

# Application of Large Language Models for Spam Detection

MSc Research Project  
Cybersecurity

Jose Merida  
Student ID: 23271621

School of Computing  
National College of Ireland

Supervisor: Michael Prior

**School of Computing**

Student Name:	Jose Fernando Merida Ramos		
<b>Student ID:</b>	23271621		
Programme:	Cybersecurity	Year:	2024
Module:	MSc Research Project		
<b>Supervisor:</b>	Michael Prior		
Submission Due Date:	12/12/2024		
Project Title:	Application of Large Language Models for Spam Detection		
Word Count:	5213 <b>Page Count</b> 20		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

<b>Signature:</b>	Jose Fernando Merida Ramos
<b>Date:</b>	12 <sup>th</sup> of December

## PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Application of Large Language Models for Spam Detection

Jose Fernando Merida Ramos

23271621

## Abstract

Spam messages in emails and SMS are a raising problem, often containing scams, malware, or unwanted ads. Detecting spam is essential to protect users and improve communication. This project combines BERT (Bidirectional Encoder Representations from Transformers) and SVM to enhance SMS spam detection. BERT processes messages to capture their meaning, while SVM classifies them as spam or ham.

By using the SMS Spam Collection dataset, the study compares the BERT-SVM model with Traditional Text Classification. The results demonstrate that BERT-SVM outperforms older techniques in precision, recall, and accuracy. An API was also built to test the model's real-world performance. This project emphasizes the potential of large language models in spam detection and recommends exploring lighter versions of BERT for future use.

**Keywords** – *Spam, BERT, SVM, Large Language Models, Spam detection.*

## 1 Introduction

Spam messages, both in emails and SMS, are a growing problem in today's digital world. These messages often include harmful links, scams, or unwanted advertisements, causing trouble for individuals and businesses. Detecting spam effectively is important to protect users from cyber risks. Traditional methods like Support Vector Machines (SVM) have been widely used, but new paradigms such as BERT (Bidirectional Encoder Representations from Transformers) have made great progress in processing language. This project focuses on using these advanced algorithms to make spam detection better, especially for SMS messages.

Studies show that spam messages are a significant issue worldwide. For instance, approximately 55% of all email traffic is classified as spam, according to cybersecurity reports (Labonne & Moran, 2023). SMS spam is also increasing rapidly, presenting risks such as spreading malware or stealing personal information (Gadde et al., 2021). Phishing a type of spam aimed at stealing sensitive information accounts for a significant portion of these messages, with over 3.4 billion phishing emails sent daily (Irwin, 2023). According to the Anti-Phishing Working Group (APWG), there were 963,994 phishing attacks reported in the first quarter of 2024, with social media platforms being the most targeted sector, representing 37.4% of all phishing attacks (APWG / *Phishing Activity Trends Reports*, n.d.). These messages not only waste time but also create security concerns for users. The main challenge lies in distinguishing legitimate (ham) messages from spam, especially in datasets that often include informal language, spelling errors, and abbreviations. Whilst many existing solutions achieve good accuracy, they often struggle to address these complexities effectively (AbdulNabi & Yaseen, 2021).

This project aims to answer the question: "How effective is BERT combined with machine learning techniques such as SVM to enhance SMS spam detection?" The goal is to study how well this combination works compared to traditional text classification. By using BERT's ability to understand context and SVM's strong performance in classification, the project tests their accuracy, precision, and recall to see if they can better detect spam.

The paper is organized into separate sections, which provide a coherent framework for the study. First, the section on related work evaluates previous researches with regard to spam detection, including traditional machine learning approaches and recent works on extensive language models. The methodology section talks about the experimental setup, which gives a detailed description of the dataset, preprocessing steps, and how BERT was combined with SVM for classification. Then, the implementation section reveals how the model was developed and tested in building an API for classifying messages as spam or ham. The evaluation section presents the results of the model proposed, comparing its performance with the traditional approach based on some metrics: precision, recall, F1-score, and accuracy. At last, discussion and conclusion sections sum up all that has been found out so far, emphasize the benefit of combining BERT-SVM, and recommend the future research studies on other lighter versions of BERT and real-world applications for the model.

## 2 Related Work

In this section, we will review the relevant projects in the industry related to machine learning and spam classification. The projects described in this section introduce the reader to understand the purpose of this paper.

This section contains two parts, the machine learning approach to spam classification where we examine the projects related to ML for classification and their results. The second part is the noble methodology with interesting projects using large language models for spam detection.

### 2.1 Machine Learning Approach to Spam Classification

In order to classify spam, there are several Machine Learning (ML) methods to apply. To begin with (Almeida et al., 2011) in their work, the main goal is to determine the ML algorithm with the highest accuracy in the SMS Spam classification problem. In this project, the researchers use a dataset from *UCI Machine Learning Repository* that contains 5,574 messages 425 are Spam from the *Grumbletext* website, *Ham or Non-Spam* 3,375 from NUS SMS, 450 from Carolines Tag's PhD thesis and 1,002 from the SMS Spam Corpus v0.1 Big dataset. The algorithms applied are Naïve Bayes, Support Vector Machine (SVM), and k-Nearest Neighbour (k-NN), and also the study explores Random Forest (RF) and AdaBoost in order to boost performance. The project reveals that Naïve Bayes hit the best accuracy by 98.88% followed by SVM 98.86%. The researchers conclude that Naïve Bayes with Laplace smoothing and SVM with a linear kernel are the best solutions for detecting SMS spam. Although ensemble methods such as Random Forest and *AdamBoost* give better results in this model is not the case. This means complexity did not necessarily provide better performance in this dataset.

The study presented (Trivedi, 2016) exposes ML algorithms for spam classification. In order to perform this action, the author uses a dataset combination of 6,000 emails 50% Spam and the rest Ham also the dataset contains Headers and Body. The data processing stages are used for converting into string word vectors (vectorization) and removing HTML tags. Moreover, applying the *Greedy Stepwise Subset Evaluation Method* for reducing the feature up to 49 characters and also helping to add or remove features (columns) based on relevance. Machine Learning classifiers applied in this study are Naïve Bayes, Support Vector Machine (SVM), and Decision Tree. The metrics for comparing the results are F-Measure, a combination of precision and recall, False Positive, and training time. As a result, the best algorithm for this dataset is SVM with an F-Measure of 93.3% and a false positive of 6.5%. Showing that SVM is a powerful algorithm for this problem.

The paper “Email Spam Detection Using Machine Learning Algorithms” focuses on using ensemble methods such as Random Forest and Bagging and Boosting. The authors (Kumar et al., 2020) utilized different datasets from Kaggle, skelarn and self-collected sources in total the dataset contains 5,573 email samples and 2 columns (spam and ham). Furthermore, the data processing step is very similar compare with the other projects that contain cleaning, top word removal and tokenization. One of the key points of this paper is in the feature extraction step which contains “Bag of Words (BOW)” that works similarly to a dictionary but with the vocabulary of all unique words from the dataset. The algorithms used in this project are Naïve Bayes, Support Vector Machine, Decision Tree, K-Nearest Neighbours by applying Random forest and Bagging and Boosting for improving the classification. The evaluation part applies hyperparameter tuning in order to optimize the performance of each classifier algorithm, also the metrics for measuring the models are accuracy, precision, recall and F1-score. To sum up, the paper shows Naïve Bayes as the classifier with robust performance with high accuracy, combined with the ensemble methods applied such as *AdaBoost* improves the accuracy by reducing the variance.

The paper called “SMS Spam Detection using Machine Learning and Deep Learning Techniques” proposed by (Gadde et al., 2021) demonstrates the use of Machine Learning for SMS spam. The dataset called SMS Spam Collection from the UCI contains 5,572 messages divided into “spam” or “ham”. The Machine Learning models analysed are Naive Bayes, Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbours, and Support Vector Machine. The paper demonstrates the use of tokenization, stopword removal and stemming for text preprocessing. Stemming is a technique for reducing the word to its stem for instance, the word “playing” can be changed as “play”, after this step the embedding part can be applied. The authors apply embedding techniques such as Count Vectorize, TF-IDF Vectorizer and Hashing Vectorize. As mentioned Count Vectorizer is used for removing special symbols and converting lower case, also TF-IDF Vectorizer uses two parameters, Term Frequency and Inverse Document Frequency. TF is the number of times the sentence appears and IDF is inverse document frequencies and the Hashing Vectorizer technique is for converting text to vector. The results presented the SVM as the best accuracy rate with 97%, however applying LTM the accuracy moves up to 98.5%. Furthermore, the paper shows the importance of the preprocessing stage and how it affects the ML algorithm. For future work, the authors plan to evaluate the models using different datasets.

## 2.2 Large Language Models (LLMs)

The paper “Spam-T5: Benchmarking Large Language Models for Few-Shot Email Spam Detection” compares Large Language Models such as RoBERTa, SetFit and Flan-T5 against traditional models for email spam detection. According to this paper, the Flan-T5 model (family of models based on T5) was implemented with the HuggingFace the Sentence piece tokenizer. In this project the authors (Labonne & Moran, 2023) Consider different datasets, such as Ling-Spam with 2,893, SMS Spam Collection with 5,574 tagged “spam or ham,” Spam Assassin Public Corpus with 6,047 messages, and Enron Email 33,716 emails that contain 17,171 spam emails and 16,545 ham emails. Having different types of distribution channels allows for the wide testing of the models’ performance, from emails to SMS. Furthermore, the based models selected for this study are Nave Bayes (NB), Logistic Regression (LR), K-Nearest Neighbors (KNN), SVM, XGBoost, and LightGBM. The results of this paper show that Spam-T5 (Flan-T5 plus HuggingFace) reaches the highest F1 score at 97.4% followed by SVM at 95.2%. The authors emphasise the potential of LLMs and particularly Spam-T5 in addressing the spam detection challenges. For future work, the investigators mention focusing on decreasing the computational needs of the LLMs.

The authors (Xu et al., 2019) focus on the application of a BiLSTM, which is the abbreviation for Bidirectional Long Short-Term Memory, model as a way to improve the sentiment analysis for comment texts. Compared with the traditional models of CNN (Convolutional Neural Networks), RNN (Recurrent Neural Networks), and LSTM (Long Short-Term Memory networks), the BiLSTM has captured contextual information in both forward and backward directions. It consequently finally achieves a very high F1-score of 92%, which means a substantial improvement in performance. Further, the study goes one step further by enhancing word representation by directly combining the sentiment information and TF-IDF algorithm, ensuring an improved sentiment identification at the word level. While the model is superior to all other models in terms of accuracy, precision, and recall, it has to be highlight that training such a model is really time-consuming and requires extensive computational resources. The high requirement can make this model less applicable to real-world scenarios/applications that need fast response times. The authors give future research directions by pointing out that there is an imperative need to improve the efficiency of the model. Improvement in this aspect should focus on how to reduce the time needed for the training of the model to increase its applicability, especially in large-scale sentiment analysis tasks.

The study on Large Language Models (LLMs), such as GPT, demonstrates how these models have revolutionized human-machine communication by processing natural language through advanced techniques like tokenization, stemming, and topic modelling. These models have shown great promise in various fields, such as chatbots, healthcare data analysis, and text summarization. The study “Large Language Model (LLM) & GPT, A Monolithic Study in Generative AI” uses multiple datasets to evaluate the models' effectiveness, including the Ling-Spam dataset with 2,893 messages, SMS Spam Collection (5,574 messages tagged as spam or ham), the SpamAssassin Public Corpus (6,047 messages), and the Enron Email Dataset with 33,716 emails (17,171 spam and 16,545 ham). By applying these NLP techniques, LLMs have outperformed traditional models in handling complex language patterns and context. However,

challenges such as high computational costs and reliance on proprietary systems still hinder their broader adoption. The authors (Mohammad et al., 2023) suggest that transitioning to open-source LLMs could address these limitations, reduce costs, and foster innovation. This shift could enhance accessibility, democratize the use of LLMs, and encourage further advancements, especially in domains such as speech recognition and chatbots.

The research titled “Spam Email Detection Using Deep Learning Techniques” focuses on the use of advanced deep learning models, specifically BERT (Bidirectional Encoder Representations from Transformers) and BiLSTM (Bidirectional Long Short-Term Memory), for detecting spam emails. Two datasets were used: the Spambase dataset, containing 5,569 emails with 745 labelled as spam, and the Kaggle Spam Filter dataset, consisting of 5,728 emails with 1,368 spam messages. A balanced training dataset of 2,000 spam and 3,000 ham emails was prepared, along with a separate test set of 226 emails. The BERT model outperformed all others, achieving 98.67% accuracy and an F1-score of 98.66%, while BiLSTM achieved 96.43% accuracy. Traditional models, such as Naive Bayes and KNN, were less effective, showing the strength of deep learning in identifying complex patterns in text. Preprocessing steps, including tokenization and stopword removal, improved the models' performance. However, the study highlights challenges such as the computational intensity of BERT and its limited ability to process long emails due to a sequence length cap of 300 tokens. The authors (AbdulNabi & Yaseen, 2021) suggest increasing the sequence length, applying the models to other languages, and using more diverse datasets in future research to further improve spam detection systems. This study demonstrates the importance of deep learning in modern spam filtering.

### **3 Research Methodology**

This section explains how the experimental setup was created. We will review the dataset, the steps followed to evaluate the traditional technique. Also, we will determine BERT and how works in the proposed model.

#### **3.1 Dataset Description**

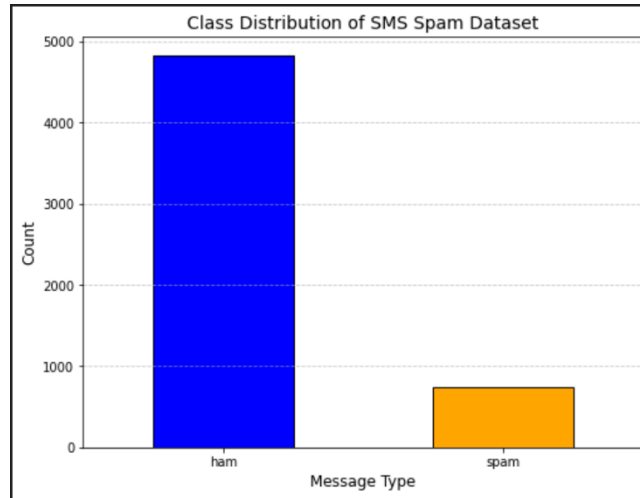
The dataset used in this paper is the “SMS Spam Collection Dataset”, sourced from Kaggle. It contains 5,572 SMS messages, in English, designed for binary purposes to classify “spam” or “ham”. In this context:

- Spam refers to illegitimate messages
- ham refers to legitimate messages

##### **3.1.1 Imbalance**

As we analysed the dataset we detected that it is imbalanced: The messages tagged as Ham 4,827 and Spam 747. In order to avoid this problem, we use a parameter (`class_weight`) in the Scikit-learn library that helps us to handle it. In the next paragraph, we will extend it more. Additionally, the dataset reveals informal SMS communication, containing spelling errors,

abbreviations, and informal language. These characteristics introduce additional complexity to the classification task and emphasise the importance of robust preprocessing techniques.



**Figure 1 Dataset distribution**

In order to contrast our proposed method against traditional text classification, we have pre-processed the data. The traditional method requires different steps in order to clean the data to be classified by any classifier. This includes performing steps such as:

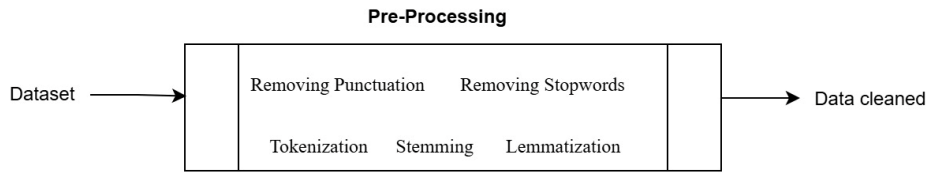
- Removing Punctuation
- Tokenization
- Removing Stop words
- Stemming
- Lemmatization

### **3.2 Traditional Text Classification**

We will call traditional text classification the act of preprocessing the dataset. This step usually includes removing punctuation, tokenisation, steaming and removing stopwords. Performing this action can take time and will not reflect a better classification.

The flowchart presents the preprocessing, which includes the dataset or raw data to be cleaned and the token as an output. In our evaluation, we have included the five stages appearing in the figure below.





**Figure 2 Flowchart of traditional pre-processing**

The first step is tokenization, which involves breaking the input text into smaller parts, such as words or phrases. The second step is to remove stopwords, which are common words like "the," "and," or "in" that do not add much meaning. This helps make the text simpler without losing important information. The final step is applying either stemming or lemmatization. Stemming removes prefixes and suffixes to reduce words to their root form, whereas lemmatization transforms words to their base or dictionary form.

The steps listed are considerably important since SVM receives the dataset cleaned and tokens. In case the dataset is not treated it can lead to a misclassification.

## Removing Punctuation

This function helps to delete punctuation such as commas, full stops, question marks and so forth. To perform this action we have used the library 'string' which has a list of common punctuation symbols. In order to clean each message it is verified and delete the character.

## Tokenization

The main goal of this function is to split into individual tokens. The function uses the 're.split()' method to separate the sentence or message from everything that is not a word character such as spaces. Also, the function converts to lowercase in order to make sure that the whole words are being treated in the same circumstances.

## Removing Stop words

This function its principal target is to remove common words. The common words that we want to delete are "the", "for", "and", etc. To perform this action we are using the NLTK library.

## Stemming

The purpose of this function is to reduce the word to the root form. The function uses the PorterStemmer from the NLTK library that removes the 'ed', 'ing' or 'ly' from the words. The purpose for applying this technique is to reduce the number of unique words in the dataset.

## Lemmatization

The function cleans the dataset by changing the word endings. In order to be classified accurately the ‘lemmatisation’ will help to convert the words into their basic form called lemma.

### **3.3 Large Language Models**

#### **3.3.1 BERT**

The paper “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” (Devlin et al., 2019), introduces the term transformers. They are used in natural language processing to work with sequences like text or speech. BERT stands for Bidirectional Encoder Representations from Transformers, is a type of transformer that understands context by looking at the words before and after each word. It uses tokens and embeddings to process text. This study has two steps: first, creating embeddings with the base version of BERT, which is smaller and faster, and second, using SVM, a supervised machine learning method, for classification.

#### **3.3.2 Pre-trained Model and Tokenizer**

In this paper, we use a pre-trained model called *BertModel* and its tokenizer, *BertTokenizer*. The model is based on *Bert-base-uncased*, a version made for English that does not differentiate between lowercase and uppercase letters. It was trained on a large dataset, including Wikipedia. The model has 12 layers, each with 768 dimensions, 12 attention heads, and about 110 million parameters.

## **4 Design Specification**

This section is divided into three parts. The first part explains the libraries used in the proposed model and how they are applied in the project. The second part describes the evaluation approach for the model. Finally, the third part explains the model design and the entire process.

### **4.1 Libraries**

#### **4.1.1 Pandas**

This library needs to be part of our project due to it can manage data. Pandas is one of the most used libraries for data handling. In this project, we use pandas for storing our dataset and covert the dataset to a data frame.

#### **4.1.2 Numpy**

The project uses numpy to convert the embeddings to a 2D array. Afterwards, the 2D array is sent to the SVM for training and testing the model.

### 4.1.3 Transformers

This library provides access to the methods BertTokenizer and BertModel. Embeddings are generated from the model to be used for SVM..

### 4.1.4 Sklearn

Is the most useful library for machine learning. This library, includes different machine learning algorithms such as SVM. We have used SVM for classification of our dataset.

### 4.1.5 Matplotlib

This library is used for mathematical solutions. In the project is used to generate the confusion matrix to plot the information after classification and evaluate the model.

## 4.2 Evaluation

To verify if the proposed model is classifying accurately, we need to evaluate it. For this, we will use a mathematical method to analyze the results and make conclusions based on the data. This section explains the formulas used for this evaluation. We will use acronyms for the formulas: TP as True Positive, FP as False Positive, FN as False Negative.

### 4.2.1 F1 Score

By calculating the precision and Recall we can obtain the relation between these metrics called F1 score. In other words, the F1 Score provides the harmony of Precision and Recall.

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

### 4.2.2 Precision

Precision is the measure of the accuracy of true positives. In our project how many inputs marked as spam are actually spam in comparison to the total spam values.

$$Precision = \frac{True\ Positive}{True\ Positives + False\ Positives}$$

### 4.2.3 Recall

This metric provides the true-positive rate showing the actual positives messages that were classified correctly. In broad terms, the recall tells how well the model could identify the spams.

$$Recall = \frac{True\ Positive}{True\ Positives + False\ Negative}$$

### 4.2.4 Accuracy

This metric provides general information about our model, and how well our model predicts in other words the model's performance in one number.

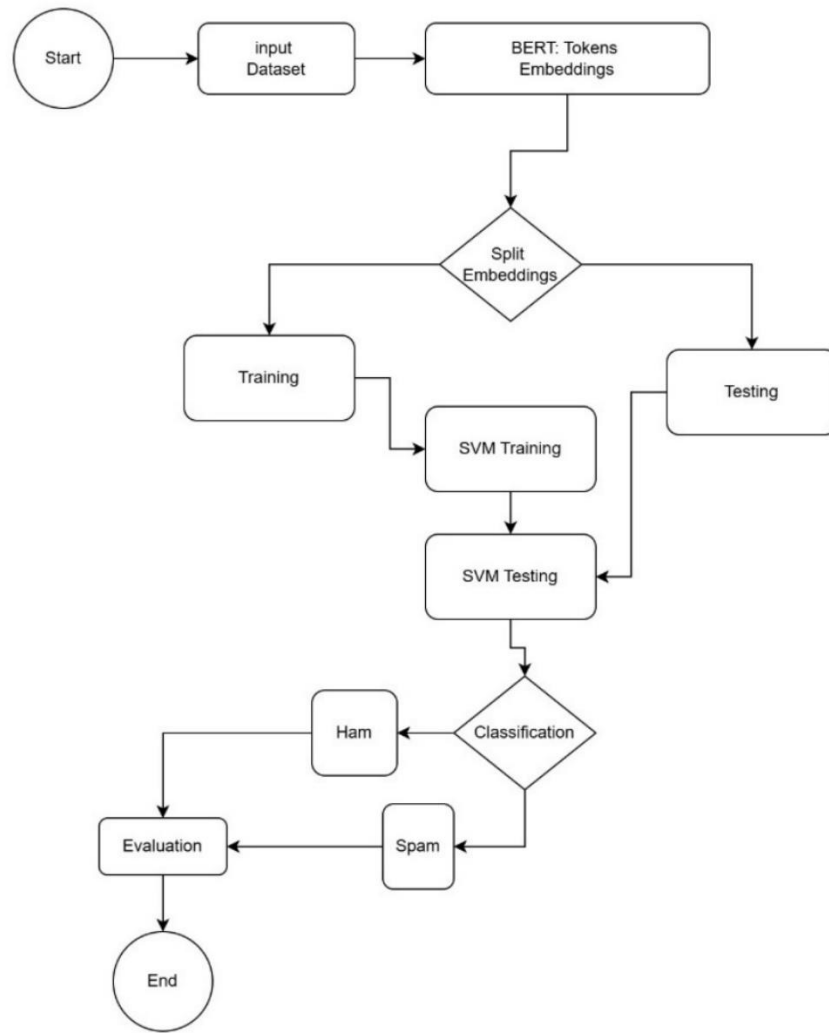
$$Accuracy = \frac{True\ Positive + True\ Negative}{TP + TN + FP + FN}$$

## 4.3 Model Design

The system design is leveraged on BERT, and we have established the following stages:

- Input Dataset
- BERT to generate tokens and embeddings
- Splitting the dataset into training and testing
- SVM Training
- SVM Testing
- Evaluation, of the model against traditional text classification

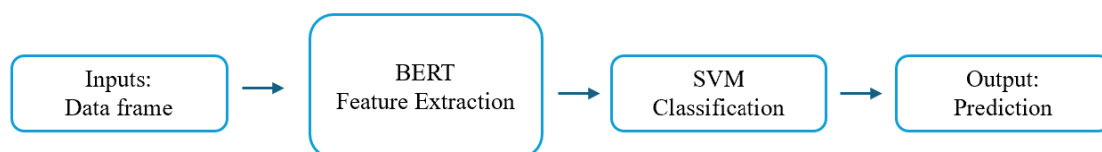
In the stages mentioned above, the generation of tokens and embeddings is working as a feature extraction. Since the SVM needs embeddings BERT is an accurate solution for this project. And also, the SVM training uses 80% of the dataset and for the testing 20%. Last, for the evaluation stage we have included the metrics, Accuracy, F1-score, Precision, recall and the Confusion Matrix. Next figure, demonstrates the graphical process.



**Figure 3 Flowchart of the Model Design**

## 5 Implementation

In this chapter, we will discuss the system, and how BERT and SVM work together to improve spam detection. According to our research, we have established our system is compound in four stages: Data frame, Bert Feature Extraction, SVM Classification and Prediction. The image below shows each step followed by the next step to be implemented.



**Figure 4 Flowchart of Model Implementation**

## 5.1 Data frame

The dataset contains information on SMS messages. We leveraged a widely well-known spam dataset from Kaggle which includes two columns first one contains the classification spam or ham and the second column the messages. The dataset was converted into data frame by using pandas.

## 5.2 BERT Feature Extraction

In order to generate embeddings we have to call the methods to interact with BERT. Establishing the methods *BertTokenizer* and *BertModel* is fundamental since these are the pretrained model which generates the tokens and embeddings. The model we are using named *bert-base-uncased* offering a good balance between performance and computational.

This project uses a function called *get\_bert\_embeddings* to create embeddings, which are numerical representations of text. The function begins by sending the input the dataset to a tokenizer, which splits it into smaller pieces called tokens and turns them into tensors with a maximum length of 512 characters.

Furthermore, these tensors are then passed through the BERT model to produce embeddings that capture the meaning of the text. This process is applied to the dataset column containing spam and ham messages, and the resulting embeddings are stored in a new column called embeddings.

```
# Function to generate embeddings
def get_bert_embeddings(text):
    # Tokenize and encode the text
    inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True, max_length=512)

    # Get the BERT output
    with torch.no_grad():
        outputs = model(**inputs)

    # Take the embedding of the [CLS] token
    cls_embedding = outputs.last_hidden_state[:, 0, :].squeeze()
    return cls_embedding
```

Figure 5 Snippet Function to generate embeddings

The image below demonstrates how BERT has converted the text into embeddings. This part is crucial and it is the keystone of the model proposed for improving spam detection. Embeddings are the essential information for training the classifier.

	v2	embeddings
0	Go until jurong point, crazy.. Available only ...	[tensor(-0.1513), tensor(-0.3229), tensor(0.18...
1	Ok lar... Joking wif u oni...	[tensor(-0.1238), tensor(0.3435), tensor(-0.00...
2	Free entry in 2 a wkly comp to win FA Cup fina...	[tensor(-0.5031), tensor(-0.2829), tensor(0.57...
3	U dun say so early hor... U c already then say...	[tensor(0.0646), tensor(0.5769), tensor(0.3411...
4	Nah I don't think he goes to usf, he lives aro...	[tensor(0.0947), tensor(0.3897), tensor(0.0389...
...	...	...
5567	This is the 2nd time we have tried 2 contact u...	[tensor(-0.0725), tensor(0.0551), tensor(0.620...
5568	Will l_b going to esplanade fr home?	[tensor(-0.0089), tensor(0.1375), tensor(0.270...
5569	Pity, * was in mood for that. So...any other s...	[tensor(-0.0464), tensor(0.0716), tensor(-0.28...
5570	The guy did some bitching but I acted like i'd...	[tensor(0.2887), tensor(0.2153), tensor(-0.353...
5571	Rofl. Its true to its name	[tensor(-0.6254), tensor(0.0154), tensor(0.062...

Figure 6 Embeddings

### 5.3 SVM Classification

This we will review the classification process. The SVM classification section takes place after generated embeddings. As mentioned the dataset includes one column with spam or ham, we convert the labels into numeric as spam equals 1 or ham equals 0.

Converting the embeddings to the 2D array. In this step, we have turned the dataset into a 2D array by using the library *Numpy*. Also, we combined the labels and embeddings, for the X-part embeddings generated by the model and Y values labels (0 or 1).

The following step it is to split the dataset. We need to divide the dataset for different purposes such as training and testing. The splitting part will help us to separate randomly the data for training which we decide to use 80% and test 20%. We have performed this action by *sklearn.model\_selection* calling the method *train\_test\_split*.

By using the library *sklearn* we call the method *svm* which is the machine learning classifier. For the training of our classification model we use a linear kernel for maintaining the efficiency and balancing the dataset by *class\_weight*. After the steps mentioned, we can pass to the prediction. As we separated the dataset into the 80:20 ratio we are able to use 20% of our dataset to test the model.

```
# Initialize and train the SVM classifier
svm_model = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto', class_weight='balanced' )
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred))
```

Figure 7 Snippet SVM Classification

## 5.4 Output Prediction

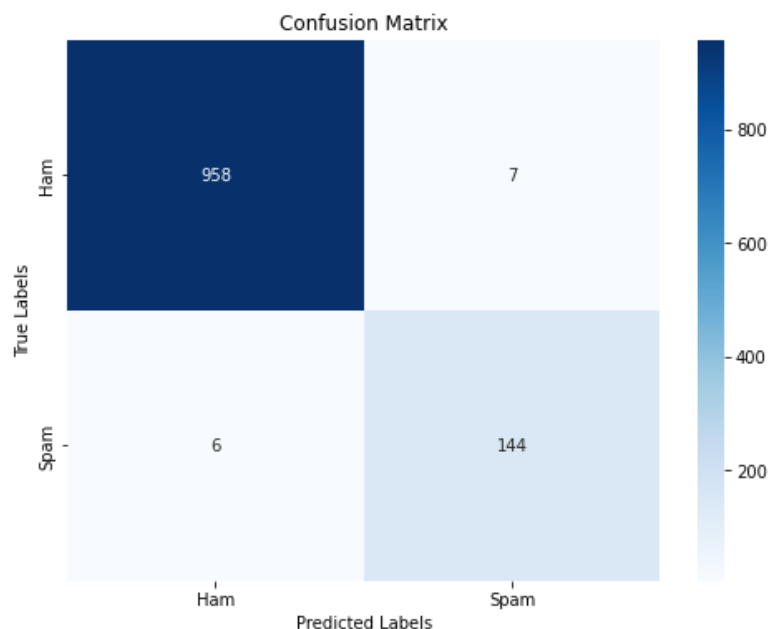
The last stage in this implementation is to determine how well our model predicts. We will evaluate the model using the metrics Accuracy, F1-score, Precision and Recall. However, we can introduce the Confusion Matrix to evaluate the performance of a classification model as well.

As shown in our table results, by applying BERT we have an Accuracy of 99% which is a high accuracy. The model will predict 99% of the time accurately.

**Table 1 Metrics For BERT combined with SVM**

	BERT + SVM		
	Precision	Recall	F1-score
Ham	0.99	0.99	0.99
Spam	0.95	0.96	0.96
Accuracy			0.99

As we can learn from the confusion matrix we have 144 True Positives (TP) which means the values that have been labelled as spam and the model classify accurately. Also, the True Negatives (TN) are 958 messages where the dataset labelled them as Ham and the model classified correctly. Furthermore, the False Positives (FP) and False Negatives (FN) are 7 and 6 respectively, FP was labelled as ham and the model was classified as spam the other way around for the FN was tagged as spam and the model was classified as ham.



**Figure 8 Confusion Matrix**

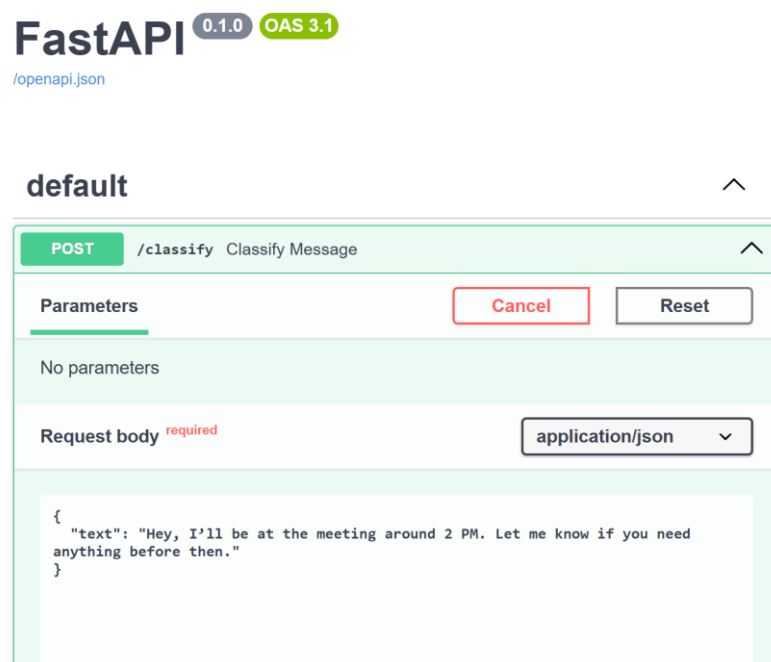


## 6 Evaluation

In this chapter, we will present the solution to evaluate the model, besides the metrics explained earlier. By API (Application, Program Interface) which interacts with our model and will return a response to the message and a classification as ham or spam. We will provide one spam message and one ham.

### Ham Message

The image below shows the message sent to the API: Hey, I'll be at the meeting around 2 PM. Let me know if you need anything before then.



**Figure 9 Ham Message**

As we can observe in the image below, the model responds as Ham which is accurate. That demonstrates that the model predicts well the hams messages.



Code	Details
200	<div>Response body</div> <pre>{   "message": "Hey, I'll be at the meeting around 2 PM. Let me know if you need anything before then.",   "classification": "ham" }</pre> <div>   </div> <div>Response headers</div> <pre>content-length: 125 content-type: application/json date: Sun,01 Dec 2024 16:33:59 GMT server: uvicorn</pre>

Figure 10 Ham classification

## Spam Message

Congratulations! You've been selected for a FREE gift card worth \$1,000! Click here to claim

**FastAPI** 0.1.0 OAS 3.1  
/openapi.json

default ^

POST /classify Classify Message ^

Parameters Cancel Reset

No parameters

Request body required application/json v

```
{
  "text": "Congratulations! You've been selected for a FREE gift card worth $1,000! Click here to claim"
}
```

Figure 11 Spam Message

As we can learn from the image below, the model responds to the classification as spam. The model classifies as precise.

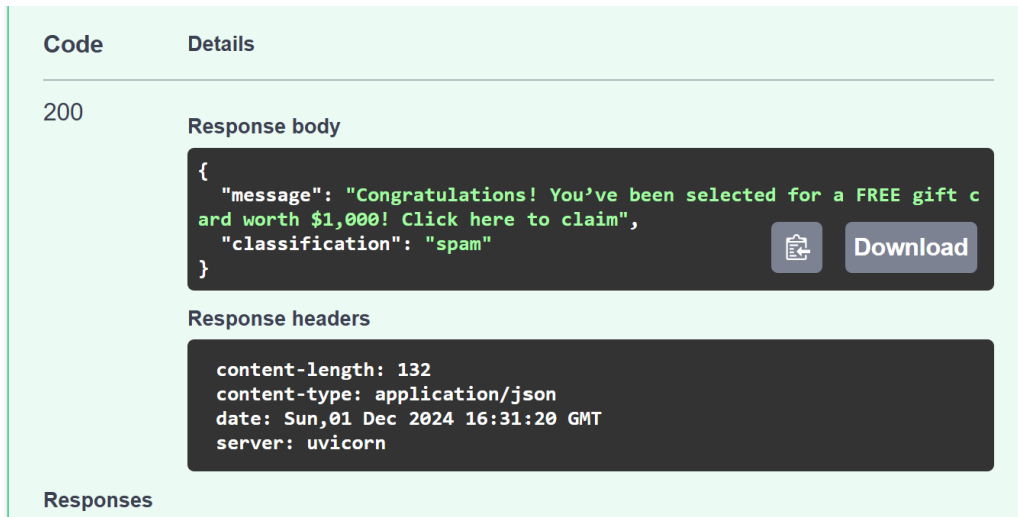


Figure 12 Spam Classification

## 6.1 Discussion

In this section, we will go through the metrics generated by the two approaches for spam classification.

From the table below we can learn the following metrics; regarding the proposal model, the Precision are 0.99 and 0.95 for ham and spam respectively whereas the traditional method has 0.98 for ham and 0.96 for spam showing that the model with BERT has better classification for spam messages that are truly spam. Furthermore, Recall shows us 0.99 ham and 0.96 spam for the new model in contrast to the traditional method that shows 0.99 and 0.89 for ham and spam respectively we can deduct from these numbers for the combined model does well classifying the real positives accurately. Moreover, the F1-score presents 0.99 ham and 0.96 spam regarding the merged model, whilst the other model appears 0.99 and 0.92 demonstrating that our model has a higher balance between precision and recall. Finally, the Accuracy gives us a general idea of how well the models work, the number of the proposal model reaches 0.99 while the traditional method is 0.97 giving us a 0.02 difference in the general performance.

Table 2 Results

BERT Combined SVM			
	Precision	Recall	F1-score
Ham	0.99	0.99	0.99
Spam	0.95	0.96	0.96
Accuracy			0.99
Traditional Text Classification			
Ham	0.98	0.99	0.99
Spam	0.96	0.89	0.92
Accuracy			0.97

## 7 Conclusion and Future Work

In this paper, we went through different approaches to spam detection. From the traditional text classification in Machine learning such as Naïve Bayes, KNN, and SVM to noble solutions such as Large Language Models. The proposed model in this project applies BERT as feature extraction, passing over the traditional data cleaning such as tokenization, removing stopwords and so forth. As we explained in this study, BERT generates embeddings after we apply a machine learning algorithm, in this case, SVM, to classify. We trained the model with 80% of our data and the rest for testing. After training and testing our model, we used the same data but this time applied the traditional text classification, the result shows the proposal model's accuracy exceeds by 0.02 versus the traditional method.

Additionally, we built a local API to test the model, by sending two different messages, spam and ham, the algorithm was able to detect spam and classify it. To sum up, by using BERT we demonstrate the effectiveness of spam detection and also how will work in an API.

For future work, we will use a mix of datasets and also apply DistilBERT which is a lightweight version of BERT in order to minimize the computational resources. Moreover, the local API could be deployed in a real environment such as Gmail API. Our project is ready to be implemented in real circumstances.

## References

- AbdulNabi, I., & Yaseen, Q. (2021). Spam Email Detection Using Deep Learning Techniques. *Procedia Computer Science*, 184, 853–858.  
<https://doi.org/10.1016/j.procs.2021.03.107>
- Almeida, T. A., Hidalgo, J. M. G., & Yamakami, A. (2011). Contributions to the study of SMS spam filtering: New collection and results. *Proceedings of the 11th ACM Symposium on Document Engineering*, 259–262.  
<https://doi.org/10.1145/2034691.2034742>
- APWG / *Phishing Activity Trends Reports*. (n.d.). Retrieved 9 December 2024, from <https://apwg.org/trendsreports/>

- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. <https://doi.org/10.48550/arXiv.1810.04805>
- Gadde, S., Lakshmanarao, A., & Satyanarayana, S. (2021). SMS Spam Detection using Machine Learning and Deep Learning Techniques. *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), 1*, 358–362. <https://doi.org/10.1109/ICACCS51430.2021.9441783>
- Irwin, L. (2023, June 8). *51 Must-Know Phishing Statistics for 2023 / IT Governance*. IT Governance UK Blog. <https://www.itgovernance.co.uk/blog/51-must-know-phishing-statistics-for-2023>
- Kumar, N., Sonowal, S., & Nishant. (2020). Email Spam Detection Using Machine Learning Algorithms. *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, 108–113. <https://doi.org/10.1109/ICIRCA48905.2020.9183098>
- Labonne, M., & Moran, S. (2023). *Spam-T5: Benchmarking Large Language Models for Few-Shot Email Spam Detection* (arXiv:2304.01238). arXiv. <http://arxiv.org/abs/2304.01238>
- Mohammad, A. F., Clark, B., & Hegde, R. (2023). Large Language Model (LLM) & GPT, A Monolithic Study in Generative AI. *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*, 383–388. <https://doi.org/10.1109/CSCE60160.2023.00068>
- Trivedi, S. K. (2016). A study of machine learning classifiers for spam detection. *2016 4th International Symposium on Computational and Business Intelligence (ISCBI)*, 176–180. <https://doi.org/10.1109/ISCBI.2016.7743279>

Xu, G., Meng, Y., Qiu, X., Yu, Z., & Wu, X. (2019). Sentiment Analysis of Comment Texts

Based on BiLSTM. *IEEE Access*, 7, 51522–51532. IEEE Access.

<https://doi.org/10.1109/ACCESS.2019.2909919>