UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

# Deep Learning for NLP – Twitter Sentiment Analysis with (Distil)BERT

Student name: *Nikitas Rafail Karachalios*
*sdi: -*

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Spring Semester 2025*

## Contents

## 1. Abstract

This project is all about using BERT and DistilBERT for Twitter sentiment analysis. The goal was to build a model that can tell if a tweet is positive or negative. We used the same dataset as in the previous project, but this time we went for a much more powerful approach: fine-tuning a pre-trained BERT model.

We started with a simple baseline and then improved things step by step, always checking our results and making changes based on what worked best. We also used Optuna for hyperparameter tuning to squeeze out the best possible performance. In the end, we got a model that was not only accurate but also pretty efficient. The report will walk you through all the experiments, what we learned, and how we made decisions along the way.

## 2. Data Processing and Analysis

### 2.1. Initial Scan

We start by printing the label distribution of the training and validation dataset. (Test dataset does not have labels). Also, we check for missing/duplicate values so as to be sure about the data quality.

```
Basic Statistics for Train Dataset:
Number of samples: 148388
Positive labels: 74196 (50.00%)
Negative labels: 74192 (50.00%)
Missing values in each column:
ID 0
Text 0
Label 0
dtype: int64
Duplicate tweets: 0 (0.00%)

Basic Statistics for Validation Dataset:
Number of samples: 42396
Positive labels: 21199 (50.00%)
Negative labels: 21197 (50.00%)
Missing values in each column:
ID 0
Text 0
Label 0
dtype: int64
Duplicate tweets: 0 (0.00%)

Basic Statistics for Test Dataset:
Number of samples: 21199
Missing values in each column:
ID 0
Text 0
```

dtype: int64
Duplicate tweets: 0 (0.00%)

It is clear that we don't have any issues with data quality. Also, both datasets (training and validataion) are perfectly balanced.

## 2.2. Pre-processing

Just like last time, we had to clean up the tweets before feeding them to BERT. But since BERT is already pretty smart about language, we didn't need to go as crazy as with Word2Vec. Still, some steps really helped:

1. **Lowercasing**: BERT-base-uncased expects lowercase input.
2. **URL removal**: URLs are just noise for sentiment.
3. **User mentions removal**: Usernames don't matter for sentiment.
4. **Hashtag symbol removal (keeping word)**: Hashtags can be important, so we keep the word.
5. **Emoticon replacement**: We replaced emoticons like ":)" with words like "happy" so BERT can understand the feeling.
6. **Negation handling**: Expanding things like "can't" to "can not" helps BERT catch the negation.
7. **Corrections and abbreviations**: Stuff like "lol" becomes "laugh out loud", "u" becomes "you", etc.
8. **Repeated character handling**: "soooo" becomes "soo" to reduce noise.
9. **Extra spaces**: Removed for tidiness.

We didn't remove stopwords or do lemmatization, since BERT can handle those just fine. The preprocessing was all about making sure the tweets were as clear as possible for the model, but not overdoing it.

## 2.3. Tokenization and Data Partitioning

(Distil)BERT comes with its own tokenizer, so we used that. The datasets were already split into train, validation, and test sets, so we just loaded them and applied our preprocessing. For experimenting though, apart from the whole datasets, we used a mini-train dataset (10% of the actual dataset) so as to test a wide variance of hyperparameter values and combinations and avoid time-consuming trials. Also, during some of the early fine-tuning and hyperparameter search experiments, we used a 10% sample of the validation set for faster evaluation. We later realized this is not best practice, as it can make validation metrics less reliable, but for early steps and key differences, this method helped a lot. For all final model evaluations, we used the full validation set.

## 2.4. Further Analysis, preprocessing results

We remind from the previous project that our datasets are perfectly balanced and not duplicate or empty tweets appear so no issues with data quality. Now, let's see our preprocessing in action by comparing the head of the training dataset before and after preprocessing.

```
0  189385      @whoisralphie dude  I'm so bummed ur leaving!      0             dude i am so bummed your leaving !
1   58036  oh my god, a severed foot was foun in a wheely...      0  oh my god a severed foot was found in a wheely...
2  190139  I end up &quot;dog dialing&quot; sumtimes. Wha...      1  i end up quot dog dialing quot sometimes. what...
3   99313                        @_rachelx meeeee toooooo!      0                                       mee too !
4  157825  I was hoping I could stay home and work today,...      0  i was hoping i could stay home and work today ...
```

Above we can clearly see the effect of the preprocessing. "I" is same as "i" for the reduction of vocabulary size, spelling mistakes are corrected ("sumtimes" to "sometimes"), consecutive letters are reduced to 2, and punctuation is removed except for "!" and "?" since these contribute to sentiment.

**Most frequent words**

Let's see the visualization of training and validation for most common words (excluding stop words) before and after preprocessing to feel even more confident about our text cleaning.

For training dataset, before preprocess (with or without) stopwords we see noisy, imbalanced data with useless punctuation.

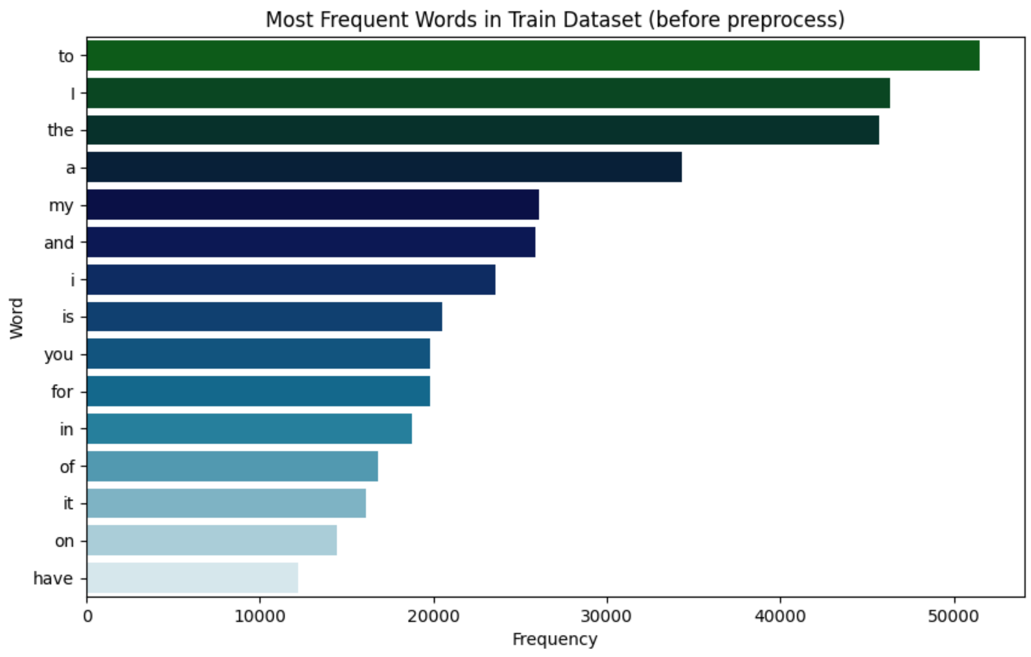

Figure 1: Train Dataset no preprocess, no stopword removal

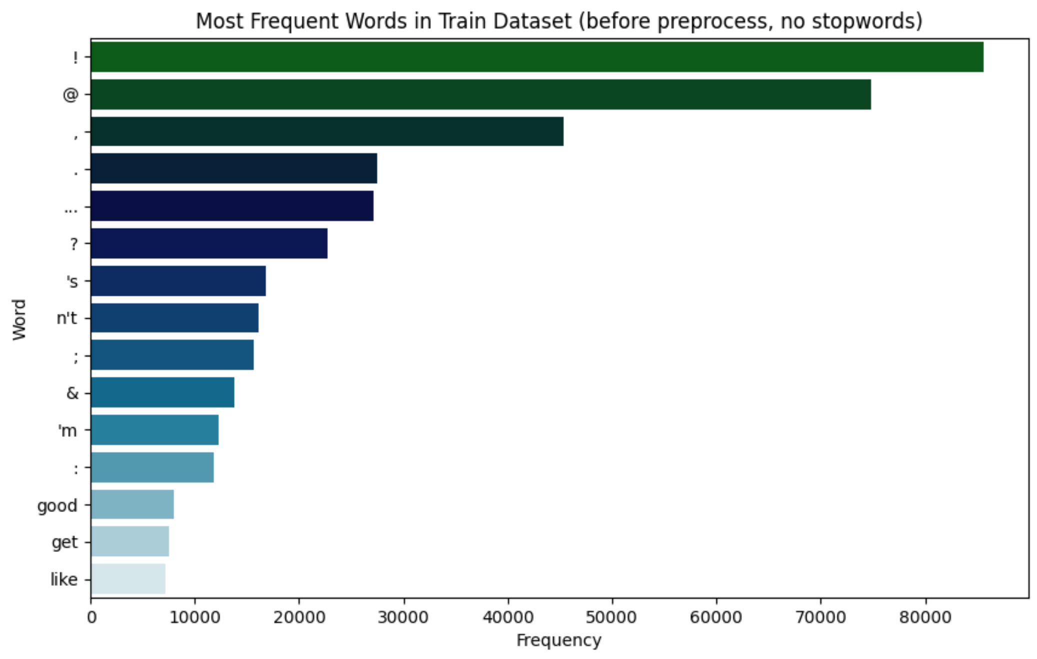

Figure 2: Train Dataset no preprocess, with stopword removal

Same observations for the validation dataset, as well:

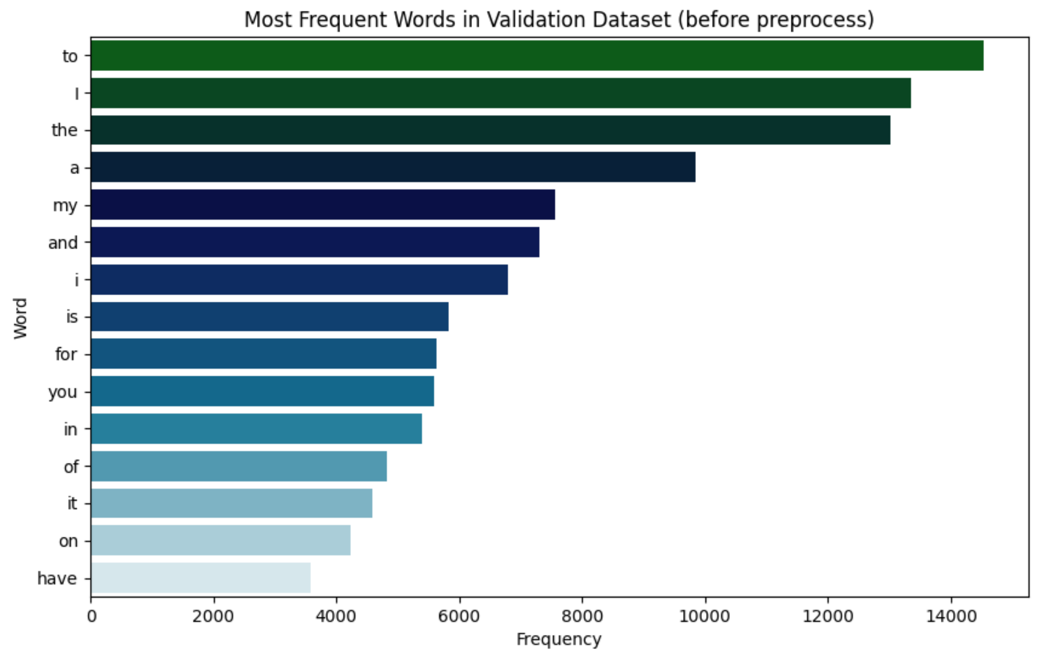

Figure 3: Validation Dataset no preprocess, no stopword removal

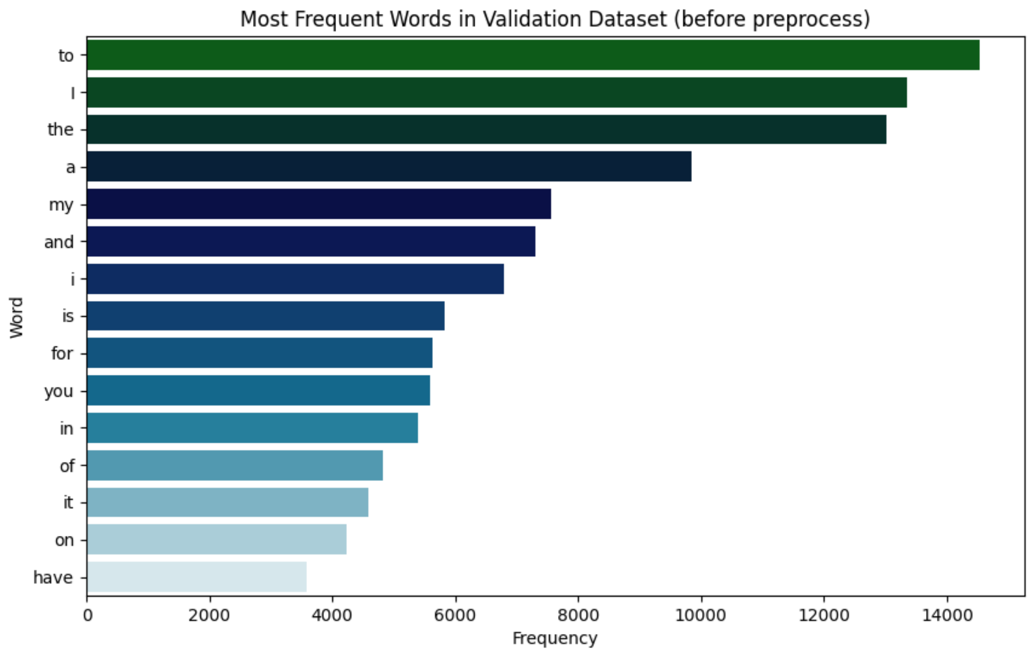

Figure 4: Validation Dataset no preprocess, with stopword removal

Now, it's time to see our preprocessed data for training and validation. Below, it is obvious (especially when stop words are removed and we have a better picture) that the data are less noisy, the useless punctuation is cleared, the meaningful one which contributes to sentiment ("!", "?") is kept. Also, "I" and "i" are treated in the same way. In figure 6, in which we removed stopwords, noisy symbols/words such as "@", ",", ".", "n't" (etc.) are cleared and crucial words for sentiment such as "good", "like", "laugh" (etc.) popped up. These key differnces can be seen, also, in the below validation set visualisations. Although data without stopwords seem more organised, the removal of stopwords led to worse performance so we decided to keep them.

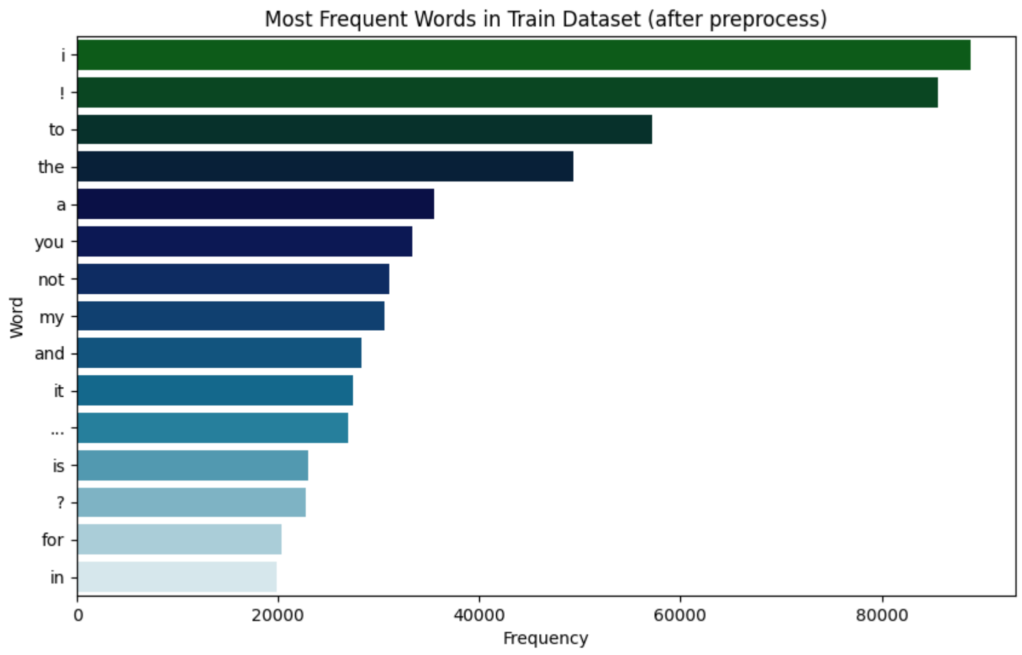

Figure 5: Train Dataset preprocessed, no stopword removal

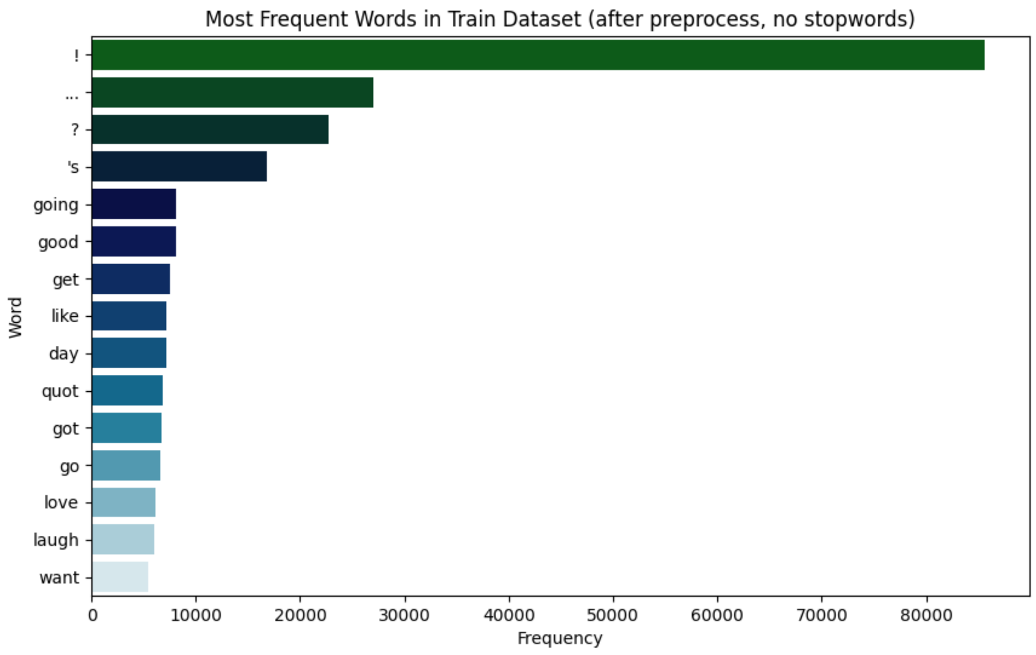

Figure 6: Train Dataset preprocessed, stopword removal

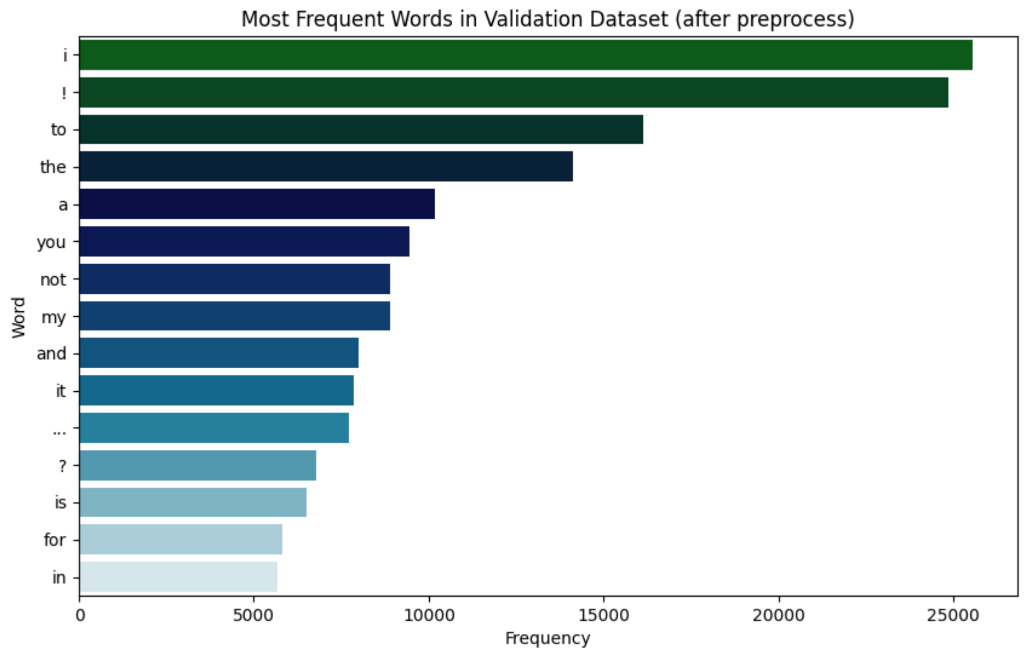

Figure 7: Validation Dataset preprocessed, no stopword removal

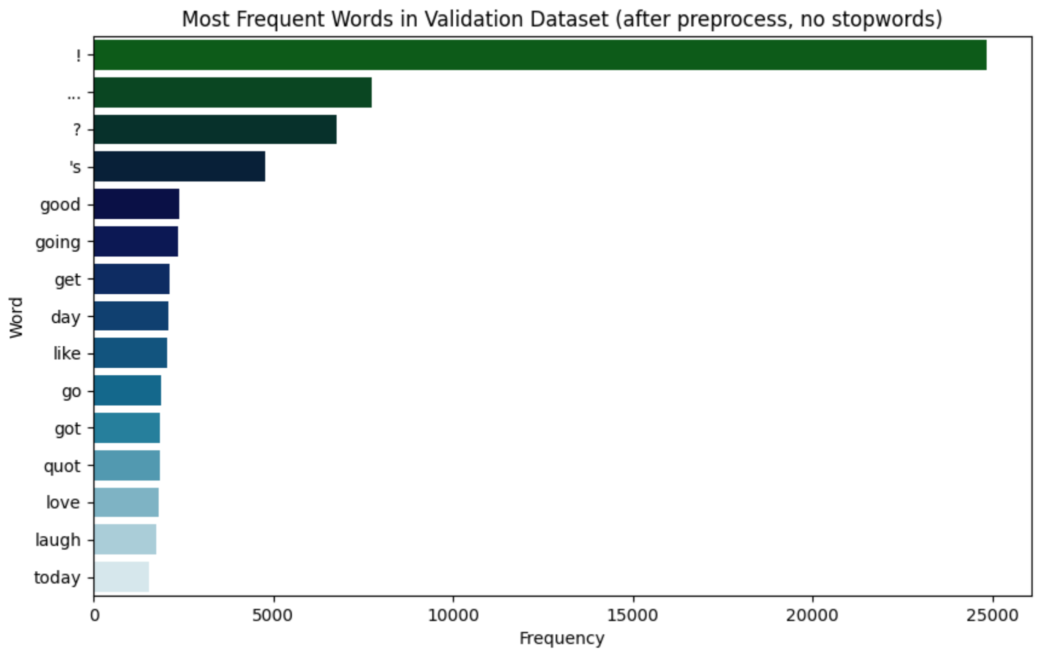

Figure 8: Validation Dataset preprocessed, stopword removal

## 3. Model and Experiments

### 3.1. Token Count Distribution and Sequence Length Justification

Before running any big experiments, we checked how long tweets actually are after preprocessing. The stats were eye-opening:

- Average token length: 20 tokens per tweet
- 95th percentile: 35 tokens
- Max: 70 tokens

This means our default sequence length of 128 was total overkill! Most tweets are way shorter, so we were just padding a lot. This analysis led us to try shorter sequence lengths (64, then 48), which made training much faster and didn't hurt performance. Sometimes, it also improved performance. The pattern in the train data set sample and in the full train data set is almost identical, making the sample possibly more reliable.

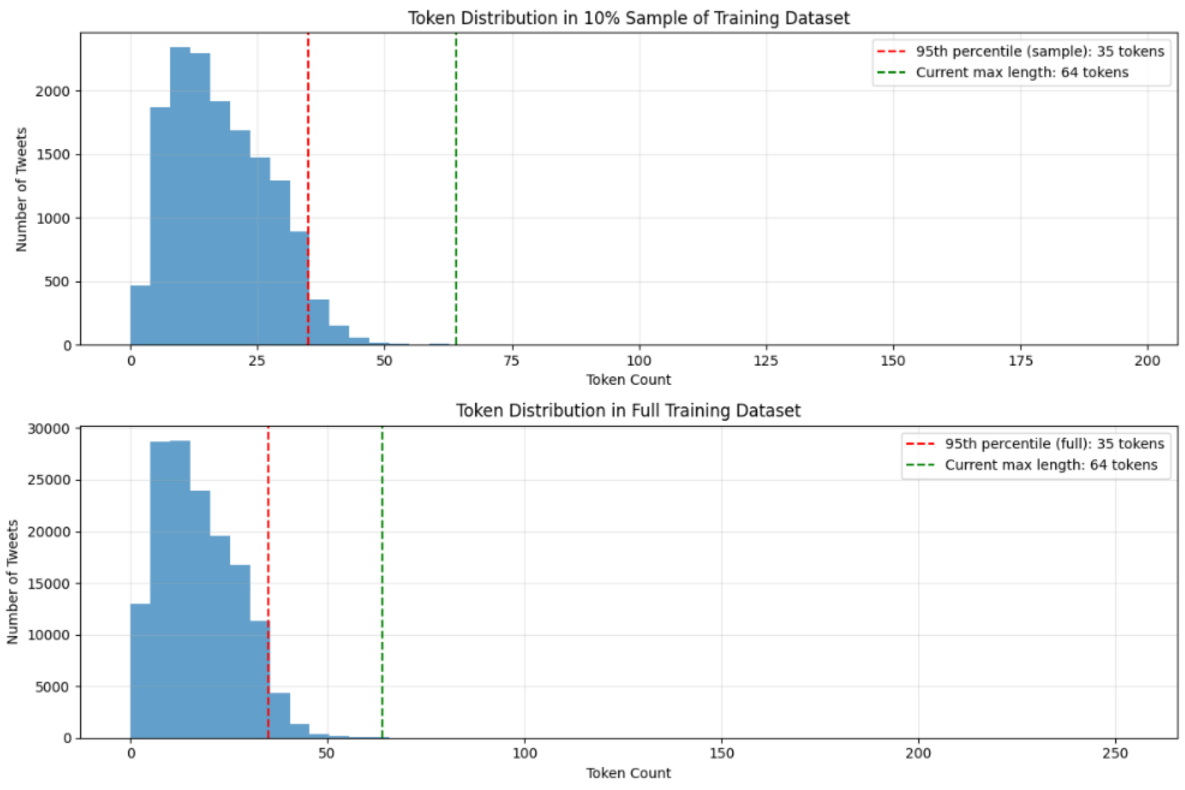

Figure 9: Distribution of token count

### 3.2. Table of Key Trials

We ran a ton of experiments, always using the results to decide what to try next. Here's a table of the most important trials (see below for discussion):

**BERT**

| Trial | Seq. Len | Batch | LR | Optimizer | Weight Decay | F1 Score | Notes |
|---|---|---|---|---|---|---|---|
| Baseline | 128 | 32 | 2e-5 | AdamW | 0 | 0.8250 | Overfits after one epoch |
| Preproc+ | 128 | 32 | 2e-5 | AdamW | 0 | 0.8233 | Improved preprocess func |
| EarlyStop+Sched | 128 | 32 | 2e-5 | AdamW | 0 | 0.8273 | Early stopping, scheduler |
| LR=5e-5 | 128 | 32 | 5e-5 | AdamW | 0 | 0.8164 | Overfits faster, worse F1 |
| Batch=64 | 128 | 64 | 1e-4 | AdamW | 0 | 0.8158 | Lower precision, higher recall |
| SeqLen=64 | 64 | 32 | 2e-5 | AdamW | 0 | 0.8273 | Same F1, faster |
| WeightDecay | 64 | 32 | 2e-5 | AdamW | 0.01 | 0.8248 | Slight recall boost |
| Dropout=0.3 | 64 | 32 | 2e-5 | AdamW | 0 | 0.8200 | Too much regularization |
| SeqLen=48 | 48 | 32 | 2e-5 | AdamW | 0 | 0.8311 | Best on 10% sample |
| FullData Seq64 | 64 | 32 | 2e-5 | AdamW | 0 | 0.8524 | Best overall F1 |
| FullData Seq48 | 48 | 32 | 2e-5 | AdamW | 0 | 0.8512 | 30% faster |
| Optuna (10%) | 48 | 32 | 1.52e-5 | Adam | 0 | 0.8314 | Best on 10% sample |
| Optuna Full | 48 | 32 | 1.52e-5 | Adam | 0 | 0.8521 | Best precision, 2 epochs |

Table 1: Key BERT experiment trials and results.

**DistilBERT**

| Trial | Seq. Len | Batch | LR | Optimizer | Dropout | F1 Score | Notes |
|---|---|---|---|---|---|---|---|
| Baseline | 64 | 32 | 2e-5 | AdamW | 0.1 | 0.8187 | Overfits quickly after 2 epochs |
| SeqLen=48 | 48 | 32 | 2e-5 | AdamW | 0.1 | 0.8156 | Faster training, slight F1 loss |
| LR=1e-5 | 48 | 32 | 1e-5 | AdamW | 0.1 | 0.8176 | More stable training |
| Batch=64 | 48 | 64 | 1e-5 | AdamW | 0.1 | 0.8110 | Lower performance |
| Scheduler | 48 | 32 | 1e-5 | AdamW | 0.1 | 0.8195 | Improved metrics |
| FullData Manual | 48 | 32 | 1e-5 | AdamW | 0.1 | 0.8455 | Manual full-data run (3 epochs) |
| Optuna (best) | 48 | 32 | 1.94e-5 | AdamW | 0.2 | 0.8246 | Best config on 10% sample |
| FullData Optuna | 48 | 32 | 1.94e-5 | AdamW | 0.2 | 0.8464 | Final best DistilBERT model |

Table 2: Key DistilBERT experiment trials and results.

### 3.3. Timeline: Step-by-Step Experimental Reasoning and Discussion

***3.3.1. BERT.*** **Baseline:** We started with a classic BERT fine-tuning setup (128 tokens, AdamW, 2e-5, batch 32). The F1 score was 0.8250, but the model overfit after just one epoch: training loss kept dropping, but validation loss doubled by epoch 4. This is a textbook case of overfitting, where the model memorizes the training data but can't generalize.

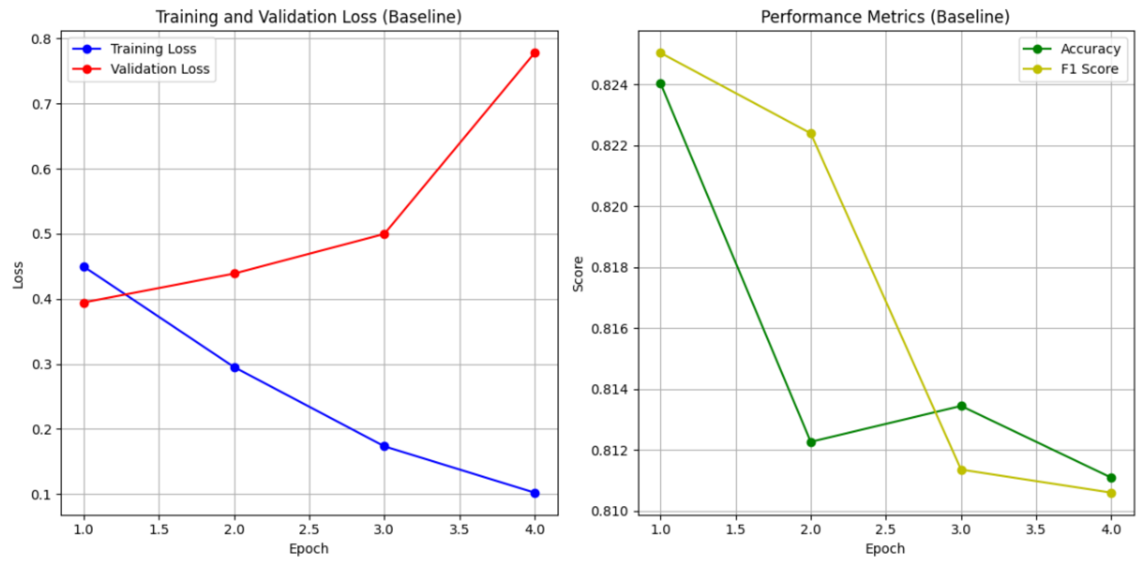

Figure 10: Baseline learning curves - BERT

**Preprocessing tweaks:** We tried more advanced preprocessing (extra emoticons, contractions, abbreviations). It made the data cleaner, and f1 and accuracy increased.

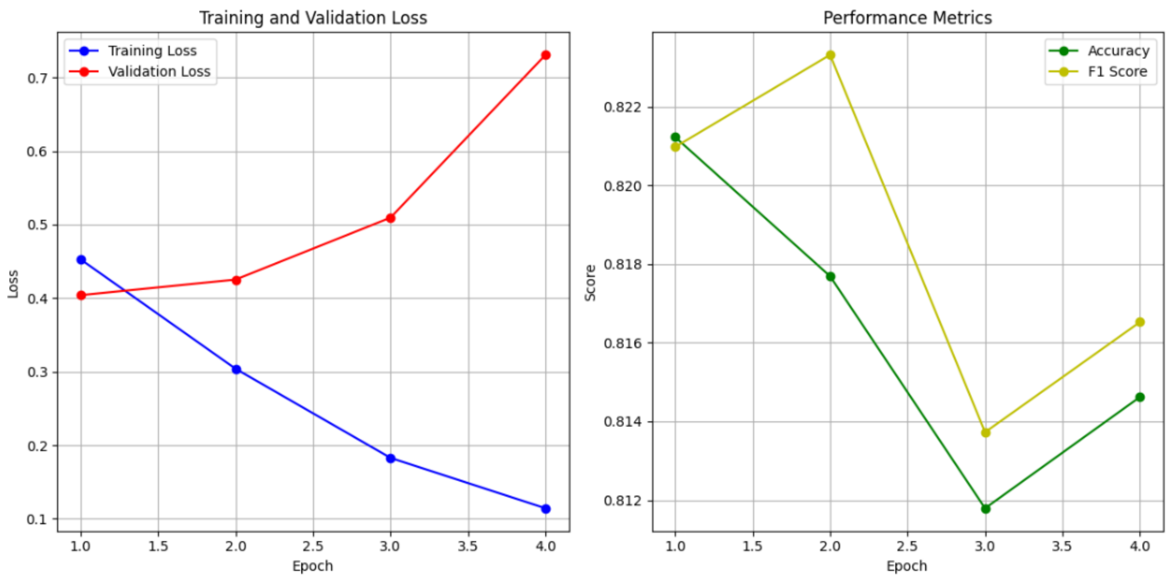

Figure 11: Enhanced Preproccessing - BERT

**Early stopping and scheduler:** To fight overfitting, we added early stopping (patience=2) and a linear learning rate scheduler. This helped a lot: the model stopped before it started to memorize, and F1 went up to 0.8273. Also, improved balance between precision and recall compared to previous experiments.

**Learning rate and batch size:** We tried a higher learning rate (5e-5), maybe the most effective hyperparameter as mentioned in class, but the model overfit even faster and F1 dropped. A bigger batch size (64) also hurt performance, and confusion matrix as you can see in figure 14 was imbalanced. This matches the theory: too high a learning rate can make the model jump over good solutions, and big batches can make updates less noisy but also less effective for generalization. Batch size 16 was not more capable to generalize and its metrics were worse.

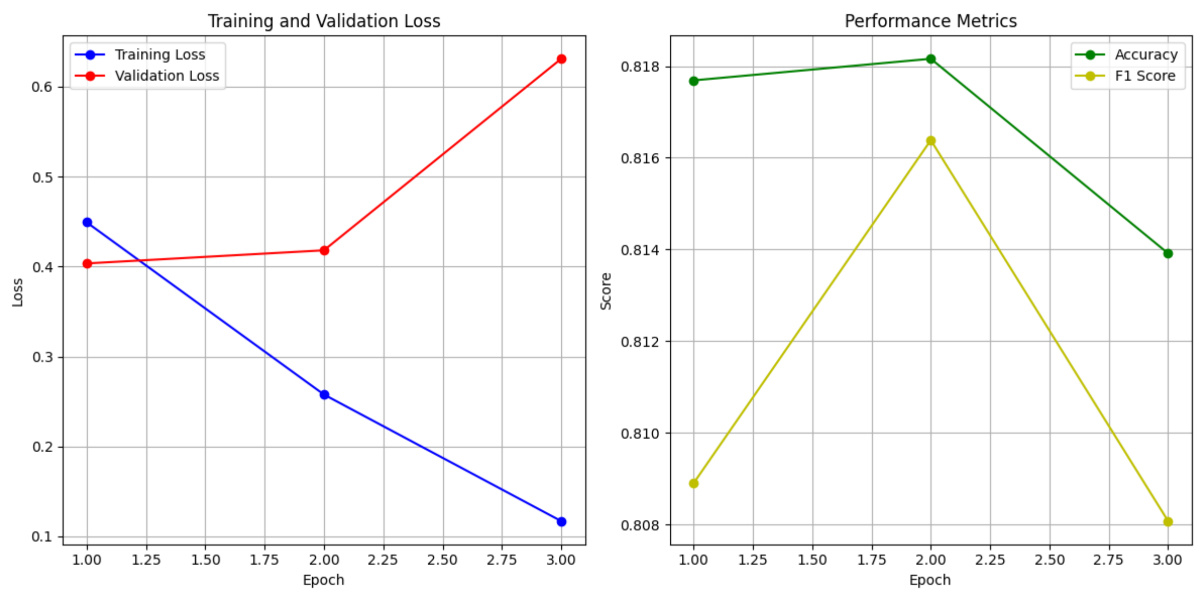

Figure 12: Learning Rate (5e-5) - BERT

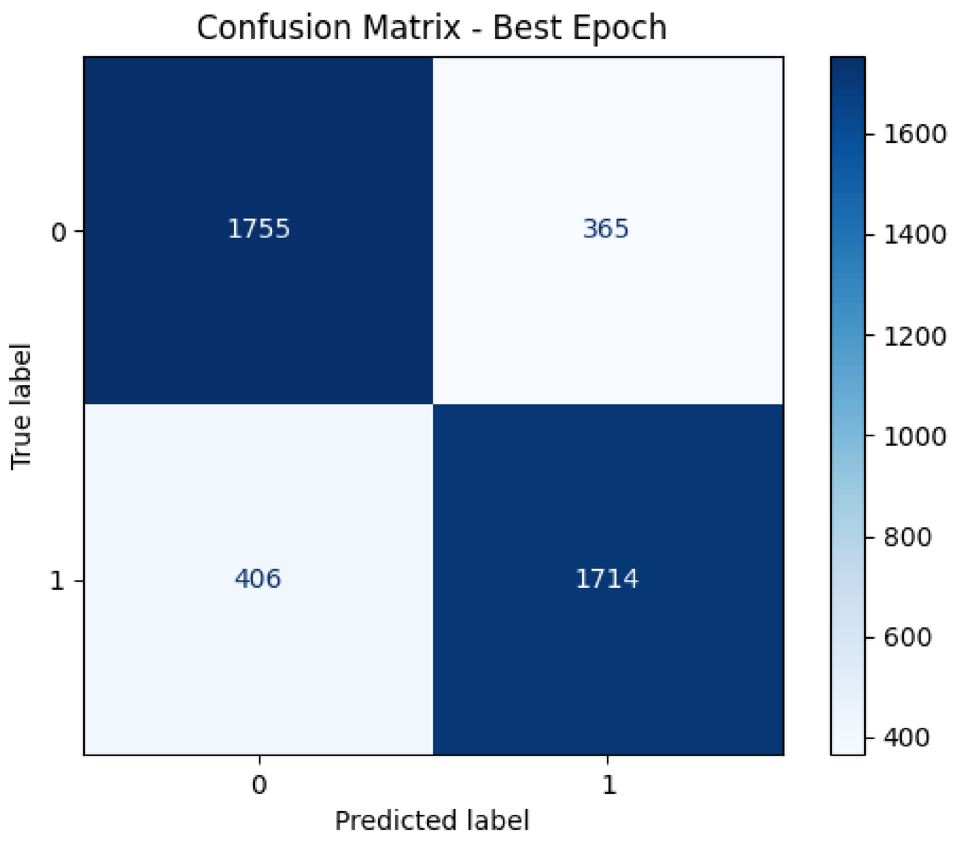

Figure 13: Conf Matrix in mini val dataset - BERT

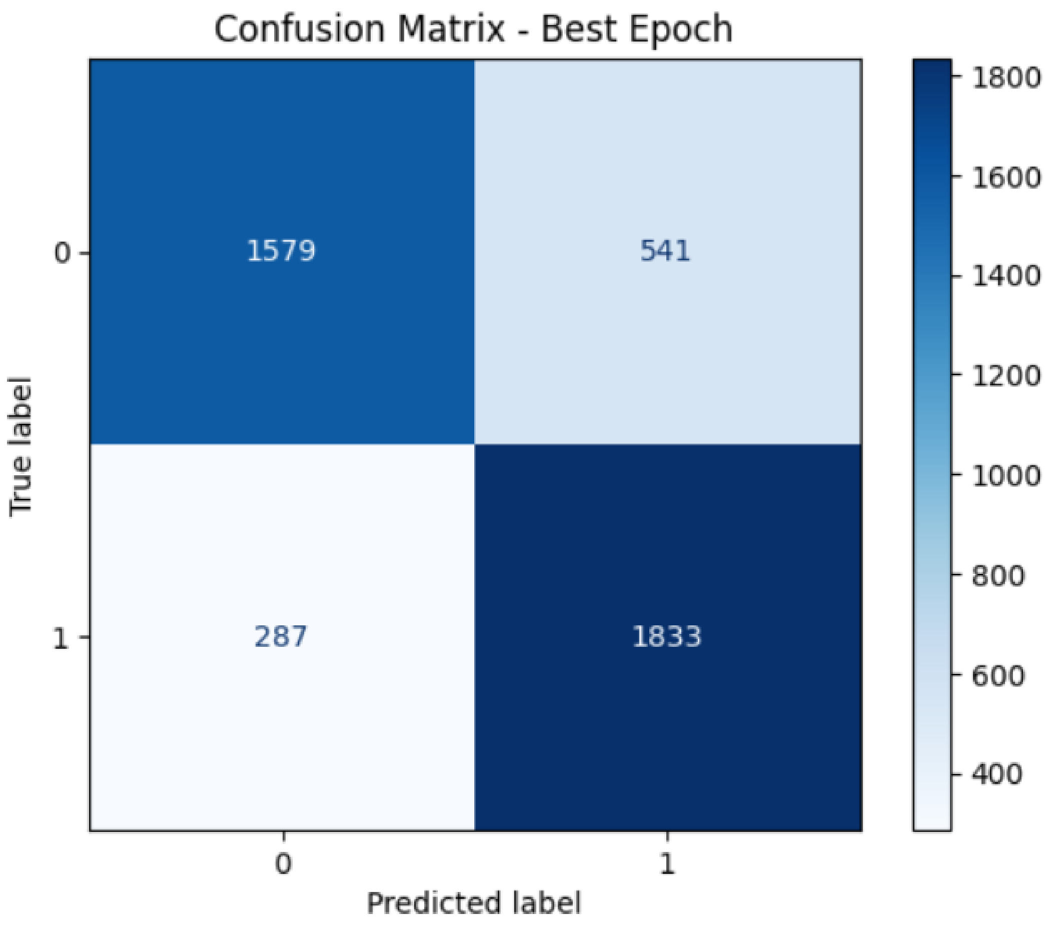

Figure 14: Batch size 64 in mini val dataset - BERT

**Sequence length optimization:** After analyzing the token distribution, we realized 128 tokens was way too much. We dropped to 64, then 48. F1 stayed the same or even improved, and training was much faster. Based on visualisations, it is clear that 48 was the right value. The 48-token model outperformed the 64 one on all key metrics and its training time was surely 2 times faster.

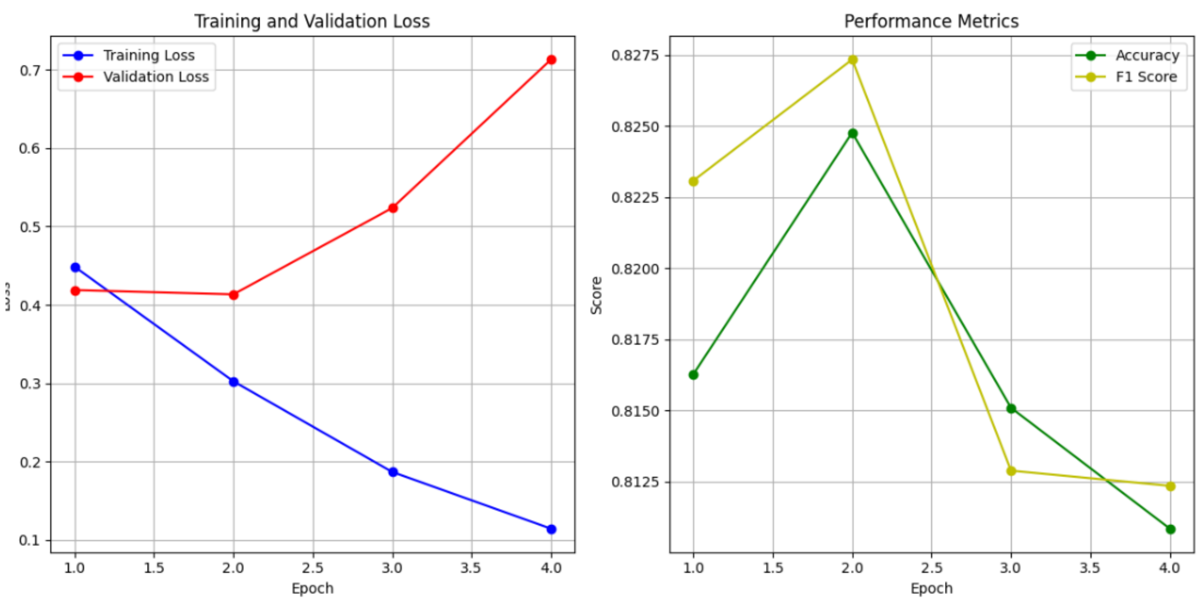

Figure 15: LR with max seq len 64 - BERT

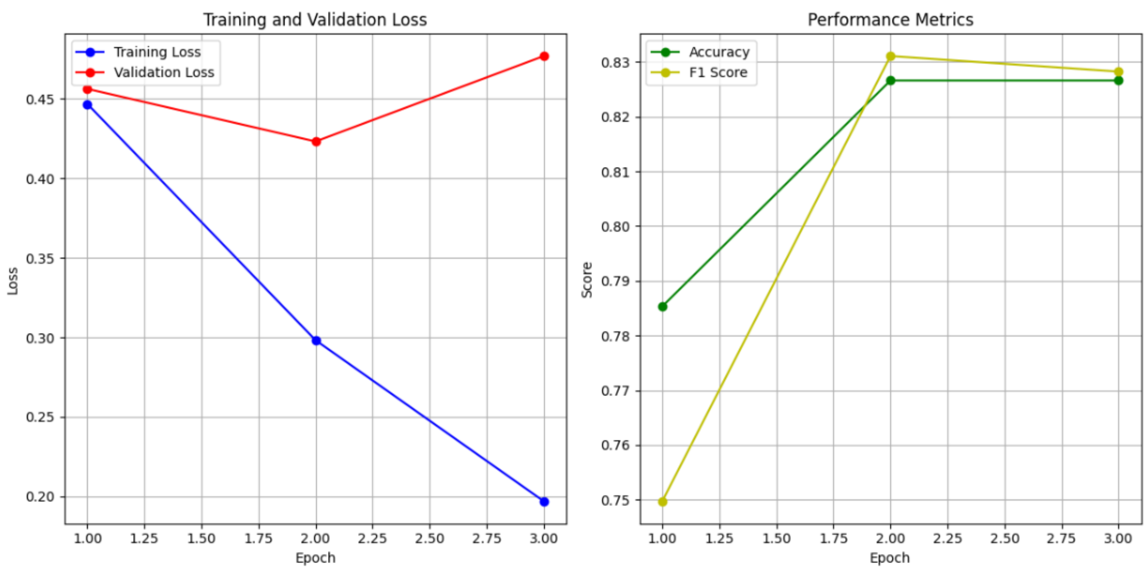

Figure 16: LR with max seq len 48 - BERT

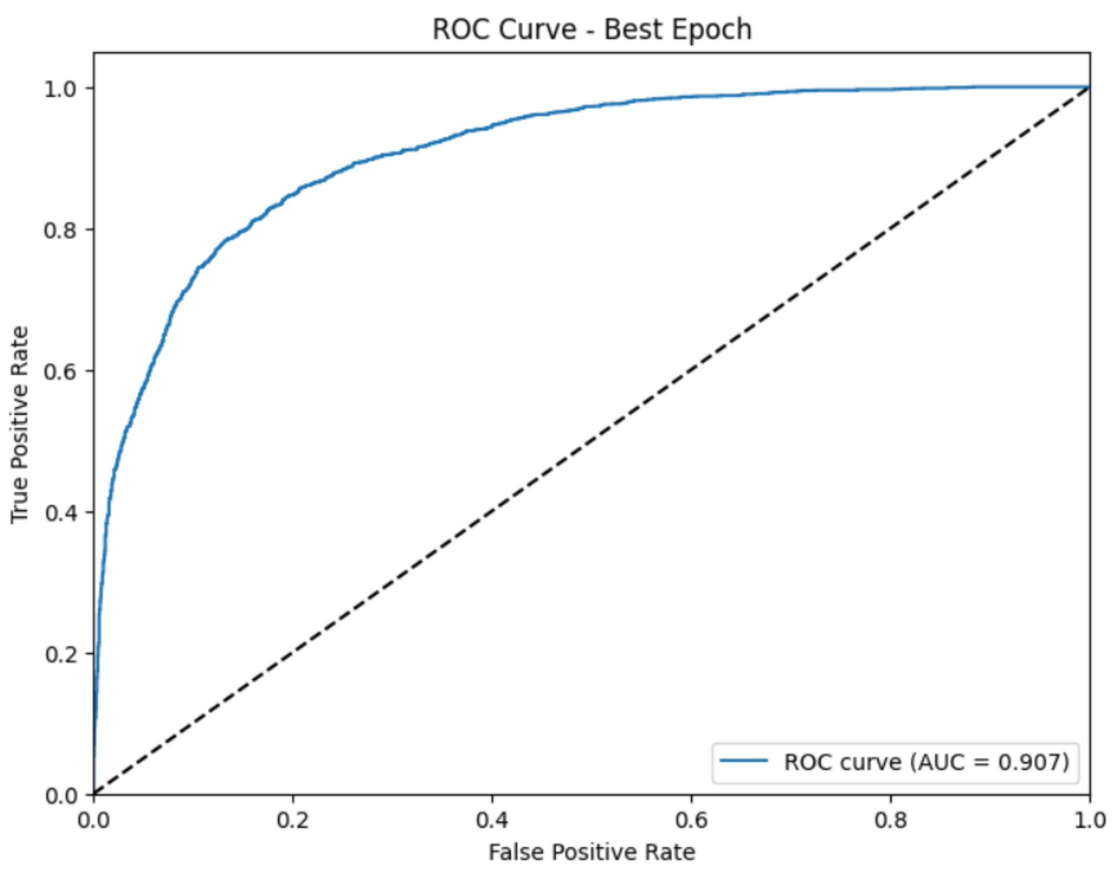

Figure 17: ROC Curve with max seq len 64 - BERT

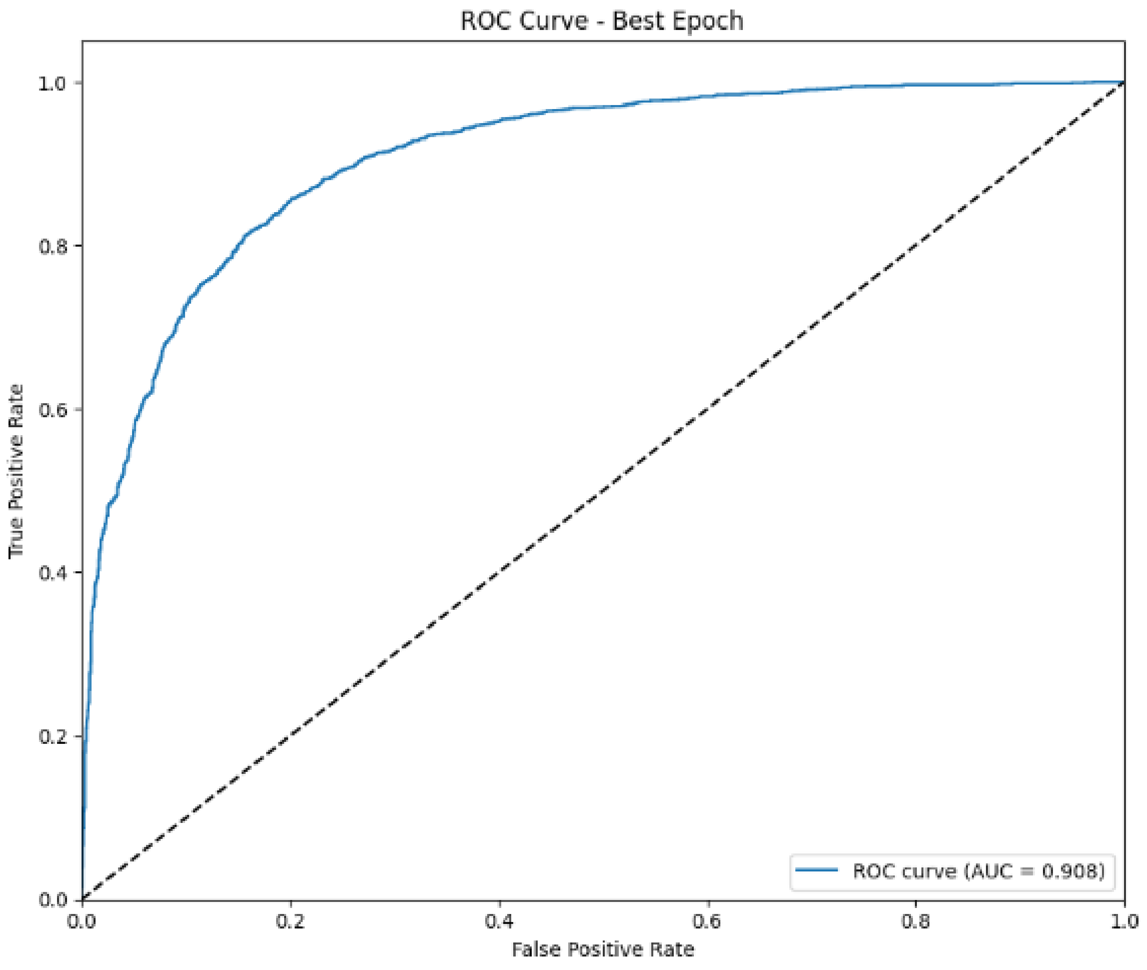

Figure 18: ROC curve with max seq len 48 (+0.001) - BERT

**Weight decay and dropout:** We tried adding weight decay (0.01) and increasing dropout (0.3). Both made the model generalize a bit better (higher recall), but overall F1 dropped. Too much regularization can actually hurt, especially with a strong model like BERT and a balanced dataset. Below, the AUC score is lower than the above results.

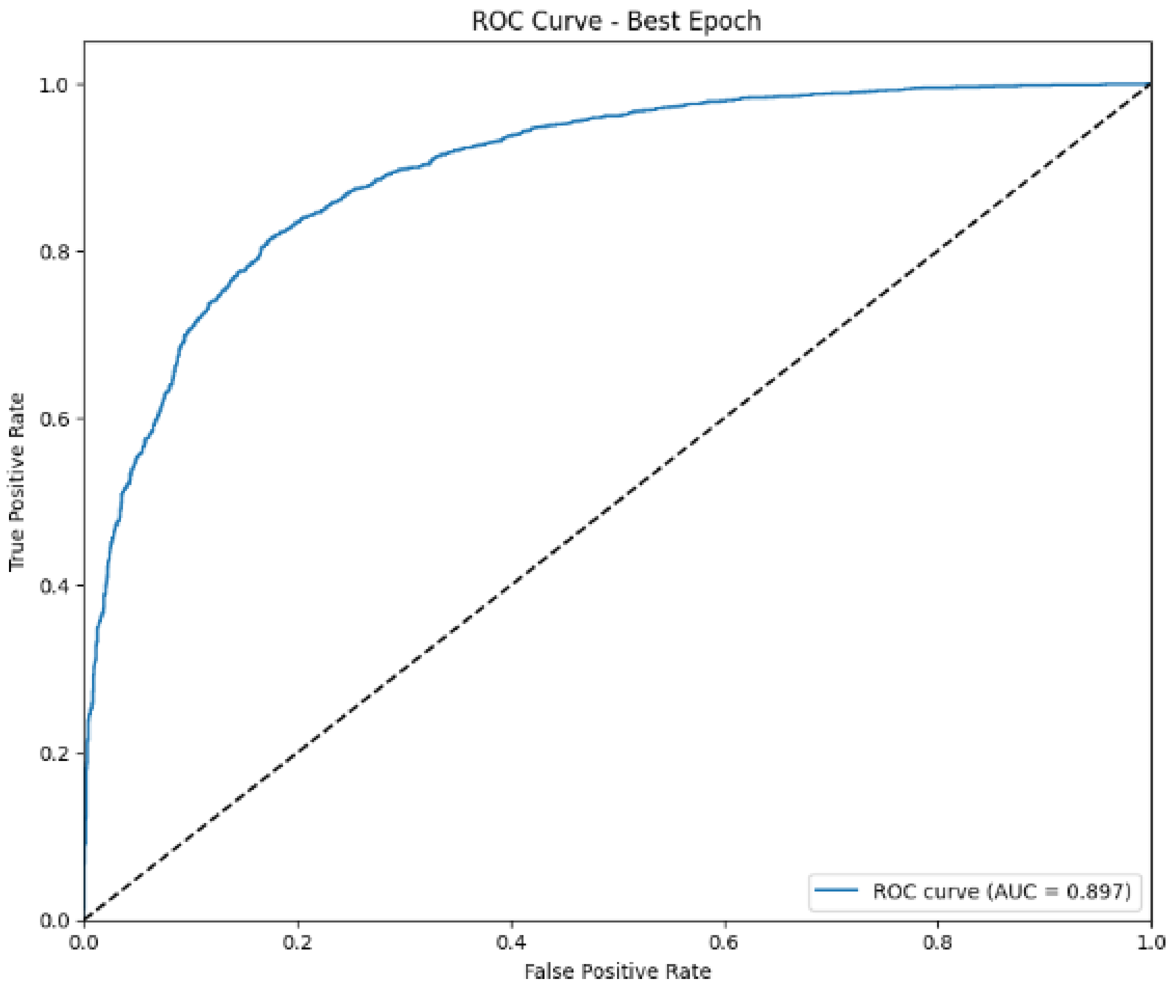

Figure 19: ROC with dropout 0.03 - BERT

**Full dataset training:** Once we had the best settings, we trained on the full dataset. F1 jumped to 0.8524 (with 64 tokens) and 0.8512 (with 48 tokens). The difference was tiny, but 48 tokens was much faster (about 30% less training time), so we chose the 2nd option.

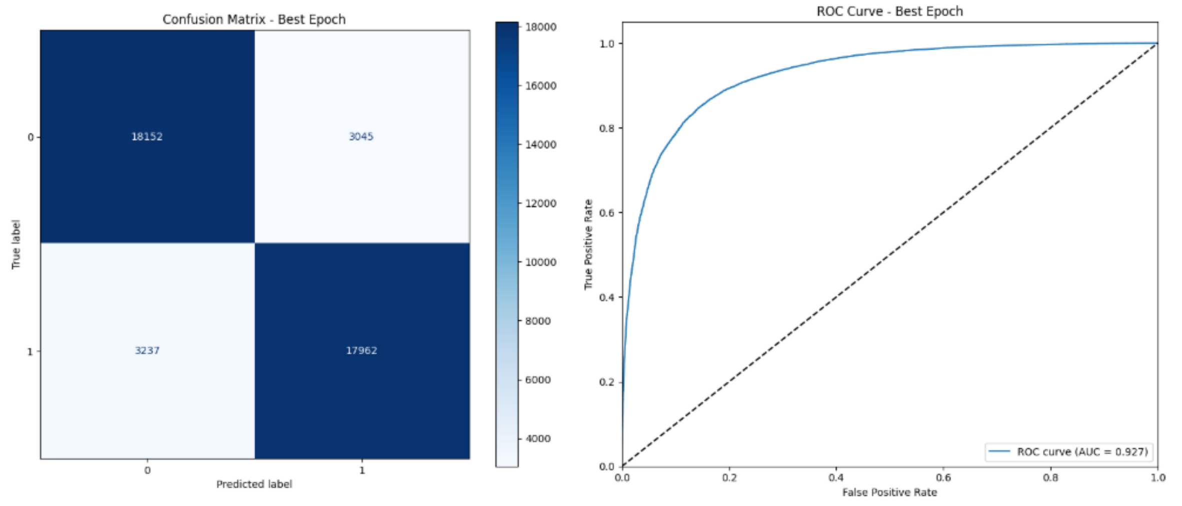

Figure 20: Seq Max Len = 48 - BERT

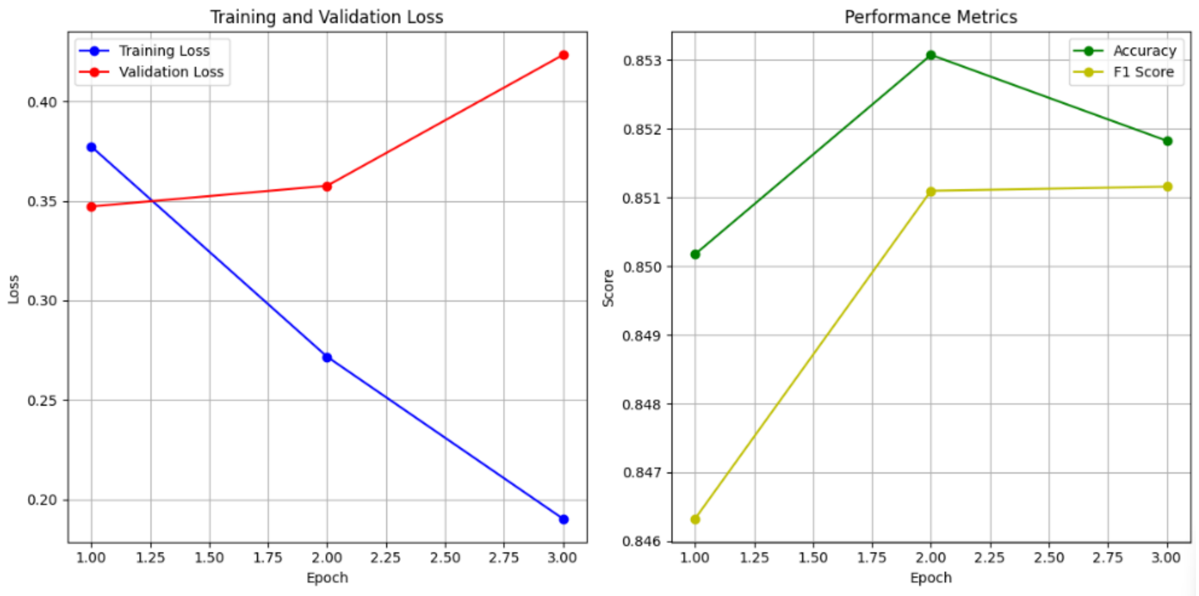

Figure 21: Seq Max Len = 48 - BERT

**Optuna hyperparameter search:** To make sure we weren't missing anything, we used Optuna to search for the best learning rate, optimizer, and weight decay.

```
1. HYPERPARAMETER OPTIMIZATION WITH OPTUNA:
   Based on our results, we'll focus on a targeted
   optimization with Optuna on a 10\% sample.

a) Learning Rate: [1e-5, 3e-5]
   * Narrower range around our current best (2e-5)
   * Fine-tuning this parameter has consistently shown impact

b) Optimizer Choice: [Adam, AdamW]
   * Compare standard Adam against AdamW

c) Weight Decay: [0, 0.01, 0.03]
   * Regularizing to remove signs of overfitting in the last epochs
```

The best config was Adam, learning rate 1.52e-5, and no weight decay. This got us an F1 of 0.8521, basically matching our best manual results but with even better efficiency. The top 5 Optuna trials were all very close (within 0.0015 F1 - see history below), showing our manual tuning was already pretty good.

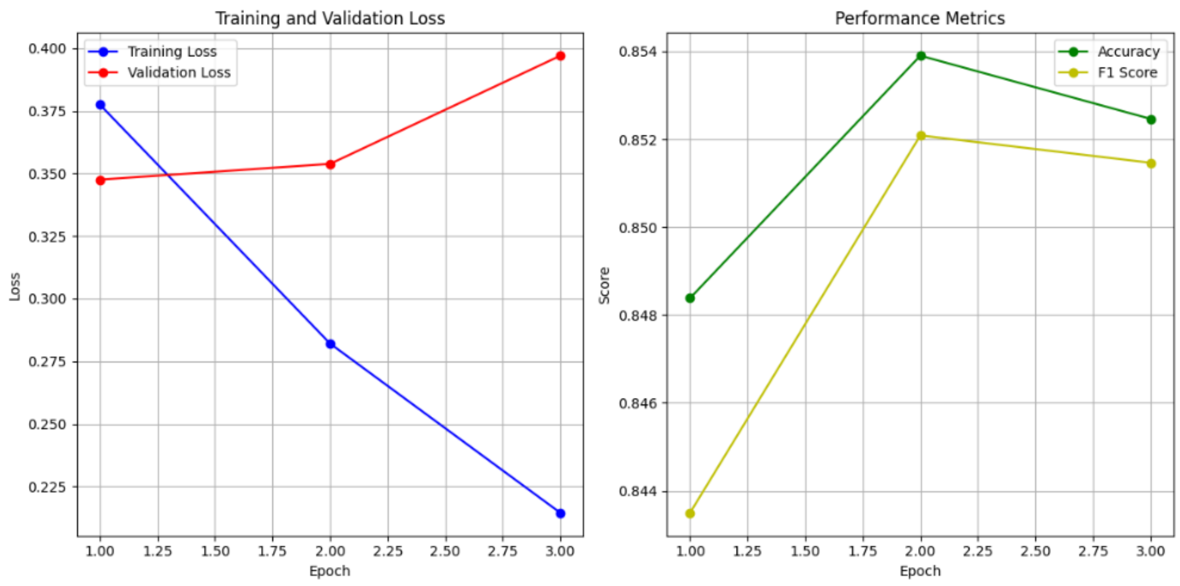

Figure 22: Optuna results - BERT

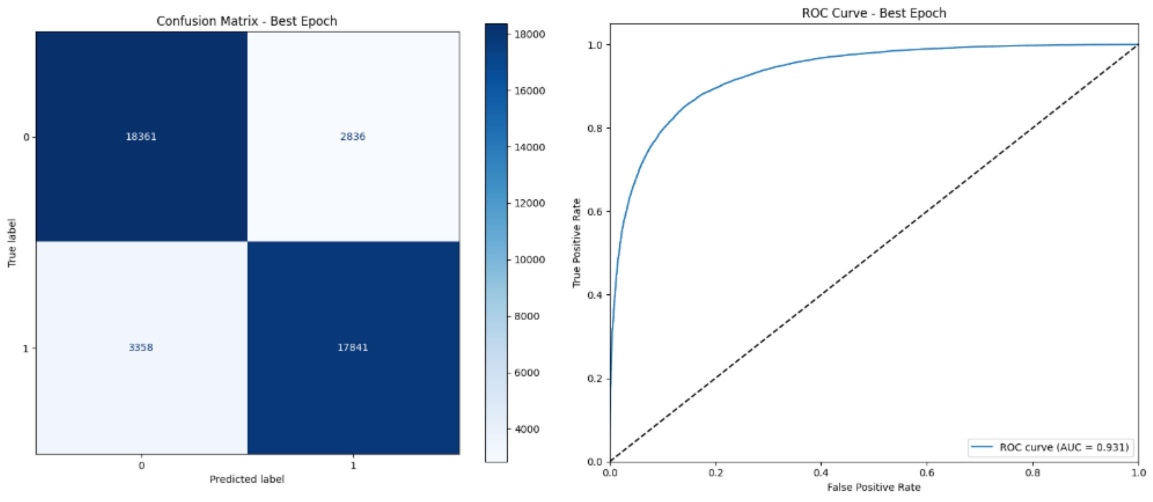

Figure 23: Optuna results - BERT

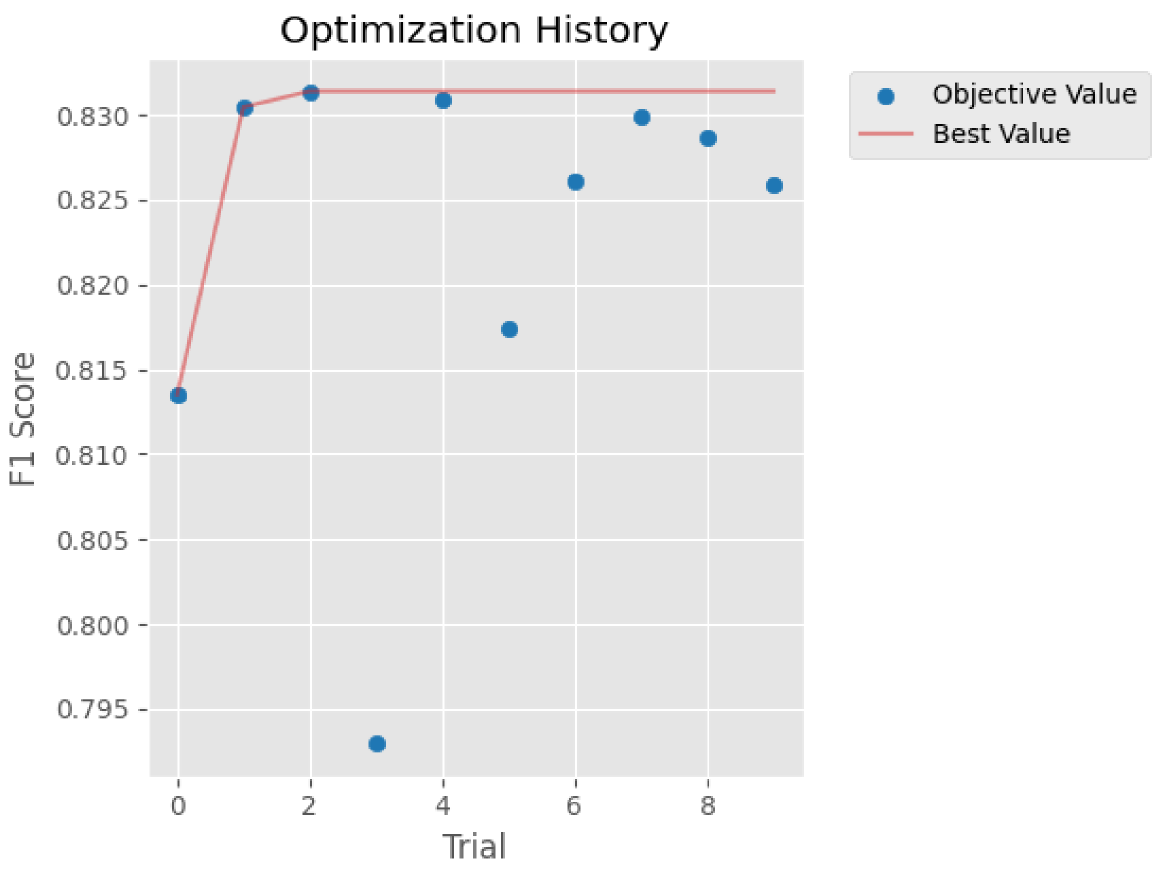

Figure 24: Optuna results History - BERT

**Final model:** We trained the final model for 2 epochs, since f1 peaked at epoch 2 and it was clear that after epoch 2 the model was overfitting in the training data (the validation loss increased and the training loss decreased rapidly). More about final model at 3.7.

***3.3.2. DistilBERT.*** We already learned many things from BERT, so our DistilBERT experiments were fewer and more focused. Here's how they went:

**Baseline:** We started DistilBERT with default dropout (0.1), batch size 32, sequence length 64, learning rate 2e-5, and got F1=0.8187. The overfitting in training data was pretty clear from the learning curves plot. The generalization gap was a bit smaller than BERT and this is because BERT was capable to understand better the training data (and eventually tend to overfit earlier).

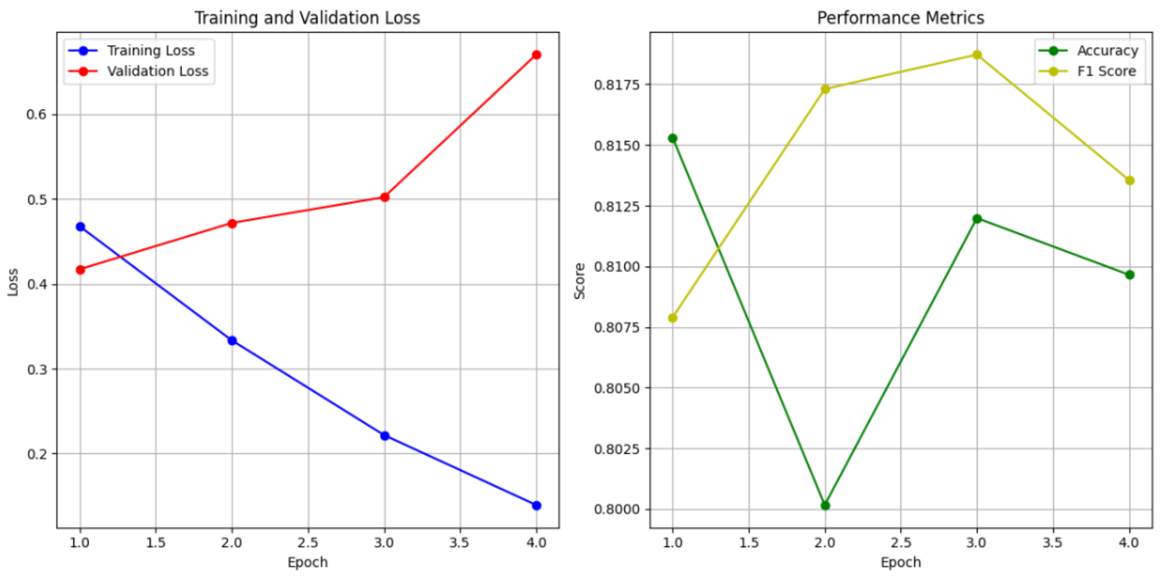

Figure 25: Baseline - DistilBERT

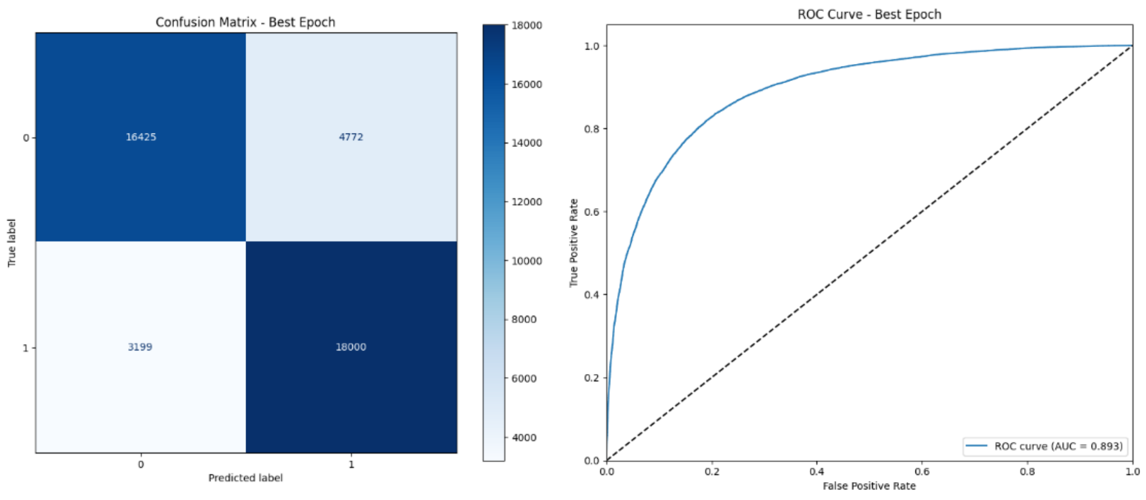

Figure 26: Baseline - DistilBERT

**Sequence length 48:** Reducing sequence length to 48 made training significantly faster with almost no loss in F1 (0.8156). Clearly, shorter tweets need fewer tokens. Early stopping was triggered this time in epoch 2 since with seq-len dropped to 48, each batch has less data to process and the model may learn the main patterns much faster.There's simply less context it can overfit on. In other words, we have faster convergence because each forward/backward pass is cheaper and the model's parameters settle into a good configuration more quickly and earlier plateau because the model exhausts its ability to improve on the validation set sooner. After epoch 1 it's already picked up the strongest signals, and by epoch 2 any further tweaks just overfit the limited context (same pattern can be seen in BERT as well after this change).

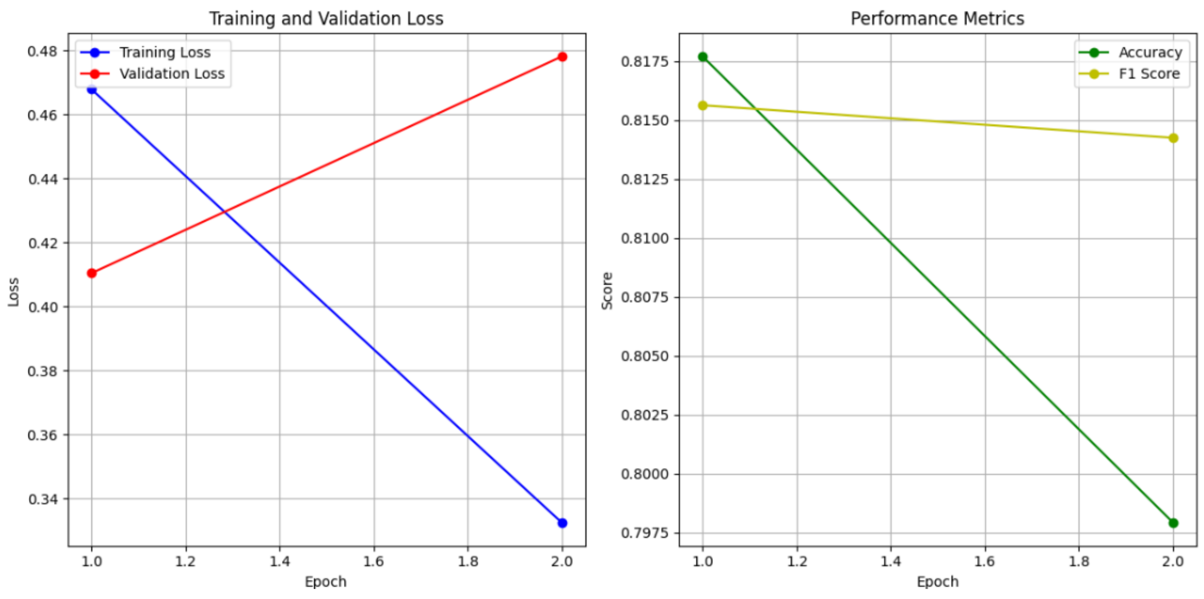

Figure 27: Seq Len to 48 - DistilBERT

**Learning rate adjustments:** With the lower LR of 1e-5, training loss drops smoothly each epoch, but validation loss only peaks at epoch 2 and then comes back down at epoch 3, so the model isn't overfitting as fast. F1 rises from 0.8102 to 0.8176 in epoch 2 and then holds steady, while accuracy dips slightly at epoch 2 before recovering. In short, the smaller learning rate gives more stable learning and a smaller generalization gap, suggesting we could even explore a few more epochs without immediately overfitting.

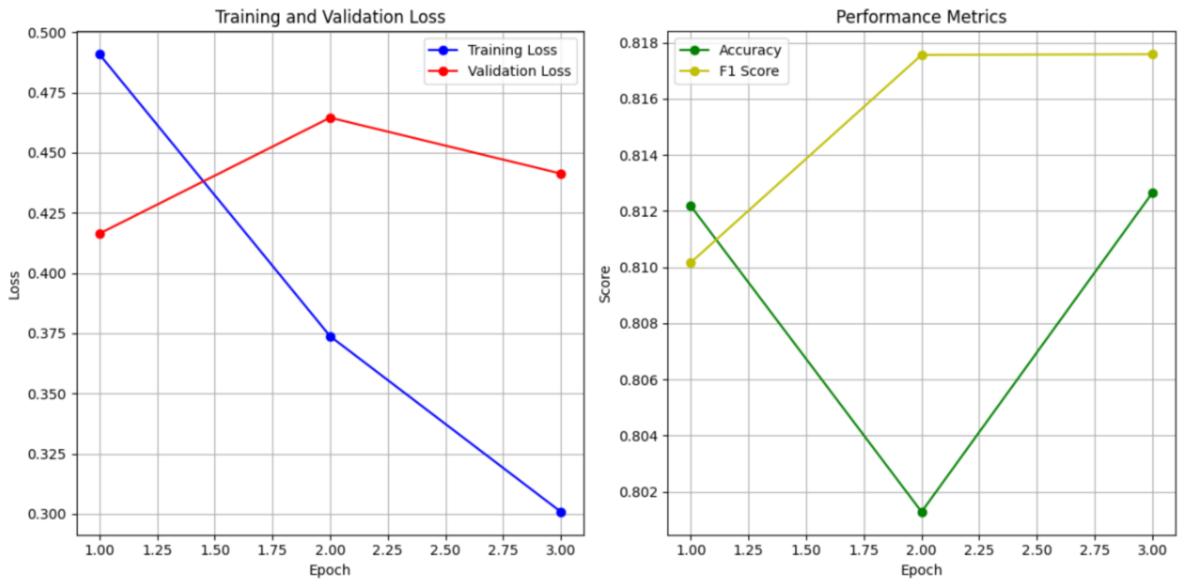

Figure 28: Reduced Learning Rate - DistilBERT

**Batch size tests:** Increasing batch size to 64 reduced performance (F1=0.8110), matching BERT's results—smaller batches generalize better.

**Scheduler and regularization:** Adding a learning rate scheduler slightly improved the F1 score (0.8195). Optuna then found the best dropout rate (0.2), giving us our best trial yet (F1=0.8246).

**Manual Final Model:** From experimenting and manual hyperparameter tuning, the results we can be seen below.

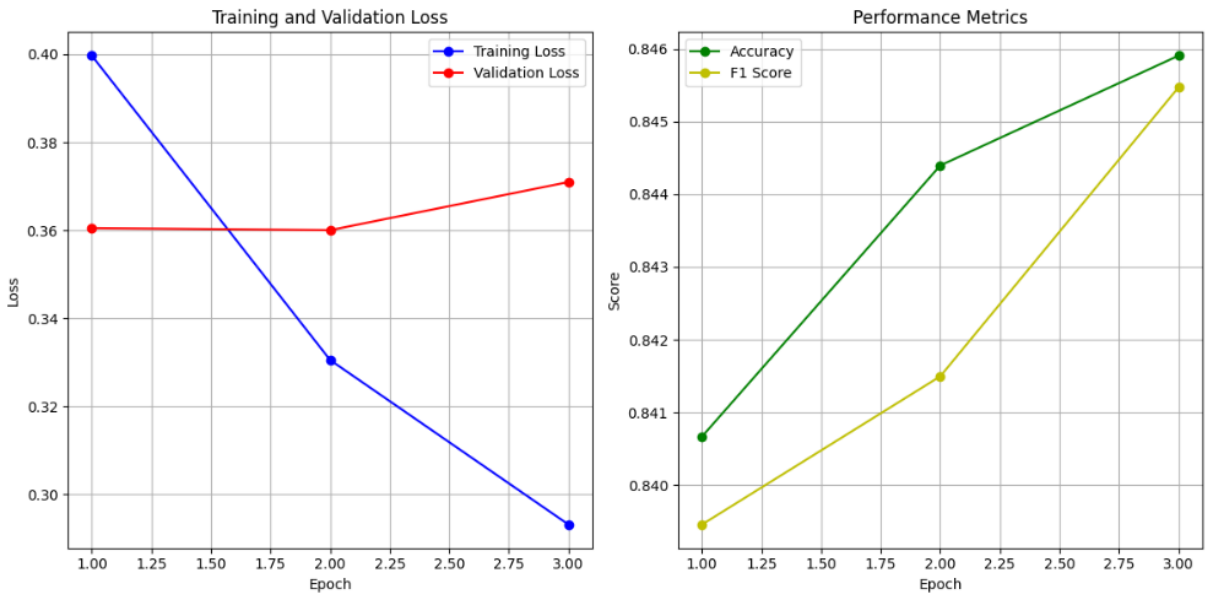

Figure 29: Manual Final - DistilBERT

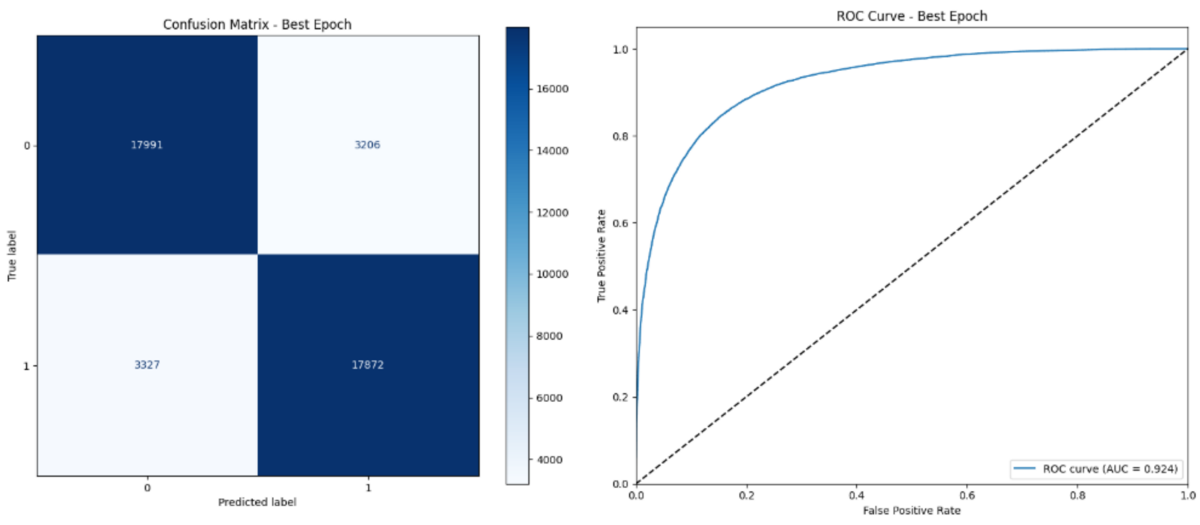

Figure 30: Manual Final - DistilBERT

**Final model - Optuna search:** With the optimal configuration from Optuna (sequence length 48, learning rate 1.94e-5, dropout 0.2, batch 32), we got our final model, more about it in 3.7. We saw that we were in the right direction while manual hyperparameter tuning, since the optuna suggestion was pretty similar with our model. The only difference was that Optuna gave a higher learing rate, a more aggresive approach we could say, so as to maximize the f1-score safely, without huge overfit risk and big generalization gap, since dropout was increased to 0.2, as well. In DisilBERT, we increased the number of trials to 15, since the training was faster, again aiming to maximize f1-score. This approach is explained here 3.5.

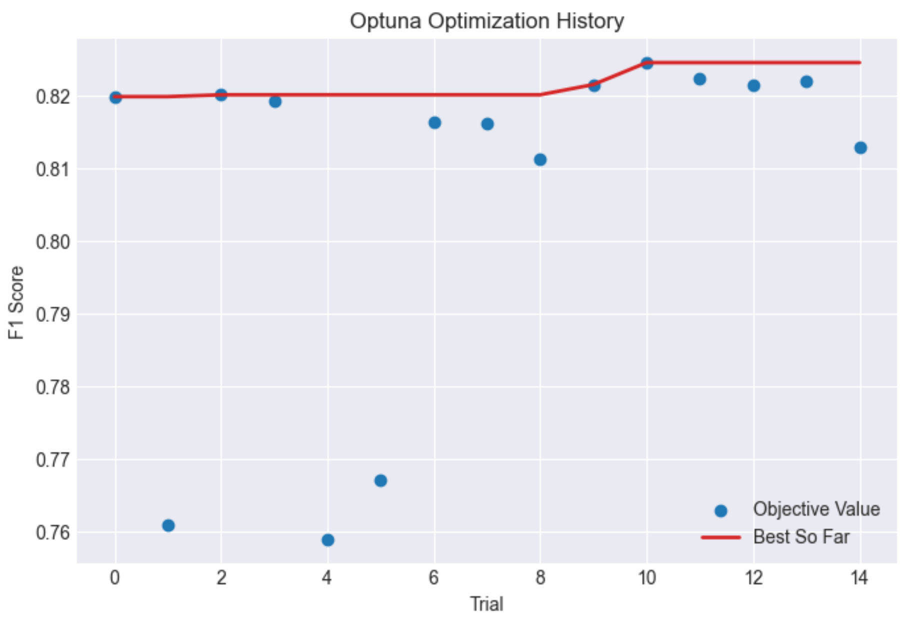

Figure 31: Manual Final - DistilBERT

## 3.4. Fine Tuning and Optimization Techniques

Here is a short summary of what each hyperparameter did and the conclusions we drew from both BERT and DistilBERT experiments:

- **Learning Rate:** Controls how big each update step is.
  - Too high (e.g. 5e-5) $\rightarrow$ very fast over-fitting.
  - Lower learning rates were better. Models were able to perform with any normal learning rate value, lower ones were used to prevent fast overfit and not blowing generalization gap.
- **Batch Size:** How many samples per gradient step.
  - Small batches (32) $\rightarrow$ better generalization, higher F1.
  - Bigger batches (64) $\rightarrow$ smoother loss but often lower F1.
  - Even bigger batches (128+) $\rightarrow$ worse metrics.
- **Sequence Length:** Number of tokens per tweet.
  - 128 was overkill (lots of padding).
  - 64 gave same F1, 48 cut training time by 30% and kept performance.
- **Dropout:** Prevents co-adaptation of neurons.
  - Default 0.1 good for BERT.
  - DistilBERT needed 0.2 to safely push learning rate up without over-fitting.
- **Weight Decay:** Penalizes large weights for regularization.
  - 0.0 was best in almost all runs—no decay needed on this balanced dataset.
- **Early Stopping:** Stop training when F1 stops improving.
  - Patience=2 for BERT (peaked at epoch 2).
  - Patience=1 for DistilBERT kept it from over-fitting too long.
- **Scheduler:** Linear warmup helps avoid big jumps early on.
  - Gave a small F1 lift by smoothing the learning rate decay.
- **Optimizer:** Which algorithm updates model weights.
  - **BERT:** Adam (no weight decay) let us push learning without complicating the update rule.
  - **DistilBERT:** AdamW handled the small weight decay term cleanly and worked best with dropout=0.2.
  - In both cases, the two optimizers gave very close F1 scores, so either is fine if paired with the right decay and dropout.

### 3.5. Why F1 Score?

Even if our data have the same number of positive and negative tweets, we pick F1 score as our main metric because it looks at two things at once:

- **Precision**: how many of the tweets we say "positive" really are positive.
- **Recall**: how many of the real positive tweets we actually find.

Validation loss tells us if the model is confident or not in its probabilities, but it does not say if the label is right or wrong. In our runs, sometimes validation loss went up a little but F1 still went up. That means the model became more confident (maybe too confident), yet it was still predicting the right sentiment more often.

Because our goal is to get the correct sentiment label (positive or negative) and not to have perfect probability numbers, we focus on F1. We also watch validation loss as a warning sign of over-fitting, and we use early stopping based on F1 so we do not train too much.

**Difference between BERT and DistilBERT**

- With **BERT**, we stopped at **2 epochs** because we observed a large generalization gap—the training loss kept dropping while the validation loss rose sharply. BERT's size and capacity make it prone to over-fitting quickly.
- With **DistilBERT**, the generalization gap was much smaller, so we could safely train for **3 epochs**. DistilBERT is a lighter, faster version of BERT and did not over-fit as fast, allowing us to gain a little extra F1 in the third epoch.

### 3.6. What worked and what didn't

- **+ Reducing sequence length** was a huge win for speed, with almost no loss in accuracy.
- **+ Early stopping** and a good learning rate were key to avoiding over-fitting.
- **+ Optuna** helped confirm our manual tuning and made the process more systematic.
- **! Validation loss vs F1:** Don't panic if loss goes up (a bit!) but F1 improves, for this sentiment classification task, we prefer F1!
- **– Too much regularization** (dropout, weight decay) actually hurt performance.
- **– Large batch sizes** didn't work at all. 32 and 64 were the best values in all trials.

### 3.7. Best Model Summary

Let's summarize the final models for BERT and DistilBERT.

***3.7.1. BERT.*** Here's the final config for our best BERT model:

- **Model:** BERT-base-uncased
- **Sequence length:** 48
- **Batch size:** 32
- **Learning rate:** 1.52e-5
- **Optimizer:** Adam
- **Weight decay:** 0.0
- **Epochs:** 2
- **Early stopping:** patience=2 (but stopped at 2)
- **F1 score:** 0.8528
- **Accuracy:** 0.8539
- **Precision:** 0.8591
- **Recall:** 0.8465

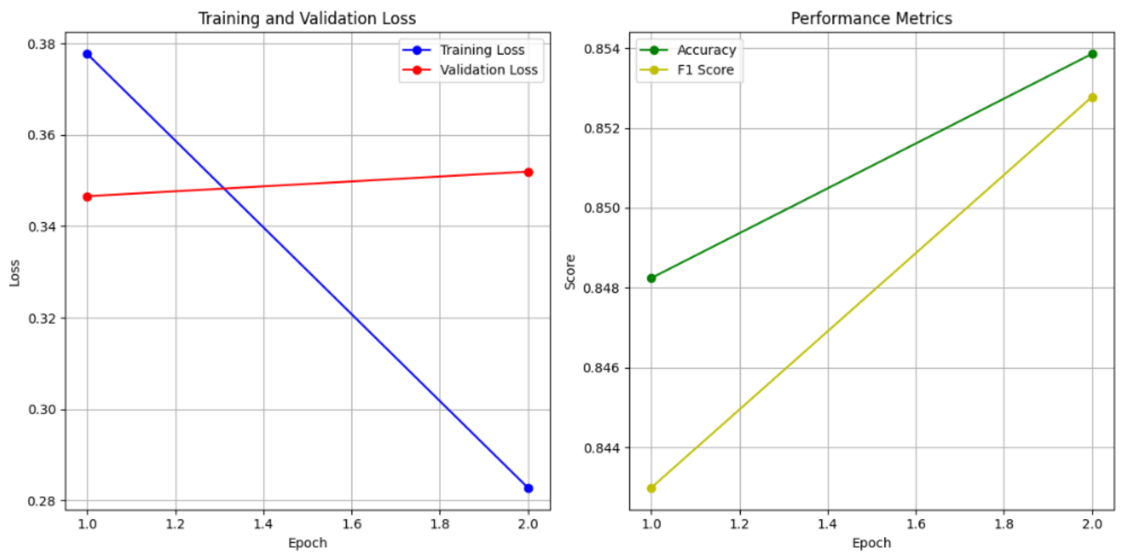

Figure 32: Final Model - BERT

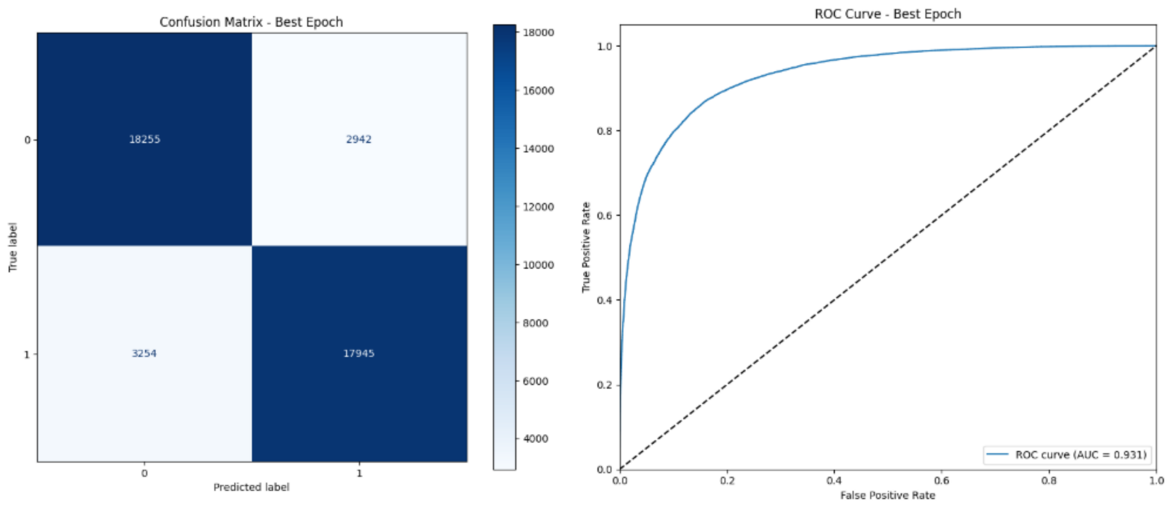

Figure 33: Final Model - BERT

### 3.7.2. *DistilBERT.*

Here's the final config for our best DistilBERT model:

- **Model:** DistilBERT-base-uncased
- **Sequence length:** 48
- **Batch size:** 32
- **Learning rate:** 1.94e-5
- **Optimizer:** AdamW
- **Weight decay:** 0.0
- **Epochs:** 3
- **Dropout rate:** 0.2
- **Early stopping:** patience=2 (no stop, ran all 3)
- **F1 score:** 0.8464
- **Accuracy:** 0.8478
- **Precision:** 0.8543
- **Recall:** 0.8387

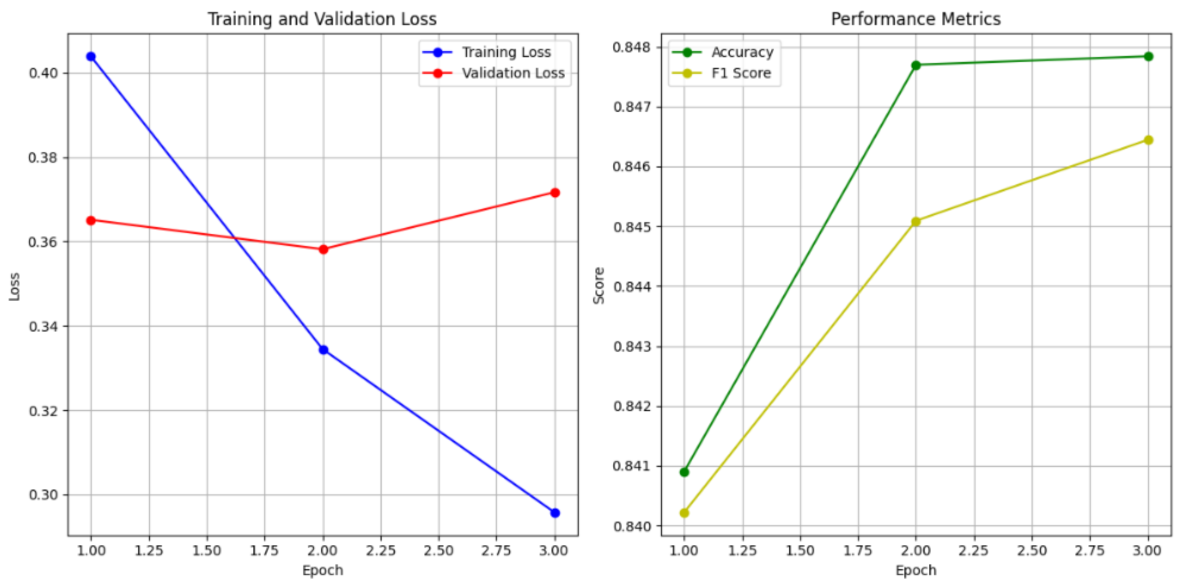

Figure 34: Final Model - DistilBERT

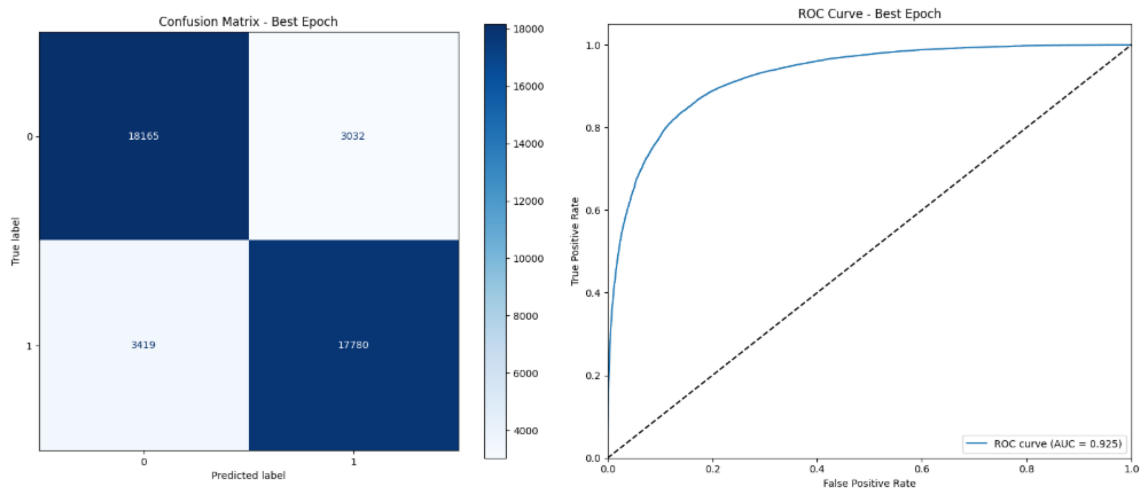

Figure 35: Final Model - DistilBERT

## 4. Overall Analysis

### 4.1. Comparison with the first project

In the first project, we just used TF–IDF vectors and a Logistic Regression model, and that got me about **0.804** accuracy on validation. Switching to BERT was like night and day: we hit F1 = 0.8528 and accuracy = 0.8539. It's a huge jump because BERT actually "gets" word order and picks up on little sentiment signals that TF–IDF can't see. Plus, BERT handled all the noisy, weird tweets way better—its pretraining on tons of text really pays off.

## 4.2. Comparison with the second project

In project two, we moved up to Google Word2Vec embeddings with a small DNN and got around 0.78 for both F1 and accuracy. That was better than TF–IDF but still left a gap. BERT crushed that too, thanks to its deep contextual layers and transfer learning. You can clearly see the difference in both the numbers and the confusion matrix/ROC curves. BERT and DistilBERT are just way more capable on this kind of Twitter data.

## 4.3. Final thoughts

This project taught me a lot about fine-tuning big models, watching the generalization gap, and picking the right metric. DistilBERT almost matched full BERT's performance but trained faster and overfit less. In the end, good preprocessing, early stopping, and a focused Optuna search were the real MVPs for squeezing out the best results. In conclusion, the models were more than enough to handle these datasets, and the metrics were improved significantly compared to previous projects, even from the baseline models with 10% training dataset.

**The whole project in one picture:**

Figure 36: Best part of the report

## 5. Bibliography

### References

[1] Manolis Koumparakis. Artificial intelligence ii, deep learning for natural language processing, advanced artificial intelligence. https://cgi.di.uoa.gr/~ys19/#homework, 2025.

[2] Despina-Athanasia Pantazi. Tutorials for homework 3. 2025.

[1] [2]