pretrained on large corpus for contextual embeddings. - Architecture: RoBERTa Encoder → Dropout → Linear Classification Head (2 logits: Positive / Negative). - Produces embeddings that capture word meaning in context.
Tokenizer	**RoBERTa Tokenizer (Byte-Level BPE)**: - Converts text into subword tokens. - Outputs Input IDs & Attention Masks. - Pads/truncates sequences to max length 256.
Loss Function	**Cross-Entropy Loss**: - Measures difference between predicted probability and true label. - Suitable for binary classification. - Encourages the model to assign high probability to correct sentiment.
Inference Calibration	Temperature scaling (T = 2.5) applied to logits to calibrate probabilities and avoid overconfident predictions.
Decision Metric	Decision Strength (Logit Margin): Absolute difference between Positive and Negative logits. Flags ambiguous or borderline reviews.
Deployment	Streamlit Web Application for interactive input, real-time sentiment prediction, confidence display, and decision strength warnings.

Problem Statement

1. What is the problem that I tried to solve?

This project addresses sentiment analysis of IMDB movie reviews, classifying them as positive or negative. The challenge lies in handling contextual nuances, negations, and mixed sentiments, which traditional rule-based or bag-of-words methods cannot reliably capture.

2. Why is it a problem?

Manual analysis of large volumes of reviews is infeasible, and traditional NLP approaches often fail to capture the semantic meaning and context of text. Accurate sentiment detection is crucial for business insights, social media analysis, and research applications.

3. What is my solution?

I re-implemented and fine-tuned RoBERTa-base model with a linear classification head trained on the IMDB dataset. The model uses RoBERTa tokenizer for preprocessing and Cross-Entropy Loss for optimization. A Streamlit web application enables interactive inference, displaying predicted sentiment, confidence, and decision strength. Temperature scaling (T=2.5) improves probability calibration. The optimized model achieved an F1-score of 0.9435 on the test set.

4. Why is it a solution?

Transformer-based deep learning captures contextual and semantic nuances that classical methods cannot. RoBERTa's pretrained embeddings allow efficient transfer learning, achieving high performance on ambiguous or complex text. Deep learning is necessary for state-of-the-art accuracy, adaptability, and robust real-world deployment.

Preprocessing Pipeline

Steps Implemented:

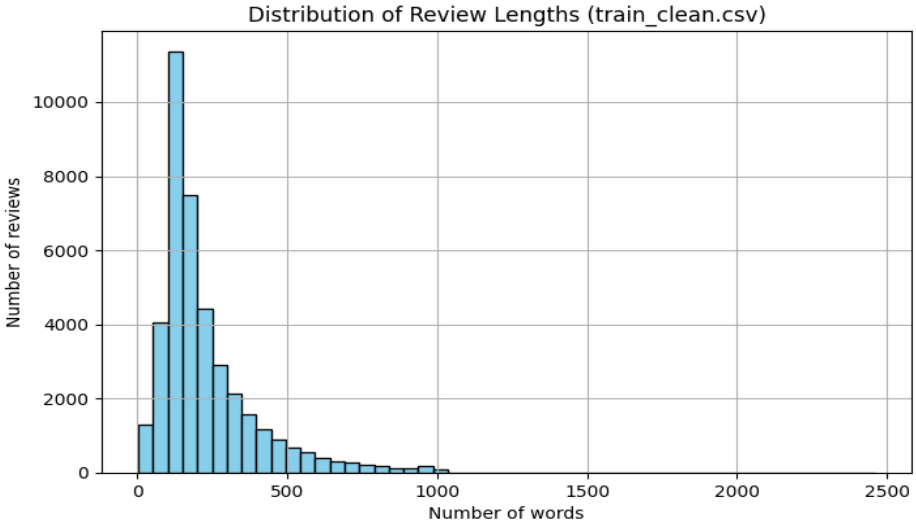
1. Loading Dataset
2. Train / Validation / Test Split
    - Split before preprocessing to avoid leakage (Train: 80%, Validation: 10%, Test: 10%)
    - Stratified split to maintain class balance.
    - Saved raw splits in split_raw_data/.
3. Text Cleaning
    - Removed HTML tags like
.
    - Normalized multiple spaces into a single space.
    - No lowercasing, punctuation removal, stopword removal, or stemming, as RoBERTa is pretrained on raw text.
    - Saved cleaned CSVs in preprocessed_clean_data/.
4. Tokenization
    - Used RoBERTa tokenizer to convert text into:
        - Input IDs (subword tokens)
        - Attention masks
    - Padded/truncated sequences to max_length=256.
    - Saved .pt files for train/val/test in tokenized_files/.
5. Label Encoding
    - Sentiment labels manually encoded as:
        - 0 → negative
        - 1 → positive
    - And then saved as tensors for training and evaluation.

Preprocessing & Postprocessing Tests

Implemented unit tests using **pytest**:

1. **Preprocessing tests**
    - Verify CSV splits exist.
    - Check split sizes match original dataset.
    - Confirm cleaned texts have no
 tags and normalized spaces.

- Validate tokenized tensors exist and shapes are correct.
- Check label integrity (only 0/1 and match input size).
- Histogram of review lengths (train set)



2. **Postprocessing tests**
   - Tiny forward/backward pass with RoBERTa on CPU.
   - Check logits shape and predictions validity.
   - Metrics calculation (accuracy, precision, recall, F1).
   - Save/load CSV files and plot confusion matrix.
   - Confusion matrix plotting

Run tests with:

python -m pytest -v -s test_preprocessing.py

python -m pytest -v -s test_postprocessing.py

All tests passed, ensuring preprocessing and postprocessing pipelines are correct.

Model Architecture

The model is **RoBERTa-base** with a linear classification head.

Architecture Details:

- RoBERTa encoder outputs contextual embeddings for each token
- Dropout layer prevents overfitting
- Linear classification head maps embeddings to **2 output classes** (positive/negative)

Loss Function:

- **Cross-Entropy Loss:** Measures the difference between predicted class probabilities and true labels.

- Encourages confident correct predictions and penalizes wrong ones

Optimizer & Scheduler:

- AdamW optimizer
- Learning rate scheduler (linear or cosine)
- Warmup ratio: 0.06 and 0.1

Model Training, Validation & Hyperparameter Optimization

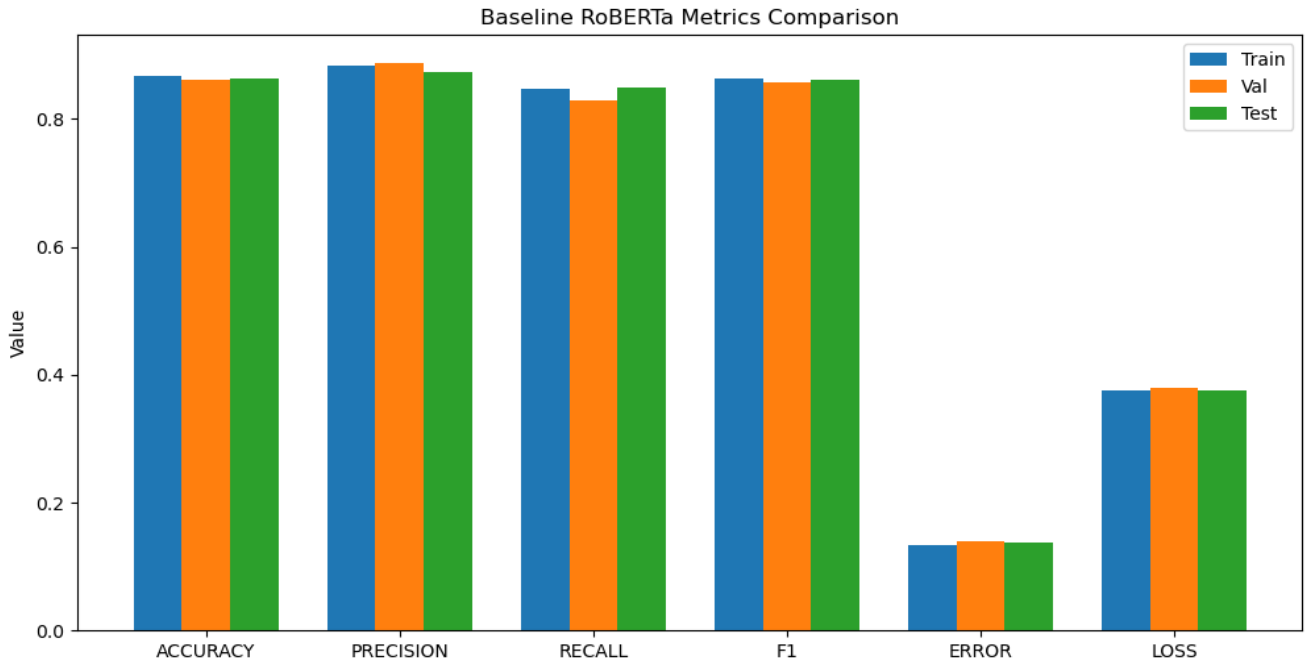
Baseline Model: RoBERTa with linear classification head. Trained head only for 5 epochs. Saved metrics in baseline_model_results/

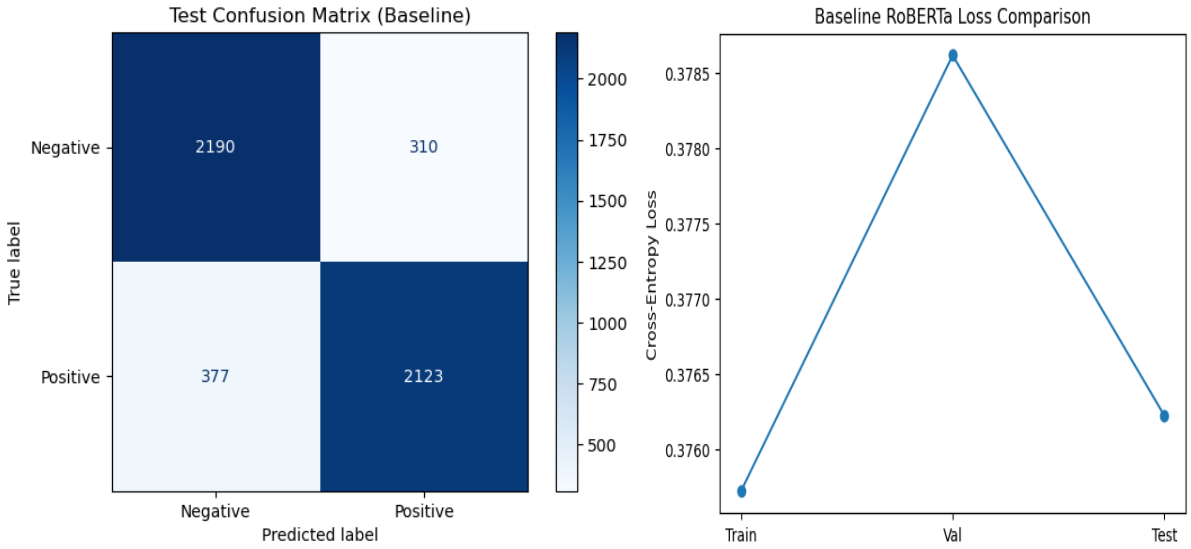
Baseline Results Table:

Split	Accuracy	Precision	Recall	F1-score	Error	Loss	Runtime (s)
Train	0.8667	0.8824	0.8462	0.8639	0.1333	0.3757	492.02
Val	0.8611	0.8867	0.8280	0.8564	0.1389	0.3786	54.66
Test	0.8626	0.8726	0.8492	0.8607	0.1374	0.3762	60.74

The baseline model of RoBERTa showed good results given it was trained for 5 epochs with minimal training only for head classifier and with default hyperparameters. I kept batch_size = 32 same as the optimized model for better comparison.

Below are the plots of the baseline model:





Optimized Model (Reimplemented/Fine-Tuned)

Fine-tuned RoBERTa with: Dropout, weight decay, learning rate, scheduler, pooling tuned etc. Ran 8 configurations over ~15 hours.

Hyperparameters tested (configurations):

- Learning rate: 1e-5, 2e-5, 3e-5, Batch size: 16, 32, Dropout: 0.1, 0.2, Weight decay: 0, 0.01, Scheduler: cosine, linear, Pooling: CLS / mean, Warmup ratio: 0.06 and 0.1

Results Table (All 8 Configurations):

lr	batch_size	dropout	weight_decay	scheduler	warmup_ratio	pooling	train_acc	val_acc	runtime (s)
1e-05	32	0.2	0.01	linear	0.06	mean	0.9929	0.9429	8036
3e-05	32	0.1	0	cosine	0.06	mean	0.9896	0.9344	5772
3e-05	16	0.1	0.01	cosine	0.06	mean	0.9642	0.9233	3680
3e-05	16	0.2	0.01	linear	0.06	cls	0.9811	0.9320	4906
1e-05	32	0.1	0.01	cosine	0.06	mean	0.9955	0.9427	9184
2e-05	32	0.1	0.01	cosine	0.10	cls	0.9980	0.9420	9182
3e-05	32	0.2	0	linear	0.10	cls	0.9975	0.9384	9150
1e-05	32	0.1	0	cosine	0.06	cls	0.9862	0.9429	5722

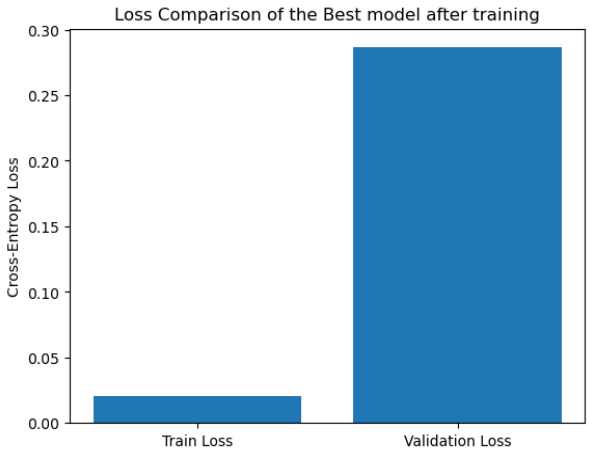
Best configuration after training: { "lr": 1e-05, "batch_size": 32, "dropout": 0.1, "weight_decay": 0.0, "scheduler": "cosine", "warmup_ratio": 0.06, "pooling": "cls" }

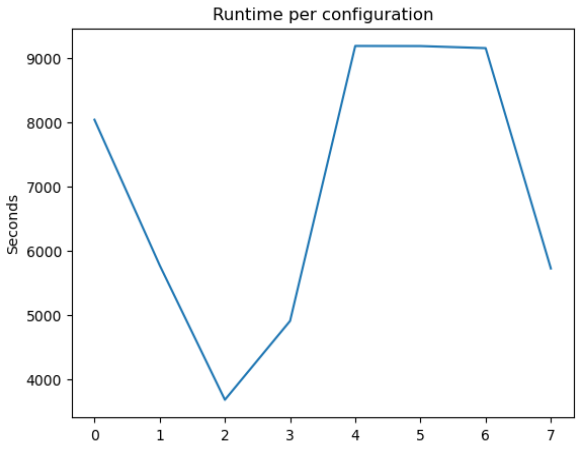
Training & Validation Results (Best Model)

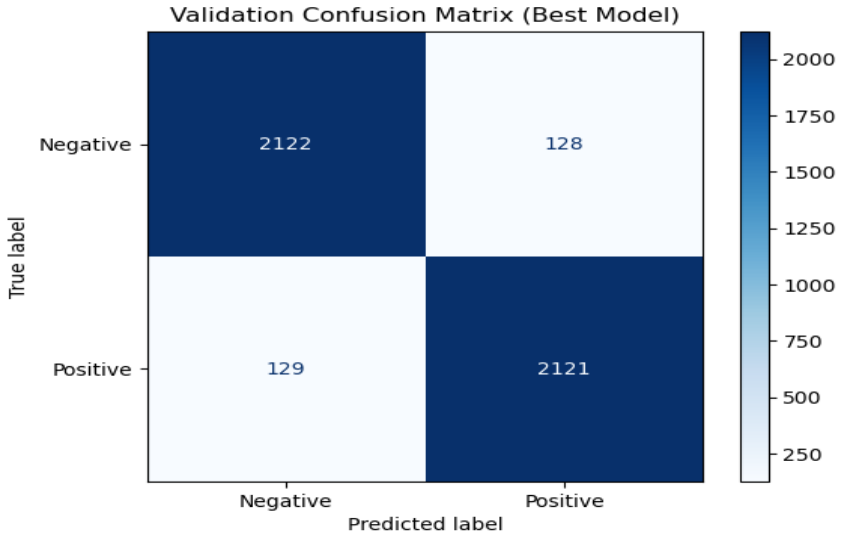
Here I only put the results of the best configuration in the table for further comparison.

Metric	Train	Validation
Accuracy	0.9862	0.9429
Precision	0.9850	0.9431
Recall	0.9875	0.9427
F1-score	0.9862	0.9429
Error	0.0138	0.0571
Loss	0.0209	0.2864
Runtime (s)	5722.76	5722.76

From the table it can be seen that the training metrics are quite high. Also, the training time took around 2 hours for each configuration, for some it took less due to early stopping when even after some epochs their performance metrics were not increasing. The validation set showed a bit less performance than the training set which is understandable as the size of the validation set is 8 times smaller than training set. Below are the plots for the training of the model, validation confusion matrix and runtimes:



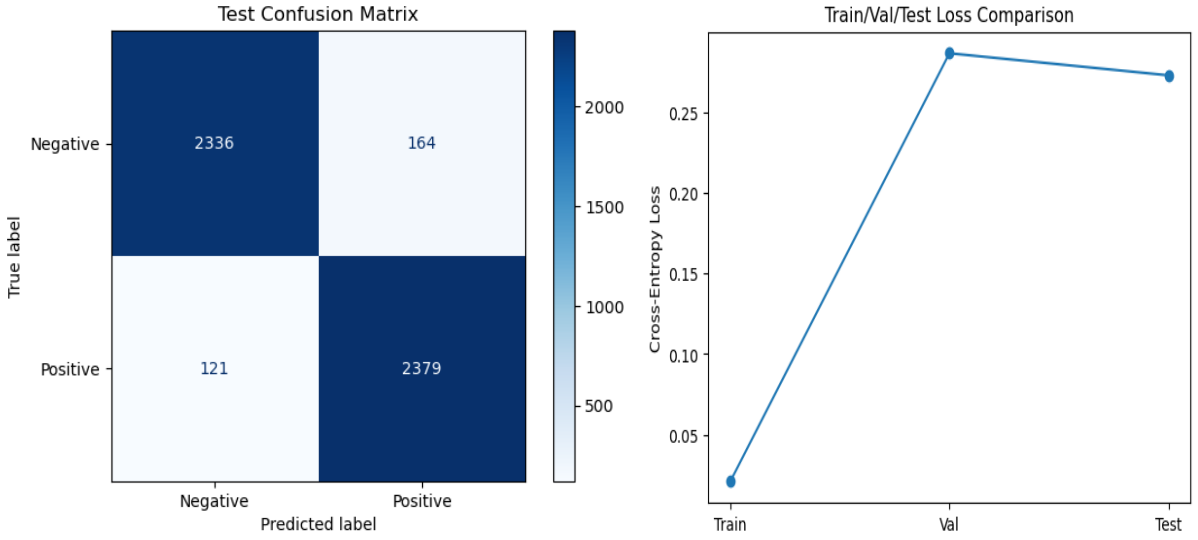


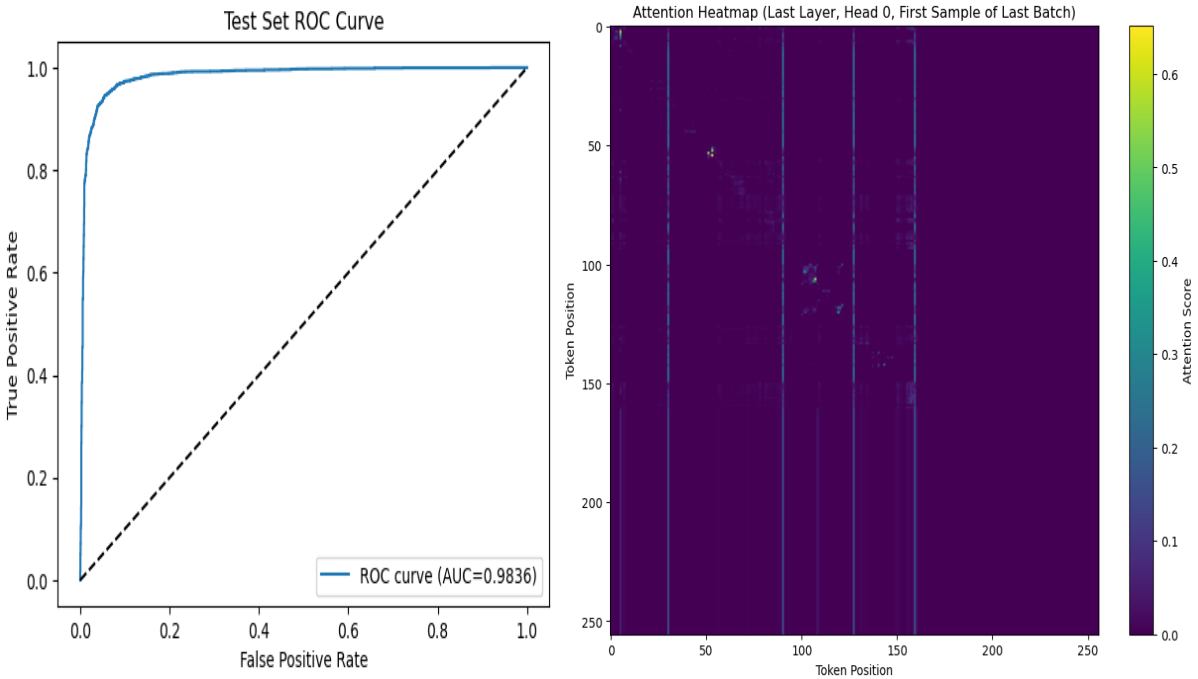


Test Results (Best Model):

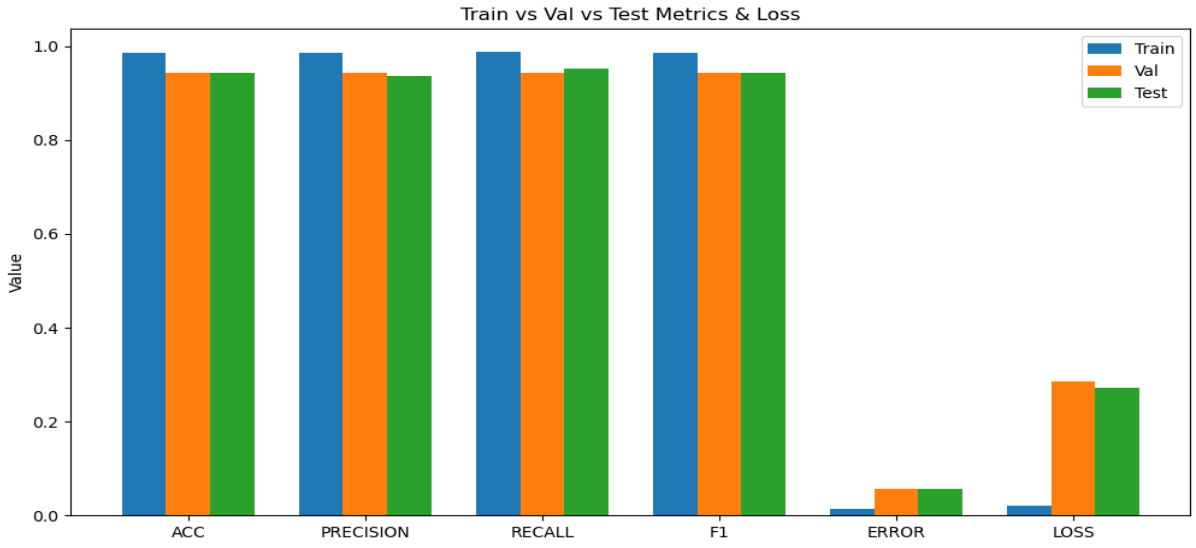
Split	Accuracy	Precision	Recall	F1-score	Error	Loss	Runtime (s)
Test	0.9430	0.9355	0.9516	0.9435	0.0570	0.2727	197.21

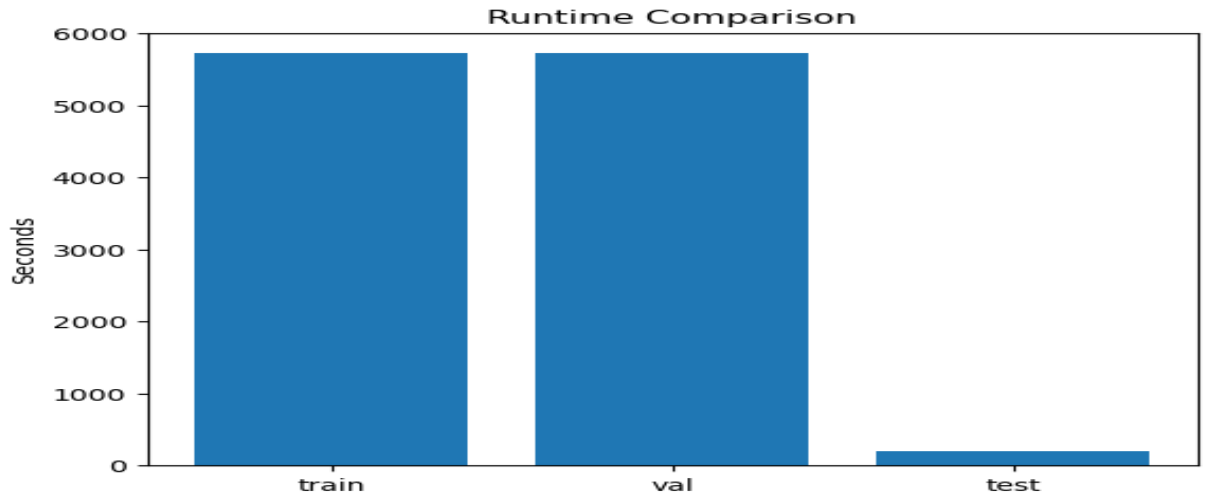
So, from the above tables it can be seen that the F1 score improved a bit for the test set and Cross Entropy Loss also improved a little. For training I used early stopping patience = 2 which helped to stop the epochs whenever the metrics started decreasing. Below are the plots for the testing of model:





Below is the plot comparing the metrics, runtimes and loss of training, validation and testing of the optimized (fine-tuned) model:





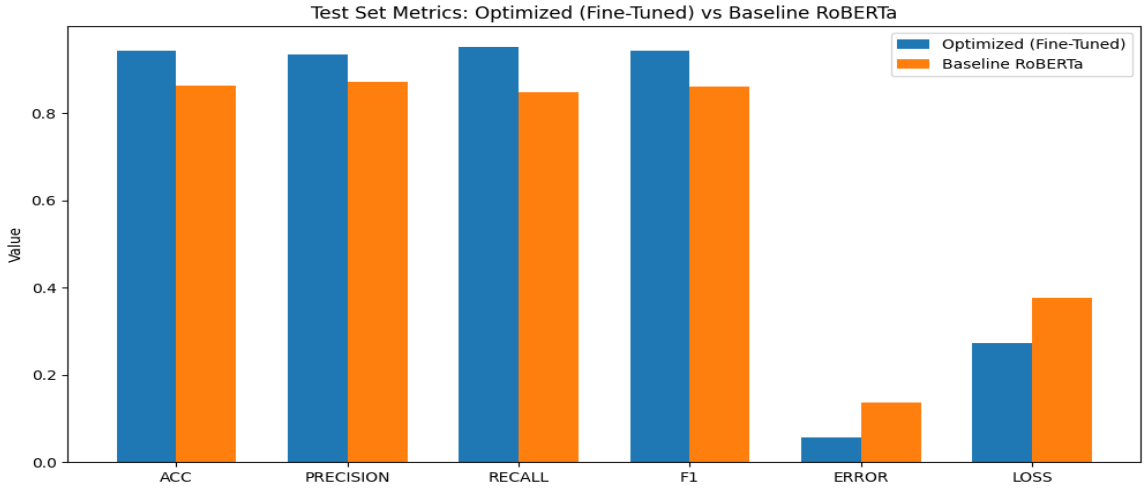
Comparison: Optimized vs Baseline

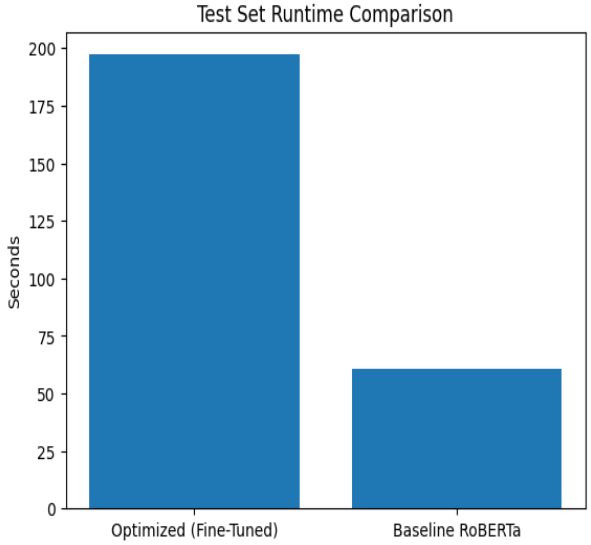
Test Set Metrics Comparison Table to Baseline

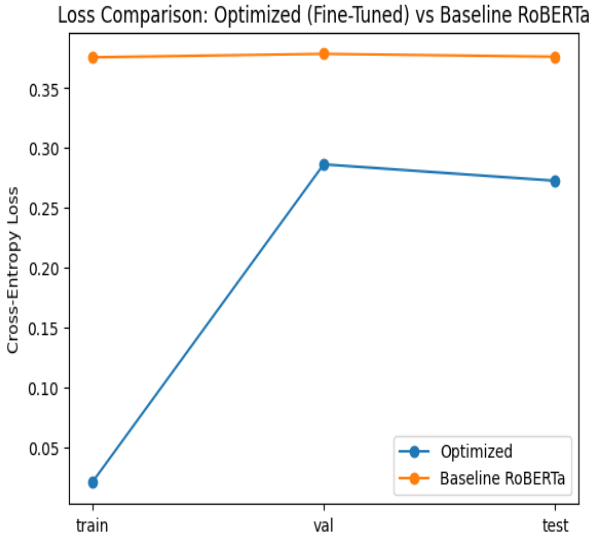
Metric	Optimized (Fine-Tuned)	Baseline RoBERTa
Accuracy	0.9435	0.8626
Precision	0.9355	0.8726
Recall	0.9516	0.8492
F1-score	0.9435	0.8607
Error	0.0570	0.1374
Loss	0.2727	0.3762
Runtime (s)	197.21	60.74

So, from the above table it can be clearly seen that my reimplemented and fine Tuned RoBERTa model performed quite better than the SOTA baseline RoBERTa model. The cross-entropy loss for optimized model is very less as compared to the base model. The Primary metric for comparison I took was F1 score and it can be seen that the Optimized model has higher F1 score as compared to the baseline model. So, the optimized model significantly outperforms the baseline across all metrics. Fine-tuning all layers and hyperparameter tuning led to higher F1-score (0.9435 vs ~0.8607).

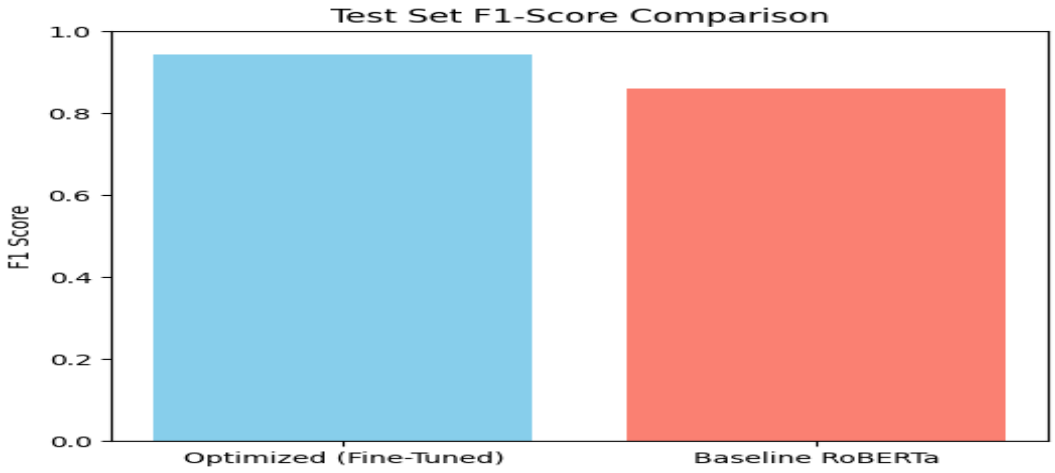
The below plots show this behaviour and show how the optimized model perfoems better than the baseline RoBERTa in all aspects:







Finally, below is the plot which shows the comparison of the **Main Metric (F1)** of comparison between the Optimized and the Baseline RoBERTa model. The F1 score of the Optimized model is significantly higher and thus shows better performance than the base model as can be noticed from the plot below:



Deployed Model and Inference Setup

Model Used

- Model: Fine-tuned RoBERTa-base transformer
- Source: Best-performing model from Assignment 2 (best_model.pt)
- Task: Binary sentiment classification (Positive / Negative)
- Loss Function (Training): Cross-Entropy Loss
- RoBERTa encoder
- Dropout layer
- Linear classification head producing two logits

Tokenization and Input Handling

- Tokenizer: RoBERTa tokenizer (byte-level BPE)
- Max sequence length: 256 tokens
- Outputs: Input IDs and attention masks.

Prediction Pipeline and Calibration (Sentiment Prediction Pipeline)

The deployed pipeline follows these steps:

1. User enters a movie review in the web interface
2. Text is tokenized using the RoBERTa tokenizer
3. Tokenized inputs are passed through the fine-tuned model
4. Model outputs raw logits for positive and negative sentiment
5. Temperature scaling is applied to logits
6. Final prediction, confidence score, and decision strength are displayed on the web app

(To avoid overconfident predictions and better reflect real-world sentiments, temperature scaling is applied): Temperature used: T = 2.5

Decision Strength (Logit Margin)

This metric:

- Measures internal model certainty
- Is independent of probability calibration
- Identifies borderline or ambiguous sentiment cases

The web app uses this value to issue warnings when predictions are weak or uncertain.

Web Application Implementation (Streamlit-Based Deployment)

The sentiment analysis system is deployed as a Streamlit web application, offering:

- Text input for user reviews
- Real-time prediction
- Confidence score display
- Decision strength visualization
- Warning messages for ambiguous sentiment

Key Take-Aways and Insights

- Preprocessing effort is significant: Even minimal text cleaning, dataset splitting, and tokenization required substantial care to avoid data leakage.
- Transformer fine-tuning is powerful but expensive: Small hyperparameter changes resulted in large performance differences, but training required long runtimes (~16 hours total).
- F1-score is more informative than accuracy: Especially for sentiment classification, F1-score provided a better balance between precision and recall.
- Probability calibration matters: Raw SoftMax outputs were often overconfident. Temperature scaling greatly improved interpretability.

What Would Be Done Differently

- A three-class sentiment dataset (positive / neutral / negative) would be preferred to better handle ambiguous or neutral reviews
- More systematic hyperparameter search methods could reduce training time.
- Earlier integration of deployment considerations could reduce refactoring effort later.
- The model was trained only on movie reviews. Fine-tuning on larger and more domain-diverse datasets (e.g., product reviews or social media text) could improve generalization and robustness across different sentiment styles.

Conclusion

This project demonstrates a complete end-to-end deep learning workflow: from dataset selection and preprocessing to model training, optimization, evaluation, deployment, and reflection. The fine-tuned RoBERTa model achieved strong performance on the IMDB dataset and was successfully transformed into a usable web application.

The project highlights both the strengths and limitations of deep learning for sentiment analysis and provides practical insights into real-world NLP system design.