ENHANCED SMS SPAM DETECTION WITH MODIFIED TRANSFORMER MODEL USING MULTI-HEAD-SELF-ATTENTION MECHANISM

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

KARNATI SURYA KIRAN REDDY(Reg. No.: 225003194)
SANGATI YASWANTH REDDY(Reg. No.: 225003175)
R UDAY KUMAR(Reg. No.: 225003180)

May 2024

Department of Computer Science and Engineering

SRINIVASA RAMANUJAN CENTRE

Kumbakonam, Tamil Nadu, INDIA - 612 001

Department of Computer Science and Engineering

SRINIVASA RAMANUJAN CENTRE

Kumbakonam, Tamil Nadu, INDIA - 612 001

May 2024

Bonafide Certificate

This is to certify that the report titled **"Enhanced SMS Spam Detection with Modified Transformer Model using Multi-Head-Self-Attention Mechanism"** submitted as *in partial fulfilment of the requirements for the award of the degree of B.Tech.,* Computer Science and Engineering to the SASTRA Deemed to be University, is a bonafide record of the work done by **Mr. Sangati Yaswanth Reddy (225003175), Mr. R Uday Kumar (225003180), Mr. Karnati Surya Kiran Reddy (225003194)** during the final semester of the academic year 2023-2024, in the Srinivasa Ramanujan Centre, under my supervision. This project report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor :

Name with Affiliation : **Smt. M Kamala Devi/AP-II/CSE/SRC/SASTRA**

Date :

Mini Project *Viva voce* held on :

Examiner 1 **Examiner 2**

Department of Computer Science and Engineering

SRINIVASA RAMANUJAN CENTRE

Kumbakonam - 612 001

Declaration

We declare that the project report titled **"Enhanced SMS Spam Detection with Modified Transformer Model using Multi-Head-Attention Mechanism"** was submitted by us in an original work done by us under the guidance of **Smt. M Kamala Devi**, **Assistant Professor-II, Department of CSE, SRC, SASTRA Deemed to be University, Kumbakonam,** during the sixth semester of the academic year 2023-24, in the **Srinivasa Ramanujan Centre.** The work is original and wherever we have used the materials from other sources, we have given due credit and cited them in the text of the project report. This project report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of the candidates :1.

2.

3.

Name of the candidates : 1. Sangati Yaswanth Reddy

2. R Uday Kumar

3. Karnati Surya Kiran Reddy

Date :

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam and Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli,** Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. Ramaswamy,** The Dean and **Dr. A. Alli Rani,** Associate Dean, Srinivasa Ramanujan Centre for their constant support and suggestions when required without any reservations.

We express my sincere gratitude to **Dr. V. Kalaichelvi,** Associate Professor, Department of Computer Science and Engineering for her unconditional support and encouragement for completing the project.

Our guide **Smt. M Kamala Devi,** Assistant Professor-II, Department of Computer Science & Engineering, Srinivasa Ramanujan Centre was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped me in making progress throughout our project work.

We also thank the Project Coordinator **Dr. S. Priyanga,** Assistant Professor, Department of Computer Science & Engineering, Srinivasa Ramanujan Centre for her ever-encouraging spirit and meticulous guidance for the completion of the project. We also thank the panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the Srinivasa Ramanujan Centre who have either directly or indirectly helped us in the completion of the project. We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me with an opportunity to showcase my skills through the project.

List of Figures

Figure No.	Title	Page No.
1.1	Architecture Diagram	6
1.2	Model Architecture	7
1.3	Solution Approach	7
4.1.1	Testing and Training data after Balancing	20
4.1.2	Original and Pre-processed Text Length	20
4.2.1	Model Summary	21
4.2.2	Comparison of Testing and Training accuracies	21
4.5	Comparison of Model accuracies	23
4.6.1	Twitter data Analysis	24
4.6.2	Tweet length for training dataset	24
4.6.3	Tweet length for testing dataset	25
4.7	Testing and Training accuracies for Twitter dataset	25
5.1.1	Result Interpretation for spam dataset	27
5.1.2	Result Interpretation for twitter dataset	27

Abbreviations

FNN	Feedforward Neural Networks
NLP	Natural Language Processing
TF-IDF	Term-Frequency Inverse Document Frequency
ReLU	Rectified Linear Activation Function
NB	Naïve Bayes
RF	Random Forest

Abstract

The Short Message Service (SMS) was one of primary communication channel over the past few years. However, its widespread use given rise to the SMS spam, marked by intrusive message that may pose risks like credential theft and data compromise. To address this persistent threat we propose implementing SMS spam detection model based on pre-trained Machine Learning and Deep Learning frameworks. “Modified Transformer Model” employs various techniques for text classification and keyword analysis, it explores the potential traditional Machine Learning, Deep Learning algorithms and modified Transformer model. The objective is to identify the most appropriate algorithm for discerning between spam and authentic text messages accurately. The solution not only enhances traditional spam filtering but also adapts to evolving spam strategies ensuring a robust defends against potential security threats in the realm of SMS communication.

Keywords: SMS Spam detection, Multi-Head-Self-Attention, Transformer Model, SPAMHAM, Feedforward Neural Networks.

Table of Contents

Title	Page No.
Bonafide Certificate	ii
Declaration	iii
Acknowledgements	iv
List of figures	v
Abbreviations	vi
Abstract	vii
1. Summary of the base paper	1
2. Merits and Demerits of the base paper	10
3. Source Code	12
4. Snapshots	20
5. Results and Discussions	27
6. Conclusion and Future Plans	28
References	29
7. Base paper	30

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title: A Spam Transformer Model for SMS Spam Detection

Name: IEEE.

Publisher: Xiaoxu Liu, Haove Lu, Amiya Naya.

Year: 2021.

1.1 SUMMARY:

- In this base paper, the authors proposed a Modified Transformer model using multi-head-self-attention mechanism that will learn message label from the text message using deep learning methodology and techniques which helps to improve the performance of the model.
- The dense layers are used in the form of linear fully-connected layers before applying the final activation function for prediction and sigmoid function, which maps the output to single numeric probability value for binary classification of spam messages.
- The Transformer model is trained using Adam optimization and gradient descent algorithm which helps in minimizing loss function by updating the models parameters .
- The modified transformer model and text processing techniques in model analysed.
- Data pre-processing and balancing has been handled carefully during the training phase.
- The SPAM detection model has been optimized by the proposed transformer model architecture.
- The proposed system given in base paper has successfully predicted age using machine learning and deep learning algorithms.

1.2 BACK GROUND:

Some of the key concepts are listed that are useful for implementation of the project and reason behind usage of concepts are described.

1.3 PROPOSED SOLUTION:

SPAM TRANSFORMER MODEL:

The Spam Transformer model is a neural network architecture explicitly developed for detecting SMS spam. The model is based on the Transformer architecture, which was developed under the task of natural language processing, which allows the Spam

Transformer to interpret and analyze SMS messages quickly and in parallel. At the heart of the Spam Transformer is the multi-head self-attention mechanism, which enables it to learn the intricate dependencies between words and word groups in the present text. Specifically, this mechanism allows the transformer to give a different weight to each word in terms of other words in the message, which creates a comprehensive representation of the content's meaning and context. In addition, the model is equipped with the positional encoding option to keep track of the original order of words in a message. This way, the Spam Transformer analyzes the whole SMS.

1.4 INTRODUCTION

The purpose of this SPAM Transformer model is that it will learn to detect the message as spam or ham by using deep learning methodology and techniques which are used to improve the performance of the model to clone the behaviour of the user input and implement proper environment.

Input Embedding:

The Transformer architecture begins by transforming input tokens into dense vector representations, referred to as input embeddings. In other words, the embeddings provide an initial encoding of the textual signal present in the input sequence. Formation any unique token is projected in a high-dimensional vector; the map is produced by an embedding layer. By utilizing embeddings, the model is able to preserve semantic proximities between tokens and thus process text data in the subsequent layers.

Positional Encoding:

After the input embeddings have been generated, positional encoding is used to integrate information about the sequence number of tokens into the input representations. This step is necessary to enable the model to adequately distinguish between tokens by their position in the input sequences. Positional encoding is frequently implemented as closed, linear combinations of various sinusoidal functions of different frequencies and phases. Having established positional information in the embeddings, the model has a sense of the tokens' positions concerning one another, allowing for the sequential arrangement of the underlying input order.2. Activation Map: It was created by sliding the each and every filter over input volume's spatial dimension during a forward pass-through FNN. If a neuron decides to pass information across, a numerical value is obtained.

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Encoder Layer:

The encoder layer is in charge of reading the input sequence and producing representation vectors for each token with contextual information. Many of the same encoder layers are stacked on top of one another. It consists of two essential components: self-attention with multiple heads and a feedforward neural network that is position-wise. First, in self-attention, every token of the input sequence pays attention to each other to learn dependencies between them. Then the attention vector proceeds through a feed-forward network to learn more about the data patterns and interactions. Layer normalization and residuals connections are executed to stabilize the training process and introduce data summarizing logic.

Decoder Layer:

Another example where this is the case is machine translation. To account for that, the Transformer architecture has a component responsible for reading the input sequence and producing the output sequence, which is the decoder layer. Just as the encoder has the sub-layer with proven effectiveness, the decoder similarly consists of parallel sub-layers. These include the multi-head self-attention mechanism with masking, the multi-head attention mechanism to give weight to the encoder output, and the position-wise feedforward neural network . Using the mentioned components, the decoder pays attention to the necessary parts of the input sequence and generates the output sequence token by token but still in the context of the input, allowing the model to expand input context into the output.

Output Layer:

The output layer is the last building block of the Transformer architecture; it is where the model's predictions or outputs are produced. For example, in the context of the SMS spam detection task, the output layer usually contains a soft-max activation function that provides probabilities for each class, such as spam or not spam. The class with the highest probability is then used as the model's output. This layer is conceptually the tip of the architecture, as the encoder or decoder layers generate contextualized representations that are processed into predictions or classes.

Multi-Head-Self-Attention Mechanism:

The multi-head self-attention mechanism is a key component of the Transformer architecture, allowing the model to capture dependencies between different tokens in the input sequence.

Query, Key, and Value Projections:

The input embeddings are transformed into three sets of vectors: query (Q), key (K), and value (V). These projections are linear transformations of the input embeddings

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_K$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_V$$

Where $\mathbf{W}_Q, \mathbf{W}_K, \text{and } \mathbf{W}_V$ represents learnable weight matrices

Scaled Dot-Product Attention:

- For each query, the attention scores are computed by taking the dot product of the query with all keys, scaled by the square root of the dimension of the key vectors to stabilize gradients

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)$$

- The resulting attention scores determine the importance of each token in the input sequence for a given query.

Weighted Sum of Values:

- The attention scores are then used to compute a weighted sum of the corresponding values

$$\mathbf{Z} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})\mathbf{V}$$

- Here, Z represents the output of the attention mechanism, capturing the contextualized representations of tokens based on their relationships with other tokens.

Multi-Head Attention:

- To capture different types of dependencies and relationships, the attention mechanism is applied multiple times in parallel, with separate sets of weight matrices for queries, keys, and values.
- The outputs of these parallel attention heads are concatenated and linearly transformed

$$\mathbf{Z}_{\text{multi}} = \text{Concat}(\mathbf{Z}_1, \mathbf{Z}_2, ..., \mathbf{Z}_h)\mathbf{W}_O$$

- Here, Z_{multi} represents the final output of the multi-head self-attention mechanism, h is the number of attention heads, and WO is another learnable weight matrix.

Feedforwarding Neural Networks:

The Feedforward neural network is a class of the neural network in which no feedback connections are there, and the information flows in one direction. The FFNN is employed as a position wise feedforward layer in the case of Transformer and placed after the self-attention mechanism. It consists of two linear forward layers with activations such as ReLU.

Input Layer:

The input layer receives the input data and passes it to the subsequent hidden layers. Each neuron in the input layer represents a feature or input variable.

Hidden Layer:

The hidden layers perform computations on the input data using activation functions and learnable parameters (weights and biases).

$$\mathbf{Z}^{(1)} = \mathrm{ReLU}(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

Output Layer:

The output layer produces the final output of the neural network. The number of neurons in the output layer depends on the task (e.g., binary classification, multi-class classification, regression).

$$\mathbf{Z}^{(2)} = \mathrm{softmax}(\mathbf{Z}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$$

1.5 PROBLEM DEFINTION:

Detecting and classifications of messages as spam/ham is very crucial from keeping the users from unwanted and potentially harmful content. As there are various machine learning algorithms available for the detection of spam but there are some demerits of those models. To solve this problem advanced deep learning architectures such as Modified Transformer model is enhanced. This model mainly focus on the improving the accuracy and performance of spam detection systems. The deep learning and attention mechanisms, the proposed model seeks to solve the key challenges in SMS detection, including the presence of unknown words, casual language, and abbreviations that are common in SMS messages

- To conduct a review of the existing solutions for detection of the spam messages in the SMS services, with an attention in the machine learning and deep learning approaches.

- To describe the algorithmic schema of the proposed classification system i.e, Modified Transformer model, explaining the various types of the mechanisms used in the model for classification.
- To analyze the architecture for the model, considering various methods namely Attention Mechanisms, Data Augmentation and so on to optimize the accuracy and performance of the proposed system.
- To analyze the accuracy of decision making on the basis of each aggregation method by conducting experiments with various models and comparing the effectiveness and performance of them in classifying the messages for two different sets of datasets.

1.6 PROPOSED ARCHITECTURE:

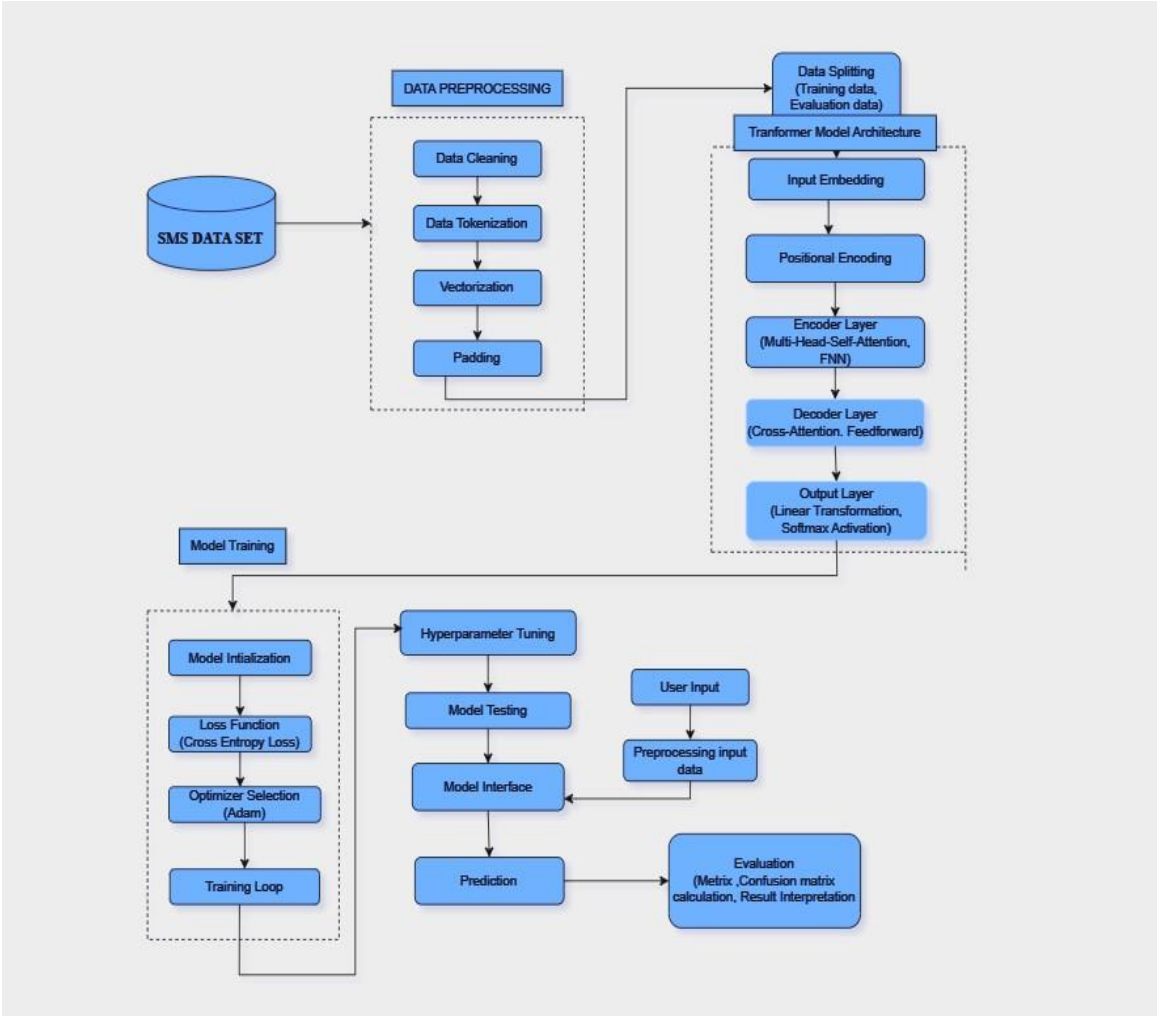

Fig. 1.1. Architecture Diagram

Spam Transformer Model Architecture :

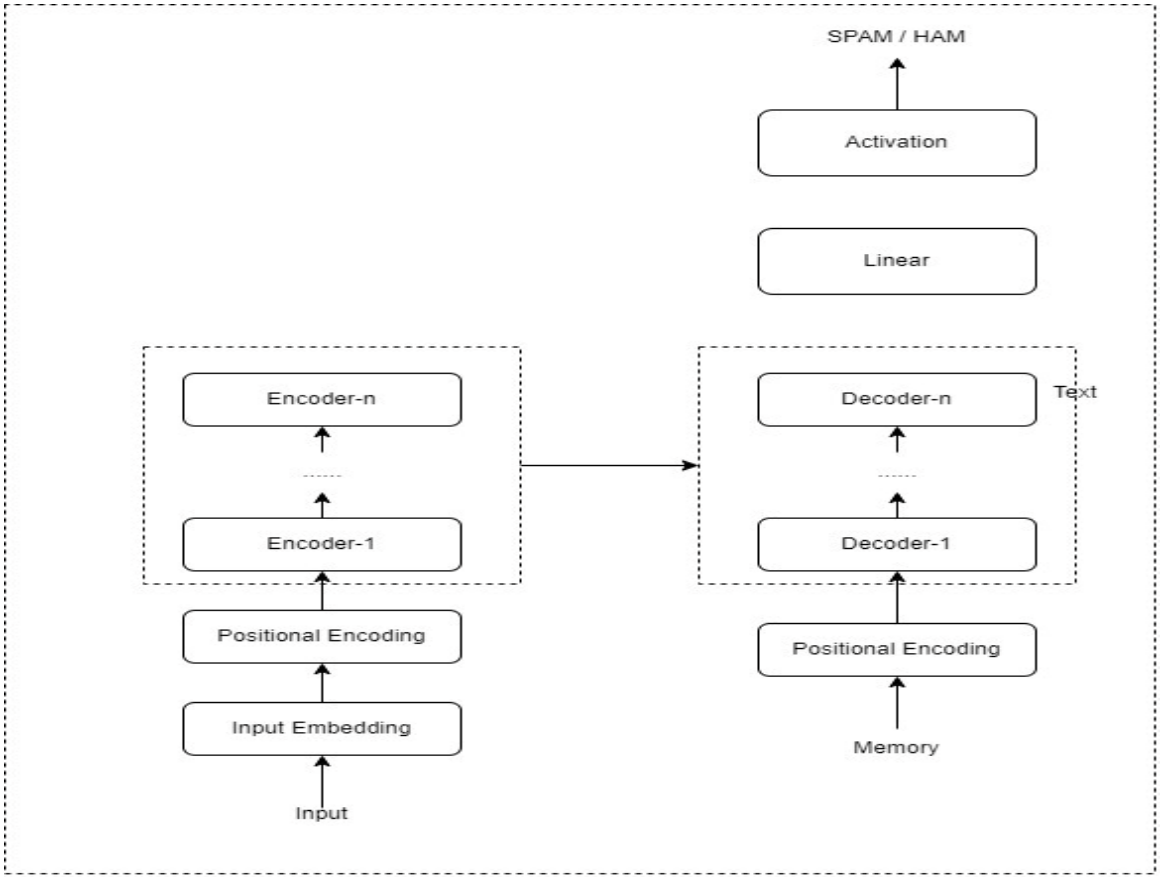

Fig 1.2. Model Architecture

1.7 SOLUTION APPROACH :

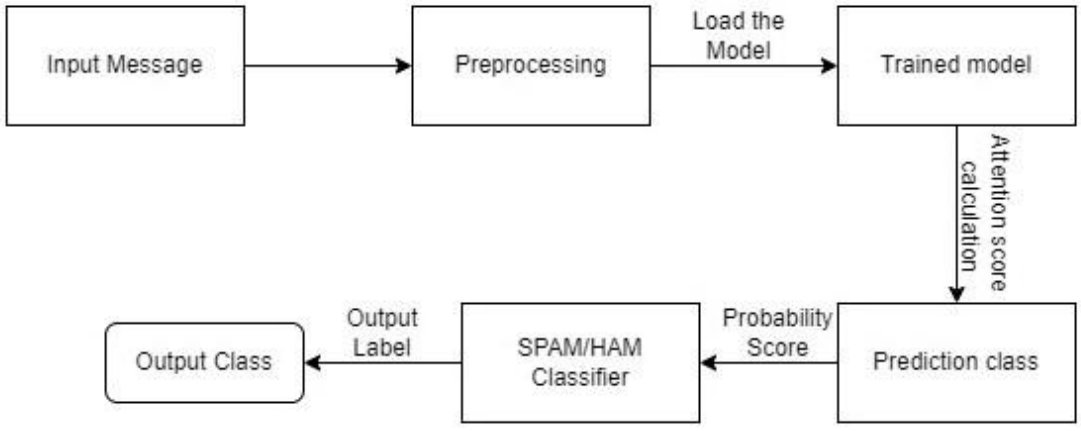

Fig.1.3 Solution Approach

The solution for the above approach is :

- Utilize the spam collection v.1 and Twitter UtkMIs datasets for training models by computing attention scores on each text message.
- Implement the spam transformer model in to the program for prediction.
- The user input message is pre-processed by applying necessary preprocessing techniques.
- The message tensors are passed to trained model for computing attention scores and capturing the dependencies of each token.
- The self-attention score is passed to the prediction class for returning the probability value of input text message.
- Spam classifier classifies the input message as SPAM or HAM by using the probability value.

Dataset :

The datasets used for this project is spam collection v.1 and twitter UtkMl's datasets. In spam collection v.1 dataset contains 5573 data points of two columns called v1(label) and v2(Text message). In twitter UtkMl's dataset there are 14900 datapoints for training and 786 datapoints for testing of 6 columns called Tweet, following, followers, actions,is_retweet, location, type.

1.8 TECHNOLOGIES USED:

There are different libraries in python those are used in Machine Learning and some of them are used to improve our project performance few of them are mentioned below

1.NUMPY: It is used for computing numerical values in python. It also supports multidimensional arrays and matrices, along with mathematical functions to operate on these arrays

2.TORCH: Pytorch is deep learning framework which provides tensor computation. It helps to include variety of layers, loss functions, activation functions and other utilities for constructing deep learning models.

3.SKLEARN: The sklearn.utils library helps to implement various functions which used in scikitlearn. It also provides functions for resampling data, manipulating miscellaneous utilities.

The machine used for this project is Google Colab, Kaggle and laptop with following specifications.

- Processor: Intel i5 Core -2100CPU @ 3.10GHz
- Random Access Memory (RAM) : 8GB

- Operating System : Windows 11 64-bit

1.9 Functions used in source code:

Binary Cross Entropy Function:

Binary Cross-Entropy, also known as Binary Log Loss. In SMS spam detection, the binary cross-entropy function serves as a crucial component for increasing the performance of the model in distinguishing between spam and non-spam messages.

$$\text{Binary Cross-Entropy Loss} = -\left(y \log(p) + (1-y) \log(1-p)\right)$$

- y is true binary value (0 or 1)
- p is predicted probability that belongs to class label 1

The binary cross entropy loss penalizes the transformer model more heavily when it makes a confident incorrect prediction. It penalizes less severely when the model's prediction is close to the actual label.

Rectified Linear Unit:

- Rectified Linear Unit function is non-linear activation function used in neural networks it returns the max value from 0 to x.

$$f(x) = \max(0, x)$$

- It is used applied as activation function for introducing non-linearity in hidden layers.

- It is implemented within hidden layers of feedforwarding neural networks after the linear transformation done.

SoftMax Function:

The SoftMax function is implemented in output layer of classification models, particularly in multiclass classification tasks. It takes input as a vector of real-valued scores and transforms them into a probability distribution on multiple classes. The output of the SoftMax function represents the likelihood of each class being the correct classification.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- z_i is the input score of class i.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1 MERITS :

- The proposed model in base paper is Transformer Model which is a Deep Learning Architecture.
- Utilizes a well-defined data splitting strategy for training, validation and testing.
- Lots of preprocessing methods and techniques were implemented.
- This system can identify spam and regular messages easily, even when the dataset is uneven and balanced.
- It proposes by increasing the variety of words used and replacing unused ones with similar words to improve classification performance.
- Tests during the training phase can be clearly seen that both the training and validation losses are lower compared to other models. This shows that the proposed model, which subjected finetuning, is performed better.
- Uses the validation set for selection of model and overfitting is prevented by early stopping.
- Shows the perfect results in precisely identifying spam messages and maintaining high scores in recall and F1-Score on both of the datasets i.e, SMS Spam Collection v.1 and UtkMl's Twitter datasets.
- It clearly shows that the model can handle even very large datasets and provide more accurate results and better performance.
- The model's entire flow, from training to validation and testing, is clearly explained, showing a better understanding of how the proposed model works and executes.

2.2 DEMERITS :

- The model's ability to handle noisy or improper inputs, which is important for real-life applications where the data quality may vary, is not thoroughly examined.
- For improving the performance of the model, we need more extensive datasets.
- When messages are very short, unknown words have a strange effect, which creates problems while predicting and identifying the messages.
- The performance is varied for different datasets due to the usage of different languages and abbreviations in the datasets.
- The behavioural problems related to spam detection algorithms, such as privacy issues or the unexpected negative effects of wrongly estimated messages, are not clearly shown.

- It is not clearly provided how the model reacts to different hyperparameters or traits of the training data. This could affect how it adapt to new situations and be stable.
- The model should be trained completely on different kind of paths and on different conditions in which generalization is feasible in that situations.

CHAPTER 3

SOURCE CODE

#samples from source code

3.1 Import Libraries :

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re

#download required packages
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
sms_data = pd.read_csv('spam.csv', encoding="latin-1")
```

3.2 Data Preprocessing and Balancing:

```
# Preprocessing data
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    tokens0 = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    tokens1 = [word for word in tokens0 if word not in stop_words]

    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens1]

    preprocessed_text = ' '.join(tokens)
    return preprocessed_text

sms_data['clean_text'] = sms_data['v2'].apply(preprocess_text)

spam_data = sms_data[sms_data['v1'] == 'spam']
ham_data = sms_data[sms_data['v1'] == 'ham']

num_samples = min(len(spam_data), len(ham_data))

spam_data_resampled = resample(spam_data, replace=True, n_samples=num_samples, random_state=42)
ham_data_resampled = resample(ham_data, replace=True, n_samples=num_samples, random_state=42)

balanced_data = pd.concat([spam_data_resampled, ham_data_resampled])

X_balanced = balanced_data['clean_text']
y_balanced = balanced_data['v1'].map({'ham': 0, 'spam': 1})

X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.2, random_state=42)

count_vectorizer = CountVectorizer()
X_train_vectorized = count_vectorizer.fit_transform(X_train)
X_test_vectorized = count_vectorizer.transform(X_test)

X_train_tensor = torch.tensor(X_train_vectorized.toarray()).float()
X_test_tensor = torch.tensor(X_test_vectorized.toarray()).float()
y_train_tensor = torch.tensor(y_train.values).long()
y_test_tensor = torch.tensor(y_test.values).long()
```

3.3 Transformer Model definition:

```
# model definition
import torch.nn as nn
import torch.nn.functional as F

class InputEmbedding(nn.Module):
    def __init__(self, vocab_size, d_model):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, d_model)

    def forward(self, x):
        x = x.long()
        return self.embedding(x)

class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5573):
        super().__init__()
        self.dropout = nn.Dropout(dropout)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-torch.log(torch.tensor(10000.0)) / d_model))
        pe = torch.zeros(max_len, d_model)
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(1).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:, :x.size(1)]
        return self.dropout(x)
```

3.3.1 Encoder Layer:

```
class TransformerEncoderLayer(nn.Module):
    def __init__(self, d_model, nhead, dim_feedforward=2048, dropout=0.1):
        super().__init__()
        self.self_attn = nn.MultiheadAttention(d_model, nhead, dropout=dropout)
        self.linear1 = nn.Linear(d_model, dim_feedforward)
        self.dropout = nn.Dropout(dropout)
        self.linear2 = nn.Linear(dim_feedforward, d_model)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout1 = nn.Dropout(dropout)
        self.dropout2 = nn.Dropout(dropout)
        self.activation = F.relu

    def forward(self, src, src_mask=None, src_key_padding_mask=None):
        src2 = self.self_attn(src, src, src, attn_mask=src_mask, key_padding_mask=src_key_padding_mask)[0]
        src = src + self.dropout1(src2)
        src = self.norm1(src)
        src2 = self.linear2(self.dropout(self.activation(self.linear1(src))))
        src = src + self.dropout2(src2)
        src = self.norm2(src)
        return src

class TransformerEncoder(nn.Module):
    def __init__(self, encoder_layer, num_layers):
        super().__init__()
        self.layers = nn.ModuleList([copy.deepcopy(encoder_layer) for _ in range(num_layers)])

    def forward(self, src, mask=None, src_key_padding_mask=None):
        for layer in self.layers:
            src = layer(src, src_mask=mask, src_key_padding_mask=src_key_padding_mask)
        return src
```

3.3.2 Decoder Layer:

```
class TransformerDecoderLayer(nn.Module):
    def __init__(self, d_model, nhead, dim_feedforward=2048, dropout=0.1):
        super().__init__()
        self.mha_self = nn.MultiheadAttention(d_model, nhead, dropout=dropout)
        self.ln1 = nn.LayerNorm(d_model)
        self.mha_src = nn.MultiheadAttention(d_model, nhead, dropout=dropout)
        self.ln2 = nn.LayerNorm(d_model)
        self.ffn = nn.Sequential(
            nn.Linear(d_model, dim_feedforward),
            nn.ReLU(),
            nn.Linear(dim_feedforward, d_model)
        )

    def forward(self, trg, enc_src, trg_mask=None, src_mask=None, trg_key_padding_mask=None, src_key_padding_mask=None):
        x, _ = self.mha_self(trg, trg, trg, attn_mask=trg_mask, key_padding_mask=trg_key_padding_mask)
        x = self.ln1(x)
        x, _ = self.mha_src(x, enc_src, enc_src, attn_mask=src_mask, key_padding_mask=src_key_padding_mask)
        x = self.ln2(x)
        x = x + self.ffn(x)
        return x

class TransformerDecoder(nn.Module):
    def __init__(self, decoder_layer, num_layers):
        super().__init__()
        self.layers = nn.ModuleList([decoder_layer for _ in range(num_layers)])

    def forward(self, x, enc_src, trg_mask=None, src_mask=None, trg_key_padding_mask=None, src_key_padding_mask=None):
        for layer in self.layers:
            x = layer(x, enc_src, trg_mask=trg_mask, src_mask=src_mask, trg_key_padding_mask=trg_key_padding_mask, src_key_padding_mask=src_key_padding_mask)
        return x
```

3.3.3 Model Initialization:

```
class TransformerModel(nn.Module):
    def __init__(self, vocab_size, d_model, nhead, num_layers, dim_feedforward=2048, dropout=0.1):
        super().__init__()
        self.embedding = InputEmbedding(vocab_size, d_model)
        self.pos_encoding = PositionalEncoding(d_model, dropout)
        self.encoder = TransformerEncoder(TransformerEncoderLayer(d_model, nhead, dim_feedforward, dropout), num_layers)
        self.decoder = TransformerDecoder(TransformerDecoderLayer(d_model, nhead, dim_feedforward, dropout), num_layers)
        self.fc = nn.Linear(d_model, 2)

    def forward(self, x, enc_x=None):
        x = self.embedding(x) * torch.sqrt(torch.tensor(self.embedding.embedding.embedding_dim, device=x.device))
        x = self.pos_encoding(x)
        if enc_x is not None:
            enc_x = self.encoder(enc_x)
            x = self.decoder(x, enc_x)
        x = self.fc(x[:, 0, :])
        return x
class SimpleClassifier(nn.Module):
    def __init__(self, input_dim):
        super(SimpleClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

3.4 Model Training:

```
import matplotlib.pyplot as plt
accuracies=[]
input_dim = X_train_tensor.shape[1]
model = SimpleClassifier(input_dim)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)

train_accuracies = []
test_accuracies = []

#training model
def train(model, criterion, optimizer, X_train, y_train, X_test, y_test, epochs=10, batch_size=64):
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        correct_train = 0
        total_train = 0
        correct_test = 0
        total_test = 0
        for i in range(0, len(X_train), batch_size):
            inputs = X_train[i:i+batch_size]
            labels = y_train[i:i+batch_size]
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, predicted_train = torch.max(outputs.data, 1)
            total_train += labels.size(0)
            correct_train += (predicted_train == labels).sum().item()
        epoch_loss = running_loss / len(X_train)
        epoch_acc_train = correct_train / total_train
        train_accuracies.append(epoch_acc_train)
```

3.5 Model Evaluation:

```
        model.eval()
        with torch.no_grad():
            for i in range(0, len(X_test), batch_size):
                inputs_test = X_test[i:i+batch_size]
                labels_test = y_test[i:i+batch_size]
                outputs_test = model(inputs_test)
                _, predicted_test = torch.max(outputs_test.data, 1)
                total_test += labels_test.size(0)
                correct_test += (predicted_test == labels_test).sum().item()
            epoch_acc_test = correct_test / total_test
            test_accuracies.append(epoch_acc_test)

        print(f'Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}, Train Accuracy: {epoch_acc_train:.4f}')

train(model, criterion, optimizer, X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor)

# evaluate overall accuracy
def evaluate_overall_accuracy(model, X_test, y_test, batch_size=64):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for i in range(0, len(X_test), batch_size):
            inputs = X_test[i:i+batch_size]
            labels = y_test[i:i+batch_size]
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    overall_accuracy = correct / total
    print(f'Overall Accuracy: {overall_accuracy:.4f}')
    accuracies.append(overall_accuracy*100)
    return overall_accuracy

overall_accuracy = evaluate_overall_accuracy(model, X_test_tensor, y_test_tensor)
```

3.6 Prediction class Declaration:

```
def predict_message_simple_classifier(model, message, count_vectorizer):
    message = preprocess_text(message)

    message_vectorized = count_vectorizer.transform([message])

    message_tensor = torch.tensor(message_vectorized.toarray()).float()

    with torch.no_grad():
        output = model(message_tensor)
        predicted_class = torch.argmax(output)

    if predicted_class == 0:
        prediction = "ham"
    else:
        prediction = "spam"

    return prediction


user_message = input("Enter your text message: ")
prediction = predict_message_simple_classifier(model, user_message, count_vectorizer)
print(f"The predicted class is: {prediction}")
```

3.7 Metric Scores Evaluation:

```
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score

def evaluate_model(model, X_test, y_test, batch_size=64):
    model.eval()
    predictions = []
    with torch.no_grad():
        for i in range(0, len(X_test), batch_size):
            inputs = X_test[i:i+batch_size]
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            predictions.extend(predicted.cpu().numpy())
    y_pred = np.array(predictions)

    f1 = f1_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    print(f'F1 Score: {f1:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print('Confusion Matrix:')
    print(cm)

evaluate_model(model, X_test_tensor, y_test_tensor)
```

3.8 Importing Models:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, f1_score, recall_score, confusion_matrix

#random Forest
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train_vectorized, y_train)
rf_predictions = rf_classifier.predict(X_test_vectorized)

#naive Bayes
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_vectorized, y_train)
nb_predictions = nb_classifier.predict(X_test_vectorized)

#logistic Regression
logistic_classifier = LogisticRegression()
logistic_classifier.fit(X_train_vectorized, y_train)
logistic_predictions = logistic_classifier.predict(X_test_vectorized)

#evaluate models
def evaluate_model(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    cm = confusion_matrix(y_true, y_pred)
    return acc, f1, recall, cm

rf_acc, rf_f1, rf_recall, rf_cm = evaluate_model(y_test, rf_predictions)

nb_acc, nb_f1, nb_recall, nb_cm = evaluate_model(y_test, nb_predictions)

logistic_acc, logistic_f1, logistic_recall, logistic_cm = evaluate_model(y_test, logistic_predictions)

print("Random Forest Classifier:")
print(f"Accuracy: {rf_acc}")
print(f"F1 Score: {rf_f1}")
print(f"Recall: {rf_recall}")
print(f"Confusion Matrix:\n{rf_cm}")
```

```
print("\nNaive Bayes Classifier:")
print(f"Accuracy: {nb_acc}")
print(f"F1 Score: {nb_f1}")
print(f"Recall: {nb_recall}")
print(f"Confusion Matrix:\n{nb_cm}")

print("\nLogistic Regression Classifier:")
print(f"Accuracy: {logistic_acc}")
print(f"F1 Score: {logistic_f1}")
print(f"Recall: {logistic_recall}")
print(f"Confusion Matrix:\n{logistic_cm}")

accuracies.append(rf_acc*100)
accuracies.append(nb_acc*100)
accuracies.append(logistic_acc*100)
print(accuracies)
```

3.9 Twitter Dataset :

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    tokens0 = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    tokens1 = [word for word in tokens0 if word not in stop_words]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens1]
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text

train_data['clean_tweet'] = train_data['Tweet'].apply(preprocess_text)
test_data['clean_tweet'] = test_data['Tweet'].apply(preprocess_text)

# apply balanceing the dataset
spam_data = train_data[train_data['Type'] == 'Spam']
quality_data = train_data[train_data['Type'] == 'Quality']
num_samples = min(len(spam_data), len(quality_data))
spam_data_resampled = resample(spam_data, replace=True, n_samples=num_samples, random_state=42)
quality_data_resampled = resample(quality_data, replace=True, n_samples=num_samples, random_state=42)
balanced_data = pd.concat([spam_data_resampled, quality_data_resampled])

X = balanced_data['clean_tweet']
y = balanced_data['Type'].map({'Quality': 0, 'Spam': 1})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

vectorizer = TfidfVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

X_train_tensor = torch.tensor(X_train_vectorized.toarray()).float()
X_test_tensor = torch.tensor(X_test_vectorized.toarray()).float()
y_train_tensor = torch.tensor(y_train.values).long()
y_test_tensor = torch.tensor(y_test.values).long()
```

CHAPTER 4

SNAPSHOTS

For spam messages dataset:

4.1 Balancing and Preprocessing of data :

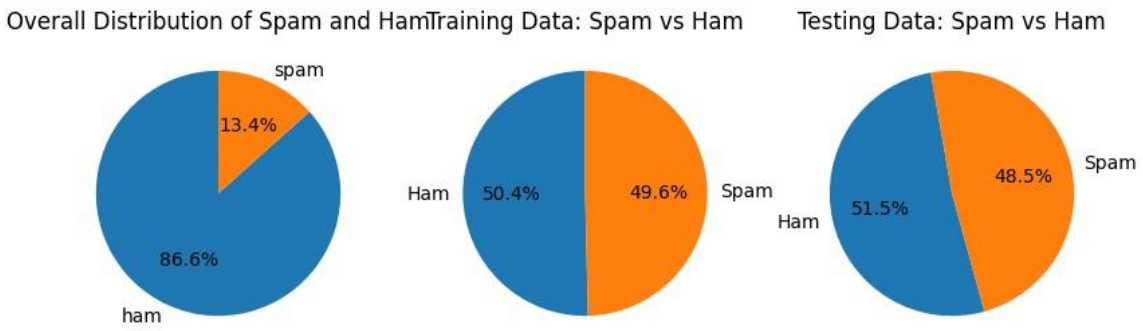

Fig 4.1.1. Testing and Training data after balancing

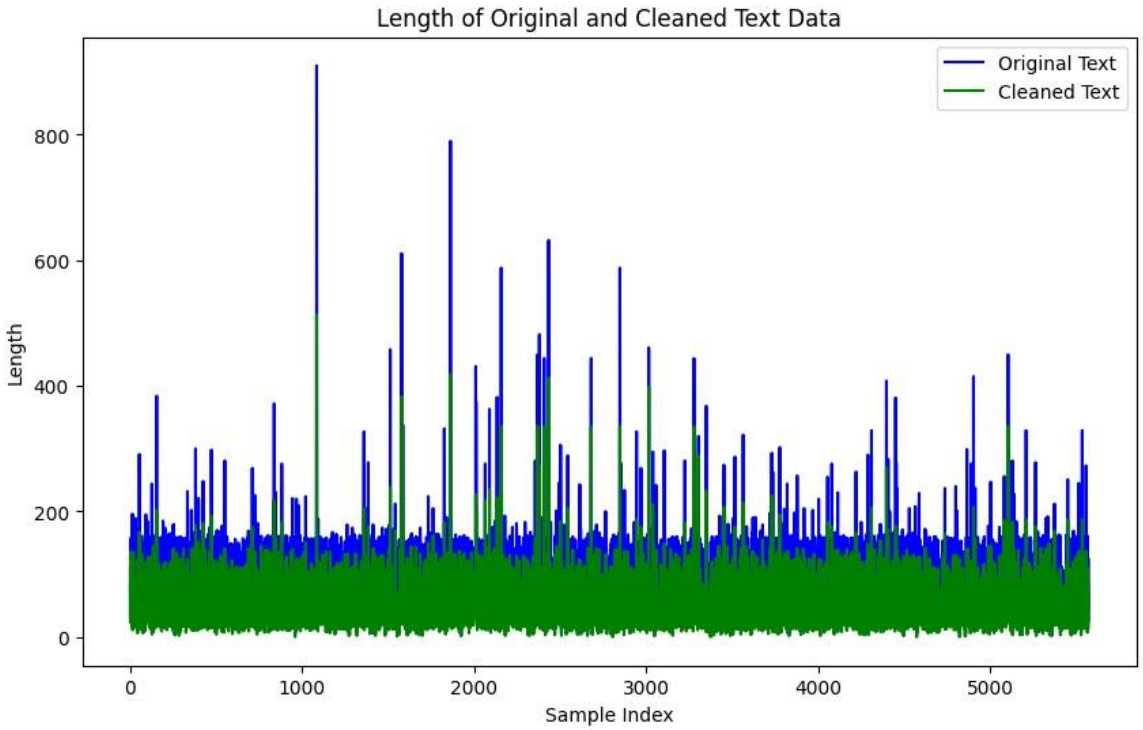

Fig 4.1.2 Output of original text length and preprocesses text length

4.2 Model Summary, Testing and Training Accuracies :

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
         Embedding-1          [-1, 1, 5573, 512]       4,440,064
    InputEmbedding-2          [-1, 1, 5573, 512]               0
            Dropout-3          [-1, 1, 5573, 512]               0
PositionalEncoding-4          [-1, 1, 5573, 512]               0
            Linear-5               [-1, 5573, 2]           1,026
================================================================
Total params: 4,441,090
Trainable params: 4,441,090
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.02
Forward/backward pass size (MB): 87.16
Params size (MB): 16.94
Estimated Total Size (MB): 104.13
----------------------------------------------------------------
```

Fig 4.2.1 Summary of the Transformer Model

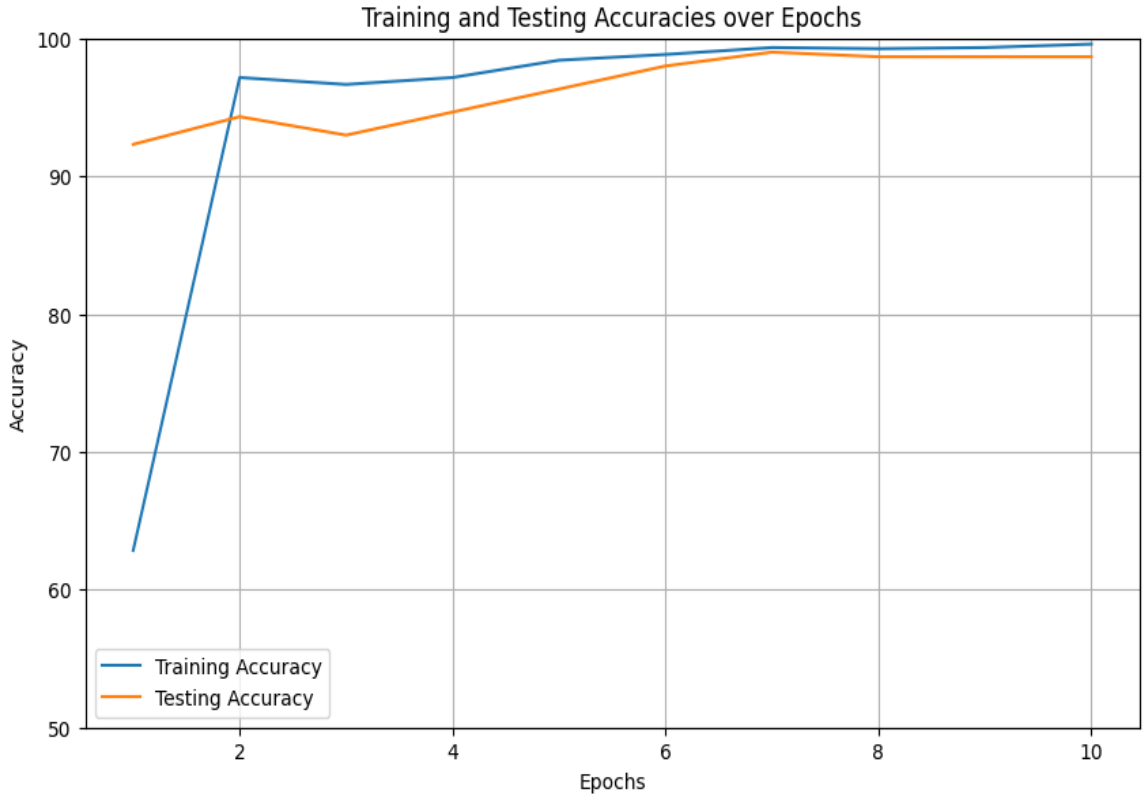

Fig 4.2.2 Comparison of Testing and Training Accuracies over 10 epochs

4.3 Metric Scores:

Confusion Matrix :

	SPAM	HAM
SPAM	151	1
HAM	3	142

```
F1 Score: 0.9861
Precision: 0.9930
Recall: 0.9793
```

Fig 4.3.1 Evaluation Metrics

4.4 Models implemented for comparison :

1. Logistic Regression
2. Naïve Bayes
3. Random Forest

4.1.1 Confusion Matrix:

	Logistic Regression		Naïve Bayes		Random Forest	
	SPAM	HAM	SPAM	HAM	SPAM	HAM
SPAM	154	0	143	11	152	2
HAM	5	140	1	144	5	140

4.1.2 Accuracies and Metric Scores :

	Accuracy	F1 Score	Recall	Precision
Logistic Regression	97.65	0.9752	0.9517	1.00
Random Forest	97.65	0.9756	0.9655	0.9928
Naïve Bayes	95.98	0.96	0.9931	0.9290

4.5 Comparison of accuracies :

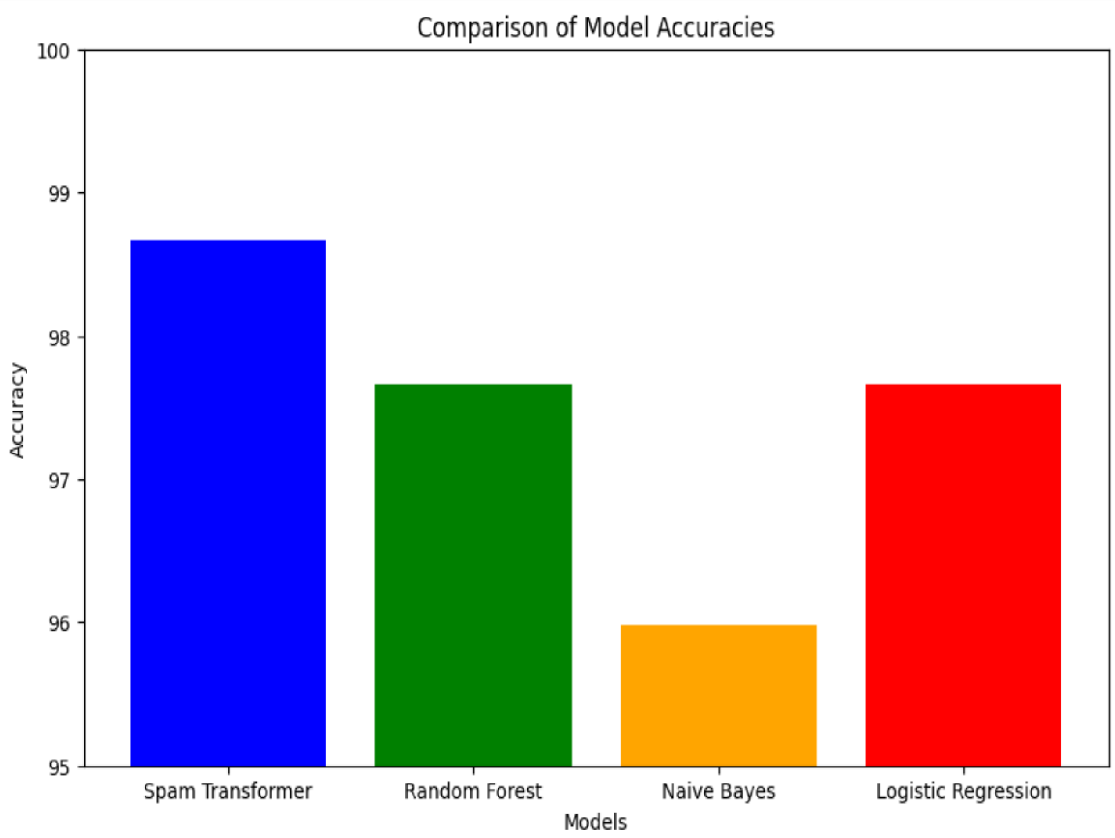

Fig 4.5 Comparison of accuracies of the different models

For UtkMl's Twitter dataset :

4.6 ANALYSIS

4.6.1 Tweets Analysis:

In this dataset the number of tweets used are 14,900 and it trained the model by this dataset(train.csv). In this tweet analysis the number of tweets are based on type of tweet (Quality/Spam). And the classification is shown in below fig 4.6.1.

Fig 4.6.1 Tweet Analysis

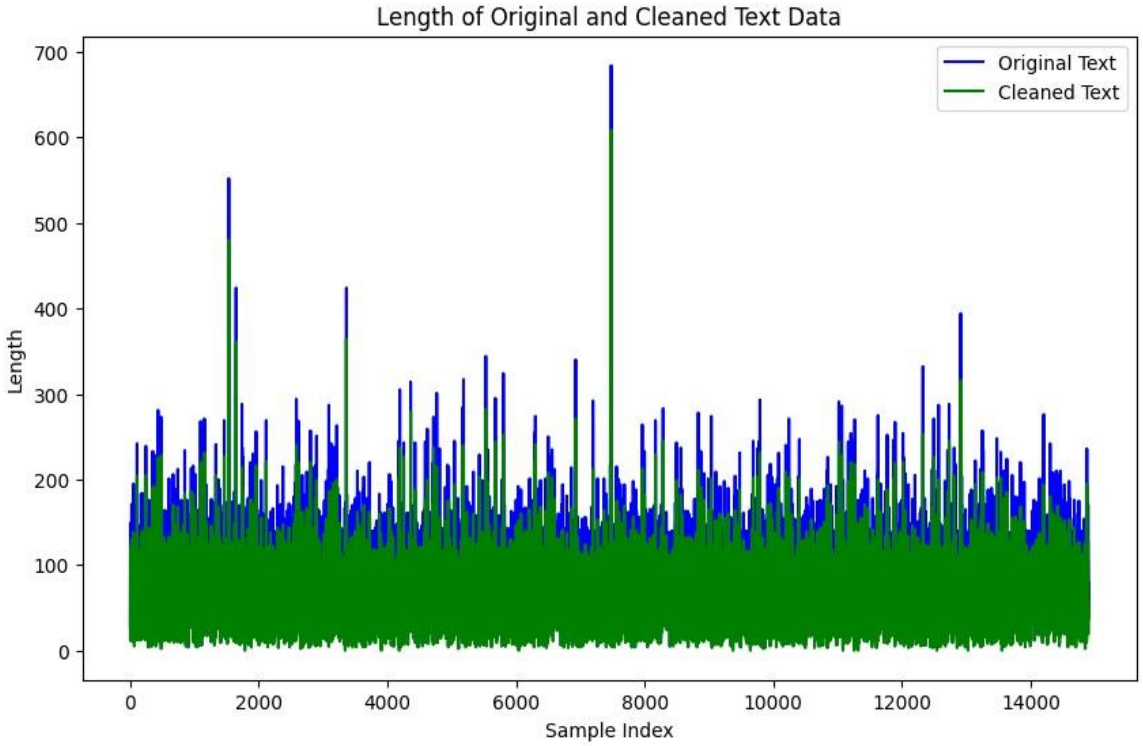

Fig 4.6.2 Tweet lengths in bar graph for training data

The below graph shows the preprocessing of Tweets in the dataset test.csv which is used for testing the transformer model.

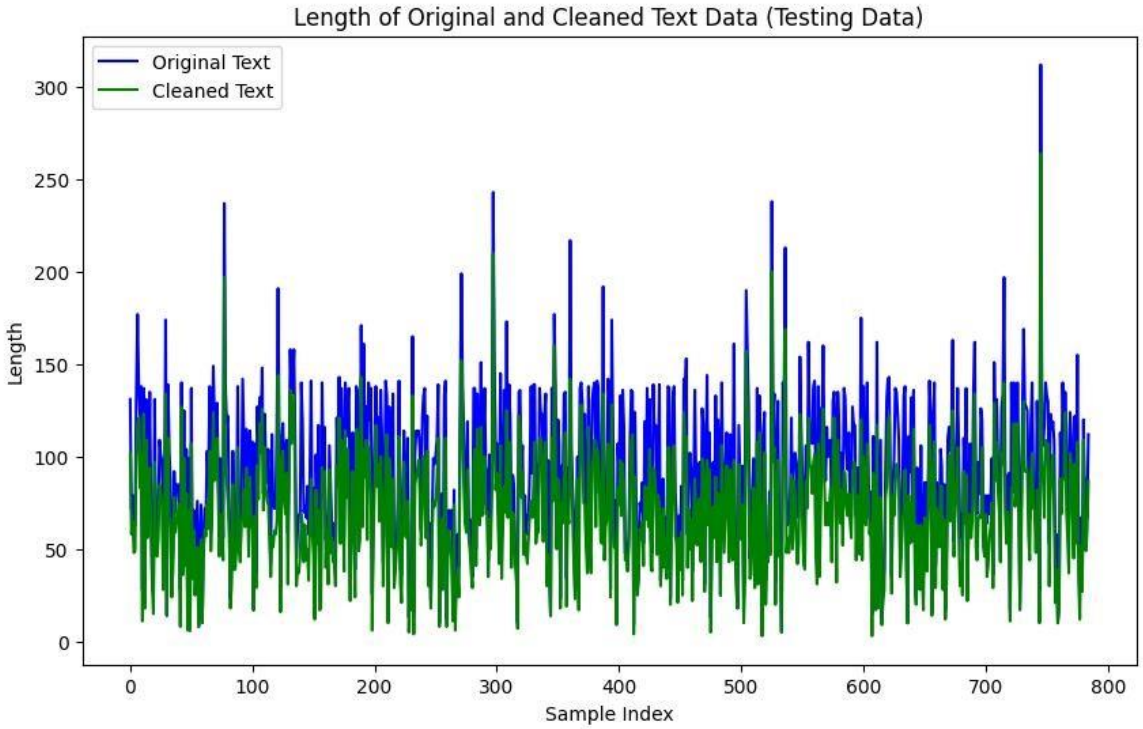

Fig 4.6.3 Tweet lengths in bar graph for testing data

4.7 Testing and Training Accuracies :

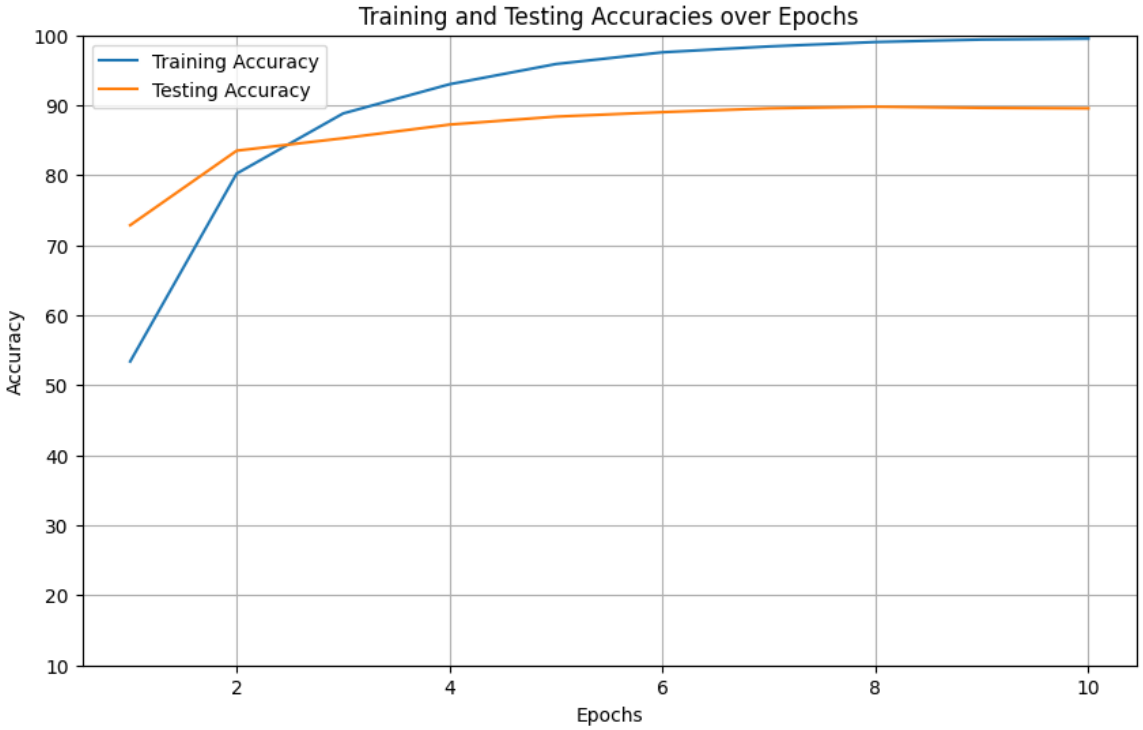

Fig 4.7.1 Representation of Testing and Training accuracies over 10 epochs

4.8 Evaluation Metrics for the Model and Result Interpretation:

4.8.1. Transformer Model :

```
F1 Score: 0.8981
Precision: 0.8990
Recall: 0.8972
Confusion Matrix:
[[1329  151]
 [ 154 1344]]
```

4.8.2 Metrics for other models used for comparison :

Confusion Matrix:

	Logistic Regression		Naïve Bayes		Random Forest	
	SPAM	HAM	SPAM	HAM	SPAM	HAM
SPAM	1310	170	1310	170	1381	99
HAM	263	1235	220	1278	218	1280

4.8.3 Evaluation Metrics for other models:

	Accuracy	F1 Score	Recall	Precision
Logistic Regression	85.46	0.8508	0.8244	0.8790
Random Forest	89.35	0.8898	0.8544	0.9282
Naïve Bayes	86.90	0.8676	0.8531	0.8825

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Result Analysis:

- Utilizes the Self-Attention scores for the binary classification, differentiating the SPAM Message and HAM Message.
- The SoftMax function is used as activation function during training for binary classification.
- Binary cross entropy function is used for outlier detection

For Spam dataset:

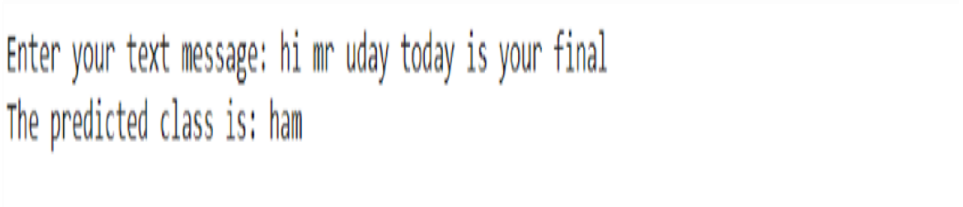

Fig 5.1.1 Result Interpretation for spam

For Twitter dataset :

Fig 5.1.2 Result Interpretation for Twitter

CHAPTER 6

CONCLUSION AND FUTURE PLANS

6.1 CONCLUSION:

In conclusion, the SMS spam detection system is developed using advanced machine learning techniques which utilise the modified transformer model with multi-head-self-attention mechanism. Overall, the presented system covers essential aspects, such as careful data preprocessing, feature extraction using a transformer model, and classification using the extracted representations. The system presented in this paper shows high accuracy for distinguishing between spam and legitimate messages. The transformer model is effective for such a task due to the fine-tuning process. Furthermore, multi-head self-attention perfectly covers the complex structure and dependencies in textual data.

The integration of the transformer model dramatically enhances the system's ability to capture contextual nuances and semantic subtleties present in SMS messages, which results in improving the accuracy of spam detection by numerous significant orders of magnitude. Furthermore, the use of advanced deep learning approaches ensures the system's scalability and generalization, making it applicable for use in real-world settings, where the quick and accurate identifying of spam is necessary to ensure the privacy and safety of users.

The discussion above demonstrates that the system has already shown promising results and can significantly contribute to the fight against SMS spam, efficiently protecting users from unwanted messages. However, the integration of additional features or the use of ensemble learning approaches can provide further improvements and increase the system's generalizability to different SMS datasets.

6.2 FUTURE PLAN:

In future this project will be useful for large and various types of datasets providing more performance and prediction rate by having the optimizations in the model and searching various advanced Transformer architectures.

By analysing the user's message history and preferences, the system can be able to provide accurate suggestions for both spam and useful messages, thereby improving the applicability and usefulness of its suggestions.

In future the model will not only useful for message services but also can be capable of detecting the messages, tweets in the social media apps like WhatsApp, Twitter, Facebook which are primary sources for getting attacked by opening spam messages. The messages and tweets are crafted in such a way that one is incapable of identifying them. The model will be well trained for those datasets and prevent the user from getting exploited. By using advanced architectures, the identification of them will be very easy for anyone.

Detecting the message type is not so easy. To detect messages, train our model very well and train with large data set of SMS messages.

For real time applications, our model should be well trained to predict for even short length messages which was very difficult to predict.

REFERENCES:

[1] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, A comparison of sequence-to-sequence models for speech recognition, in Proc. Inter speech, Aug. 2017, pp. 939943.

[2] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, Sequence to sequence video to text, inProc. IEEEInt.Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 45344542.

[3] D. Bahdanau, K. H. Cho, and Y. Bengio, Neural machine translation by jointly learning to align and translate, in Proc. 3rd Int. Conf. Learn. Represent. (ICLR), 2015.

[4] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, Show, attend and tell: Neural image caption generation with visual attention, in Proc. 32nd Int. Conf. Mach. Learn., vol. 3, 2015, pp. 20482057.

[5] T.Luong,H.Pham,andC.D.Manning, Effectiveapproachestoattention based neural machine translation, in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2015, pp. 14121421.

[6] E. S. D. Reis, C. A. D. Costa, D. E. D. Silveira, R. S. Bavaresco, R. D. R. Righi, J. L. V. Barbosa, R. S. Antunes, M. M. Gomes, and G. Federizzi, Transformers aftermath, Commun. ACM, vol. 64, no. 4, pp. 154163, Apr. 2021.

[7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, Improving neural networks by preventing co adaptation of feature detectors, 2012, arXiv:1207.0580. [Online]. Available: http://arxiv.org/abs/1207.0580

[8] I. Loshchilov and F. Hutter, Decoupled weight decay regularization, 2017, arXiv:1711.05101. [Online]. Available: http://arxiv.org/abs/1711. 05101

[9] UtkMls Twitter Spam Detection Competition | Kaggle, UtkMl.

[10] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, spaCy: Industrial-strength natural language processing in python, 2020, doi: 10.5281/zenodo.1212303.

CHAPTER 7

BASE PAPER

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A Spam Transformer Model for SMS Spam Detection

XIAOXU LIU[1], HAOYE LU[1] (Member, IEEE), AND AMIYA NAYAK.[1], (Senior Member, IEEE)

[1]School of Electrical Engineering and Computer Science, University of Ottawa, Canada, K1N 6N5 (e-mail: xliu301, hlu044, nayak@uottawa.ca)

Corresponding author: Haoye Lu (e-mail: hlu044@uottawa.ca).

ABSTRACT In this paper, we aim to explore the possibility of the Transformer model in detecting the spam Short Message Service (SMS) messages by proposing a modified Transformer model that is designed for detecting SMS spam messages. The evaluation of our proposed spam Transformer is performed on SMS Spam Collection v.1 dataset and UtkMl's Twitter Spam Detection Competition dataset, with the benchmark of multiple established machine learning classifiers and state-of-the-art SMS spam detection approaches. In comparison to all other candidates, our experiments on SMS spam detection show that the proposed modified spam Transformer has the optimal results on the accuracy, recall, and F1-Score with the values of 98.92%, 0.9451, and 0.9613, respectively. Besides, the proposed model also achieves good performance on the UtkMl's Twitter dataset, which indicates a promising possibility of adapting the model to other similar problems.

INDEX TERMS SMS spam detection, Transformer, attention, deep learning

I. INTRODUCTION

A. MOTIVATION AND OBJECTIVE

THE Short Message Service (SMS) has been widely used as a communication tool over the past few decades as the popularity of mobile phone and mobile network grows. However, SMS users are also suffering from SMS spam. The SMS spam, also known as drunk message, refers to any irrelevant messages delivered using mobile networks [1]. There are several reasons that lead to the popularity of spam messages. Firstly, there is a large number of users who use mobile phones in the world, making the potential victims of the spam messages attack also high. Secondly, the cost of sending out spam messages is low, which could be good news to the spam attacker. Last but not least, the capability of the spam classifier on most mobile phones is relatively weak due to the shortage of computational resources, which limits them from identifying the spam message correctly and efficiently.

Machine learning is one of the most popular topics in the last few decades, and there are a great number of machine learning based classification applications in multiple research areas. Specifically, spam detection is a relatively mature research topic with several established methods. However, most of the machine learning based classifiers were dependent on the handcrafted features extracted from the training data [2].

As a class of machine learning techniques, deep learning has been developing rapidly recently thanks to the surprising growth of computational resources in the last few decades. Nowadays, deep learning based applications play a significant part in our society, making our lives much easier in many aspects. As one of the most effective and widely used deep learning architectures, Recurrent Neural Network (RNN), as well as its variants such as Long Short-Term Memory (LSTM), were applied to spam detection and proved to be extremely effective during the last few years.

The Transformer [3] is an attention-based sequence-to-sequence model that was originally designated for translation task, and it achieved great success in English-German and English-French translation. Moreover, there are multiple improved Transformer-based models such as GPT-3 [4] and BERT [5] proposed recently to address different Natural Language Process (NLP) problems. The accomplishments of the Transformer and its successors have proved how powerful and promising they are. In this paper, we aim to explore whether it is possible to adapt the Transformer model to the SMS spam detection problem. Therefore, we propose a modified model based on the vanilla Transformer to identify SMS spam messages. Additionally, we analyze and compare the performance of SMS spam detection between traditional machine learning classifiers, an LSTM deep learning solution, and our proposed spam Transformer model.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

B. RELATED WORK

There are several different machine learning based classification applications proposed in the last few decades [6] [7] [8] [9]. In the field of SMS spam detection, a great number of these approaches are based on traditional machine learning techniques, such as Logistic Regression (LR), Random Forest (RF) [10], Support Vector Machine (SVM) [11], Naïve Bayes (NB), and Decision Trees (DT). Recently, with the prosperity of the deep learning techniques, an increasing number of methods have been introduced to address the SMS spam problem using deep learning based solutions such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM), which is a successful variant of RNN.

In [12], Gupta et al. compared the performance of 8 different classifiers including SVM, NB, DT, LR, RF, AdaBoost, Neural Network, and CNN. The experimental tests on the SMS Spam Collection v.1 [13] dataset that was conducted by the authors shows that the CNN and Neural Network are better compared to other machine learning classifiers, and the CNN and Neural Network achieved an accuracy of 98.25% and 98.00%, respectively.

In [14], Jain et al. proposed a method to apply rule-based models on the SMS spam detection problem. The authors extracted 9 rules and implemented Decision Tree (DT), RIPPER [15], and PRISM [16] to identify the spam messages. According to the experimental results from the authors, the RIPPER outperformed the PRISM and the DT, yielding a 99.01% True Negative Rate (TNR) and a 92.82% True Positive Rate (TPR).

In [1], Roy et al. aimed to adapt the CNN and LSTM to the SMS spam messages detection problem. The authors evaluated the performance of CNN and LSTM by comparing them with Naïve Bayes (NB), Random Forest (RF), Gradient Boosting (GB) [17], Logistic Regression (LR), and Stochastic Gradient Descent (SGD) [18]. The experiments that were conducted by the authors showed that the CNN and LSTM perform significantly better than the tested traditional machine learning approaches when it comes to SMS spam detection.

In [2], the authors proposed the Semantic Long Short-Term Memory (SLSTM), a variant of LSTM with an additional semantic layer. The authors employed the Word2vec [19], the WordNet [20], and the ConceptNet [21] as the semantic layer, and combined the semantic layer with the LSTM to train an SMS spam detection model. The experimental evaluation that was conducted by the authors claimed that the SLSTM achieved an accuracy of 99% on the SMS Spam Collection v.1 dataset.

In [22], Ghourabi et al. proposed the CNN-LSTM model that consists of a CNN layer and an LSTM layer in order to identify SMS spam messages in English and Arabic. The authors evaluated the CNN-LSTM by comparing it with the CNN, LSTM, and 9 traditional machine learning solutions. The experimental tests that were conducted by the authors showed that the CNN-LSTM solution performed better than other approaches and yield an accuracy of 98.3% and an F1-Score of 0.914.

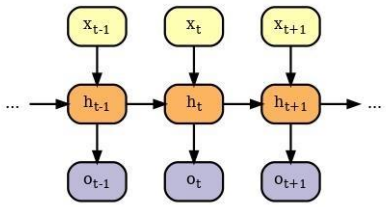

FIGURE 1: Structure of a typical Recurrent Neural Network.

C. PAPER ORGANIZATION

The rest of the paper is organized as follows. Section II provides the backgrounds and details of the LSTM and our spam Transformer approaches. Concretely, Section II-A introduces the architecture of RNN, followed by one of its most successful variant LSTM in Section II-B. We then introduce Sequence-to-Sequence in Section II-C, attention mechanism in Section II-D, and the original version of Transformer for translation tasks in Section II-E. Furthermore, Section III discusses the modified spam Transformer that we proposed in detail. Afterward, Section IV demonstrates the experiment designs, results and analysis. Finally, we conclude in Section VI and describes the future work in Section VII.

II. DEEP LEARNING APPROACHES

While the traditional machine learning techniques do perform well in many fields, they are still much interference or guidance from human specialists required when people try to apply these technologies to address problems. For instance, extracting and representing the features from data is always a challenging but indispensable work for machine learning scientists. In another word, the inadequate capacity of many traditional machine learning classifiers is a major limitation to a more effective and massive application. However, many deep learning techniques are able to not only learn much more amount of features but also extract more higher-level features that are formed by the composition of lower-level features. With an effective training process, the deep learning techniques are more capable to consume and make good use of a large amount of data and thus perform better especially in coping with difficult jobs compared to the traditional machine learning approaches.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

A. RECURRENT NEURAL NETWORK

As is known to all, shuffling the order of words in a sentence can severely influence the meaning of the entire sentence, which could potentially turn a legitimate message into spam messages, and vice versa. Therefore, in many Natural Language Process (NLP) problems, the order of words is no less important than the words themselves. To address this problem, we need a new kind of model that is capable to effectively learn from prior knowledge to improve the understanding of the data. Although the classical feed-forward neural network is a powerful deep learning technique that generally works well in many areas, it cannot utilize the information from the past. Derived from the feed-forward neural network, recurrent neural network (RNN) [23] has the ability to reuse the saving information at the time of processing input values. Additionally, unlike the traditional feed-forward neural network supports only the input sequence with a fixed length, RNN is capable to handle the input sequence with different length.

The Fig. 1 shows the typical structure of RNN models, with the input sequence, output sequence, and the hidden layers at time t are represented by x_t, o_t, and h_t respectively. At time t, the current hidden layers state h_t is calculated based on the current input sequence x_t and the last hidden layers h_{t-1}. After the calculation of h_t is finished, the output at the current time step o_t is generated and the hidden layers state h_t will get involved in the calculation at the next time step t+1. Unlike the normal neural network, where the neurons in the same layer of the hidden layers are independent of each other, RNN models usually allow the data flows within the same layer. In another word, connections between neurons in the same layers or even self-connections are allowed generally allowed in RNN based models.

A major advantage of RNN models is that they are able to utilize the information from previous input and apply it at the current time, which is significantly useful in NLP problems since the context can help us understand the sentence better. However, a major drawback of the vanilla RNN is the vanishing and exploding gradients [24]. In back-propagation training process, the vanishing gradients refers to gradients go exponentially close to 0, while the exploding gradients refers to the gradients go exponentially increase. The vanishing and exploding gradients are usually caused by the multiplication of multiple derivatives in training process. Although there are several approaches [25] existing to address the vanishing and exploding gradients problem, in practice, it is still difficult for vanilla RNN to memorize and learn the features from long distance, which is described as long-term dependencies problem. In order to deal with the long-term dependencies problem, many researchers have proposed multiple variants of RNN, such as the Long Short-Term Memory (LSTM) [26], the Gated Recurrent Unit (GRU) [27], and the Clockwork RNN (CW-RNN) [28].

B. LONG SHORT-TERM MEMORY

The Long Short-Term Memory (LSTM) is a famous variant of RNN. The main idea of the LSTM is the introduction of gate units, which are the structures that can determine to keep or discard the current information. A typical LSTM network consists of multiple memory cells, and each memory cell is formed by an input gate, a forget gate, and an output gate. In LSTM, at time t, the state of a memory cell c_t is calculated based on the input x_t and the last hidden state h_{t-1}. The state of input gate, output gate, and forget gate at time t are represented as i_t, o_t, f_t, respectively. Therefore, the LSTM transition functions are defined as follows [29]:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ q_t &= tanh(W_q \cdot [h_{t-1}, x_t] + b_q) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot q_t \\ h_t &= o_t \odot tanh(c_t) \end{aligned} \tag{1}$$

The σ denotes the sigmoid function, and the operator $\odot$ denotes the element-wise multiplication. The sigmoid function is a logistic function with the returning value between 0 and 1. The sigmoid function is defined as follow:

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{2}$$

When the output of a gate unit is close to 1, the information is more likely to be memorized. On the contrary, a returning value close to 0 from a gate unit means that the information should not be kept. The input gate i_t is the gate unit that controls how much information should be stored at this time. The forget gate f_t is responsible to determine to what extent the memory from the last time c_{t-1} should be kept at time t. The output gate o_t at time t is designed to be used in the computation of the output (hidden state) based on the memory cell state.

In our LSTM approach for SMS spam detection, the input message embedding is fed into an LSTM network as an input sequence. Meanwhile, the LSTM network saves the important features and outputs a sequence with the same length as the input sequence. The output sequence is then fed into a feed-forward fully connected layer with a single neuron since SMS spam detection is a binary classification problem. Finally, a sigmoid function is applied to the output of the single neuron to produce a final prediction.

C. SEQUENCE-TO-SEQUENCE MODELS

Sequence-to-sequence (Seq2Seq) [30] was introduced in 2014 by Sutskever et al. aiming to find a mapping between two sequences for translation tasks. Seq2Seq models employed the Encoder-Decoder architecture, which consists of an encoder stack, a hidden state, and a decoder stack. Fig. 2 presents a typical Encoder-Decoder architecture. The encoders take the input sequence and produce a hidden state with critical information, which is consumed by the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

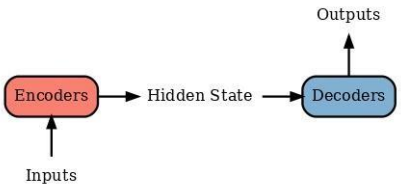

FIGURE 2: Structure of Encoder-Decoder architecture

decodes to generate the output sequences. One of the crucial advantages of the Encoder-Decoder architecture is that the input sequence and output sequence can be different in terms of size or format, which provides much more flexibility and possibility. In reality, the Seq2Seq models have been proved themselves in language translation [30], Speech Recognition [31], and Video to Text [32]. Undoubtedly, Seq2Seq architecture is designed to fit translation tasks exceptionally well, since it can extract the relationship between the sequences in one language and the sequences in a different language. The vanilla version of the Seq2Seq model proposed in 2014 choose LSTM as both encoder and decoder, because LSTM has the ability to successfully learn on data with long-term dependencies [30].

D. ATTENTION MECHANISM

The main purpose of the attention mechanism is to find out the most important part from the input sequence. Concretely, the attention mechanism produces weights that represent the importance of the elements based on their correlation with the context. The attention mechanism makes it possible to focus on the key elements.

In [33], the attention mechanism was introduced as an improvement of the RNN Encoder-Decoder model hidden state in Neural Machine Translation (NMT). The most important contribution of the attention mechanism in NMT is that it computes the weights based on all the hidden states generated by the encoder, and the decoder consumes the weighted combination of all the hidden states instead of focusing only on the latest one. The introduction of the attention mechanism greatly boosts the performance of NMT.

There are also other forms of attention mechanism proposed. In [34], the attention mechanism is applied to the field of computer vision by Xu et al., and they also proposed two different approaches of attention named "soft attention" and "hard attention". In [35], Luong et al. proposed global attention and local attention. The global attention is similar to the model of Bahdanau et al. in [33] with a simpler architecture, while the local attention is a combination of soft and hard attention from Xu et al. in [34].

E. THE TRANSFORMER MODEL

The Transformer [3] model is a sequence-to-sequence (Seq2Seq) model that was proposed in 2017 by Vaswani et al., as an approach to English-German and English-French translation tasks. Compared to those previous Seq2Seq models, the main innovation of Transformer is that it completely relies on the attention mechanism to efficiently learn from the most informative elements [36].

Though LSTM and some other RNN variants were proved to perform well as encoders and decoders in Seq2Seq based models, the high training consumption of recurrent models becomes a significant limitation. At time t, the computation of hidden state h_t relies on the previous hidden state h_{t-1}, which is the sequential computation nature of recurrent models. This sequential computation nature prevents the computing of RNN variants from parallelization, leading to the limitation on computational efficiency during the training process.

In order to address the computational efficiency limitation of RNN variants, the Transformer uses only multi-head attention mechanism instead of RNN variants as encoders and decoders. This not only greatly reduces the cost of training through parallelization, but also surprisingly improves the performance in translation tasks as is mentioned in [3].

In Transformer, the attention function takes a query Q and a set of key-value pairs (K, V) as input, and computes the weighted sum of values as output, where the weights are calculated based on the queries and keys. Particularly, Scaled Dot-Product Attention is used in Transformer as the attention function. The Scaled Dot-Product Attention is the dot-product attention [35] with a scaling factor of $\frac{1}{\sqrt{d_k}}$, which aims to counteract the massive growth of dot-product when dimensions of queries and keys d_k is large.

Another important innovation of Transformer is the Multi-Head Attention. In the previous practice, the attention is directly performed on the queries, keys, and values, where their dimension is d_{model}. In this way, there is only a single attention function calculated at one turn. However, Transformer finds an effective way to apply multiple attention functions at once. Specifically, the queries, keys, and values are sent to some different learned linear layers to be projected h times to the dimension of d_k, d_k, and d_v, respectively. In another word, the projection linear layers are individually learned, and output projections have dimensions of d_k, d_k, and d_v, where $d_k = d_v = d_{model}/h$. After that, a number of h attention functions are performed in parallel on these projected queries, keys, and values, resulting in h different output values. Finally, all these h values are concatenated together and then projected back to a dimension of d_{model}. The entire process of the attention mechanism in the Transformer

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

is defined as follows [3]:

$$
\begin{aligned}
&Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \\
&MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \\
&head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)
\end{aligned}
\tag{3}
$$

The W_i^Q, W_i^K, and W_i^V are parameters matrices in linear projection layers, where they are used to project d_{model}-dimension queries, keys, and values to d_k, d_k, and d_v dimension, respectively. In both vanilla Transformer and our modified Transformer for SMS spam detection, $d_k = d_v = d_{model}/h$.

In RNN, the computation of the hidden states is based on the previous states, making it available to learn from the order of words naturally. However, there is no recurrent or convolutional structure in Transformer. Therefore, Transformer introduces a positional encoding function based on $sine$ and $cosine$ functions of different frequencies.

In vanilla Transformer model designed for language translation tasks, source language texts and shifted right target language texts are first sent to embedding layers as input sequence and output sequence. Secondly, positional information is injected into the input and output sequence in the positional encoding layer. After that, the input and output sequence is fed into encoders and decoders, respectively. Then, the Multi-Head Attention layers and fully-connected feed-forward layers, combined as a single encoder or decoder, produce the output of dimension of d_{model}. The results of decoders are passed to a linear layer. Finally, the softmax function is performed on the output of the linear layer, producing the translation in the target language.

III. PROPOSED MODIFIED TRANSFORMER MODEL FOR SMS SPAM DETECTION

In Fig. 3, the main architecture of the modified Transformer model for SMS spam detection is described. In order to apply the Transformer model to the SMS spam detection task, two major modifications are done to the vanilla Transformer model, which is described in Section III-A and Section III-B, respectively. After that, several implementation details are discussed.

A. MEMORY

The first modification for the SMS spam detection task is the introduction of memory. Since there is no output sequence (target sequence) in the SMS spam detection task, we used a list of trainable parameters named "memory" to be the substitute for output sequence embedding. The length of the memory is a configurable hyper-parameter. Each element of the memory is a vector of dimension d_{model} so that it can be adapted to the Transformer model without any extra projection. In other words, the memory is a matrix of dimension $len_{memory} \times d_{model}$. The output embedding

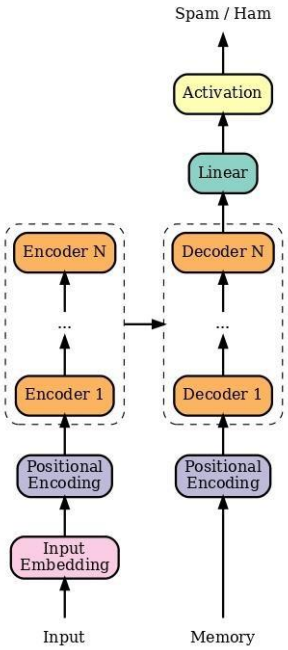

FIGURE 3: Structure of proposed modified Transformer model for SMS spam detection. The input messages embeddings and memory (trainable parameters) are positional encoded, respectively. Then, the processed message vectors are passed to encoder layers, where the self-attention is performed. The results of encoder layers are passed to decoder layers. In decoder layers, the Multi-Head Attention is executed based on the results of encoder layers and the processed memory. Then, the decoded vectors are sent to some fully-connected linear layers, followed by a final activation function for classification.

layer in the original Transformer model is also removed since there are no target sequence texts anymore to be mapped to numeric vectors. Similar to the output sequence in the vanilla Transformer model, the positional information is injected into the memory at the positional encoding layer before being fed into decoders.

During the training process, the parameters of memory are trained, and the memory matrix is expected to contain the important information that can help to predict whether or not a message is a spam. Therefore, in the decoders of the modified spam Transformer model, with the help of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

attention mechanism, the memory can contribute to locate the significant part of the output sequence of the encoder stack that summarized the message, and eventually help to classify the spam SMS messages.

B. LINEAR LAYERS AND FINAL ACTIVATION FUNCTION

The second modification is the final activation function. In the vanilla Transformer, the dimension of outputs of decoder layers is $T \times d_{model}$, where T is the target sequence length and d_{model} is the model size (number of features). Therefore, intuitively, it is a promising approach to use the linear layers to map the output to a vector that has the same dimension as the number of words in the dictionary and apply a softmax function on the vector to find the closest candidate word from the dictionary.

However, the SMS spam detection task is a binary classification problem. Therefore, to convert the output from the decoder stacks with dimension d_{model} into a single probability of the message being spam, the linear layers after the decoders are also modified. Instead of mapping the output of the decoder stack to a vector, the linear layer in the modified Transformer model for SMS spam detection has only one single neuron in the last layer. Thus, the outputs of the decoder stack are converted into a single numeric probability value.

Additionally, the final activation function needs to be replaced with a function that can map the result to a binary outcome. Thus, in the modified Transformer for SMS spam detection, a sigmoid function, which is defined in Equation (2), as the final activation function, is applied to the output of the linear layers after decoders, generating a binary result that predicts whether or not the message is spam.

C. DROPOUT

Dropout [37] is a powerful technique published by Hinton et al. in 2012 in order to prevent over-fitting in a large feed-forward neural network. Concretely, the Dropout refers to randomly omit some nodes in those large feed-forward layers on each specific training case. The modified spam Transformer model that we proposed employs multiple feed-forward layers. Thus, the Dropout technique is also implemented in the feed-forward layers of our spam Transformer model. Besides, the Dropout technique is also used in positional encoding and calculation of attention function.

D. BATCHES AND PADDING

During each epoch of training on our proposed models, the whole training set is divided into multiple batches. As the length of the message with the same batch should be the same, some padding words (empty words) should be added into the shorter message vectors, interfering with the detection to some extent. Therefore, the algorithm of dividing the training set into batches is designed to minimize the padding words. Specifically, the training data is sorted by the message length first, and the batches are created to minimize the padding words based on the sorted messages.

Admittedly, adding padding words may pose a negative influence on the model. However, using batch has been proved to be a good idea for model training as it increases the training speed extraordinarily. In fact, a larger batch size accelerates a ton for the training speed. Additionally, the negative influence of padding words is addressed by minimizing the use of padding words. Besides, the padding masks are also passed into the model along with the training batches so that the Transformer model can ignore the padding words during training.

E. OPTIMIZATION AND LEARNING RATE

The gradient descent is employed to optimize our modified spam Transformer model. The main idea of the gradient descent algorithm is to minimize the loss function of the model by updating the parameters along the opposite way of the gradient to the loss function, where the gradient is the partial derivatives of the loss function of the parameter. There are plenty of variant optimizers of gradient descent. We use the AdamW [38] optimizer for our proposed modified spam Transformer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$. Learning rate is a critical hyper-parameter in machine learning. It is defined as the step size of updating parameters, which basically represents the speed of learning of the model. Having the learning rate set too high will lead to the situation that the model fails to locate the best parameters (weights and biases), while a learning rate that is too small sticks the model around the local optimal point rather than finding a better parameter solution. For the modified spam Transformer model, the same way of determining the learning as mentioned in [3] is utilized. The learning rate lr first increases linearly until reaching the $warmup\_steps$ steps and then decreases proportionally to the square root of the step numbers. Concretely, we used $warmup\_steps = 8000$.

F. DATAFLOW OF MODIFIED TRANSFORMER

As is shown in Figure 3, the input messages are first converted into word embeddings using the Glove model. Following this, the memory (trainable parameters) and the embeddings of the input sequence are positionally encoded, respectively. Then, the processed message vectors are passed to encoder layers, where the multi-head self-attention is performed and the important parts of the input sequence are given larger weights. The results of encoder layers are passed to decoder layers. In decoder layers, the multi-head self-attention is computed on the memory. After that, the multi-head attention is executed based on the results of encoder layers and the processed memory. Finally, the decoded vectors are sent to some fully-connected linear layers, followed by a final activation function for classification.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

TABLE 1: The statistics of two datasets

	SMS Spam Collection v.1	UtkMl's Twitter
Spam	747	5815
Ham	4827	6153
Total	5574	11968

TABLE 2: The confusion matrix

	Predicted Spam	Predicted Ham
Actual Spam	True Positive (TP)	False Negative (FN)
Actual Ham	False Positive (FP)	True Negative (TN)

IV. EXPERIMENT

A. DATASETS

In the experiments, two different datasets are utilized. The first dataset is SMS Spam Collection v.1 [13] dataset, which is labeled SMS messages dataset collected for mobile phone message research. The second one is UtkMl's Twitter Spam Detection Competition (UtkMl's Twitter) [39] from Kaggle. Table 1 shows the overview statistics of the two datasets.

Although the Twitter posts are not precisely the same as the SMS messages, they are still in some ways common. For instance, they both have approximately less than 100 words. People tend to use more casual language and abbreviations in both Twitter posts and SMS messages. Therefore, UtkMl's Twitter dataset can also be used to test our model. Besides, we can also analyze the extensibility of our model by comparing the performance of our model on these two datasets.

In comparison with SMS Spam Collection v.1 [13] dataset, UtkMl's Twitter dataset contains more data in both spam and ham classes. Besides, UtkMl's Twitter dataset is balanced since the number of spam messages and ham messages are approximately equal. In terms of the language, although they are a lot of casual language and abbreviation used in both datasets, casual language and abbreviation appear more frequently in UtkMl's Twitter dataset. The reason for this observation may be the feature of the Twitter posts. Alternatively, it could also because of the date that the dataset was collected, as SMS Spam Collection v.1 was published in 2011.

B. EVALUATION MEASURES

In order to evaluate the performance of the proposed modified spam Transformer model, some metrics such as accuracy, precision, recall, and F1-Score are used in the experiments. All these metrics are calculated based on the confusion matrix. As is mentioned in the previous section, the spam messages in the SMS Spam Collection v.1 dataset are significantly less than the ham messages, which means that the dataset is unbalanced. Therefore, the accuracy is not sufficient as a measurement to evaluate the performance of the proposed model, and the F1-Score is employed in the experiments. The accuracy, precision, recall, and F1-Score is defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FP + FN} \tag{4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

The precision, also known as the positive predictive value, represents the percentage of the predicted positive cases that are actually positive, meaning the possibility that the classifier is correct given that it predicts positive. The recall, also known as sensitivity, denotes the number of true positives instances divided by the number of actual positive instances, which can also be described as the percentage of the positive cases that are identified successfully. The F1-Score is the harmonic mean of precision and recall, which measures the performance of a classifier in terms of precision and recall in a balanced way.

C. DATA SPLITTING

For the traditional machine learning approaches, the data is divided into training set (70%), and test set (30%). For the LSTM and our proposed modified spam Transformer model, the data is split into training set (50%), validation set (20%), and test set (30%), where the validation set is used after each epoch of training to help us select the best model and perform early stopping to avoid over-fitting.

D. DATA PRE-PROCESSING

The textual messages in the dataset are first tokenized. Tokenization refers to the task of splitting textual into meaningful words. Specifically, the SpaCy [40] library is employed for data pre-processing in order to tokenize the data.

After that, the numeric representation vectors (word embeddings) are calculated based on the textual messages. There are two major methods of calculating representation vectors are employed in our experiments.

- **TF-IDF Representation**: The TF-IDF (Term Frequency–Inverse Document Frequency) is a widely-used numerical statistic in NLP. It is designed to reflect the importance of a word to a document in the given text corpus. The Term Frequency (TF) is defined as the number of times that a term occurs in a document. A larger TF means the term is referred for more times in the given document, showing that the term is more relevant to the document. There are multiple different means to weigh the TF in order to adapt it in different applications. In our experiment, we use the raw count of the term in the document as the TF. The Inverse Document Frequency (IDF) is a value to qualify the specificity of a term, which is normally defined as the logarithmically scaled inverse fraction of the number

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

of documents that contain the term. In another word, when a term occurs in a great number of documents, the IDF is numerically low, leading to a low TF-IDF. For instance, the term "the" occurs in almost every English document, leading to a document frequency of almost 1 (100% of the documents in the corpus contain the term "the"). Thus, the IDF of "the" is close to 0, which means that its importance to any documents in the corpus is low.

- **GloVe Representation**: GloVe [41] is an unsupervised learning algorithm for obtaining vector representations for words. The main idea is to map words into a meaningful space where the distance between words is related to semantic similarity. GloVe produces a vector space with a meaningful substructure, and it can also find the relations like synonyms between words.

In our experiments, for the deep learning approaches such as LSTM and our proposed spam Transformer model, the GloVe model is employed to create representation vectors for them. Specifically, in our experiments, we used the "glove.840B.300d", a pre-trained model with 2.2 million words in the dictionary that converts textual data into 300-dimensional vectors. For benchmark machine learning algorithms, although the vectors generated by GloVe model have more dimensions and theoretically contain more information, presumably due to the limitation of traditional machine learning classifiers, the TF-IDF representation performs better in practice. Therefore, TF-IDF representation is used for calculating representation vectors in benchmark machine learning algorithms.

E. LOSS FUNCTION

The loss function we used for deep learning approaches including LSTM and modified spam Transformer is Binary Cross Entropy function, which is defined as follow:

$$l(x_i, y_i) = -w_i[y_i \cdot logx_i + (1 - y_i) \cdot log(1 - x_i)] \quad (8)$$

The weight w_i is the rescaling factor for loss. Since the SMS Spam Collection v.1 is unbalanced, where spam messages are severely less than ham (legitimate) messages, a larger weight is given to the actual spam messages to counteract the negative effect of the unbalanced dataset. The rescaling weight is calculated based on the ratio between the number of ham messages and spam messages.

F. MODEL TRAINING

We trained our experiment models on NVIDIA GeForce RTX 3090 GPU. For the machine learning classifiers, the experiments are performed on the Scikit-learn 0.24.0 [42] environment. For deep learning approaches like LSTM and spam Transformer model, the experiments are conducted on the Ubuntu 20.04 LTS, CUDA 11.1, and PyTorch 1.7.1 [43] environment. The early stopping technique is implemented to fight against the over-fitting. Besides, we also trained and tested the CNN-LSTM SMS spam detection model proposed in [22] on both datasets as a benchmark to evaluate our modified spam Transformer model.

TABLE 3: Optimized hyper-parameters for LSTM

Hyper-parameter	SMS Spam Collection v.1	UtkMl's Twitter
LSTM units per layer	100	100
LSTM layers	1	2
LSTM Dropout	0.1	0.1
Linear Dropout	0.4	0.1

TABLE 4: Initial and optimized hyper-parameters for modified spam Transformer on SMS Spam Collection v.1

Hyper-parameter	Initial	Optimized
Encoder layers	6	6
Decoder layers	6	6
Model size	512	600
Feed-forward size	2048	1200
Attention head size	8	8
Transformer Dropout	0.1	0.01
Linear Dropout	0.1	0.05

G. HYPER-PARAMETERS TUNING

In order to tune the models and find the best hyper-parameters set, the Ray Tune [44] library is employed. The Ray Tune is a hyper-parameter tuning extension tool that supports multiple machine learning frameworks. Given a candidate hyper-parameters set, the Ray Tune can find the optimized hyper-parameters set by training multiple models with different settings and comparing the results automatically. In our experiments, with the help of the Ray Tune, we first explored optimal settings for the overall architectural hyper-parameters such as Encoder layers, Decoder layers, and Model size. After that, other hyper-parameters such as the rate of dropout and Feed-forward layer size are tuned under the candidate optimal model settings.

For the LSTM model, the optimized parameters on both datasets are shown in Table 3. For our modified spam Transformer model on SMS Spam Collection v.1, Table 4 presents the initial hyper-parameters that we started from and the optimized values when the better result was achieved after tuning. Table 5 demonstrates the initial as well as the optimized hyper-parameters of modified spam Transformer on UtkMl's Twitter dataset.

V. RESULTS AND ANALYSIS

TABLE 5: Initial and optimized hyper-parameters for modified spam Transformer on UtkMl's Twitter

Hyper-parameter	Initial	Optimized
Encoder layers	6	2
Decoder layers	6	4
Model size	600	600
Feed-forward size	1200	1200
Attention head size	8	8
Transformer Dropout	0.01	0.01
Linear Dropout	0.05	0.1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

A. EVALUATION

We demonstrate the performance of the modified spam Transformer model by comparing it on two datasets with some other typical spam detection classifiers, including Logistic Regression, Naïve Bayes, Random Forests, Support Vector Machine (classifier), and Long Short-Term Memory. Besides, for the SMS Spam Collection v.1 dataset, we also compare our models with the CNN-LSTM approaches in [22], since they aim to solve the same problem on the same dataset with us.

Table 6 summarizes the results on SMS Spam Collection v.1 dataset. For accuracy, our modified spam Transformer model achieved the best value of 98.92%. Concerning precision, the best score was from the Random Forests classifier with a value of 1.0, and our proposed spam Transformer got a value of 0.9781. When it comes to recall, the optimal result came from the spam Transformer model with a value of 0.9451, and the same value came from the Naïve Bayes classifier as well. Finally, in terms of F1-Score, our spam Transformer also achieved the best value of 0.9613. The experiment of CNN-LSTM [22] that was conducted by Ghourabi et al. on the same dataset, are also included in Table 6. In Table 8, we demonstrate the confusion matrix of all the approaches that we tested in the experiments on SMS Spam Collection v.1 dataset.

Table 7 summarizes the results on UtkMl's Twitter dataset. The modified spam Transformer model outperformed all other candidates in all four aspects that we tested with the values of 87.06%, 0.8746, 0.8576, and 0.8660 on the accuracy, precision, recall, and F1-Score, respectively. The confusion matrix of the modified spam Transformer model on UtkMl's Twitter is presented in Table 9.

B. ANALYSIS

Although the experimental results show an improved performance of the proposed spam Transformer model compared to other candidates, the false predictions also indicate the drawback of the proposed model. We analyzed the content of the false prediction samples including false positive and false negative samples and found that there were a great number of the UNK marks in the data passed to the model, which is produced because the words are never seen in the training data. In other words, the unknown words obstruct the model from understanding the messages. Besides, the SMS messages are usually short, which increases the influence of every single word and makes the unknown words more influential. Actually, due to the unknown words, the model did not have enough information to detect spams in many false prediction cases.

Though our proposed model performs better than other candidate algorithms on UtkMl's Twitter dataset, the results are still not as good as that in case of SMS Spam Collection v.1 dataset. From our observation, the major cause is also the unknown words. Compared to SMS Spam Collection v.1 dataset, there are more casual language and abbreviations in UtkMl's Twitter dataset, which may be caused by the feature of Twitter posts or the date of collection of the dataset, as is discussed in Section IV-A. Therefore, the negative influence from casual language and abbreviation is more severe on UtkMl's Twitter dataset, and that is the major cause of more unknown words and eventually worse performance from our perspective.

TABLE 6: Results obtained on SMS Spam Collection v.1

Classifiers	Accuracy	Precision	Recall	F1-Score
Logistic Regression	98.56%	0.9863	0.9113	0.9473
Naïve Bayes	98.38%	0.9411	**0.9451**	0.9431
Random Forests	97.90%	**1.0**	0.8535	0.9209
SVM	98.62%	0.9908	0.9113	0.9494
LSTM	98.56%	0.9570	0.9409	0.9489
CNN-LSTM [22]	97.66%	0.9159	0.9198	0.9178
Spam Transformer	**98.92%**	0.9781	**0.9451**	**0.9613**

In addition, Table 8 and Table 9 show the excellent robustness of our model to classify both the spams and hams effectively on no matter balanced (UtkMl's Twitter) or unbalanced (SMS Spam Collection v.1) datasets.

TABLE 7: Results obtained on UtkMl's Twitter

Classifiers	Accuracy	Precision	Recall	F1-Score
Logistic Regression	81.51%	0.8441	0.7615	0.8007
Naïve Bayes	83.21%	0.8316	0.8221	0.8269
Random Forests	79.28%	0.8449	0.7044	0.7683
SVM	82.68%	0.8681	0.7604	0.8107
LSTM	81.04%	0.8594	0.7307	0.7898
CNN-LSTM [22]	79.45%	0.8182	0.7438	0.7792
Spam Transformer	**87.06%**	**0.8746**	**0.8576**	**0.8660**

VI. CONCLUSION

In this paper, we proposed a modified Transformer model that aims to identify SMS spam. We evaluated our spam Transformer model by comparing it with several other SMS spam detection approaches on the SMS Spam Collection v.1 dataset and UtkMl's Twitter dataset. The experimental results show that, compared to Logistic Regression, Naïve Bayes, Random Forests, Support Vector Machine, Long Short-Term Memory, and CNN-LSTM [22], our proposed spam Transformer model performs better on both datasets.

On the SMS Spam Collection v.1 dataset, our spam Transformer has a better performance in terms of accuracy, recall, and F1-Score compared to other classifiers. Specifically, our modified spam Transformer approach accomplished an exceeding result on F1-Score.

Additionally, on the UtkMl's Twitter dataset, the results from our modified spam Transformer model demonstrate its improved performance on all four aspects in comparison to other alternative approaches mentioned in this paper. Concretely, our spam Transformer does exceptionally well on recall, which contributes to a distinct F1-Score.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

TABLE 8: The confusion matrices on SMS Spam Collection v.1

	Logistic Regression		Naïve Bayes		Random Forests		SVM		LSTM		CNN-LSTM		Spam Transformer	
Act. \ Pred.	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**
Spam	216	21	224	13	204	35	216	21	211	26	218	19	224	10
Ham	3	1433	14	1422	0	1434	2	1434	17	1419	20	1416	5	1431

TABLE 9: The confusion matrices on UtkMl's Twitter

	Logistic Regression		Naïve Bayes		Random Forests		SVM		LSTM		CNN-LSTM		Spam Transformer	
Act. \ Pred.	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**	**Spam**	**Ham**
Spam	1332	417	1438	311	1232	517	1330	419	1278	471	1301	448	1500	249
Ham	246	1592	291	1547	226	1612	202	1636	209	1629	289	1549	215	1623

VII. FUTURE WORK

Although the experimental results in this paper have shown an improvement of our proposed spam Transformer model in comparison with some previous approaches on SMS spam detection, we still believe that there is great potential in the model we proposed.

Firstly, since our current two datasets contain only thousands of messages, in the future, we plan to extend our spam Transformer model to a larger dataset with more messages or even other types of content, for the purpose of better performance.

Besides, in our proposed model, we flattened the outputs from decoders and applied linear fully-connected layers before applying the final activation function and getting the prediction. We believe that some dedicated designs or implementations instead of simple flattening and linear layers could absolutely boost the performance, which would be one of the most important future works.

Additionally, although the experimental results show that our modified model based on the vanilla Transformer performs well on SMS spam detection and confirms the availability of the Transformer on this problem, the model is still far from optimal. There are some improved models based on the Transformer with more complex architecture such as GPT-3 [4] and BERT [5] that could be explored in the future. Specifically, the BERT seems to be a promising starting point of future work as it has fewer features and is easier to be fine-tuned.

Finally, as is discussed in Section V-B, the proposed model is severely influenced by the unknown words in many cases of false prediction. To address this problem, more data preprocessing techniques could be applied. For instance, a larger vocabulary with more words could be a good option, and some semantic operations such as replacing unknown words with their synonyms could also be explored. Besides, there are some other data-preprocessing and feature extraction techniques that could be done, such as the extraction and analysis of the abbreviation, URLs, tags, or emoji in data.

REFERENCES

[1] P. K. Roy, J. P. Singh, and S. Banerjee, "Deep learning to filter SMS Spam," *Future Generation Computer Systems*, vol. 102, pp. 524–533, 2020.

[2] G. Jain, M. Sharma, and B. Agarwal, "Optimizing semantic LSTM for spam detection," *International Journal of Information Technology (Singapore)*, vol. 11, no. 2, pp. 239–250, 2019.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5999–6009.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," *arXiv*, may 2020. [Online]. Available: http://arxiv.org/abs/2005.14165

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT)*, vol. 1, pp. 4171–4186, oct 2019.

[6] G. Sonowal and K. S. Kuppusamy, "SmiDCA: An Anti-Smishing Model with Machine Learning Approach," *The Computer Journal*, vol. 61, no. 8, pp. 1143–1157, aug 2018.

[7] J. W. Joo, S. Y. Moon, S. Singh, and J. H. Park, "S-Detector: an enhanced security model for detecting Smishing attack for mobile computing," *Telecommunication Systems*, vol. 66, no. 1, pp. 29–38, sep 2017.

[8] S. Mishra and D. Soni, "Smishing Detector: A security model to detect smishing through SMS content analysis and URL behavior analysis," *Future Generation Computer Systems*, vol. 108, pp. 803–815, jul 2020.

[9] C. Li, L. Hou, B. Y. Sharma, H. Li, C. S. Chen, Y. Li,

X. Zhao, H. Huang, Z. Cai, and H. Chen, "Developing a new intelligent system for the diagnosis of tuberculous pleural effusion," *Computer Methods and Programs in Biomedicine*, vol. 153, pp. 211–225, jan 2018.

[10] T. K. Ho, "Random decision forests," in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, vol. 1, 1995, pp. 278–282.

[11] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[12] M. Gupta, A. Bakliwal, S. Agarwal, and P. Mehndiratta, "A Comparative Study of Spam SMS Detection Using Machine Learning Classifiers," in *2018 11th International Conference on Contemporary Computing (IC3)*, 2018.

[13] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, "Contributions to the study of sms spam filtering: New collection and results," in *Proceedings of the 11th ACM Symposium on Document Engineering*, 2011, p. 259–262.

[14] A. K. Jain and B. B. Gupta, "Rule-Based Framework for Detection of Smishing Messages in Mobile Environment," in *Procedia Computer Science*, vol. 125, 2018, pp. 617–623.

[15] W. W. Cohen, "Fast Effective Rule Induction," in *Machine Learning Proceedings*, 1995, pp. 115–123.

[16] J. Cendrowska, "PRISM: An algorithm for inducing modular rules," *International Journal of Man-Machine Studies*, vol. 27, no. 4, pp. 349–370, 1987.

[17] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[18] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT'2010*. Physica-Verlag HD, 2010, pp. 177–186.

[19] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations*, 2013.

[20] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[21] H. Liu and P. Singh, "ConceptNet - a practical commonsense reasoning tool-kit," *BT Technology Journal*, vol. 22, no. 4, pp. 211–226, 2004.

[22] A. Ghourabi, M. A. Mahmood, and Q. M. Alzubi, "A hybrid CNN-LSTM model for SMS spam detection in arabic and english messages," *Future Internet*, vol. 12, no. 9, p. 156, 2020.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[24] Y. Bengio, P. Simard, and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[25] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *30th International Conference on Machine Learning (ICML)*, no. PART 3, pp. 2347–2355, 2013.

[26] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[28] J. Koutník, K. Greff, F. Gomez, and J. Schmidhuber, "A Clockwork RNN," *31st International Conference on Machine Learning (ICML)*, vol. 5, pp. 3881–3889, 2014.

[29] C. Zhou, C. Sun, Z. Liu, and F. C. M. Lau, "A C-LSTM Neural Network for Text Classification," *arXiv:1511.08630*, 2015.

[30] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *Advances in Neural Information Processing Systems*, vol. 4, no. January, pp. 3104–3112, 2014.

[31] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition," in *Proc. Interspeech 2017*, 2017, pp. 939–943.

[32] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence – video to text," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 4534–4542.

[33] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[34] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *32nd International Conference on Machine Learning (ICML)*, vol. 3, 2015, pp. 2048–2057.

[35] M. T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[36] E. S. D. Reis, C. A. D. Costa, D. E. D. Silveira, R. S. Bavaresco, R. D. R. Righi, J. L. V. Barbosa, R. S. Antunes, M. M. Gomes, and G. Federizzi, "Transformers aftermath," *Communications of the ACM*, vol. 64, no. 4, pp. 154–163, apr 2021.

[37] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv:1207.0580*, 2012.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081479, IEEE Access

[38] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," *arXiv:1711.05101*, 2017.

[39] UtkML, "UtkML's Twitter Spam Detection Competition | Kaggle."

[40] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020. [Online]. Available: https://doi.org/10.5281/zenodo.1212303

[41] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035.

[44] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A Research Platform for Distributed Model Selection and Training," *arXiv preprint arXiv:1807.05118*, 2018.

XIAOXU LIU completed his Bachelor of Computer Science and Technologies degree at Nanjing University of Posts and Telecommunications, China, and started his Master of Computer Science program in 2019 at the University of Ottawa, Canada. His research interests include machine learning, deep learning, and natural language processing.

HAOYE LU joined the University of Ottawa, Canada in 2013. He received his Bachelor of Science Joint degree in Computer Science and Mathematics in 2017 and his Master of Computer Science degree in 2019. He currently works as a research associate. His research interests include artificial intelligence and network structures.

AMIYA NAYAK received his B.Math. degree in Computer Science and Combinatorics and Optimization from University of Waterloo, Canada, in 1981, and Ph.D. in Systems and Computer Engineering from Carleton University, Canada, in 1991. Currently, he is a Full Professor at the School of Electrical Engineering and Computer Science at the University of Ottawa. His research interests include software-defined networking, mobile computing, wireless sensor networks, and vehicular ad hoc networks.

He has over 17 years of industrial experience in software engineering, avionics and navigation systems, simulation and system level performance analysis. He is now in the Editorial Board of IEEE Internet of Things Journal, IEEE Transactions on Vehicular Technology, IEEE Open Journal of the Computer Society, Future Internet, and International Journal of Distributed Sensor Networks. He has served in the Editorial Board of several journals, including IEEE Transactions on Parallel & Distributed Systems, International Journal of Parallel, Emergent and Distributed Systems, Journal of Sensor and Actuator Networks, and EURASIP Journal of Wireless Communications and Networking.