# Fine-Tuning Transformer Models for NLP Tasks

## (Sentiment Analysis)

## Team Members


- Ahmed Abdelaziz Hareedy
- Anas Ahmed Desoky
- Salah Amer Mohamed
- Menna Mohamed Abdelhady
- Horia Ahmed Abdelatief
- Mayar Mohamed Khedr

**Under Supervision:**

**Eng. Maryam Mahmoud**

## 1. Problem Statement

The goal is to build a sentiment analysis model that accurately classifies IMDB movie reviews into **positive** or **negative** sentiments. This task is valuable in understanding audience feedback for content creators and businesses.

## 2. Dataset and Preprocessing

**Dataset Used**: IMDB 50K Movie Reviews

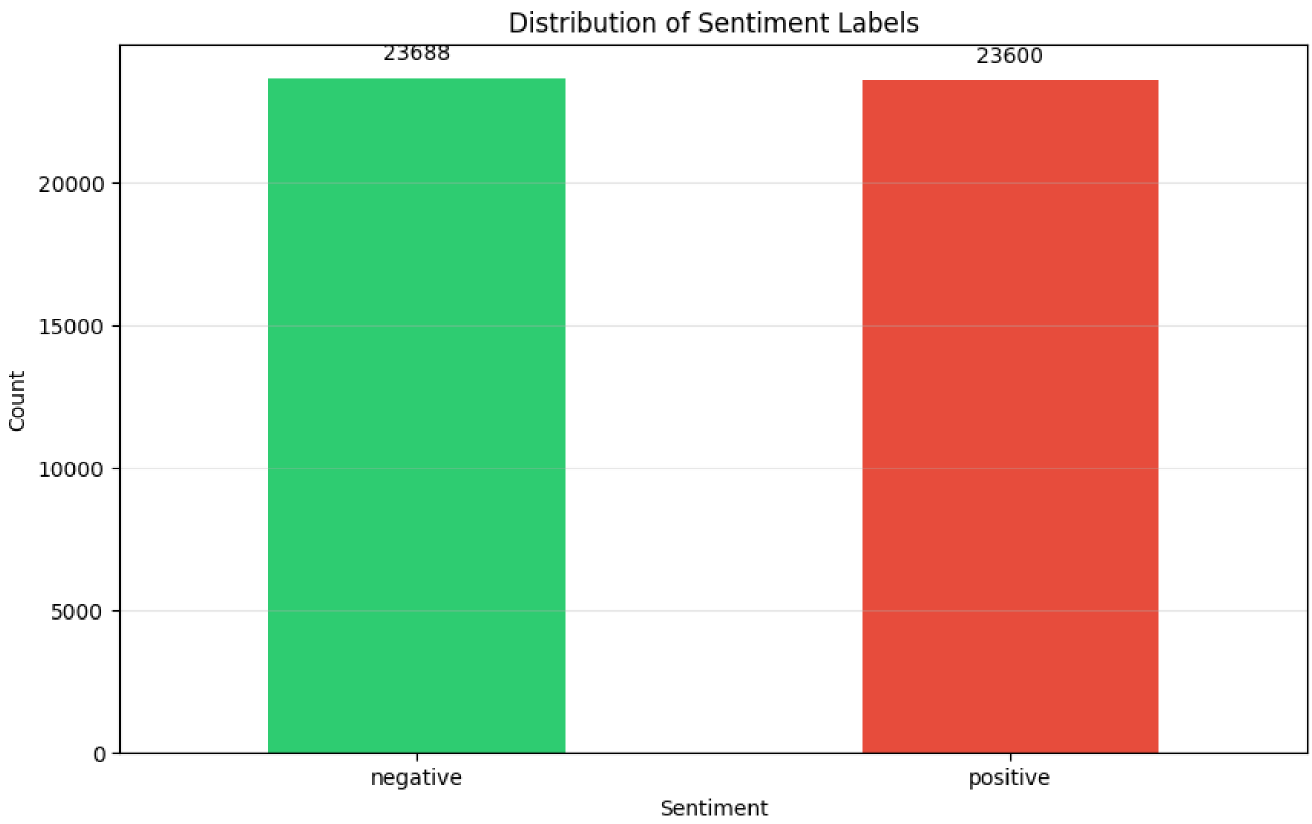
- 50,000 reviews, equally balanced (25k positive / 25k negative)
- Text + Sentiment (positive or negative)

## Preprocessing Steps:

- **HTML Tag Removal**: Strips out <div>, <br>
- **Lowercasing**: Converts all characters to lowercase to avoid duplication (Movie, movie → movie).
- **URL, Mention & Hashtag Removal**: Eliminates irrelevant patterns like links and Twitter-style tags.
- **Punctuation & Special Characters**: Removed using regular expressions.
- **Tokenization**: Splits sentences into words.
- **Stopword Removal**:
  - **First**: Removed all Common English stopwords (“the”, “is, “not”).
  - **Second in Optimization:** Remove all Common stopwords except the negative words like (not, nor, shouldn’t)

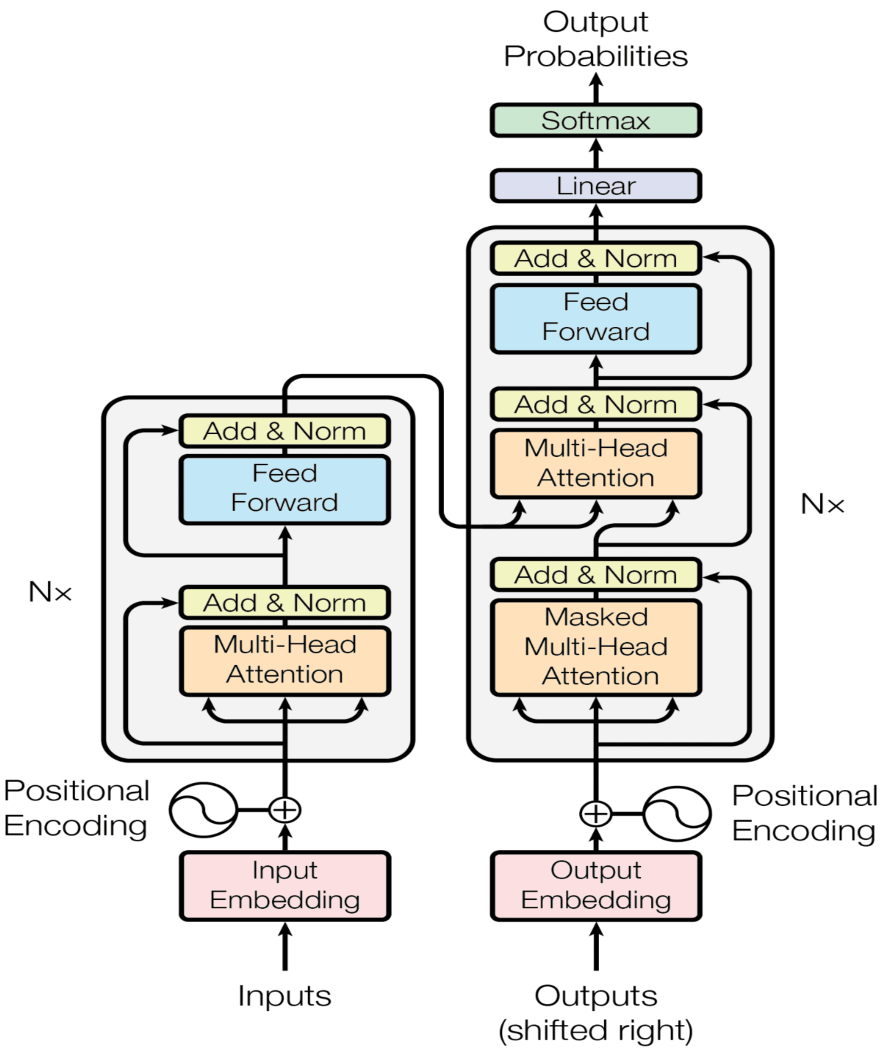
- **Lemmatization** (**Use before Optimization)** Reduces words to their base form (running → run).
- **Encoding Label**: Sentiment labels were converted to numeric form (positive → 1, negative → 0).

## Dataset After Preprocessing



## ➢ Solve our problem using Transformers

**Transformers Architecture**




## 3. Models Selection Rationale

For this sentiment analysis task, we used two transformer-based models:

- **DistilBERT**: A distilled version of BERT, chosen for its significantly reduced size and faster inference time while retaining over 95% of BERT's performance. It is ideal for real-time or resource-limited environments.
- **RoBERTa (Robustly Optimized BERT Approach)**: Selected for its enhanced training methodology, including more data,

longer sequences, and dynamic masking, resulting in state-of-the-art performance on multiple NLP benchmarks. It is ideal when accuracy is prioritized over speed.

These models were selected to compare the trade-offs between efficiency and performance in transformer-based sentiment classification.

## 4. Implementation Detail:

- **Pretrained Models:** HuggingFace Transformers library was used to load distilbert-base-uncased and roberta-base.
- Tokenization: Used the corresponding tokenizer (DistilBertTokenizerFast and RobertaTokenizerFast) to convert raw text into model-compatible input IDs and attention masks.
- Fine-Tuning:
    - Both models were fine-tuned on the preprocessed dataset.
    - Used AdamW optimizer and CrossEntropyLoss.
    - Training was conducted over 5–10 epochs with early stopping based on validation loss.
- **Train/Test Split: The dataset was split 70% Training, 15% Validation, 15% Test**

**Optimize RoBERTa Training Details:**

**Using following:**

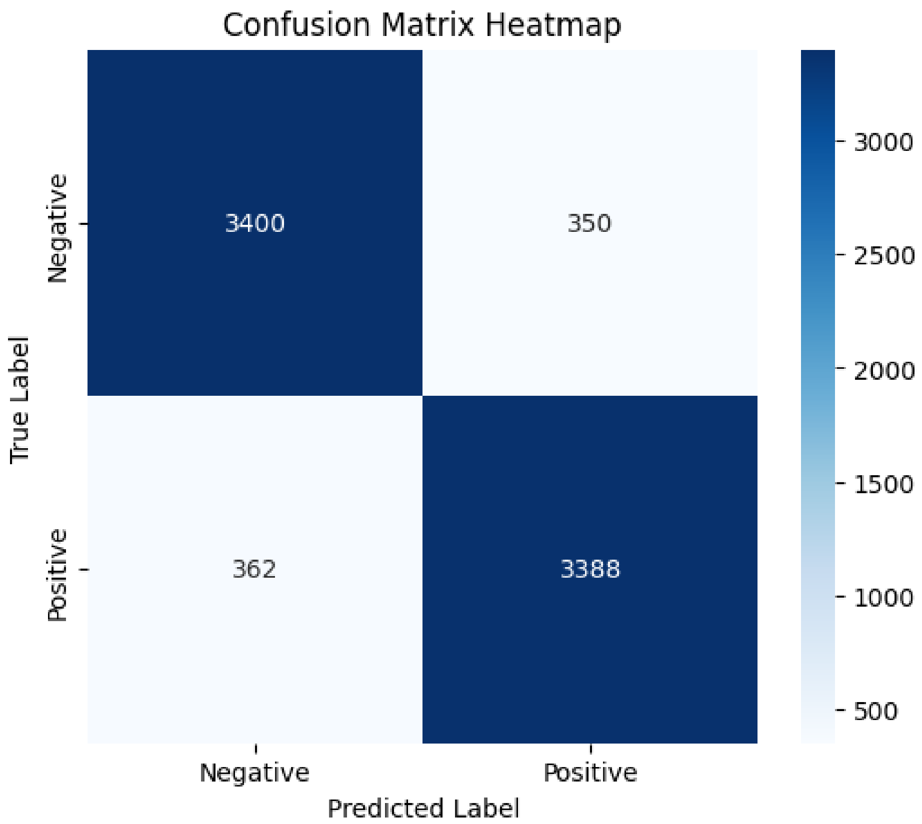
- Cosine Learning Rate Scheduler with Warmup
- Weight Decay in Optimizer : helps reduce overfitting.
- Early Stopping
- Increase number of Epochs

**Train/Test Split: The dataset was split 70% Training, 21% Validation, 9% Test**

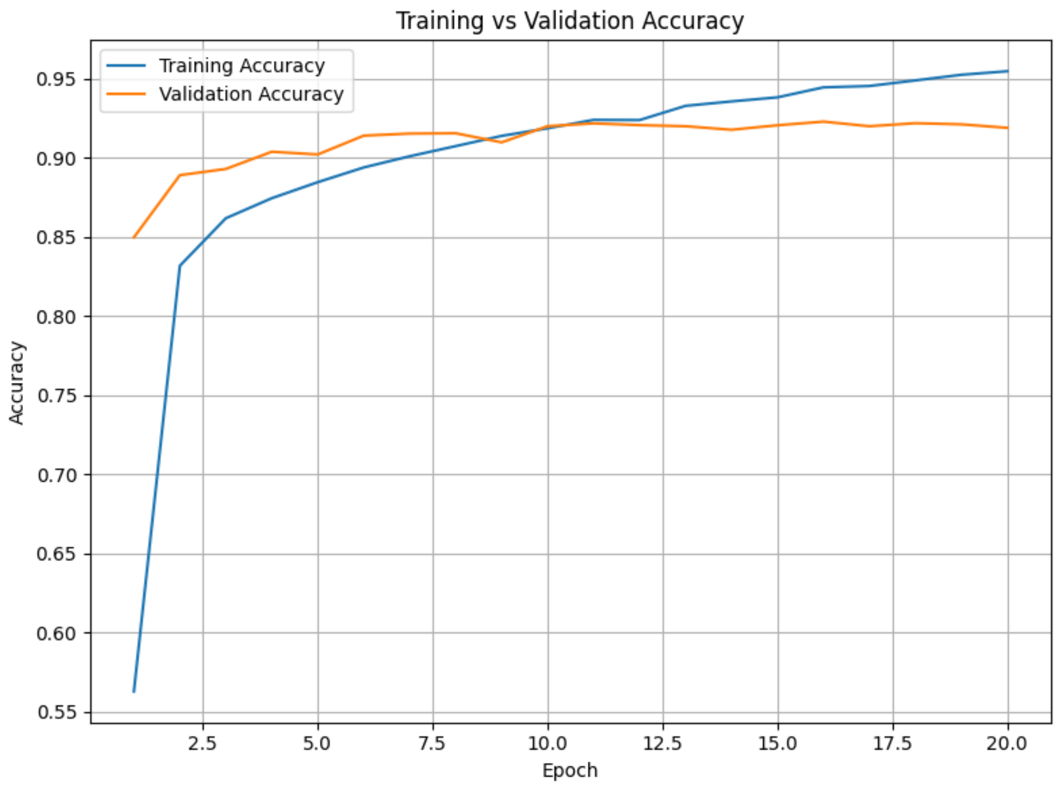
## 5. Results and Analysis

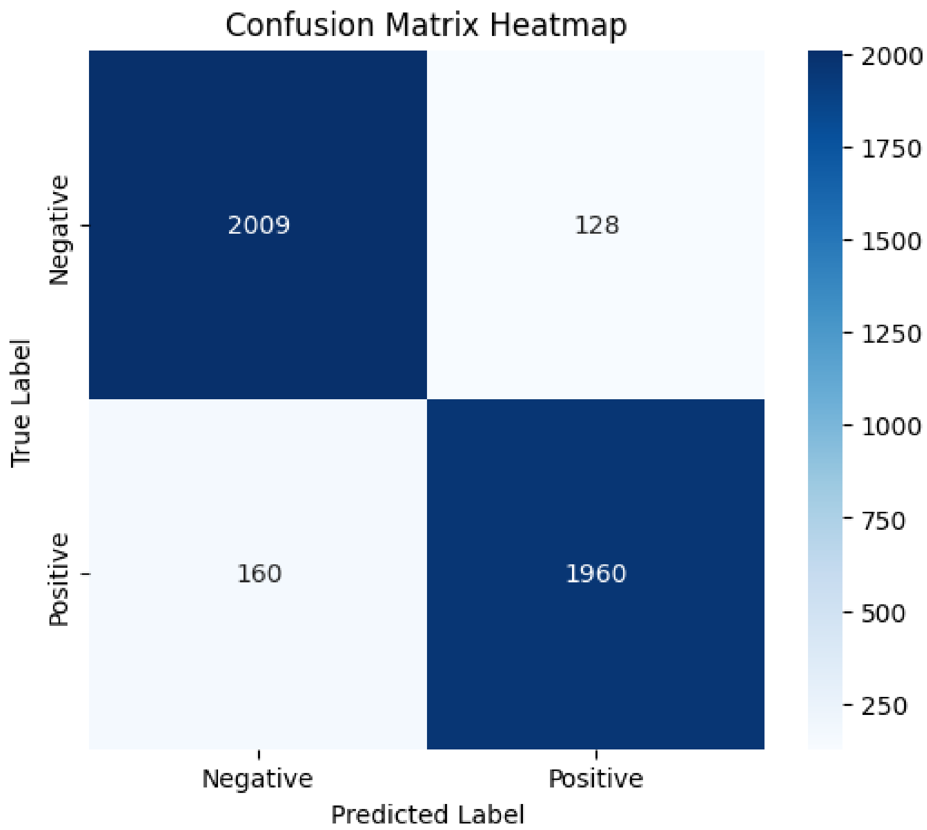
| **Metric** | **DistilBERT** | **RoBERTa** | **Optimized RoBERTa** |
|---|---|---|---|
| Accuracy | 89.83% | 90.51% | 93.27% |
| Precision | 89.83% | 90.51% | 93.28% |
| Recall | 89.83% | 90.51% | 93.28% |
| F1-Score | 89.83% | 90.51% | 93.26% |
| Avg. Inference Time | 0.0042 s | ~0.005–0.006 s | ~0.0055 s |

## RoBERTa Analysis Before Optimization




# RoBERTa Analysis After Optimization



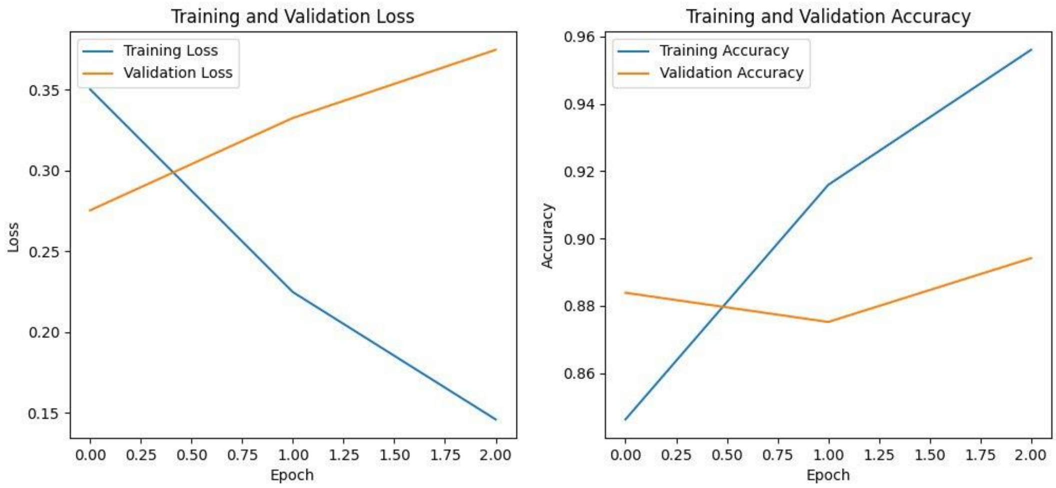


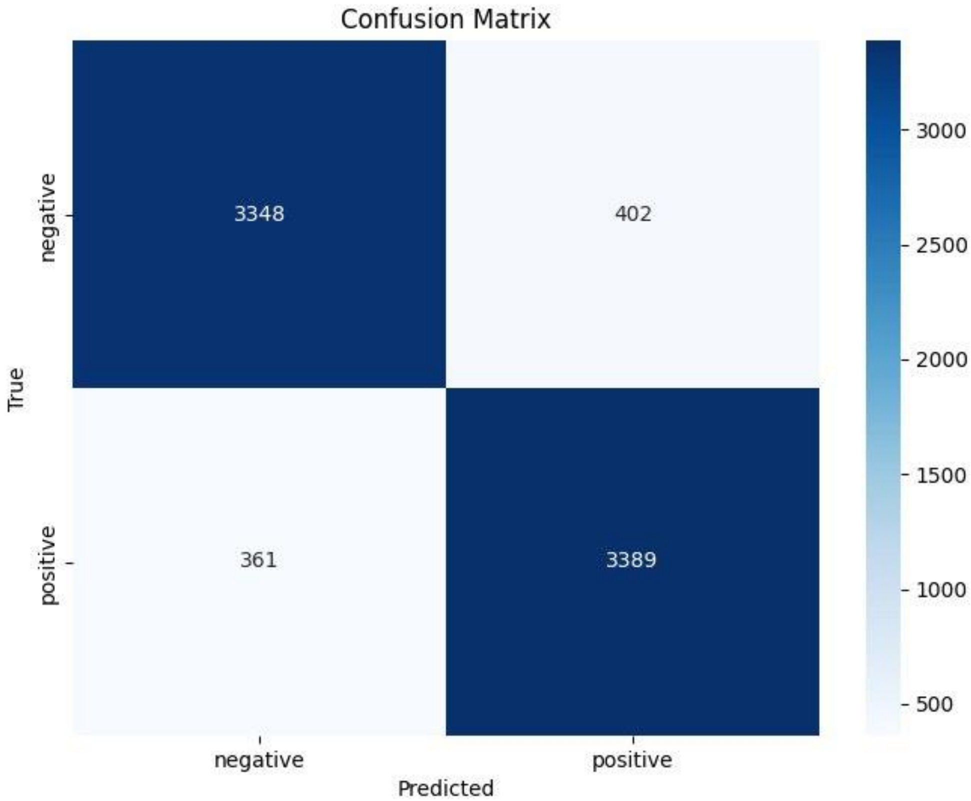

# RoBERTa Prediction

```python
# Example usage
sample_texts = [
    "I absolutely love this product! It's amazing!",
    "This is the worst experience I've ever had."
]

for text in sample_texts:
    sentiment = predict_sentiment(text, model, tokenizer, device)
    print(f"Text: {text}")
    print(f"Sentiment: {sentiment}\n")
```

```
Text: I absolutely love this product! It's amazing!
Sentiment: positive

Text: This is the worst experience I've ever had.
Sentiment: Negative
```

# DistilBERT Analysis



# DistilBERT Prediction

```python
    tokenizer = DistilBertTokenizer.from_pretrained('sentiment_model')
    for text in sample_texts:
        sentiment = predict_sentiment(text, model, tokenizer)
        print(f"Text: {text}")
        print(f"Sentiment: {sentiment}\n")
```

```
Sample Predictions:
/tmp/ipykernel_31/3517145775.py:46: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses t
  model.load_state_dict(torch.load('best_model.pt'))
Text: I absolutely love this product! It's amazing!
Sentiment: positive

Text: This is the worst experience I've ever had.
Sentiment: negative

Text: The movie was fantastic, with great acting and a compelling story.
Sentiment: positive

Text: I was really disappointed with the service at this restaurant.
Sentiment: negative
```

## ➢ Model Api Result