SMS Detection using NLP Techniques

- BENGHENIMA Hafsa
- GHANDOUZ Amina

Agenda

Introduction

Every day, millions of SMS messages are sent — many of them are spam, fraudulent, or malicious.

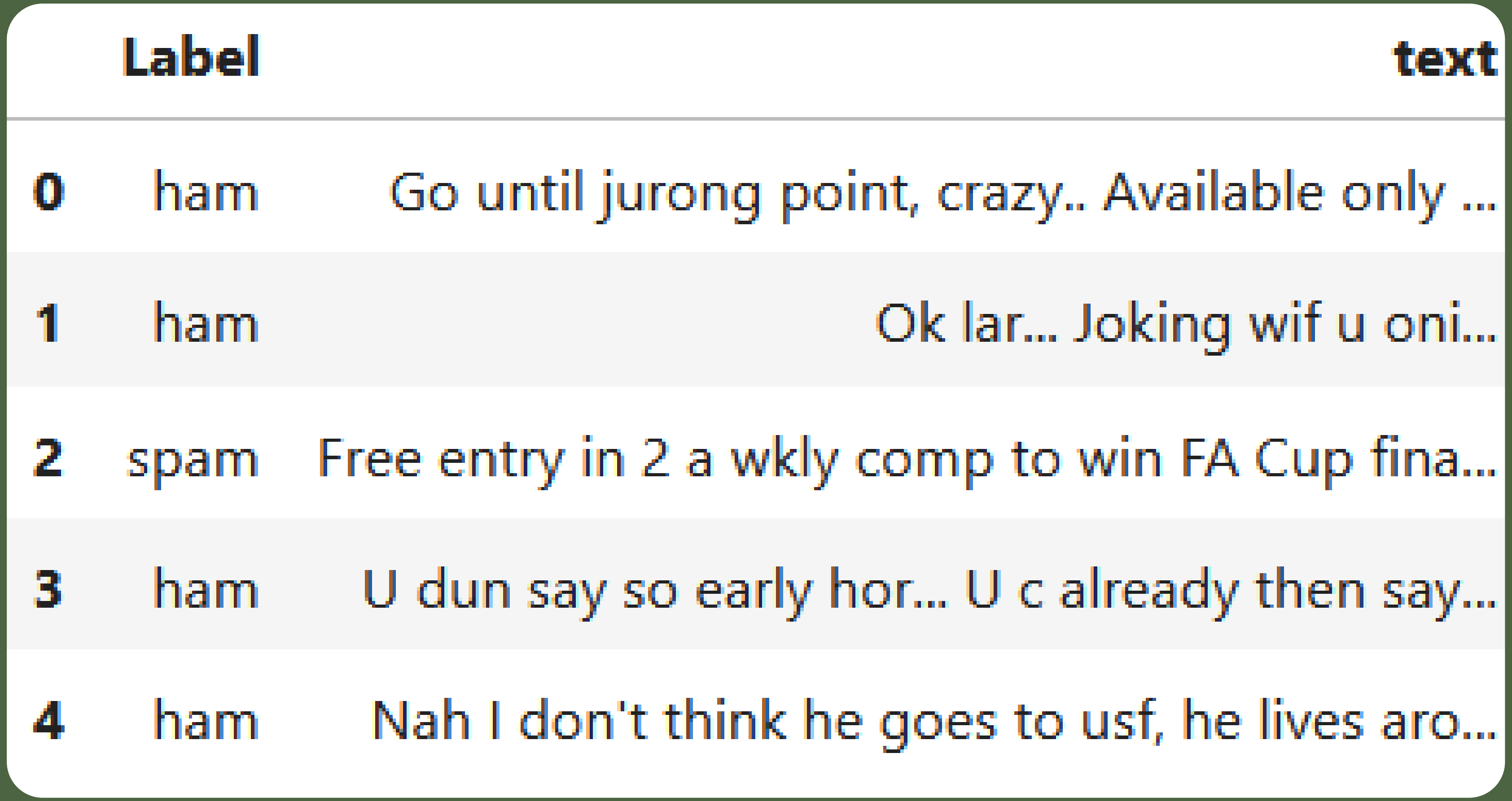
- These messages try to scam users, spread malware, or waste user time.
- Traditional keyword-based filters are ineffective, easily bypassed, and not adaptive.

We propose an NLP-based SMS spam detection system that:

- Uses Natural Language Processing (NLP) to understand message content.
- Applies machine learning and deep learning models to classify messages as spam or ham (not spam).

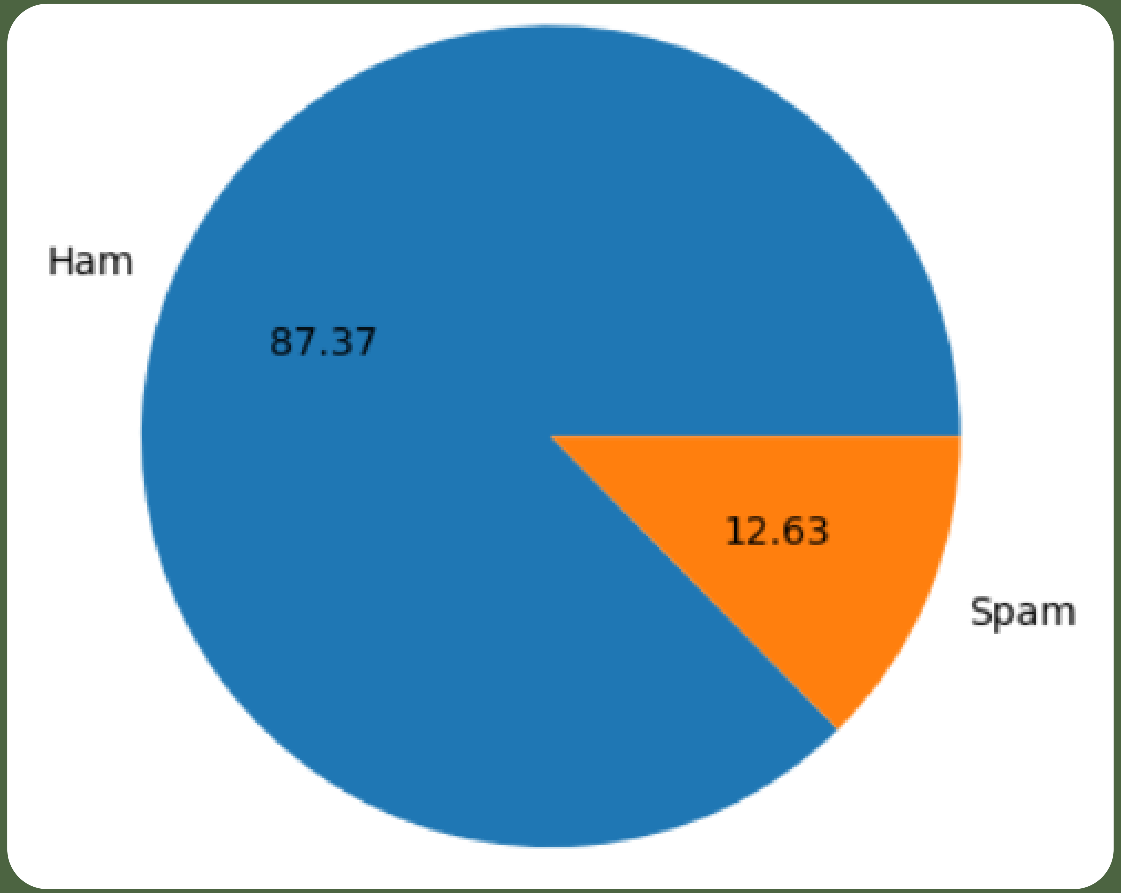
Dataset Overview

- Source: Kaggle
- Number of raws: 5,574 messages
- Classes: Spam / Ham



	Label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

- Class Distribution



Baseline Approach

1. Data Preprocessing Pipeline

1.1 Train-Test Split

- Split dataset: 80% training / 20% testing
- Used stratified splitting to maintain class balance

1.2. Text Normalization

- Lowercasing
- Removing special characters

1.3. Text Vectorization

- Transformed text using TF-IDF
- Applied only to training data to avoid data leakage and ensure test set remains unseen and evaluation is fair

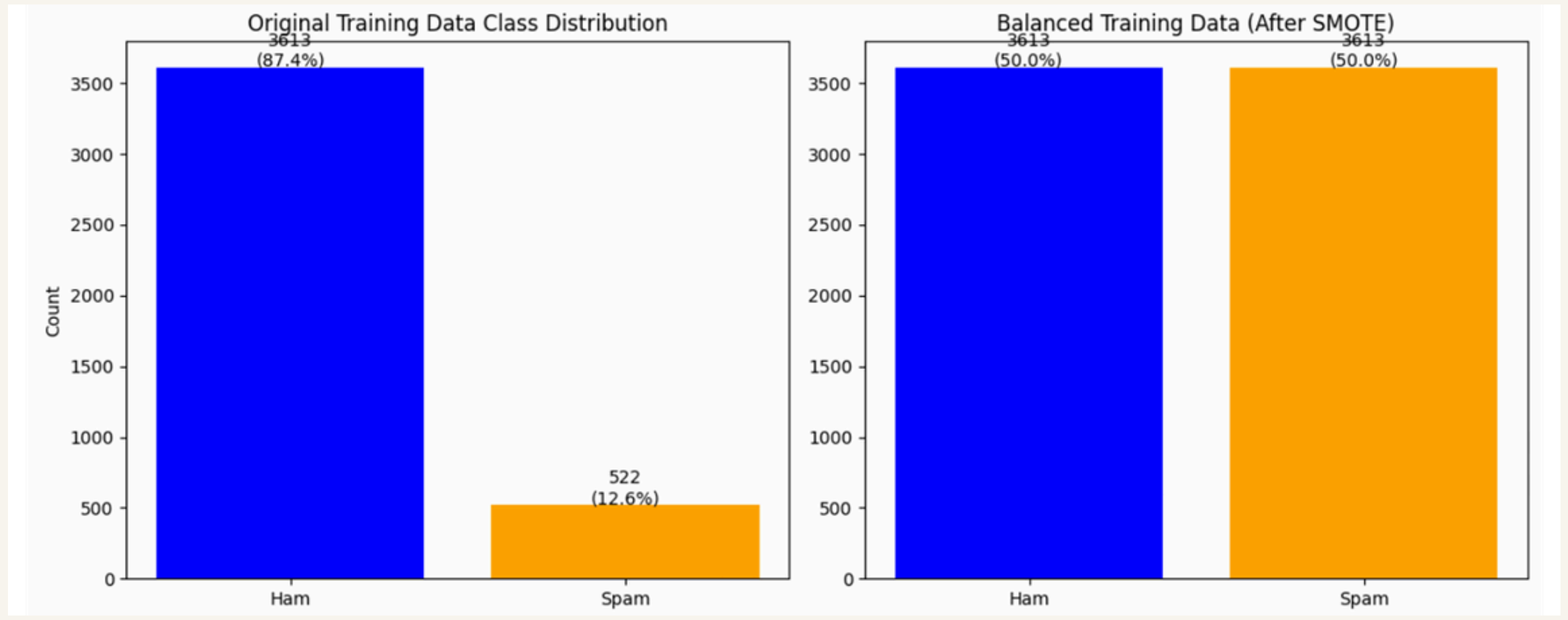
~~What is data leakage?~~

- TF: how often the word appears in a document.
- IDF: how rare the word is across all documents.

If we fit TF-IDF on the entire dataset (training + test), then the IDF part includes information from the test data, which is supposed to be unseen during training.
This creates a data leakage problem — the model indirectly "sees" part of the test set, leading to unrealistically high performance during evaluation.

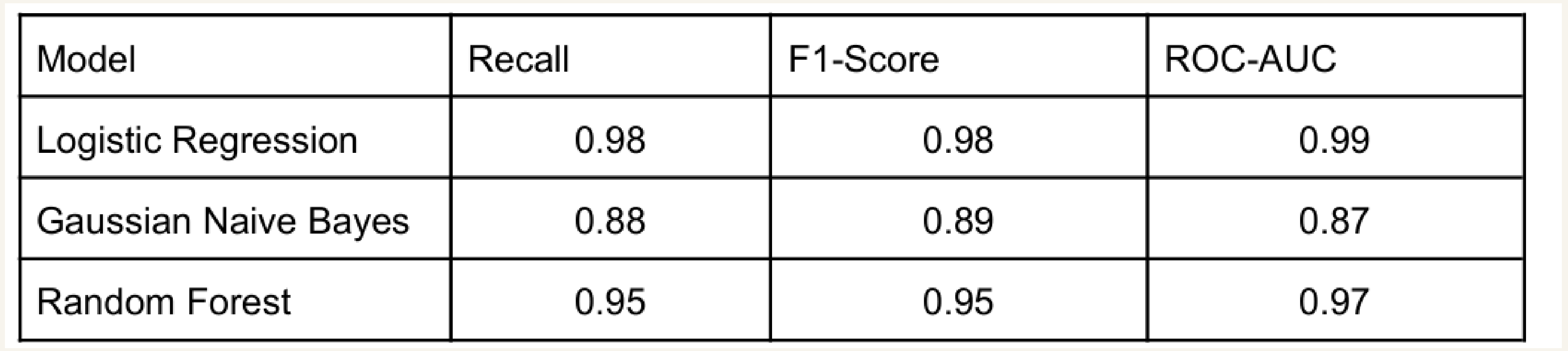
1.4 Addressing Class Imbalance

- Applied SMOTE after tf-idf to oversample minority class in training set



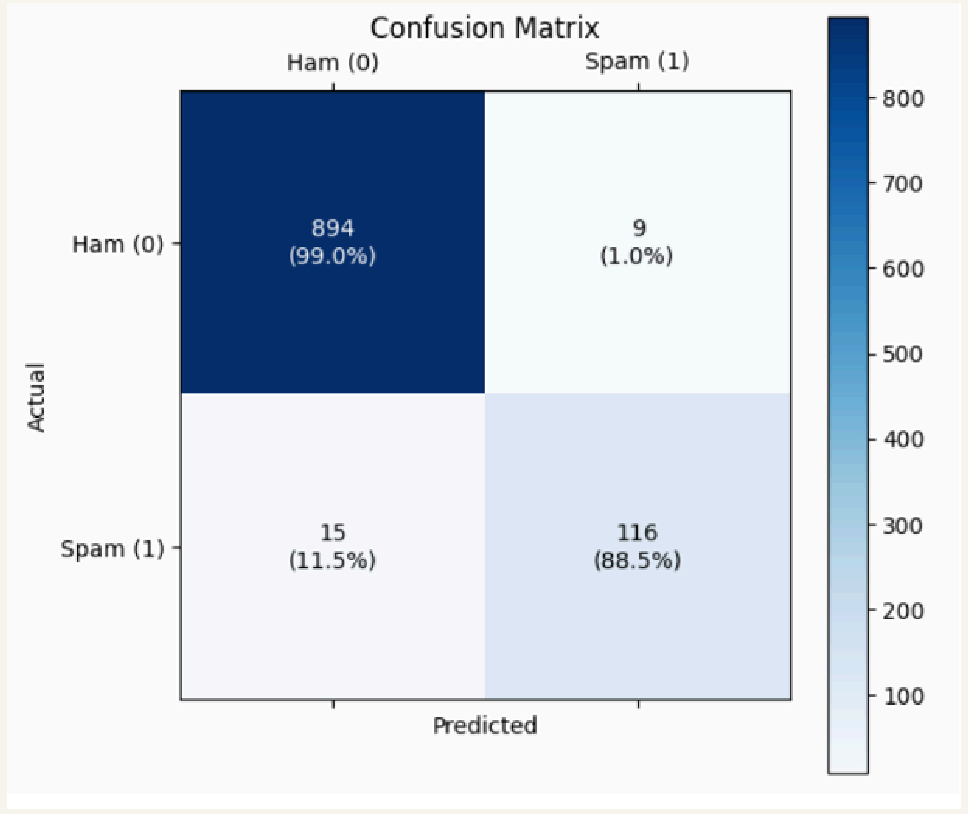
2. Model Training Process

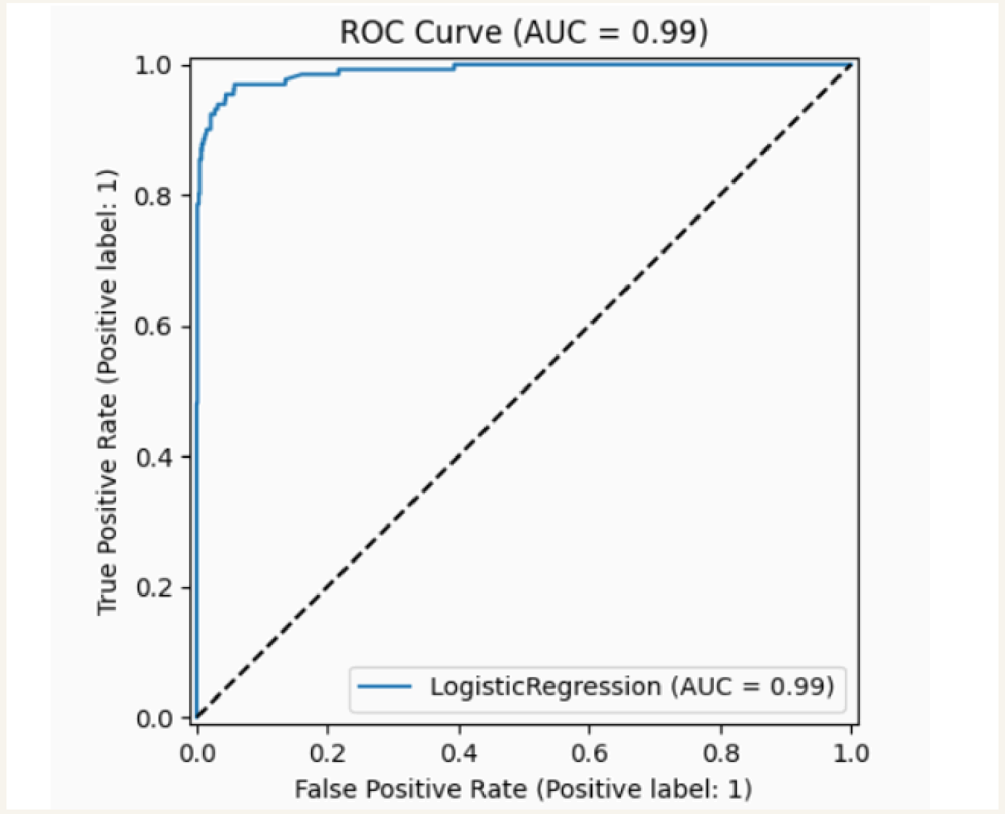
- Logistic Regression (LR)
- Gaussian Naive Bayes (GNB)
- Random Forest (RF)



Model	Recall	F1-Score	ROC-AUC
Logistic Regression	0.98	0.98	0.99
Gaussian Naive Bayes	0.88	0.89	0.87
Random Forest	0.95	0.95	0.97

After evaluating all models using comprehensive classification metrics — including accuracy, precision, recall, F1-score, confusion matrix, and ROC curves — we observed that Logistic Regression consistently outperformed the others.





Transformer-based Approach

Why Transformers for SMS Spam Detection?

- Traditional models rely on handcrafted features (like TF-IDF).
- Transformers (like BERT) understand context, semantics, and word relationships.

Transformer-based Approach

Model Used & Setup

- Model: BERT
- Library/Framework: Hugging Face Transformers and PyTorch

<u>BERT Input</u>

- Input: A sentence is tokenized using WordPiece tokenizer
- Special tokens added:
- [CLS] = Classification token added at the start of every input.
- [SEP] = Separator token added at the end (or between sentences if there are two).

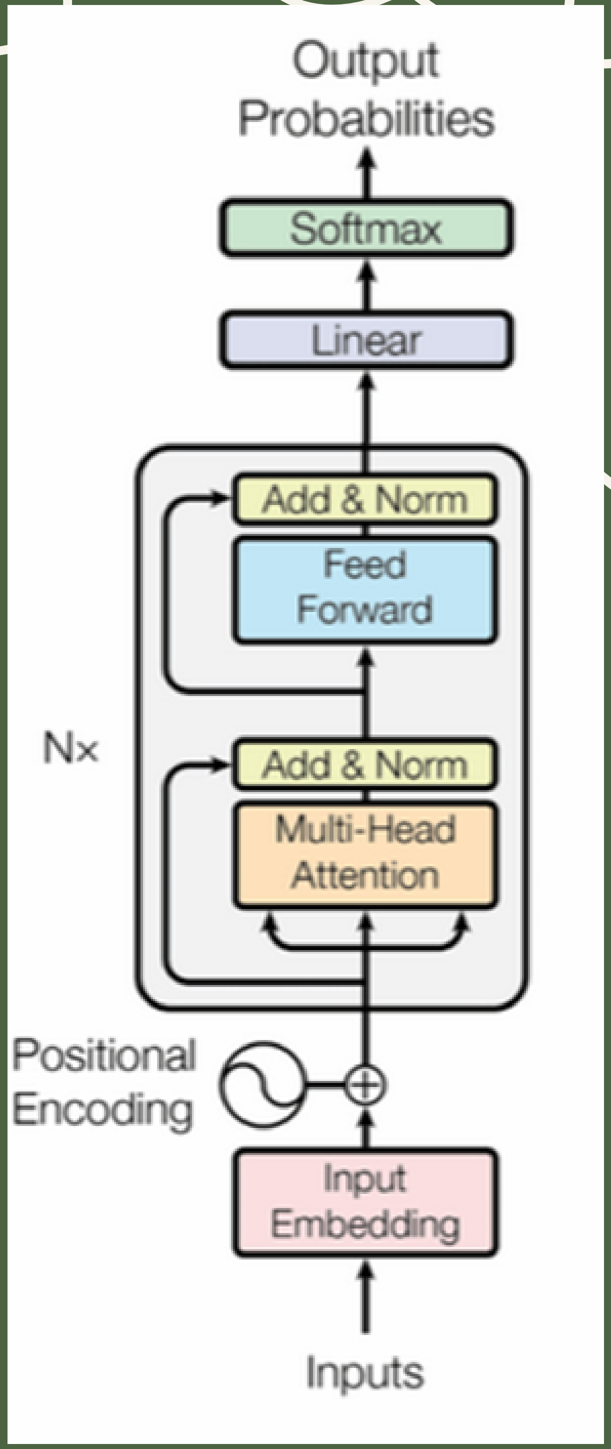
~~BERT Architecture?~~

we used BERT Base, that have:

- 12 transformer encoder layers and self-attention heads
- Add a final classification layer with 2 neurons, corresponding to the Ham and Spam classes.
- 768-dimensional hidden representations
- A total of around 110 million trainable parameters

~~How BERT Works?~~

- Each token is converted into an embedding vector.
- The whole sequence passes through multiple Transformer layers.
- The output embedding corresponding to [CCS] is used by a classifier to predict: Spam or Ham



Data preprocessing

- Used a cleaned SMS dataset (no missing values or duplicates)
- Split into 80% training / 20% test
- Applied stratified sampling to preserve spam/ham ratio across both sets

Tokenization and Input Formatting

- Used bert-base-uncased tokenizer (based on WordPiece)
- Automatically lowercases text
- Converts SMS into token IDs
- Padded/truncated to 128 tokens
- Returns:
- input_ids (tokens)
- attention_mask (1 = real token, 0 = padding)

Addressing Class Imbalance

- Computed class weights based on inverse class frequency

  weight_class = total_samples / (num_classes × samples_in_class)

- Integrated weights into the cross-entropy loss
- Spam (minority class) received higher weight
- This helps model better detect spam by penalizing misclassifications more heavily

Model Configuration and Training

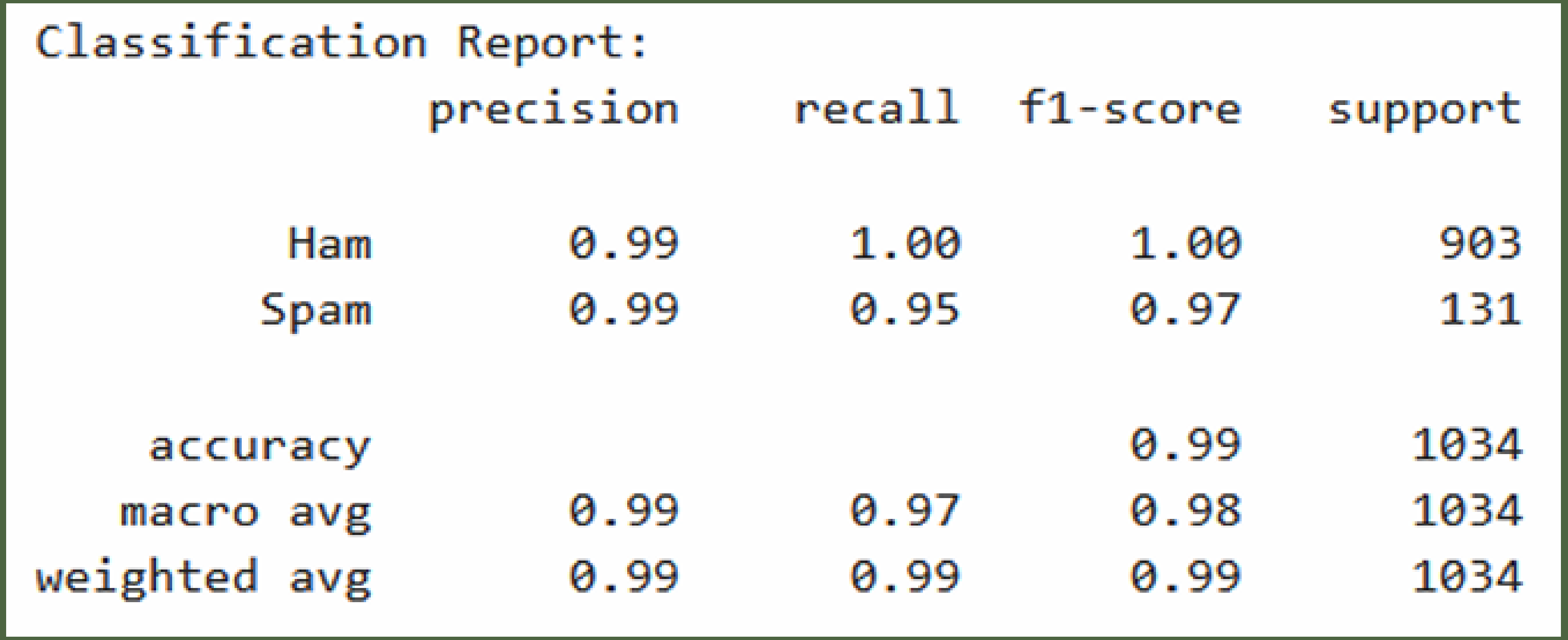
- Loss function: CrossEntropyLoss with class weights
- Optimizer: Adam with learning rate 2e-5
- Epochs: 3
- Batch size: 16

Evaluation

Accuracy: 0.9923

Coss : 0.0235

Classification report:

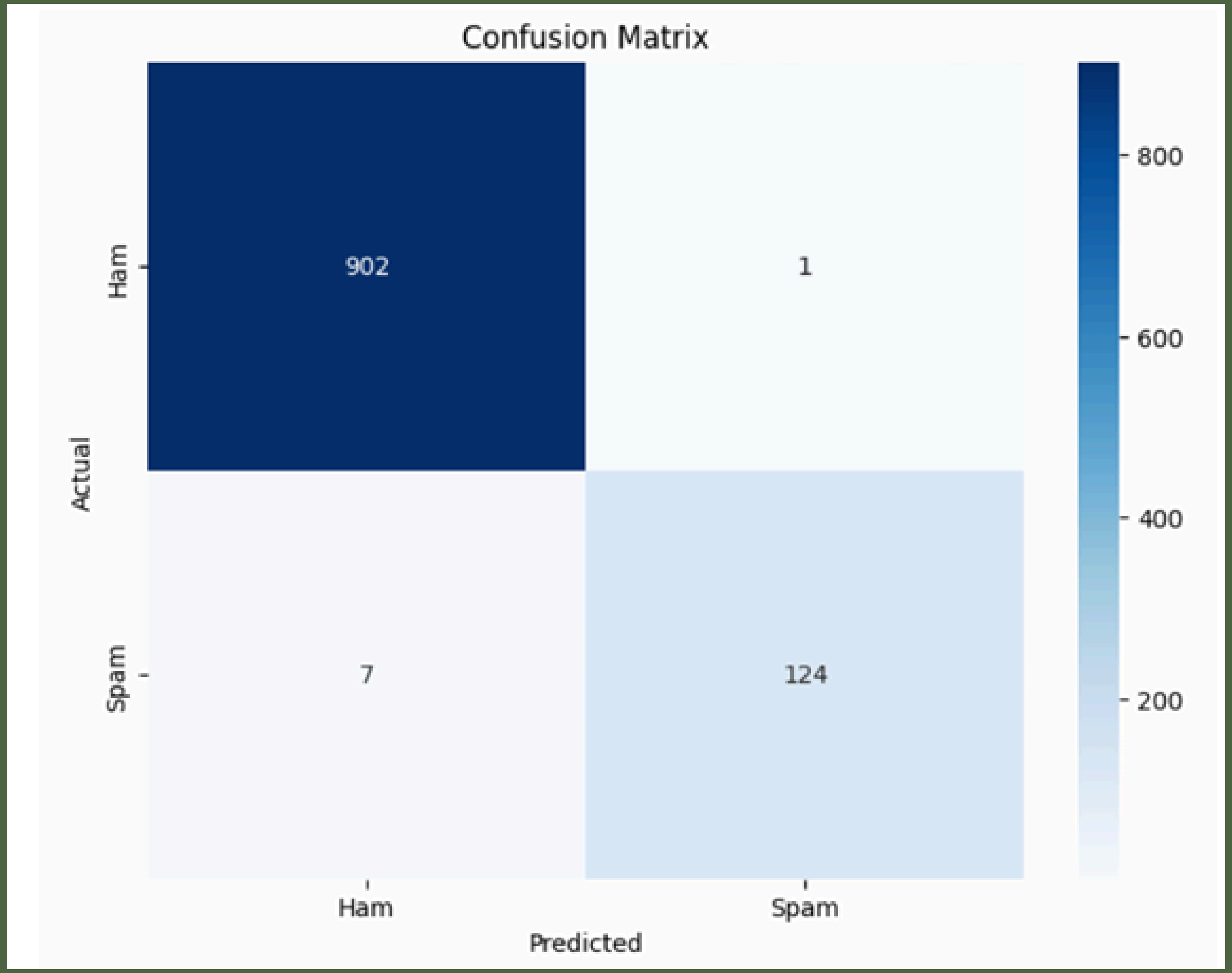


```
Classification Report:
              precision    recall  f1-score   support

         Ham       0.99      1.00      1.00       903
        Spam       0.99      0.95      0.97       131

    accuracy                           0.99      1034
   macro avg       0.99      0.97      0.98      1034
weighted avg       0.99      0.99      0.99      1034
```

Confusion matrix:



THANK YOU!