A Thesis

entitled

A Novel Spam Email Detection Mechanism Based on XLNet

by

Neeraj Shrestha

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Master of Science Degree in Cyber Security

Dr. Jared Oluoch , Committee Chair

Dr. Weiqing Sun, Committee Co-chair

Dr. Junghwan Kim, Committee Member

Dr. Scott Molitor, Interim Dean
College of Graduate Studies

The University of Toledo
August 2023

An Abstract of

A Novel Spam Email Detection Mechanism Based on XLNet

by

Neeraj Shrestha

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Master of Science Degree in Cyber Security

The University of Toledo
August 2023

Email communication is a vital component of modern-day communication. However, an increase in spam emails is a significant threat to individuals and organizations which can result in financial and resource losses. Even though the development of effective spam detection mechanisms is essential to safeguarding email security and ensuring safe and secure communication, existing spam email detection methods have some limitations such as the limited capacity to handle the high volume, complexity, and variability of natural language and requiring extensive feature engineering. Moreover, they have the limited ability to remember information from previous time steps, resulting in poor performance, including high false positive rates, low recall, and the inability to detect new types of spam emails.

This thesis proposes a novel spam email detection approach that fine-tunes XLNet through supervised training on a labeled dataset of spam and non-spam emails without requiring hand-engineered features. The fine-tuned model is used to predict the class of previously unseen emails. Additionally, the thesis proposes a spam detection model that can handle the high volume, complexity, and variability of natural language in spam emails. The proposed spam email detection model is evaluated on various benchmark datasets, including SpamAssassin, Enron, and Ling-Spam, and its performance is compared with existing models. The model either outperforms or is at least comparable to the state-of-the-art (SOTA) models, achieving an accuracy,

the area under the receiver operating characteristic curve (AUC), and F1 scores of 0.9869, 0.9817, and 0.9869 on the SpamAssassin dataset; 0.9892, 0.9893, and 0.9892 on the Enron dataset; 0.9944, 0.9967, and 0.9944 on the Ling-Spam dataset; and 0.9888, 0.9889, and 0.9888 on the combined dataset, respectively.

The proposed model's superior performance in detecting spam emails demonstrates its potential to become a key component of email security measures. The model can improve email security and protect people and organizations from potential financial and resource losses as it is able to detect spam emails with high accuracy and manage the high volume, complexity, and variability of natural language in spam emails. With implications for the wider adoption of email as a secure and reliable communication channel, the results present a promising approach for creating more effective spam detection mechanisms. The proposed model is capable of becoming an essential tool in spam email detection and ensuring safe and secure email communication.

To my parents, who have supported me in my every step of the way, and have always been my biggest cheerleaders. Their unwavering love, support, and encouragement have been my motivation throughout this journey. Their sacrifices and belief in me have made it possible for me to pursue my dreams. I dedicate this thesis to them with all my heart, and I hope that I have made them proud.

Acknowledgments

I would like to express my gratitude to my advisor, Dr. Jared Oluoch, for his invaluable guidance and support throughout my research and thesis. His expertise, patience, and commitment to my success have been instrumental in shaping my research and personal development. I would also like to thank my co-advisor and Program Director of Master's Programs in Cyber Security, Dr. Weiqing Sun, and my committee member Dr. Junghwan Kim for their valuable feedback and suggestions.

I also want to thank the professors at The University of Toledo's Department of Electrical Engineering and Computer Science, and the Department of Engineering Technology for their dedication and commitment to creating an exceptional learning environment that has equipped me with the skills and knowledge necessary for my research.

I am forever grateful to my family and friends for their constant love, support, and encouragement throughout my academic pursuits. Their encouragement and belief in me have been a constant source of inspiration. Finally, I would like to thank all those who have contributed to my research in various ways. Their support, feedback, and encouragement have been invaluable and greatly appreciated.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AR	Autoregressive
AUC	Area Under the ROC Curve
BERT	Bidirectional Encoder Representations from Transformers
BiGRU	Bidirectional Gated Recurrent Unit
BiLSTM	Bidirectional Long short-term Memory
CNN	Convolutional Neural Network
CSV	Comma-separated values
EGOA	Enhanced Grasshopper Optimization Algorithm
FN	False Negatives
FNR	False Negative Rate
FP	False Positives
FPR	False Positive Rate
FT	FastText
GRU	Gated Recurrent Unit
HAN	Hierarchical Attentional Hybrid Neural Networks
KNN	K-Nearest Neighbor
LEP	Laplacian Eigenmap
LSTM	Long Short-term Memory
MAD	Median Absolute Deviation
MAE	Median Absolute Error
MLM	Masked Language Model
MLP	Multi-layer Perceptron
MNB	Multinomial Naive Bayes

MOGA–CNN–DLAS ..... Multi-objective Genetic Algorithm and CNN-based Deep Learning Architecture Scheme
MOGOA ..... Multi-objective Grasshopper Optimization Algorithm

NB ..... Naive Bayes
NLI ..... Natural Language Inference
NLP ..... Natural Language Processing
NSP ..... Next Sentence Prediction

QA ..... Question Answering

RF ..... Random Forest
RMSE ..... Root-mean-square Error
RNN ..... Recurrent Neural Network
ROC ..... Receiver Operating Characteristic

SOTA ..... State-of-the-art
SVM ..... Support vector Machine

TN ..... True Negatives
TNR ..... True Negative Rate
TP ..... True Positives
TPR ..... True Positive Rate

List of Symbols

A ..... Number of self-attention heads
H ..... Hidden Size
L ..... Number of Layers
CLS ..... Classificaiton token
SEP ..... Separator token

Chapter 1

Introduction

1.1 Background and Motivation

With the rise of electronic communication, email has become one of the most common forms of communication globally. According to recent statistics, around 22.43 billion legitimate emails are sent daily online [2]. However, the growing popularity of email has also led to a significant increase in spam emails. Spam email is a significant problem in the world of electronic communication. Almost 85% of total emails are spam, costing businesses $20.5 billion annually [2]. Since spam emails consume time and resources and can result in the spread of malware, phishing attempts, and theft of confidential information, they pose a severe challenge to both people and organizations. Thus, spam email detection is a critical task in email security as it helps to protect people and organizations from such messages.

Spam email detection is a task that requires natural language processing to determine whether it is spam or not. Over the years, a number of methods ranging from traditional machine-learning models to deep-learning models and transformer-based models have been proposed to detect spam emails. Traditionally, spam email detection methods have relied on machine learning models such as support vector machines (SVMs), random forests, and Naive Bayes. These methods use hand-engineered fea-

tures such as the frequency of certain words or the presence of certain characters to classify emails as spam or non-spam. However, the high volume, complexity, and variability of natural language make it difficult for these models to perform efficiently and also they often require significant feature engineering to achieve high performance, which can be time-consuming.

With the introduction of deep learning, there has been a shift towards adopting neural network-based models for spam email identification. Models such as convolutional neural networks (CNNs) [3], recurrent neural networks (RNNs), and long short-term memory (LSTM) [4] can automatically learn characteristics from raw text input and be more effective than conventional machine learning models. By using word embeddings to represent words in a high-dimensional space that captures the underlying meaning of words, these models are able to learn complex representations from text input. These models do, however, have some drawbacks, including a short memory, computationally expensive, slow processing speed, and lack of parallelization.

Transformer, a deep neural network design based on attention mechanisms, was presented in 2017 to address the shortcomings of existing models [1]. This architecture has become popular in natural language processing (NLP) because it can train faster, model long-range dependencies in text, manage variable-length inputs, and capture both global and local contexts. Since its release, numerous pre-trained deep learning models based on the Transformer have been created, achieving cutting-edge performance on a variety of NLP tasks. While Google researchers introduced Bidirectional Encoder Representations from Transformers (BERT) [5], researchers from Carnegie Mellon University and Google proposed XLNet [6]. XLNet, which is a generalized auto-regressive pre-training technique that, by incorporating concepts from Transformer-XL [7] and facilitating bidirectional context learning while overcoming the restrictions of BERT through an auto-regressive formulation, beats BERT on 20

tasks [6].

To our knowledge, there has been little or no research on using XLNet for spam email classification. As a result, this thesis proposes a new XLNet-based spam email detection model and assesses its effectiveness using data from the SpamAssassin corpus [8], Enron corpus [9], Ling-Spam corpus [10], and a combination of all of the aforementioned datasets. The thesis examines the ability of XLNet to learn from data without the need for hand-engineered features and evaluates the model's performance on the task of spam email detection and compares the result with the results of the existing models.

1.2 Thesis Objectives

The objectives of this thesis are outlined below:

- Explore the various techniques proposed for detecting spam emails, including traditional machine learning models and deep learning models, and Transformer-based models.
- Propose a new spam email detection model based on the XLNet and evaluate its performance on different datasets.
- Compare the performance of the proposed XLNet-based model with the results of previously proposed models and investigate whether the proposed scheme can overcome the limitations of previously proposed SOTA models and achieve better performance on spam detection tasks.

1.3 Organization of Thesis

The thesis is structured as follows:

Chapter 1 - Introduction: Chapter 1 briefly discusses spam email, spam email detection, and the need for a good spam email detection system to protect people and organizations from losses. The chapter concludes by outlining the research objectives and the organization of the thesis.

Chapter 2 - Literature Review: In this chapter, the thesis provides a comprehensive overview of spam email detection methodology, including its development over the years. Furthermore, the chapter highlights the limitations of traditional spam email detection methods, leading to the need for a more sophisticated and robust system.

Chapter 3 - An overview of Transformer and Transformer-based models: Chapter 3 of this thesis provides a brief discussion on Transformer which is a highly powerful and state-of-the-art architecture in Natural Language Processing (NLP). This chapter also discusses some transformer-based architectures like BERT and XL-Net.

Chapter 4 - Methodology: Chapter 4 discusses the methodology used for the research, including the datasets used for the research, evaluation metrics used for the evaluation of the proposed model, proposed model architecture, and hyperparameters used for fine-tuning the model.

Chapter 5 - Results: This chapter presents the results of the experiments conducted to evaluate the proposed spam email detection model. The chapter also discusses the performance metrics of the proposed model and compares them with the results of the SOTA methods.

Chapter 6 - Conclusion: The final chapter of this thesis provides a summary of the contributions made in the research. The chapter concludes the thesis by high-

lighting the significance of the proposed model in improving spam email detection and laying out a range of potential future tasks that could build upon and extend our current work.

Chapter 2

Literature Review

2.1 Spam Email Detection System

A spam email detection system is a collection of software and/or hardware tools designed to identify and filter out unwanted or unsolicited emails (also known as spam) from legitimate emails (also known as ham) in an email system. These systems typically use a combination of techniques such as content analysis, email header analysis, machine learning algorithms, and spam detection techniques to accurately identify and filter out spam emails. The primary objective of these systems is to identify and filter out unwanted emails that could potentially harm users or waste their time and resources.

2.2 History of Spam Email Detection System

Spam email detection has been a popular research area, and various studies have been carried out to identify effective methods for detecting unwanted emails. Traditionally, machine learning algorithms like random forests, Naive Bayes, and support vector machines were used for spam detection. Recent studies have focused on improving the performance of traditional algorithms, such as Naive Bayes, for spam detection. Over the years, a number of methods ranging from traditional machine-

learning models to deep-learning models and transformer-based models have been proposed to detect spam emails which are discussed below.

2.2.1 Traditional machine learning models

In [11], the writers introduce a novel approach to identifying spam emails. They employ both support vector machine (SVM) and Naive Bayes algorithms to overcome the constraints of the independence assumptions between features that hinder spam filtering performance. The outcome of the experiment indicates that the SVM-NB algorithm is better in spam detection, with a higher accuracy rate and faster classification speed. Furthermore, the new method outperforms existing techniques in terms of both accuracy and efficiency. In [12], the authors suggest some enhancements to the Naive Bayes (NB) classifier to increase its suitability for high-precision applications, such as spam filtering. These modifications include a novel weight aggregation function based on the correlation measure and a tree-based classifier cascade. The results of the experiment demonstrate that the changes improve the overall performance and spam detection precision, surpassing traditional methods. The method presented in [13] uses a two-layered flow, consisting of a Naive Bayes ensemble using bagging, and a decision-theoretic framework to detect spam. This approach comprises a Naive Bayes bagging model with a decision tree embedded in it, a reduction process using a likelihood score bound limitation, and an improved technique based on classifier error weighting. The experimental findings indicate the effectiveness of these modifications. In [14], the authors propose a new algorithm to enhance the accuracy of the Naive Bayes spam filter by integrating semantic, keyword, and machine learning algorithms. The researchers discovered a relationship between email length and spam score, which suggests the existence of Bayesian Poisoning. In [15], a dynamic spam filter that employs the Naive Bayesian algorithm is introduced. The filter uses a supervised machine learning model with training and testing phases to classify email

messages as either standard or spam based on their content. The model achieved an accuracy rate of 98% and was deployed as a web application. The experiment results demonstrated that the Naive Bayes algorithm was the most effective in email classification, with high accuracy and precision scores. In their approach, [16] developed a text categorization model for email texts using a linear support vector machine. The model was trained and tested on a dataset split into the train (80%) and test sets (20%). Pre-processing steps included removing stop words and vectorization, followed by feature selection using weighing and selection. The model achieved an accuracy of 98.56%, a recall of 96.5%, an F1 score of 97%, and an F-beta score of 96.23%. In [17], the authors compared the performance of random forest and Naive Bayes algorithms in classifying spam emails. The experiment results indicated that random forest outperformed Naive Bayes, achieving an accuracy of 98.33% compared to Naive Bayes' 88.22%. In their work, [18] compared the performance of K-Nearest Neighbor (KNN) and Multinomial Naive Bayes (MNB) algorithms for spam email prediction using machine learning techniques. The experiment results demonstrated that KNN outperformed MNB, and both algorithms had high accuracy for spam filtering.

According to [19], traditional machine learning algorithms have limitations in detecting spam emails. These limitations include low tolerance for errors, lack of parallel processing capabilities, limited self-learning capabilities, poor performance with large datasets, and difficulties in comprehending the context and connections between words in an email, making the classification of spam emails challenging.

2.2.2 Deep learning models

The limitations of traditional methods have prompted the adoption of deep learning and neural networks in the field of email and Twitter spam detection. In [20], Debnath and Kar utilized various deep learning techniques, including LSTM, BiLSTM, and BERT, along with NLP for data pre-processing. The results of their experiments

showed that the highest accuracy was achieved using BERT with a score of 99.14%, followed by BiLSTM with 98.34% and LSTM with 97.15%. Similarly, Jacob et al. [21] presented a multi-objective genetic algorithm and a CNN-based deep learning architecture scheme (MOGA–CNN–DLAS) for effective spam detection on Twitter. The MOGA-CNN-DLAS utilizes multiple layers of CNN and a multi-objective optimization process, which demonstrated improved accuracy, precision, recall value, F-score, and optimality measures as compared to benchmarked techniques. Moreover, it also reduced the root-mean-square error (RMSE), mean absolute deviation (MAD), and median absolute error (MAE). The proposed approach in [22] utilizes a deep learning method based on an artificial neural network to categorize emails into three groups: ham, spam, or phishing. Comparative analysis revealed that this approach outperforms prior SOTA research studies. The technique presented in [23] uses a wrapper method of multi-objective grasshopper optimization algorithm (MOGOA) for feature selection and a revised enhanced grasshopper optimization algorithm (EGOA) for training multi-layer perceptron (MLP). The performance of the proposed system was verified using SpamBase, SpamAssassin, and UK-2011 datasets and resulted in a performance improvement of up to 97.5%, 98.3%, and 96.4% over established practices. The research by [24] presents a malicious email detection framework that uses deep ensemble learning to analyze all segments of an email, including the body, header, and attachments. The framework outperforms existing methods by analyzing the entire email instead of just a specific part. It uses multiple deep learning classifiers, each trained on a specific email segment, and achieves an AUC score of 0.993 and a TPR of 5% higher than human expert-based feature-based models. [25] proposes Phish Responder, a hybrid approach that combines machine learning and NLP to detect phishing and spam emails. This method uses LSTM for text-based datasets and MLP for numerical-based datasets and achieves an average accuracy of 99%, making it suitable for phishing and spam detection.

The use of deep learning algorithms has led to significant improvements in detecting spam emails, but these methods have certain limitations. These limitations include the requirement for a large amount of annotated data to train the models, difficulty generalizing the models to new data, and a slow training process. Moreover, deep learning algorithms may struggle with unstructured and highly variable email data.

2.2.3 Transformer-based models

To address the limitations of deep learning models, transformer-based models like BERT have been proposed, which better capture the context and relationships between words in an email. The personalized spam filtering approach described in [26] employs BERT and TensorFlow 2.0 to overcome these limitations. The Enron dataset was used to test the algorithm's effectiveness, and the evaluation metrics were accuracy and loss. The initial results indicate the promising potential for detecting ineffective emails using the proposed algorithm. In the study described in [27], a spam detection model is proposed that uses the BERT pre-trained model to improve classification accuracy. The model was trained on multiple datasets, including Enron, SpamAssassin, Ling-Spam, and SMS spam collection corpora, achieving high accuracy rates of 98.62%, 97.83%, 99.13%, and 99.28%, respectively. The results indicate that BERT outperforms other machine learning algorithms due to its ability to understand message context and make accurate classifications. The work by [28] presents an effective spam detection approach that uses pre-trained BERT and machine learning classifiers to classify ham or spam emails. The proposed model was tested on two public datasets, and the logistic regression algorithm achieved the best performance in both datasets, demonstrating the model's ability to detect spam emails efficiently. The study in [29] demonstrates the effectiveness of BERT in classifying spam emails. BERT outperforms baseline DNN models with BiLSTM layers and classic classifiers

like k-NN and Naive Bayes, achieving 98.67% accuracy and 98.66% F1 score. The superior performance is attributed to BERT's attention layers and contextual word embeddings, highlighting its persistence and robustness against unseen data. In [30], a universal spam detection model is introduced that uses BERT and transfer learning to classify ham or spam emails. The model was trained on four datasets (Enron, Spamassassin, Lingspam, Spamtext) combined to obtain a single model with an overall accuracy of 97% and an F1 score of 0.96.

The effectiveness of using BERT for spam email detection has been widely acknowledged due to its capability to model bidirectional contexts. However, it also has some limitations, such as neglecting the interdependence between masked positions and the gap between pre-training and fine-tuning. In order to overcome these limitations, this research proposes the use of XLNet, a more recent model than BERT that addresses these issues. XLNet employs a permutation-based training method that takes into account the relationships between all tokens in a sequence, which has been shown to outperform BERT in various tasks. Previous models for email spam detection have been developed based on individual datasets using various techniques, primarily focusing on achieving high accuracy on a single dataset. However, their performance may vary when applied to other datasets. This research addresses these limitations by integrating all datasets and training the model for better accuracy.

Chapter 3

An overview of Transformer and Transformer-based models

3.1 Transformer

The Transformer is a deep neural network architecture designed for NLP tasks, that was first introduced in a groundbreaking research paper [1] in 2017. It has become one of the most popular and influential architectures in NLP field, achieving SOTA results in tasks like language modeling, machine translation, sentiment analysis, text classification, and more.

The Transformer is a network architecture that focuses exclusively on attention mechanisms, without using recurrence or convolutions. It uses self-attention, also known as intra-attention, to relate various positions within a sequence and produce a representation of that sequence. In addition, the Transformer applies positional encoding, which includes information about the positions of tokens in the sequence, to distinguish between tokens in different positions. This is critical for accurately understanding natural language.

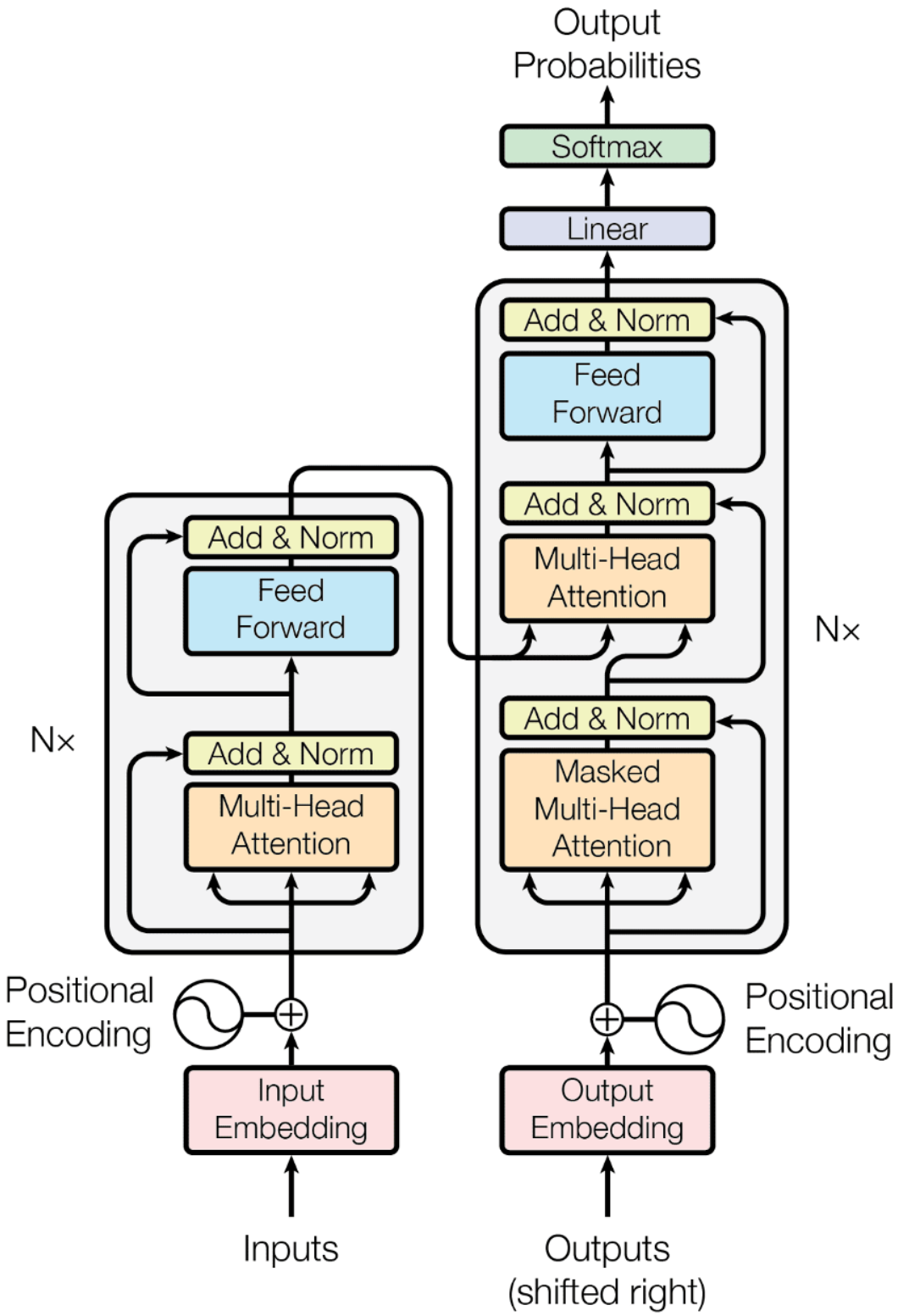

Figure 3-1: Transformer model architecture [1]

The Transformer model employs stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, as shown in Figure 3-1, with the encoder shown on the left and the decoder on the right.

3.2 Bidirectional Encoder Representations from Transformers (BERT)

BERT is a transformer-based neural network architecture that creates deep bidirectional representations of unlabeled text by considering both the left and right contexts. BERT's implementation consists of two main stages: pre-training, where it learns from unlabeled data, and fine-tuning, where the model is initialized with pre-trained parameters and then fine-tuned with labeled data for specific tasks.

BERT's model design is a multi-layer bidirectional transformer encoder based on Transformer. The model includes L layers (or Transformer blocks), with each block having a hidden size of H and A self-attention heads. BERT has two variations: BERT_{BASE}, which has 12 layers, a hidden size of 768, 12 self-attention heads, and a total of 110 million parameters, and BERT_{LARGE}, which has 24 layers, a hidden size of 1024, 16 self-attention heads, and a total of 340 million parameters.

3.2.1 Input/Output Representations

BERT is designed to manage various tasks by representing either a single sentence or a pair of sentences in its input. In BERT, a "sequence" can be a single sentence or two sentences merged together, represented using WordPiece embeddings[31] with a vocabulary of 30,000 tokens. The first token in every sequence, [CLS], is a special token and is used for classification tasks because its final hidden state represents the entire sequence. For sentence pairs, they are combined into one sequence, differenti-

ated by a special token ([SEP]) and a learned embedding showing if a token belongs to the first or second sentence.

3.2.2 Pre-training BERT

BERT is pre-trained without the use of conventional left-to-right or right-to-left language models. Instead, it is pre-trained using unsupervised tasks which are discussed below.

Task 1 - Masked Language Model(MLM): BERT's bidirectional representation is pre-trained using a masked language modeling (MLM) task, similar to a Cloze task [32]. In this process, a percentage of the input tokens (15% of WordPiece tokens in BERT's case) are randomly masked and then predicted. The output softmax is trained over the vocabulary using the final hidden vectors of the masked tokens, like in a regular language model. Unlike denoising auto-encoders [33], where the entire input is reconstructed, only the masked words are predicted in BERT's approach.

Task 2 - Next Sentence Prediction (NSP): NLP Tasks like Question Answering (QA) and Natural Language Inference (NLI) require an understanding of the relationship between two sentences, something that cannot be directly learned through language modeling. To tackle this, BERT is pre-trained on a task called the next sentence prediction (NSP) task, where it predicts if one sentence follows another in a text. This can be created from any monolingual corpus. While similar to previous works [34] and [35] that used representation learning, BERT's approach transfers all parameters to initialize the end-task model, unlike previous methods that only transferred sentence embeddings.

3.2.3 Fine-tuning BERT

Fine-tuning BERT for downstream tasks is simple because the self-attention mechanism in the Transformer can handle various tasks involving single or paired texts. To deal with text pairs, a typical method is to encode them separately and then perform bidirectional cross-attention. However, BERT encodes concatenated text pairs using self-attention, combining two stages into one. This effectively includes bidirectional cross-attention between the two sentences.

3.3 XLNet

XLNet is a generalized auto-regressive pre-training technique that incorporates concepts from Transformer-XL [7] and facilitates bidirectional context learning to overcome the limitations of BERT. XLNet is a type of pretraining method that is more advanced than BERT. It is able to learn bidirectional contexts by considering all possible orders of factorization, and it also uses an autoregressive approach to overcome BERT's limitations. In experiments, XLNet performs better than BERT on 20 various language tasks such as question answering, natural language inference, sentiment analysis, and document ranking, often by a significant margin.

XLNet's architecture builds on the Transformer-XL model, adding a segment-level recurrence mechanism and a new positional encoding scheme to manage longer sequences. In XLNet, the recurrence is replaced with a permutation-based approach that enables modeling of all possible permutations of the input sequence, allowing for the incorporation of bidirectional context into the model. The XLNet model proposes a permutation-based language modeling objective that combines the advantages of both Autoregressive (AR) language modeling and BERT. The objective enables the model to capture bidirectional context while retaining the benefits of AR models.

XLNet employs a two-stream self-attention mechanism, where one stream models

the sequential context in the standard left-to-right order, and the other stream models the context in a random permutation order. The final representation of each token is then obtained by integrating the information from both streams, allowing for the model to capture dependencies beyond the sequence order.

In terms of its architecture, XLNet has similar parameters to BERT, with two pre-trained variants: XLNet-Large (24 layers, 1024 hidden units, 16 self-attention heads, 340 million parameters) and XLNet-Base (12 layers, 768 hidden units, 12 self-attention heads,110 million parameters).

Chapter 4

Methodology

Figure 4-1 presents the architecture diagram of the proposed spam email detection mechanism, which consists of the following components:

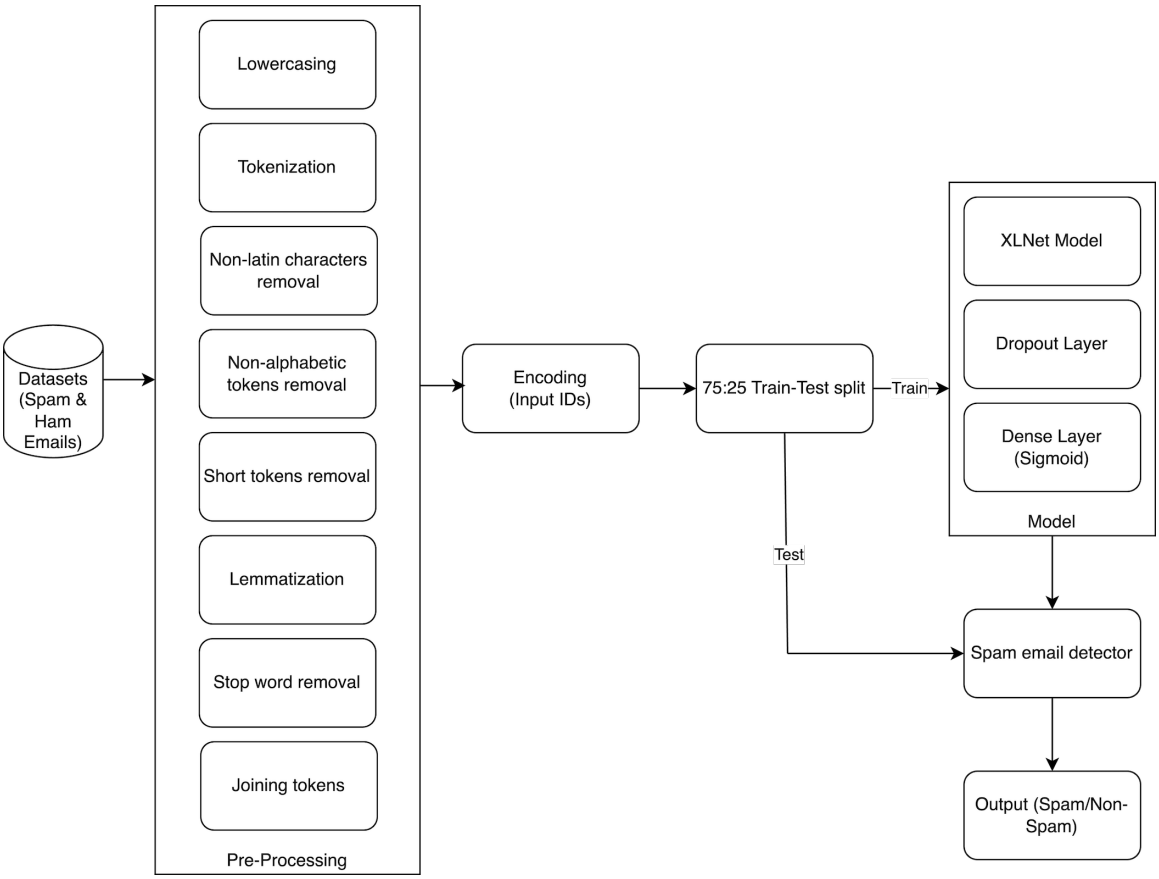

Figure 4-1: Proposed spam email detection mechanism architecture diagram

4.1 Datasets

This research used three publicly available datasets: SpamAssassin corpus [8], Enron corpus [9], and Ling-Spam corpus [10]. We tried to prepare good datasets of unique emails by checking for duplicate emails and removing them. The details of the datasets, including the total and unique number of spam and ham emails, are in given Table 4.1.

Table 4.1: Datasets Details

Dataset Name	**Total Spam**	**Total Ham**	**Total Email**	**Unique Spam**	**Unique Ham**	**Unique Email**
SpamAssassin	2397	6951	9348	1799	4322	6121
Enron	17156	16552	33708	14575	15910	30485
Ling-Spam	481	2412	2893	458	2401	2859

4.2 Data Pre-processing

First, we parsed the email files for performing further operations. One by one, we read each email file for the SpamAssassin dataset. We imported the textual data in a data frame together with their associated class and only extracted the email body, omitting the header. We extracted the whole text from every email file in the Enron dataset and added it to a data frame along with the class that it belongs to. We read the email content and class from the comma-separated values (CSV) file for the Ling-spam dataset and similarly loaded them in the data frames.

Pre-processing is essential for email classification because it prepares the raw text data for further processing. The pre-processing step involves a number of transformations that clean up the data and put it in a format suitable for modeling. The email text underwent the following pre-processing processes.

Lowercasing: To maintain consistency and reduce the dimensionality of the data, we normalized the text by making all text lowercase.

Tokenization: Using WordPunctTokenizer, we collected tokens from the normalized text. Tokenization is the process of dividing the text into sentences and then further into tokens.

Elimination of Non-Latin Characters All non-Latin characters were eliminated from the text using regex. Eliminating non-Latin letters helps to reduce data noise and enhances the accuracy of the findings.

Elimination of Non-Alphabetic Tokens: Using regex, we kept only alphabetic tokens and eliminated all other characters. It helped to get rid of extraneous characters like digits and symbols, etc.

Elimination of Short Tokens Using regex, we removed tokens shorter than three characters that did not add to the text's meaning.

Lemmatization: To convert words to their verb forms, the tokens were lemmatized using WordNetLemmatizer. For example, the word "reading" becomes "read" when lemmatized. Lemmatization reduces words to their most basic form, which lowers the dimensionality of the data and promotes consistency. This helps to improve the results' quality by deducting the impact of irrelevant variations in the words.

Eliminating Stop Words: We eliminated the stop words like "the," "is," "are," etc. Stop words are frequent words with minimal underlying actual meaning that do not add to the comprehension of a text. The dimensionality of the data is decreased and the quality of the results is improved by removing these terms.

Joining Tokens: To maintain the text's meaning and get the data ready for processing, the filtered tokens were then combined into a single string and separated by spaces.

4.3 Encoding:

The use of transformer models like XLNet requires input data to be in tokenized form with special tokens added to the original tokens. To achieve this, we utilized the encode_plus() function from the XLNetTokenizer in the Huggingface transformers[36] library. The encode_plus() function performs several operations on the input text to prepare it for use with an XLNet model, including:

Tokenization: The first step performed by the encode_plus() function is to tokenize the input text. Tokenization is the process of splitting the input text into single

words or subwords (tokens), which are the basic units of text processing. The XLNet-Tokenizer uses a specific tokenization algorithm that is optimized for use with XLNet models.

Mapping to IDs: After the input text is tokenized, each token is mapped to its respective ID using the XLNetTokenizer's vocabulary. The vocabulary is a mapping between each unique token and a unique integer ID. This step is necessary because XLNet models can only process numerical data, so the input text must be converted into a sequence of integers that the model can understand.

Truncation: The length of the input text can vary from sample to sample, which can cause issues when feeding data into a neural network. To ensure that all input sequences have the same length, the encode_plus() function pads or truncates the token IDs to a fixed length. Padding involves adding special tokens to the input sequence to make it a fixed length, while truncation involves removing tokens from the input sequence to make it a fixed length.

Attention Mask: The final step performed by the encode_plus() function is to create an attention mask. An attention mask is a binary mask that indicates which tokens should be attended to by the XLNet model and which tokens should be ignored. The attention mask is necessary because XLNet models can have variable-length inputs due to padding, and the attention mask ensures that the model only attends to the relevant tokens.

The resulting output of the encode_plus() function is a dictionary that contains the encoded token IDs, attention masks, and other necessary information for processing the input with an XLNet model. By tokenizing, mapping to IDs, padding/truncating, and creating an attention mask, the function ensures that input data is properly formatted and ready for use with the XLNet model, which can lead to improved

accuracy and efficiency in natural language processing tasks. The encoded inputs can then be fed into the model for prediction or training. XLNet's permutation operation allows for bidirectional context, making it independent of data corruption and free from pretrain-finetune discrepancy, which makes it a better choice for spam email classification.

4.4 Proposed Model

In this thesis, we present a novel spam email detector model that takes natural language text as input and predicts whether it is spam (1) or ham (0). Our detector is built using a pre-trained XLNet model, which is a transformer-based deep learning model designed to capture the context-aware and permutation-based dependencies between words in a sentence. We chose XLNet over BERT due to its superior performance on 20 NLP tasks, including text classification.

XLNet-base-cased and XLNet-large-cased are two pre-trained models of XLNet. XLNet-base-cased is a smaller model with 12 layers, 768 hidden units, 12 self-attention heads, and 110 million parameters. In contrast, XLNet-large-cased is a larger model with 24 layers, 1024 hidden units, 16 self-attention heads, and 340 million parameters. We used the XLNet-base-cased model for our spam email detector.

The proposed model comprises several components, each of which plays a crucial role in accurately classifying the emails as spam or not. The model consists of the following components:

Input Layer: The first component is the input layer, which takes in a batch of documents as its input. Each document has a maximum length of 128 tokens, and each token is represented by an int32 data type. This layer is the entry point for the data, and it is responsible for processing it before it is fed into the subsequent layers of the model.

XLNet: The second component is the XLNet language model, which is pre-trained and loaded to encode the tokens in each document. This model generates a sequence of hidden states that represent the encoding of each token in the documents. The XLNet language model, a state-of-the-art neural network architecture, can recognize intricate patterns in data and create top-quality embeddings that convey the essence of the text.

Output Layer: The third component is the output layer, which uses the last hidden state (CLS) to represent the encoding of the entire document. The final hidden state is taken and sent through a dropout layer with a dropout rate of 0.1, aiding in preventing overfitting and enhancing the model's ability to generalize. Following the dropout layer, the output goes through a dense layer with a sigmoid activation function, resulting in a probability score that indicates whether the document is spam or not.

Model Compilation: The final component of the model is the model compilation process, which involves specifying the optimizer, loss function, and evaluation metrics. In this case, the Adam optimizer is used with a learning rate of 2e-5, and the binary cross-entropy loss function is used to measure the error between the predicted and actual values. The evaluation metrics used to measure the performance of the model include binary accuracy, precision, and recall. These metrics are commonly used in classification tasks to measure the model's effectiveness in identifying spam documents.

Simply, we can say that our model is designed to accurately classify emails as spam or not by using the XLNet language model to encode the input documents and a simple yet powerful feedforward neural network, dropout regularization, and carefully selected optimization and evaluation techniques. Figure 4-2 shows our proposed spam detector model, and Table 4.2 presents a summary of the proposed spam detector

model, including the layer type, output shape, the number of layers, and the number of parameters.

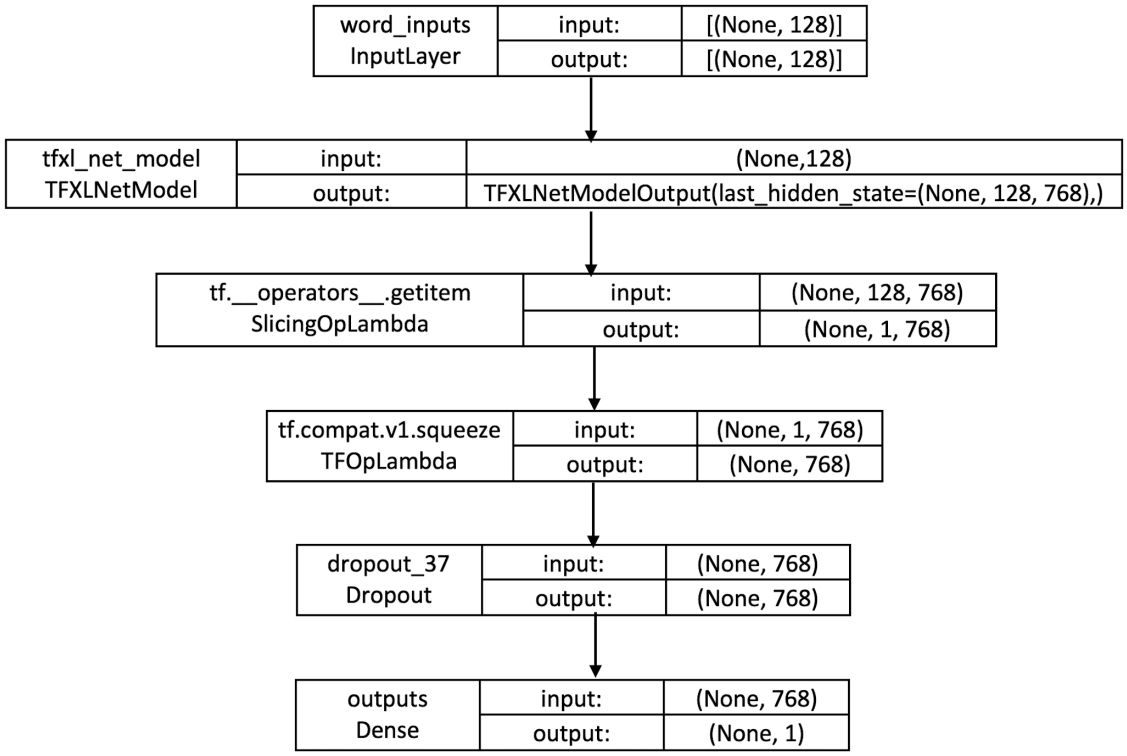

Figure 4-2: Proposed spam detector model

4.5 Hyperparameters

Our primary objective was to fine-tune the XLNet model to achieve consistently high performance across a variety of datasets. To accomplish this, we trained the model on an integrated dataset using these various hyperparameter combinations. Table 4.3 presents the optimal hyperparameter values that yielded the best performance. Our hyperparameter selection process comprises the following steps:

Table 4.2: Model Summary

Layer (type)	**Output Shape**	**Param #**
word_inputs (InputLayer)	[(None, 128)]	0
tfxl_net_model (TFXLNetModel)	TFXLNetModelOutput(last_hidden_state= (None, 128,768), mems=((128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768), (128, None, 768)), hidden_states=None, attentions=None)	116718336
tf.__operators__.getitem (SlicingOpLambda)	(None, 1, 768)	0
tf.compat.v1.squeeze (TFOpLambda)	(None, 768)	0
dropout_37 (Dropout)	(None, 768)	0
outputs (Dense)	(None, 1)	0

Data preparation: We merged three distinct datasets - SpamAssassin, Enron, and Ling-Spam - to create a combined dataset, guaranteeing a broad representation of both spam and ham emails.

Hyperparameter identification: We selected crucial hyperparameters for the XLNet model, including learning rate, dropout rate, maximum sequence length, and output activation function, among others. The selection of these hyperparameters was based on their potential influence on the model's performance and ability to generalize.

Hyperparameter exploration: We conducted manual experiments with a wide range of hyperparameter value combinations, training, and evaluating the model on the combined dataset for each specific combination. Performance metrics, such as binary accuracy, precision, recall, F1-score, and AUC, were documented for each experiment.

Analysis and selection: Following a thorough examination of the performance metrics for the various hyperparameter combinations, we determined the most effective set of hyperparameter values that led to the highest overall performance across the aggregated dataset.

By fine-tuning the XLNet model with these optimal hyperparameter values, we consistently achieved remarkable performance, proving the model's efficiency in detecting spam emails.

Table 4.3: Hyperparameters used for finetuning

Hyperparameter	**Value**
Model Architecture	XLNet-base-cased
Tokenizer	XLNetTokenizer
Maximum Sequence Length	128
Learning Rate	2e-5
Loss Function	Binary Crossentropy
Metrics	Binary Accuracy, Precision, Recall
Dropout Rate	0.1
Output Activation Function	Sigmoid
Epochs	4
Batchsize	32

4.6 Evaluation Metrics

To evaluate the performance of our spam email detection model, we used a set of evaluation metrics, also known as performance metrics. These metrics can help us determine how well the model is performing and identify areas where it may need improvement. One of the best techniques for calculating the performance of a classifier model is by creating a confusion matrix. A confusion matrix is a table that is used to compare the predicted values of the model with the actual values to evaluate the performance of the model. In the context of a spam email detection model, the confusion matrix would be used to compare the predicted labels of each email (i.e., whether the email is spam or not) with the actual labels (i.e., whether the email is actually spam or not).

Table 4.4: Confusion Matrix

	Predicted Ham (0)	**Predicted Spam (1)**
True Ham (0)	True Negative	False Positive
True Spam (1)	False Negative	True Positive

Table 4.4 shows the confusion matrix. The confusion matrix consists of four terms: True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN).

- True Positives (TP) represent the number of emails that were correctly classified as spam by the model.
- False Positives (FP) represent the number of emails that were incorrectly classified as spam by the model.
- True Negatives (TN) represent the number of emails that were correctly classified as not spam by the model.
- False Negatives (FN) represent the number of emails that were incorrectly classified as not spam by the model.

Using these four terms, we can calculate various evaluation metrics that provide different insights into the performance of the model such as True Positive Rate (TPR) or recall, True Negative Rate (TNR), False Positive Rate (FPR), False Negative Rate (FNR), accuracy, precision, F1 score, and area under the receiver operating characteristic (ROC) curve (AUC).

4.6.1 True Positive Rate (TPR)

TPR, also known as sensitivity, recall, or hit rate, quantifies the proportion of actual positive cases (spam emails) that the model accurately classifies as positive

(spam). It is calculated by dividing the number of true positive predictions by the total of true positive and false negative predictions. A high TPR value signifies that the model is effectively identifying a large proportion of spam emails.

$$TPR = \frac{TP}{TP + FN} \tag{4.1}$$

4.6.2 True Negative Rate (TNR)

TNR, also referred to as specificity, evaluates the proportion of actual negative cases (ham emails) that the model accurately classifies as negative (ham). It is calculated by dividing the true negative instances by the total number of actual negative instances. A high TNR value means that the model is effectively identifying a large percentage of ham emails.

$$TNR = \frac{TN}{TN + FP} \tag{4.2}$$

4.6.3 False Positive Rate (FPR)

FPR, also known as the false alarm rate, quantifies the percentage of emails wrongly identified as spam when they are actually ham. FPR is calculated by dividing the false positives by the total number of real ham emails. A low FPR value is desirable, as it shows that the model is not mistakenly classifying ham emails as spam.

$$FPR = \frac{FP}{FP + TN} \tag{4.3}$$

4.6.4 False Negative Rate (FNR)

False Negative Rate (FNR), or miss rate, gauges the proportion of real spam emails that the model wrongly classifies as non-spam (ham). It's determined by dividing

the false negative predictions by the total of false negatives and true positives. A low FNR value is preferable, as it shows that the model is effectively avoiding the incorrect classification of spam emails as ham.

$$FNR = \frac{FN}{FN + TP} \tag{4.4}$$

4.6.5 Accuracy

Accuracy represents the proportion of correct predictions made by the model out of all the predictions, reflecting the model's overall ability to correctly classify both spam and ham emails. It's determined by dividing the sum of true positive and true negative predictions by the total number of predictions. A higher accuracy value signifies a more effective and better-performing model.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.5}$$

4.6.6 Precision

Precision is calculated as the ratio of true positive predictions to the combined total of true positive and false positive predictions. It evaluates how accurately the model classifies spam emails. A higher precision value means that the model is effectively identifying a larger percentage of spam emails.

$$Precision = \frac{TP}{TP + FP} \tag{4.6}$$

4.6.7 F1 Score

The F1 score is the harmonic mean of precision and recall, acting as a combined measure of the model's accuracy that balances both precision and recall in its calcula-

tion. A higher F1 score indicates that the model is performing well in both precision and recall, indicating a better model performance

$$f1score = \frac{2 * (precision * recall)}{precision + recall} \tag{4.7}$$

4.6.8 Area under the receiver operating characteristic (ROC) curve (AUC)

AUC evaluates the model's capability to differentiate between positive (spam) and negative (ham) emails. It involves plotting the true positive rate (TPR) against the false positive rate (FPR) across different classification thresholds. AUC ranges between 0.5 and 1, with a value closer to 1 indicating a better-performing model. An AUC of 0.5 indicates a random classifier that cannot distinguish between positive and negative emails.

Chapter 5

Results

This study used a split of 75:25 for each dataset, where 75% of the data was used to train the model, and the remaining 25% was used for testing. The model was trained for four epochs using a batch size of 32. The study used confusion matrices and performance metrics such as accuracy, precision, f1 score, and AUC to evaluate the model's performance on different datasets. The confusion matrices and performance metrics, including accuracy, precision, f1 score, and AUC for different datasets are displayed below. The support column in the table refers to the number of email samples in each category, such as spam or non-spam (ham), in the testing dataset. This column helps in determining the number of samples used to calculate the performance metrics. The overall performance of the model can be evaluated using two types of averages: macro average and weighted average. The macro average is calculated by taking the average of all the classes without considering their individual support values. On the other hand, the weighted average is computed by considering the support value for each class while calculating the average. In this case, the performance of each class is weighted by its proportion to the total number of samples. The weighted average gives more weight to the classes with more samples, making it a better measure of overall performance for imbalanced datasets.

5.1 Datasets

5.1.1 SpamAssassin

Our model achieves an accuracy of 0.9869 on the SpamAssassin dataset, which means that it correctly identifies 98.69% of the emails as either spam or ham. It also achieves an AUC of 0.9817. Figure 5-1 is the confusion matrix, and table 5.1 is the performance metrics for the SpamAssassin dataset. Figure 5-1 shows that the model correctly classified 1075 ham emails out of 1081 ham emails (true negatives) and 436 spam emails out of 450 spam emails (true positives). However, the model also misclassified 6 ham emails as spam (false positives) and 14 spam emails as ham (false negatives).

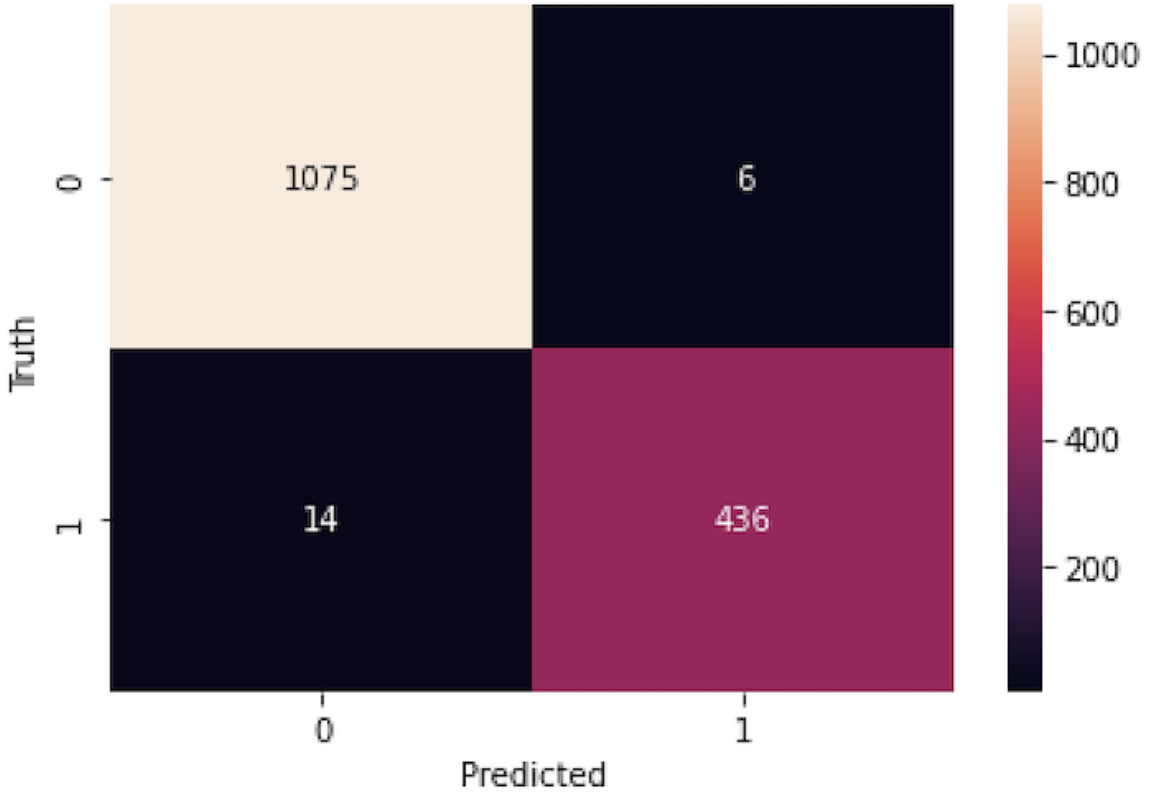

Figure 5-1: Confusion matrix for SpamAssassin

Table 5.1: Performance metrics for SpamAssassin

	Precision	**Recall**	**F1 score**	**Support**
Ham (0)	0.9871	0.9944	0.9908	1081
Spam (1)	0.9864	0.9689	0.9776	450
Macro Average	0.9868	0.9817	0.9842	1531
Weighted Average	0.9869	0.9869	0.9869	1531

5.1.2 Enron

Our model achieves an accuracy of 0.9892 on the Enron dataset, which means that it correctly identifies 98.92% of the emails as either spam or ham. It also achieves an AUC of 0.9893. Figure 5-2 is the confusion matrix, and table 5.2 is the performance metrics for the Enron dataset. Figure 5-2 shows that the model correctly classified 3927 ham emails out of 3978 ham emails (true negatives) and 3613 spam emails out of 3644 spam emails (true positives). However, the model also misclassified 51 ham emails as spam (false positives) and 31 spam emails as ham (false negatives).

Table 5.2: Performance metrics for Enron

	Precision	**Recall**	**F1 score**	**Support**
Ham (0)	0.9922	0.9872	0.9897	3978
Spam (1)	0.9861	0.9915	0.9888	3644
Macro Average	0.9891	0.9893	0.9892	7622
Weighted Average	0.9893	0.9892	0.9892	7622

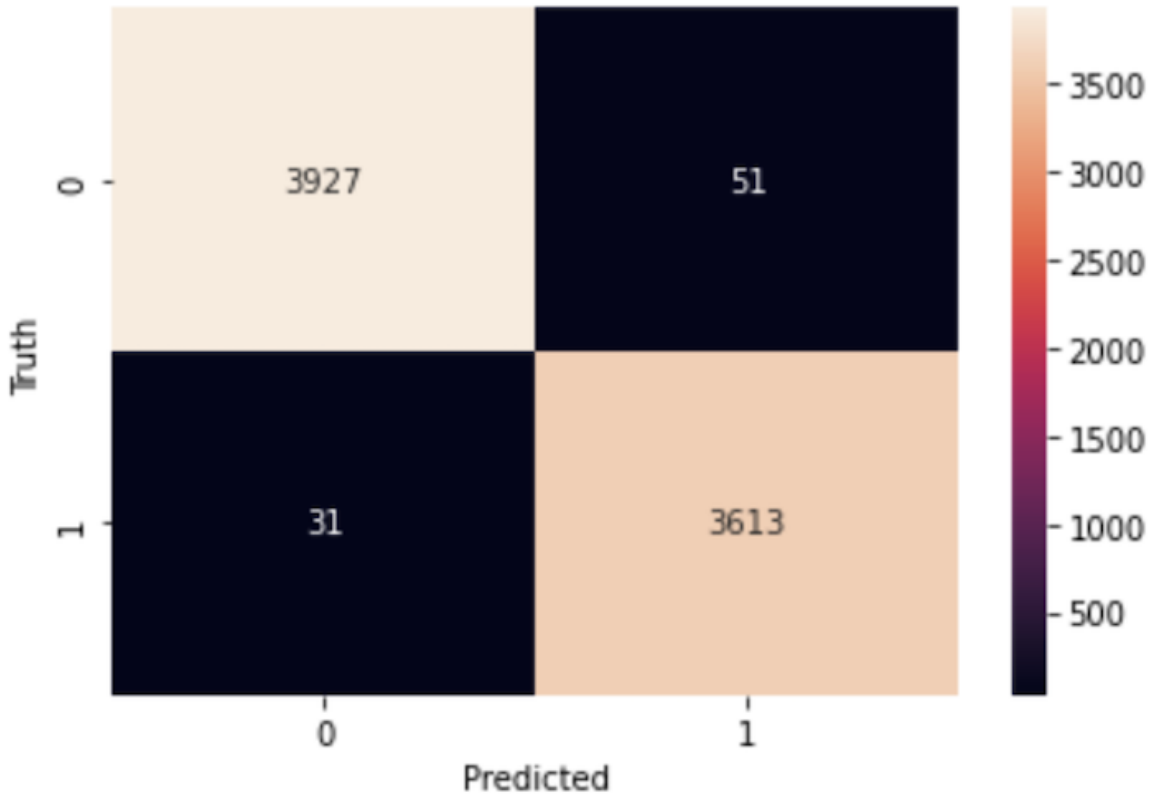

Figure 5-2: Confusion matrix for Enron

5.1.3 Ling-Spam

Our model achieves an accuracy of 0.9944 on the Enron dataset, which means that it correctly identifies 99.44% of the emails as either spam or ham. It also achieves an AUC of 0.9967. Figure 5-3 is the confusion matrix, and table 5.3 is the performance metrics for the Ling-Spam dataset. Figure 5-3 shows that the model correctly classified 596 ham emails out of 600 ham emails (true negatives) and all 115 spam emails correctly as spam emails (true positives). However, the model also misclassified 4 ham emails as spam (false positives).

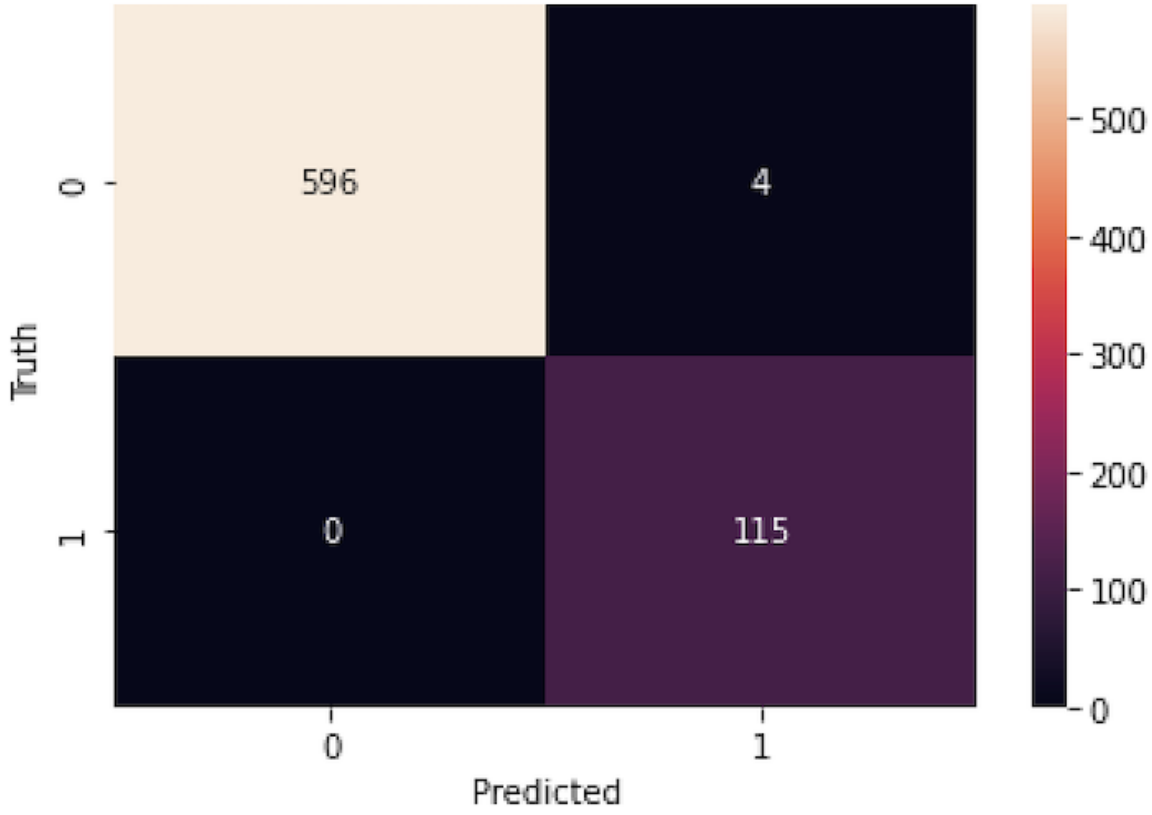

Figure 5-3: Confusion matrix for Ling-Spam

Table 5.3: Performance metrics for Ling-Spam

	Precision	**Recall**	**F1 score**	**Support**
Ham (0)	1.0000	0.9933	0.9967	600
Spam (1)	0.9664	1.0000	0.9829	115
Macro Average	0.9832	0.9967	0.9898	715
Weighted Average	0.9946	0.9944	0.9944	715

5.1.4 Combined

Our model achieves an accuracy of 0.9888 on the combined dataset, which means that it correctly identifies 98.88% of the emails as either spam or ham. It also achieves an AUC of 0.9889. Figure 5-4 is the confusion matrix, and table 5.4 is the performance metrics for the combined dataset. Figure 5-4 shows that the model correctly classified 5588 ham emails out of 5659 ham emails (true negatives) and 4168 spam emails out of 4208 spam emails (true positives). However, the model also misclassified 71 ham emails as spam (false positives) and 40 spam emails as ham (false negatives).

Table 5.4: Performance metrics for combined dataset

Class	**Precision**	**Recall**	**F1 score**	**Support**
Ham (0)	0.9929	0.9875	0.9902	5659
Spam (1)	0.9833	0.9905	0.9869	4208
Macro Average	0.9881	0.9890	0.9885	9867
Weighted Average	0.9888	0.9888	0.9888	9867

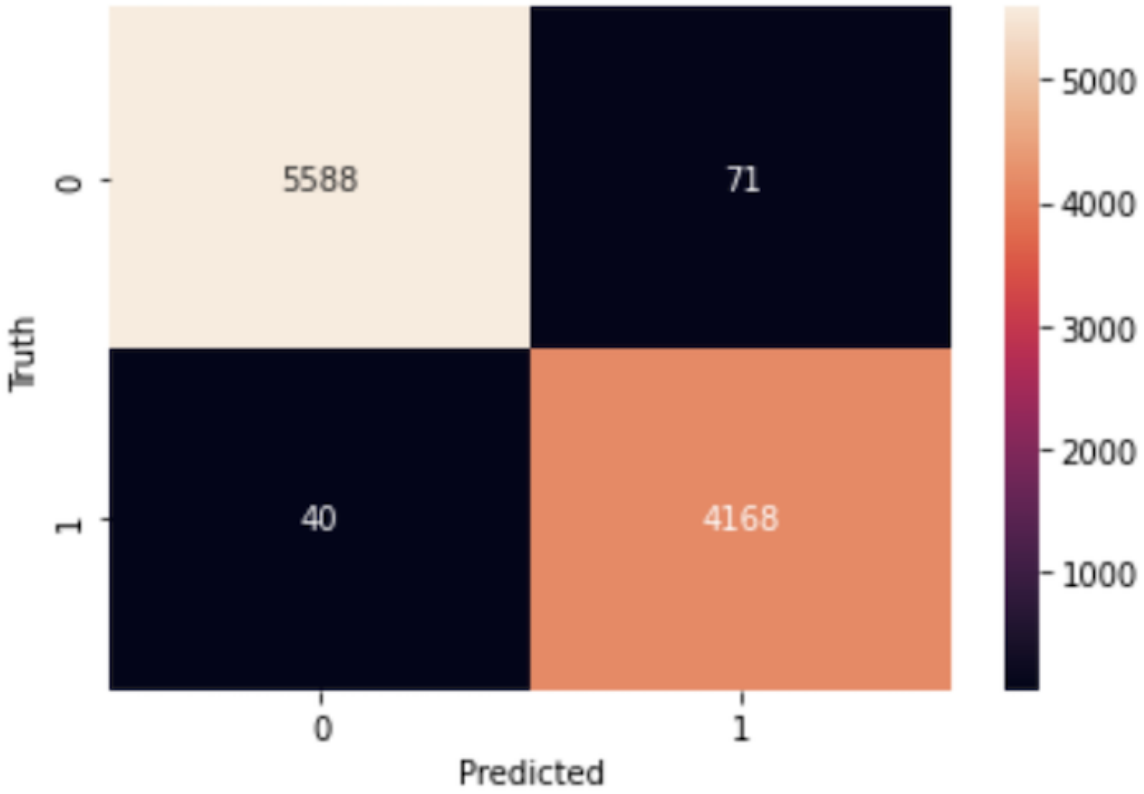

Figure 5-4: Confusion matrix for Combined

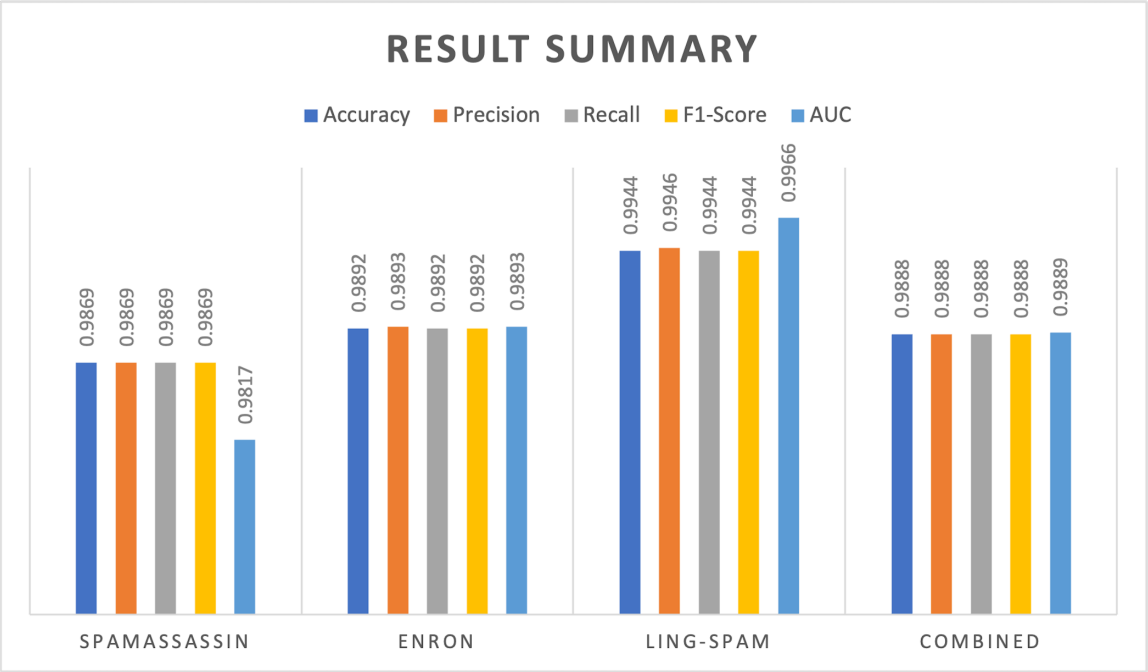

Figure 5-5: Result Summary

5.2 Result Summary

Figure 5-5 presents the summary of our model's performance metrics that show our model has consistent performance across different datasets, indicating its robustness and reliability.

5.3 Result Comparison

The model's performance has been compared with SOTA methods that use the same datasets as ours using various metrics such as accuracy, precision, recall, f1 score, and AUC as shown in table 5.5. The comparison is further illustrated through figures 5-6, 5-7, 5-8, 5-9, and 5-10. The following section provides brief descriptions of various research papers that have been compared and contrasted with our study. These comparisons serve to highlight the similarities and differences between our work and existing literature and help to describe the distinctive contributions our research brings to the field and why our method is better or at least comparable to them.

The article [37] proposes an email spam detection technique that is based on FT+HAN (Feature-based Transformer + Hierarchical Attention Network) combined with convolutional neural networks, gated recurrent units, and attention mechanisms. The model leverages the hierarchical structure of emails by applying attention techniques at both the sentence and word levels. CNNs are employed to extract meaningful features from the email content, and GRUs are used to capture temporal dependencies. Attention mechanisms enable the model to focus selectively on relevant parts of the email text during training. This study conducted experiments using five commonly used datasets: TREC 2007, GenSpam, SpamAssassin, Enron, and Ling Spam.

The study [28] proposes a proficient spam detection approach using a pre-trained BERT model combined with machine learning classifiers. The BERT model is used to represent email texts, and the features extracted from BERT outputs are utilized to classify emails as ham or spam. Four classifier algorithms (SVM, KNN, RF, and logistic regression) are employed to classify the text features into their respective categories. Two public datasets: Enron-Spam dataset, and the spam or not spam dataset are used for training and testing the proposed model. The evaluation metrics

show that the logistic regression algorithm provides the best classification performance across both datasets. The study emphasizes the proposed model's efficiency in identifying spam emails, demonstrating the effectiveness of utilizing the BERT model and supervised learning classifiers for spam detection.

The study [38] introduces a manifold learning-based method for time-efficient spam filtering. This model utilizes the Laplace feature map algorithm to extract geometric information from email text datasets and to extract crucial features. These features are then used as input for a Support Vector Machine (SVM) classifier for spam filtering. The study conducted extensive experiments on three datasets: EnronSpam, PU1, and GenSpam. The proposed model addresses the challenge of spam filtering by applying manifold learning algorithms to extract useful features from email text datasets. The study recognizes the complex spatial structures and high-degree nonlinearity present in text datasets, making traditional methods like Principal Component Analysis (PCA) ineffective.

The paper [39] proposes a spam email detection model that combines a multi-head approach with a Convolutional Neural Network (CNN) and Bidirectional Gated Recurrent Unit(GRU) network and incorporates sub-wording an context-based methods. The evaluation of the model is conducted using two datasets: Lingspam and Spamtext. Furthermore, to address the issue of catastrophic forgetting, a combination of the two datasets is used. The evaluation of the model's performance is based on metrics such as f1_score, accuracy, and the receiver operator characteristic (ROC) curve. The performance of the proposed model is evaluated using metrics such as f1_score, accuracy, and the receiver operator characteristic (ROC) curve. The results demonstrate excellent performance, with the model achieving an accuracy of 99.75% on the Spamtext dataset and 99.79% on the Lingspam dataset.

The paper [40] focuses on identifying spam emails and differentiating them from legitimate/normal emails. The study employs four machine learning models (Logis-

tic Regression, XGBoost, Support Vector Machine, Random Forest) and two deep learning models (Word Embedding, LSTM) to classify the emails. The evaluation is conducted over four datasets, including the Trec spam Dataset-2007, Enron email dataset, PU dataset, and Lingspam dataset. The study also explores identifying important keywords that are frequently present in spam emails. The models are applied to baskets of emails to effectively detect and classify spam emails using both machine learning and deep learning approaches. The study finds that XGBoost performed best among the machine learning models, while the Word Embedding layer shows superior performance in deep learning. Additionally, LSTM achieves the best results on the "Basket" dataset and hybrid systems are more effective in creating a robust spam filter.

The paper [41] proposes a hybrid approach to spam filtering using the Neural Network model Paragraph Vector-Distributed Memory (PV-DM) as a more comprehensive alternative to the Bag-of-Words (BOW) representation. The method takes into account the global context of an email and the local context of its features, combining the predictions from each representation vector to make the final classification. The experimental evaluation is conducted on two datasets: the Enron spam dataset and the Ling spam corpus. The results demonstrate that the proposed approach outperforms both PV-DM and BOW methods in terms of email classification.

The research [27] proposes a spam detector that utilizes the BERT pre-trained model to classify emails and messages based on their context. The model is trained on multiple corpora including the Enron corpus, SpamAssassin corpus, Ling-Spam corpus, and SMS spam collection corpus. The performance of the spam detector is evaluated on each corpus, achieving accuracy rates of 98.62%, 97.83%, 99.13%, and 99.28% respectively. The high performance of the model is attributed to the use of BERT, which enables a better understanding of message context and improves spam classification compared to other machine learning algorithms.

The manuscript [30] presents a novel Universal Spam Detection Model (USDM) that utilizes the pre-trained Google BERT base uncased models for efficient real-time classification of ham or spam emails. The model is trained and evaluated on four datasets: Enron, Spamassain, Lingspam, and Spamtext message classification datasets. The USDM combines the hyperparameters from individual models trained on each dataset and achieves an overall accuracy of 97% with an F1 score of 0.96. The USDM outperforms previous models trained on individual datasets, addressing the issue of performance variation across different datasets. The model demonstrates promising results and can be helpful in real-time spam classification scenarios.

Based on the comparison results, it can be inferred that our proposed model outperforms or is at least comparable to other methods in accuracy, precision, recall, F1 score, and AUC. The better performance of our model in spam email detection is attributed mainly to the use of XLNet. Below are some insights on why our model performs better than others.

XLNet's bidirectional context: XLNet leverages the Transformer architecture with a bidirectional context mechanism which enables it to capture dependencies between words in both forward and backward directions, allowing it to understand the context more effectively. This helps in spam detection tasks as it can capture complex patterns and relationships in the text data to understand the context.

Pre-training on large corpus: The XLNet model is pre-trained on a large corpus of text data, which helps it to learn general language representations and capture common patterns in text. This pre-training enables the model to better understand the underlying language semantics, improving its ability to classify them accurately.

Fine-tuning on spam detection task: The pre-trained XLNet model is fine-tuned for the specific task of spam detection using labeled emails of three datasets:

SpamAssassin, Enron, and Ling-Spam. Fine-tuning enables the model to adjust its pre-trained knowledge to match the unique characteristics and patterns of spam messages, thereby improving its performance for this specific task.

Attention mechanism: The XLNet model utilizes a self-attention mechanism that assigns different weights to different words of the input sequence. This attention mechanism helps the model focus on the most relevant information for spam detection, effectively ignoring irrelevant or noisy features. It allows the model to prioritize important words or phrases that indicate the presence of spam.

Large-scale architecture: XLNet model that we are using has a large-scale architecture with 12 layers, 768 hidden units, 12 attention heads, and 110 million parameters allowing it to capture complex hierarchical structures and dependencies in the text. This architecture enables the model to learn and represent more intricate patterns and relationships, leading to better performance in spam detection.

Table 5.5: A comparative study of performance metrics of the proposed mechanism with SOTA methods

Paper	method	Dataset	Accuracy	Precision	Recall	F1 Score	AUC
[37]	CNN + GRU + FT + HAN	SpamAssassin	0.955	0.893	0.978	0.933	0.987
		Enron	0.958	0.981	0.937	0.958	0.989
		Ling-Spam	0.980	0.933	0.948	0.940	0.997
[28]	BERT + SVM	Enron	-	0.9772	0.9769	0.9770	0.9964
	BERT + Logistic Regression		-	0.9786	0.9783	0.9784	0.9971
	BERT + KNN		-	0.9654	0.9637	0.964	0.9905
	BERT + Random Forest		-	0.9639	0.9634	0.9635	0.9946
[38]	SVM + LEP + Struc	Enron	0.947	-	-	-	
[39]	Multi-head CNN-BiGRU Network	Ling-spam (Unbalanced)	0.9979	0.9894	0.9947	0.9921	0.992
		Ling-spam (balanced)	0.9913	0.9920	0.9905	0.9912	0.989
[40]	Logistic Regression	Enron	0.5260	-	-	-	-
	Support Vector Machine		0.8472	-	-	-	-
	XGBoost		0.8846	-	-	-	-
	Random Forest		0.9228	-	-	-	-
	WordEmbeddings		0.9833	-	-	-	-
	WordEmbeddings + LSTM		0.9822	-	-	-	-
	Logistic Regression	Ling-Spam	0.8347	-	-	-	-
	Support Vector Machine		0.9036	-	-	-	-
	XGBoost		0.9589	-	-	-	-
	Random Forest		0.9606	-	-	-	-
	WordEmbeddings		1	-	-	-	-
	WordEmbeddings + LSTM		0.9994	-	-	-	-
[41]	Logistic Regression	Enron	0.9588	0.9644	0.9510	0.9576	0.99
	SVM		0.9616	0.9655	0.9559	0.9607	-
	KNN		0.9307	0.9435	0.9135	0.9283	-
	Logistic Regression	Ling-Spam	0.9827	0.9797	1	0.9897	1
	SVM		0.9827	0.9797	1	0.9897	-
	KNN		0.9827	0.9797	1	0.9897	-
[27]	BERT	Enron	-	0.9862	0.9862	0.9862	0.9862
		SpamAssassin	-	-	-	0.9783	-
		Ling-Spam	-	-	-	0.9913	-
[30]	BERT	SpamAssassin	0.98	0.96	0.99	0.9764	-
		Enron	0.97	0.96	0.98	0.9720	-
		LingSpam	0.98	0.90	0.98	0.9400	-
		Combined	0.97	0.95	0.97	0.9608	-
Our Work	XLNet	SpamAssassin	0.9869	0.9869	0.9869	0.9869	0.9893
		Enron	0.9892	0.9893	0.9892	0.9892	0.9893
		LingSpam	0.9944	0.9946	0.9944	0.9944	0.9967
		Combined	0.9888	0.9888	0.9888	0.9888	0.9889

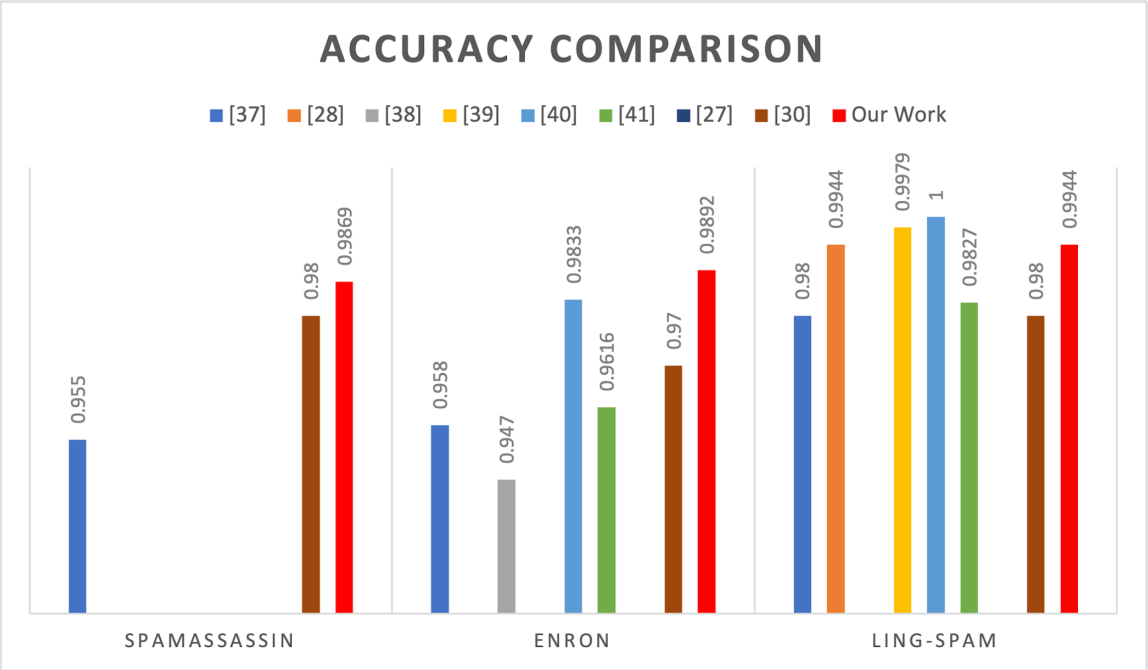

Figure 5-6: Accuracy Comparison

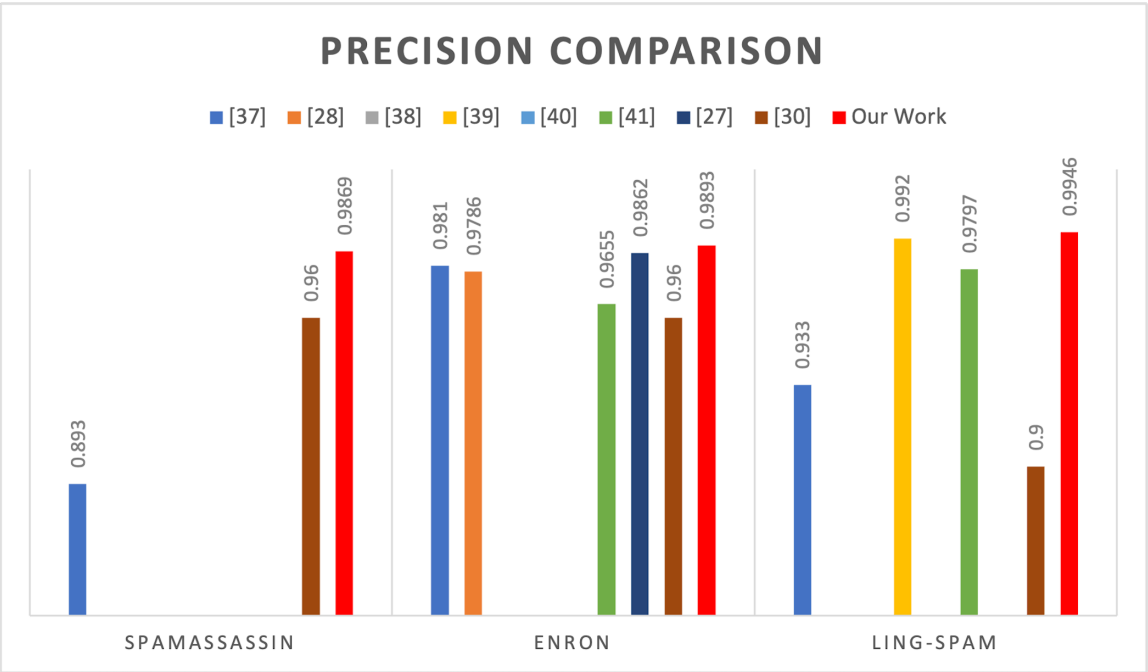

Figure 5-7: Precision Comparison

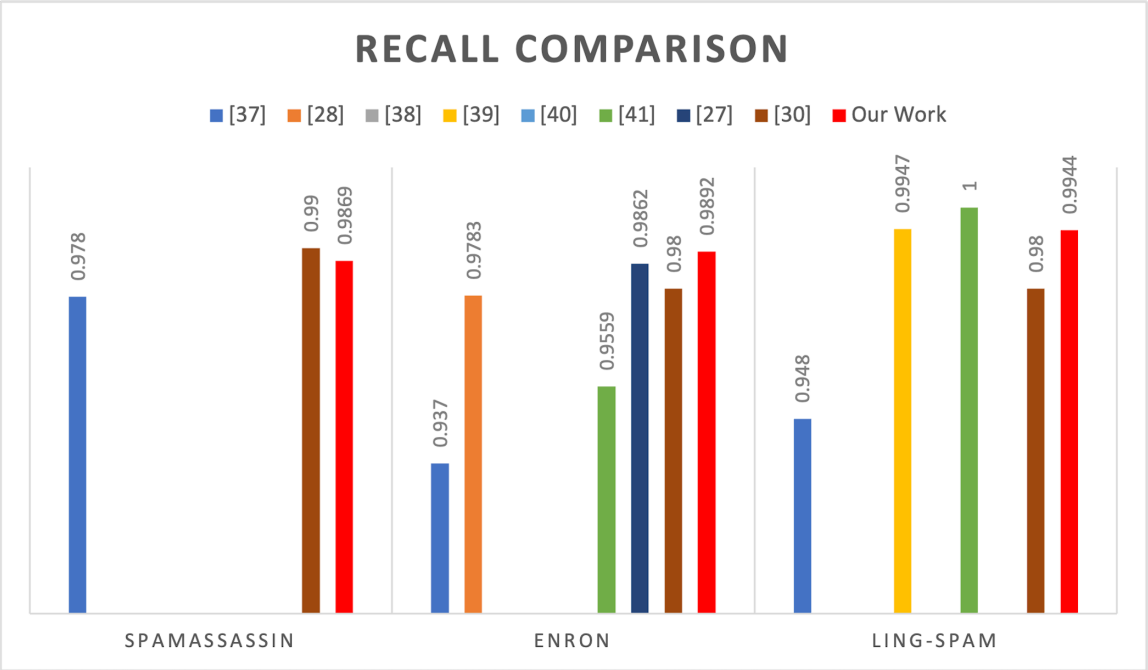

Figure 5-8: Recall Comparison

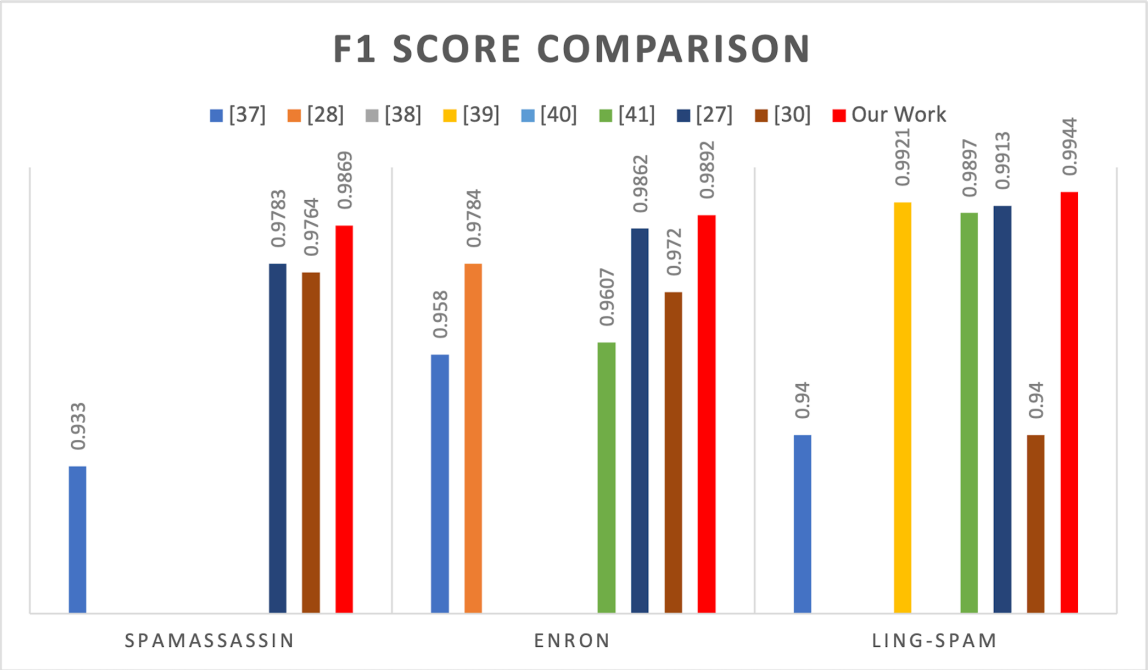

Figure 5-9: F1 Score Comparison

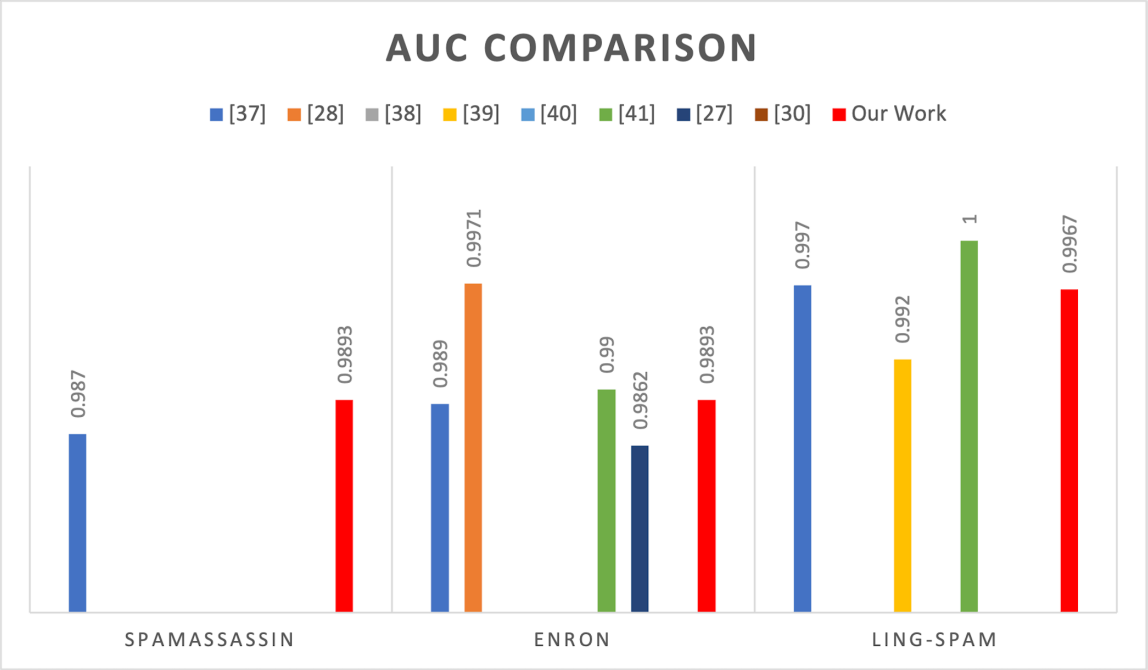

Figure 5-10: AUC Comparison

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis proposes a new spam detection model based on the XLNet model. The study aimed to develop a reliable and effective spam detection model that could accurately detect spam emails and protect users and companies from malicious activities and resource loss. To achieve this, the study utilized three publicly available datasets, namely SpamAssassin, Enron, and Ling-Spam. The datasets were parsed, preprocessed, and split into training and testing sets. The study then used the XLNet-based model, a state-of-the-art language model, and fine-tuned it with the training dataset. The trained model was then used to classify the testing dataset into spam or ham emails.

To evaluate the performance of the proposed model, the study used various evaluation metrics, including accuracy, precision, recall, f1 score, and AUC. The results showed that the proposed model achieved high accuracy, AUC, and F1 scores on all datasets, demonstrating its effectiveness in detecting spam emails accurately. On the SpamAssassin dataset, the proposed model achieved an accuracy of 0.9869, an AUC of 0.9817, and an F1 score of 0.9869. Similarly, on the Enron dataset, the model achieved an accuracy of 0.9892, an AUC of 0.9893, and an F1 score of 0.9892. On the

Ling-Spam dataset, the model achieved an accuracy of 0.9944, an AUC of 0.9967, and an F1 score of 0.9944. Finally, on the combined dataset, the proposed model achieved an accuracy of 0.9888, an AUC of 0.9889, and an F1 score of 0.9888. The study also compared the performance of the proposed model with SOTA models. The comparison showed that the proposed model outperformed or is at least comparable to the SOTA models, indicating its superiority in detecting spam emails. In conclusion, the study demonstrates the effectiveness of the proposed XLNet-based spam detection model in accurately detecting spam emails. The model's superior performance over existing models suggests its potential as a reliable and effective tool for detecting spam emails and protecting users and organizations from spam emails.

6.2 Future Work

As a future work, we can compute factors such as model complexity, training time, and ease of deployment considering the fact that these factors are also important while comparing our methods with others. To improve our model's performance, we can train our model with more data across a variety of datasets, especially the latest datasets with new types of spam emails. Furthermore, we could explore different pre-processing methods to improve the model's performance. Our current model is trained on email in the English language only. An interesting expansion could be to train the model on emails in different languages to increase its utility across the globe. Another exciting view could be training our model with all datasets except one, and seeing the prediction results on the remaining dataset. In this way, we can see the generalization power of the approach, i.e., its ability to correctly analyze totally unseen emails (emulating a zero-day phishing campaign). In addition to spam detection, we could also aim to categorize spam into different types, such as phishing emails. Our current project is only run from a script. So, developing a user-friendly

interface or a plugin for popular email clients can be a valuable addition. Finally, exploring how this model can be integrated with existing email systems (like Gmail or Outlook) to enhance their spam detection capabilities can be an interesting avenue to pursue.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] N. Cvetićanin, "What's on the other side of your inbox–20 spam statistics for 2021," 2021.

[3] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[4] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[6] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[7] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[8] A. S. Foundation, "Spamassassin," 2003. Last accessed 7 February 2023.

[9] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004. Proceedings 15*, pp. 217–226, Springer, 2004.

[10] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C. D. Spyropoulos, and P. Stamatopoulos, "A memory-based approach to anti-spam filtering for mailing lists," *Information retrieval*, vol. 6, pp. 49–73, 2003.

[11] W. Feng, J. Sun, L. Zhang, C. Cao, and Q. Yang, "A support vector machine based naive bayes algorithm for spam filtering," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8, IEEE, 2016.

[12] Y. Song, A. Kołcz, and C. L. Giles, "Better naive bayes classification for high-precision spam detection," *Software: Practice and Experience*, vol. 39, no. 11, pp. 1003–1024, 2009.

[13] Z. Yang, X. Nie, W. Xu, and J. Guo, "An approach to spam detection by naive bayes ensemble based on decision induction," in *Sixth international conference on intelligent systems design and applications*, vol. 2, pp. 861–866, IEEE, 2006.

[14] W. Peng, L. Huang, J. Jia, and E. Ingram, "Enhancing the naive bayes spam filter through intelligent text modification detection," in *2018 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*, pp. 849–854, IEEE, 2018.

[15] A. Chakraborty, U. K. Das, J. Sikder, M. Maimuna, and K. I. Sarek, "Content based email spam classifier as a web application using naïve bayes classi-

fier," in *Intelligent Computing & Optimization: Proceedings of the 5th International Conference on Intelligent Computing and Optimization 2022 (ICO2022)*, pp. 389–398, Springer, 2022.

[16] L. U. Oghenekaro and A. T. Benson, "Text categorization model based on linear support vector machine," *American Academic Scientific Research Journal for Engineering, Technology, and Sciences*, vol. 85, no. 1, 2022.

[17] Y. T. K. Reddy and S. S. Ahila, "Classification of spam emails using random forest algorithm in comparison with naive bayes algorithm," *Baltic Journal of Law & Politics*, vol. 15, no. 4, pp. 140–146, 2022.

[18] P. Charan and P. Sriramya, "Higher accuracy of spam email prediction using k-nearest neighbor algorithm comparing with multinomial naive bayes algorithm," *Baltic Journal of Law & Politics*, vol. 15, no. 4, pp. 277–286, 2022.

[19] E. G. Dada, J. S. Bassi, H. Chiroma, A. O. Adetunmbi, O. E. Ajibuwa, *et al.*, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, p. e01802, 2019.

[20] K. Debnath and N. Kar, "Email spam detection using deep learning approach," in *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*, vol. 1, pp. 37–41, IEEE, 2022.

[21] W. S. Jacob *et al.*, "Multi-objective genetic algorithm and cnn-based deep learning architectural scheme for effective spam detection," *International Journal of Intelligent Networks*, vol. 3, pp. 9–15, 2022.

[22] S. Magdy, Y. Abouelseoud, and M. Mikhail, "Efficient spam and phishing emails filtering based on deep learning," *Computer Networks*, vol. 206, p. 108826, 2022.

[23] S. A. Ghaleb, M. Mohamad, W. A. H. Ghanem, A. B. Nasser, M. Ghetas, A. M. Abdullahi, S. A. M. Saleh, H. Arshad, A. E. Omolara, and O. I. Abiodun, "Feature selection by multiobjective optimization: Application to spam detection system by neural networks and grasshopper optimization algorithm," *IEEE Access*, vol. 10, pp. 98475–98489, 2022.

[24] T. Muralidharan and N. Nissim, "Improving malicious email detection through novel designated deep-learning architectures utilizing entire email," *Neural Networks*, vol. 157, pp. 257–279, 2023.

[25] M. Dewis and T. Viana, "Phish responder: A hybrid machine learning approach to detect phishing and spam emails," *Applied System Innovation*, vol. 5, no. 4, p. 73, 2022.

[26] K. Iqbal, S. A Khan, S. Anisa, A. Tasneem, and N. Mohammad, "A preliminary study on personalized spam e-mail filtering using bidirectional encoder representations from transformers (bert) and tensorflow 2.0," *International Journal of Computing and Digital Systems*, vol. 11, no. 1, pp. 893–903, 2022.

[27] T. Sahmoud, D. Mikki, *et al.*, "Spam detection using bert," *arXiv preprint arXiv:2206.02443*, 2022.

[28] Y. Guo, Z. Mustafaoglu, and D. Koundal, "Spam detection using bidirectional transformers and machine learning classifier algorithms," *Journal of Computational and Cognitive Engineering*, 2022.

[29] Q. Yaseen *et al.*, "Spam email detection using deep learning techniques," *Procedia Computer Science*, vol. 184, pp. 853–858, 2021.

[30] V. S. Tida and S. Hsu, "Universal spam detection using transfer learning of bert model," *arXiv preprint arXiv:2202.03480*, 2022.

[31] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[32] W. L. Taylor, ""cloze procedure": A new tool for measuring readability," *Journalism quarterly*, vol. 30, no. 4, pp. 415–433, 1953.

[33] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[34] Y. Jernite, S. R. Bowman, and D. Sontag, "Discourse-based objectives for fast unsupervised sentence representation learning," *arXiv preprint arXiv:1705.00557*, 2017.

[35] L. Logeswaran and H. Lee, "An efficient framework for learning sentence representations," *arXiv preprint arXiv:1803.02893*, 2018.

[36] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[37] S. Zavrak and S. Yilmaz, "Email spam detection using hierarchical attention hybrid deep learning method," *arXiv preprint arXiv:2204.07390*, 2022.

[38] C. Wang, Q. Li, T.-y. Ren, X.-h. Wang, and G.-x. Guo, "High efficiency spam filtering: a manifold learning-based approach," *Mathematical problems in engineering*, vol. 2021, pp. 1–7, 2021.

[39] A. Gupta, J. Patil, S. Soni, and A. Rajan, "Email spam detection using multi-head cnn-bigru network," in *Advanced Network Technologies and Intelligent*

Computing: Second International Conference, ANTIC 2022, Varanasi, India, December 22–24, 2022, Proceedings, Part I, pp. 29–46, Springer, 2023.

[40] M. K. Islam, M. Al Amin, M. R. Islam, M. N. I. Mahbub, M. I. H. Showrov, and C. Kaushal, "Spam-detection with comparative analysis and spamming words extractions," in *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pp. 1–9, IEEE, 2021.

[41] S. Douzi, F. A. AlShahwan, M. Lemoudden, and B. El Ouahidi, "Hybrid email spam detection model using artificial intelligence," *International Journal of Machine Learning and Computing*, vol. 10, no. 2, 2020.

Appendix A

Python code for Spam Email Classification using XLNet

```
#install required packages
!pip install transformers
!pip install sentencepiece

#import necessary libraries
import tensorflow as tf
import pandas as pd
from pandas import DataFrame
import os
import sys
import numpy as np
import transformers
from transformers import TFXLNetModel, XLNetTokenizer
from tensorflow.keras.optimizers import Adam, SGD
import seaborn as sn
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import confusion_matrix,f1_score,roc_auc_score,
   roc_curve, classification_report,auc
from tensorflow.keras.utils import plot_model
from matplotlib import pyplot as plt
import re
import regex
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import WordPunctTokenizer
from string import punctuation
from nltk.stem import WordNetLemmatizer

#Download necessary resources from nltk library
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('stopwords')

#Define a list of stopwords and punctuations to be removed
cached_stop_words = stopwords.words("english")
wordnet_lemmatizer = WordNetLemmatizer()
for punct in punctuation:
    cached_stop_words.append(punct)

#Define the XLNet model and tokenizer
xlnet_model = 'xlnet-base-cased'
tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased',
```

```
    do_lower_case=True)

#Set seed value for reproducibility
seed_value= 1234
tf.random.set_seed(seed_value)

#Define constants
NEWLINE = '\n'
SKIP_FILES = {'cmds'}
HAM = 0
SPAM = 1

#Define the source directories containing the spam and ham emails
SOURCES = [
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/easy_ham',HAM
        ,1),
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/easy_ham 2',HAM
        ,1),
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/hard_ham',HAM
        ,1),
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/easy_ham_2',HAM
        ,1),
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/hard_ham 2',HAM
        ,1),
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/spam', SPAM,1),
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/spam 2',SPAM,1)
```

```
    ,
    ('/content/drive/MyDrive/spam/Dataset/Spamassassin/spam_2',SPAM,1)
    ,
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron1/ham',HAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron1/spam',SPAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron2/ham',HAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron2/spam',SPAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron3/ham',HAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron3/spam',SPAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron4/ham',HAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron4/spam',SPAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron5/ham',HAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron5/spam',SPAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron6/ham',HAM,2),
    ('/content/drive/MyDrive/spam/Dataset/Enron/enron6/spam',SPAM,2)

]



def read_files(path,source):
    """
    Recursively read files from a directory path and
    return a generator that yields tuples of file path
    and its content.
```

```
Args:
path (str): A string representing the path of the directory
to read files from.
source (int): An integer representing the source of the
file content.

Returns:
A generator that yields tuples of file path and its content.
"""
for root, dirs, files in os.walk(path):
    for dir_path in dirs:
        read_files(os.path.join(root, dir_path))
    for file_name in files:
        if file_name not in SKIP_FILES:
            file_path = os.path.join(root, file_name)
            if os.path.isfile(file_path):
                past_header, lines = False, []
                if source==2:
                    past_header = True
                f = open(file_path, encoding="latin-1")
                for line in f:
                    if past_header:
                        lines.append(line)
                    elif line == NEWLINE:
                        past_header = True
                f.close()
```

```
                content = NEWLINE.join(lines)
                yield file_path, content


def build_data_frame(l, path, classification,source):
    """
    Build a pandas dataframe from the file content and
    return the dataframe and the number of rows in the dataframe.

    Args:
    l (int): An integer representing the starting index of the rows
    to add to the dataframe.
    path (str): A string representing the path of the directory
    to read files from.
    classification (str): A string representing the classification
    of the files.
    source (int): An integer representing the source
    of the file content.

    Returns:
    A pandas dataframe containing the file content and classification
    and the number of rows in the dataframe.
    """
    rows = []
    index = []
    for i, (file_name, text) in enumerate(read_files(path,source)):
```

```
        rows.append({'text': text, 'class': classification})
        index.append(file_name)
    data_frame = DataFrame(rows, index=index)
    return data_frame, len(rows)

def load_data():
    """
    Load the data from multiple directories and return a pandas
    dataframe containing the file content and classification.

    Args:
    None

    Returns:
    A pandas dataframe containing the file content and classification.
    """
    data = DataFrame({'text': [], 'class': []})
    l = 0
    for path, classification, source in SOURCES:
        data_frame, nrows = build_data_frame(l, path, classification,
           source)
        data = data.append(data_frame)
        l += nrows
    data = data.reindex(np.random.permutation(data.index))
    return data
```

```
def filter_text(text):
    ''' Function to clean our data
    Clean the raw text by:
    Removing special characters, punctuations, pronouns, stopwords.
    Normalize the text by converting it into lower case.
    Extract the lemma for each word. Ex: Lemma(Reading) -> read.

    Args:
    text (str): A string representing the email text.

    Returns:
    a string of filtered text where the filtered words are joined
    together by a single space
    '''
    word_tokens = WordPunctTokenizer().tokenize(text.lower())
    filtered_text = [regex.sub(u'\p{^Latin}', u'', w) for w in
       word_tokens if w.isalpha() and len(w) > 3]
    filtered_text = [wordnet_lemmatizer.lemmatize(w, pos="v") for w in
        filtered_text if not w in cached_stop_words]
    return " ".join(filtered_text)

def encode(data,maximum_length) :
    """
    Encode each text data into a numerical format that
    can be used as input to a neural network model.
```

```
It uses the tokenizer object to encode each text data
into input_ids, and also generates attention_masks.

Args:
data (list): A list representing list of strings.
maximum_length (int): A integer value representing
maximum length of token

Returns:
A pandas dataframe containing the input_ids and attention_masks.
"""
input_ids = []
attention_masks = []
data=list(data)
for i in range(len(data)):
    encoded = tokenizer.encode_plus(
    data[i],
    add_special_tokens=True,
    max_length=maximum_length,
    truncation=True,
    padding='max_length',
    return_attention_mask=True,
  )
    input_ids.append(encoded['input_ids'])
    attention_masks.append(encoded['attention_mask'])
return np.array(input_ids),np.array(attention_masks)
```

```
#Load SpamAssassin and Enron data from files
data=load_data()

#Remove duplicate rows from spam assassin and enron dataset
data.drop_duplicates(keep="first", inplace=True)
data["text"] = data.text.apply(lambda x : filter_text(x))

#Load and preprocess Ling-Spam dataset
df= pd.read_csv("/content/drive/MyDrive/spam/Dataset/Ling-Spam Dataset
    .csv")
df['message'].replace('', np.nan, inplace=True)
df['label'].replace('', np.nan, inplace=True)
df.dropna(subset=['message'], inplace=True)
df.dropna(subset=['label'], inplace=True)
df.drop_duplicates(keep="first", inplace=True)
df["message"] = df.message.apply(lambda x : filter_text(x))
df.rename(columns={'message': 'text', 'label': 'class'}, inplace=True)

#Combine SapmAssassin, Enron and Ling-Spam dataframe
frames = [data, df]
result = pd.concat(frames)

#Split the data into training and testing sets with a 75:25 ratio
x_train, x_test, y_train, y_test = train_test_split(result['text'],
    result['class'], stratify=result['class'], random_state=1234)
```

```
#Encode both training and testing data
train_input_ids,train_attention_masks = encode(x_train,128)
test_input_ids,test_attention_masks = encode(x_test,128)

#Define an input layer for the word inputs
word_inputs = tf.keras.Input(shape=(128,), name='word_inputs', dtype='
   int32')

#Create a pre-trained XLNet model.
xlnet = TFXLNetModel.from_pretrained(xlnet_model)

#Use the XLNet model to encode the word inputs
xlnet_encodings = xlnet(word_inputs)[0]

#Collect last step from last hidden state (CLS)
doc_encoding = tf.squeeze(xlnet_encodings[:, -1:, :], axis=1)

#Apply a 10% dropout to the doc_encoding tensor
doc_encoding = tf.keras.layers.Dropout(.1)(doc_encoding)

#Add a dense layer with 1 output unit and sigmoid activation function
outputs = tf.keras.layers.Dense(1,
                                activation='sigmoid',
                                name='outputs')(doc_encoding)
```

```
#Define the evaluation metrics to be used during training
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]

#Define the model inputs and outputs, and create a model
model = tf.keras.Model(inputs=[word_inputs], outputs=[outputs])

#Compile the model with Adam optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-5),
   loss='binary_crossentropy', metrics=METRICS)

#Train the model with train_input_ids and y_train for 4 epochs
model.fit(train_input_ids, y_train, epochs=4)

#Use the trained model to make predictions on testing data
y_predicted = model.predict(test_input_ids)

#Flatten the y_predicted tensor to a 1D numpy array
y_predicted = y_predicted.flatten()

#Convert predicted probabilities to binary predictions
y_predicted = np.where(y_predicted > 0.5, 1, 0)
```

```
#Compute confusion matrix between y_test and y_predicted
cm = confusion_matrix(y_test, y_predicted)

#Plot the confusion matrix as a heatmap
sn.heatmap(cm, annot=True, fmt='d')

#Set the x-label and y-label for the confusion matrix plot
plt.xlabel('Predicted')
plt.ylabel('Truth')

#Print the model's performance classification report on the test set
print(classification_report(y_test, y_predicted,digits=4))

#Compute and print the area under the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_predicted)
print("AUC:")
print(auc(fpr, tpr))
```