Logo

Description automatically generated

**Special Topics in Cybersecurity**

**Durham College**

**Faculty Of Science, Engineering & Information Technology (SEIT)**

Group Assignment – Assignment 3: Designing an AI based Anti-phishing product based on a Large Language Model (LLM)

Course Code: INFT 1204

Due Date: August 13, 2023

Professor: Saber Amini

**Group:**

Hammed Tijani (ID: 100902204), Tehara Moonemalle (ID: 100903984),

Steffannie Egbuziem (ID: 100896975), Romoyne Watson (ID: 100895321).

Table of Contents

[INTRODUCTION 3](#_Toc142853153)

[LANGUAGE MODEL 3](#_Toc142853154)

[LIMITATIONS AND ETHICAL CONSIDERATIONS 4](#_Toc142853155)

[SOURCE CODE DOCUMENTATION 5](#_Toc142853156)

[EVALUATION METRICS 11](#_Toc142853157)

[CHALLENGES 11](#_Toc142853158)

[CONCLUSION 11](#_Toc142853159)

[REFERENCES 12](#_Toc142853160)

# INTRODUCTION

Phishing attacks are a significant cybersecurity threat, targeting users through deceptive emails and websites. In an era where digital communication plays a pivotal role in both personal and professional spheres, attackers are developing and using more advanced phishing methods in recent years mimicking individuals that are known to the target (Amini, 2023.) As Cybersecurity experts it is our role to counter these new approaches by taking advantage of automated tools such as Artificial Intelligence and ensuring the security and integrity of online interactions.

The team has developed an AI-driven Anti-phishing detection system using the BERT large language model (LLM) to successfully identify and filter malicious phishing emails based on the tone and language of the emails received. “BERT is a transformers model pre-trained on a large corpus of English data in a self-supervised fashion (Hugging Face, 2001).”

The team has trained the AI using defined email content, including both legitimate, spam, and phishing emails. Through this training process, the system can differentiate between legitimate communication from illegitimate phishing attempts.

# LANGUAGE MODEL

According to Hore (2023), “Large Language Models (LLMs) are foundational machine learning models that use deep learning algorithms to process and understand natural language. These models are trained on massive amounts of text data to learn patterns and entity relationships in the language.” For our report, we will be using the Bert LLM available through the Hugging Face API. BERT is a family of open-source transformer LLM.

"BERT, short for Bidirectional Encoder Representations from Transformers, is a Machine Learning (ML) model for natural language processing." BERT was created to help computers understand the meaning of ambiguous language in the text by using surrounding text to establish context (Lutkevich, 2020). BERT can be used on a wide variety of language tasks; however, we will be using it to predict text when writing an email to determine if an incoming email is a phishing email or not.

BERT was chosen for the following reasons:

* The model is free for use through the Hugging Face API which provides an open-source platform for natural language processing and foundation models.
* BERT has a massive dataset of 3.3 billion words which contributes to BERT’s continued success (Muller, 2022).
* BERT was specifically trained on Wikipedia (~2.5B words) and Google’s BooksCorpus (~800M words) (Muller, 2022).
* BERT is pre-trained on two different, but related, Natural Language Processing (NLP) tasks: Masked Language Modeling and Next Sentence Prediction.
* BERT uses a bidirectional approach (MLM) where the Transformer encoder reads the entire sequence of words at once, this converges slower than left-to-right approaches (read the text input sequentially (left-to-right or right-to-left) but outperforms left-to-right training after a small number of pre-training steps (Horev, 2018).

# LIMITATIONS AND ETHICAL CONSIDERATIONS

Some of the limitations of BERT include:

1. Hallucination: While BERT displays impressive language capabilities it is still capable of producing false positives if it enters an area where it lacks knowledge and will confidently conclude wrongfully. BERT can hallucinate and make incorrect information and decisions.
2. Training Data Dependency: The performance of BERT is dependent on the quality and diversity of the data that was used for training. This data will determine how accurate the LLM is as well as if there is biased data in the training data, you may get biased outputs if not properly addressed.
3. Potential Biases: The data that the LLM is trained may have societal and geographical biases encoded in them. This can cause the LLM to make biased decisions.
4. LLMs including BERT are computationally very expensive: processing a single page of text requires computations across billions of parameters, which can result in high response times, especially for longer input documents (Rotaru & Kok, n.d.).

On the other hand, BERT also comes with ethical considerations that must be considered before its use, they include:

1. Data Privacy Concerns: If used on personal data, there are privacy concerns regarding the potential exposure of sensitive information. The reliance on third-party APIs provided by vendors like Hugging Face introduces additional data privacy risks. This would mean the data (emails) is processed by, and potentially stored on, third-party servers that could be located anywhere in the world. Without proper contractual agreements, this could possibly lead to a violation of data privacy laws (Rotaru & Kok, n.d.).
2. The hallucinations can also cause some ethical concerns: BERT may generate outputs that are unexpected or unintended. An example in our case could be flagging legitimate emails. Therefore, close monitoring, validation, and human oversight should be done.

# SOURCE CODE DOCUMENTATION

**Load Libraries and Data**

!pip install -U "tensorflow==2.8.\*"

!pip install -U "tensorflow-text==2.8.\*"

!pip install transformers

!pip install -U tensorflow-text

!pip install transformers[torch]

!pip install accelerate -U

!pip install gradio

!pip install fsspec==2022.10.0

**Load Dependencies**

To begin our project, we will import the following libraries:

1. Tensorflow\_hub: This is where all TensorFlow pre-trained models are stored.
2. Tensorflow: This is used for model creation.
3. Pandas: This is used for data loading, manipulation, and wrangling.
4. Tensorflow\_text: This library allows additional NLP text processing capabilities outside the scope of tensorflow.
5. Skelarn: This library is for doing data splitting.
6. Matplotlib: This is used for visualization support.

import tensorflow\_hub as hub

import pandas as pd

import gradio as gr

import tensorflow\_text as text

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

import numpy as np

import transformers

from transformers import AutoModel, BertTokenizerFast, AutoModelForSequenceClassification, TrainingArguments

import torch

import torch.nn as nn

from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

*# specify GPU*

device = torch.device("cuda")

**Loading Data**

Now we will just load our data into a pandas dataframe:

*# load data*

df = pd.read\_csv('/content/Data/spam.csv')

df.head()

A screenshot of a computer

Description automatically generated

**Data Loading and Preprocessing**

We will be doing this by grouping the data based two categories (not phishing and phishing) and calling the value\_counts() method which will display the number of samples for each category. We will also load a dataset containing email messages and their corresponding labels (phising or not phishing) from a CSV file using Pandas. Following this, we will check the class distribution of the dataset to understand the imbalance between the two classes.

*# check count and unique and top values and their frequency*

df['Category'].value\_counts()

**A black text on a white background

Description automatically generated**

*# check count and unique and top values and their frequency*

df['Category'].value\_counts()

We then calculate and print the percentage of data that needs to be balanced for the two categories.

*# check percentage of data - states how much data needs to be balanced*

print(str(round(747/4825,2))+'%')

We then attempted to filter the dataset into two separate dataframes: one for phishing emails and one for not phishing emails.

*# creating 2 new dataframe as df\_phishing, df\_not\_phishing*

df\_phishing = df[df['Category']=='phishing']

df\_not\_phishing = df[df['Category']=='not phishing']

print("Not Phishing Dataset Shape:", df\_not\_phishing.shape)

print("Phishing Dataset Shape:", df\_phishing.shape)

*# downsampling not phishing dataset - take only random 747 example*

*# will use df\_phishing.shape[0] - 747*

df\_nphishing\_downsampled = df\_not\_phishing.sample(df\_phishing.shape[0])

df\_nphishing\_downsampled.shape

*# concating both dataset - df\_phishing and df\_nphishing\_downsampled to create df\_balanced dataset*

df\_balanced = pd.concat([df\_phishing , df\_nphishing\_downsampled])

df\_balanced['Category'].value\_counts()

df\_balanced.sample(10)

*# creating numerical repersentation of category - one hot encoding*

df\_balanced['Type'] = df\_balanced['Category'].apply(lambda x:1 if x=='phishing' else 0)

*# displaying data - phishing -1 , not phishing-0*

df\_balanced.sample(4)

*# loading train test split*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test , y\_train, y\_test = train\_test\_split(df\_balanced['Message'], df\_balanced['Type'],

stratify = df\_balanced['Type'])

**Model Creation**

Load BERT-related preprocessing and encoder layers from TensorFlow Hub using specified URLs.

*# downloading preprocessing files and model*

bert\_preprocessor = hub.KerasLayer('https://tfhub.dev/tensorflow/bert\_en\_uncased\_preprocess/3')

bert\_encoder = hub.KerasLayer('https://tfhub.dev/tensorflow/bert\_en\_uncased\_L-12\_H-768\_A-12/4')

Having downloaded the bert model, we used Keras Functional API to build our model.

*# Training using the Keras API*

text\_input = tf.keras.layers.Input(shape = (), dtype = tf.string, name = 'Inputs')

preprocessed\_text = bert\_preprocessor(text\_input)

embeed = bert\_encoder(preprocessed\_text)

dropout = tf.keras.layers.Dropout(0.1, name = 'Dropout')(embeed['pooled\_output'])

outputs = tf.keras.layers.Dense(1, activation = 'sigmoid', name = 'Dense')(dropout)

*# creating final model*

model = tf.keras.Model(inputs = [text\_input], outputs = [outputs])

*# check the summary of the model*

model.summary()

We then defined custom evaluation metrics and compile the model with the Adam optimizer, binary cross-entropy loss, and the defined metrics.

Metrics = [tf.keras.metrics.BinaryAccuracy(name = 'accuracy'),

tf.keras.metrics.Precision(name = 'precision'),

tf.keras.metrics.Recall(name = 'recall')

]

*# compiling our model*

model.compile(optimizer ='adam',

loss = 'binary\_crossentropy',

metrics = Metrics)

We then trained the model using the training data set and let it run for 5 epochs and stored the training history.

history = model.fit(X\_train, y\_train, epochs = 5)

Epoch 1/5

35/35 [==============================] - 637s 18s/step - loss: 0.7252 - accuracy: 0.5330 - precision: 0.5351 - recall: 0.5036

Epoch 2/5

35/35 [==============================] - 635s 18s/step - loss: 0.5749 - accuracy: 0.7304 - precision: 0.7094 - recall: 0.7804

Epoch 3/5

35/35 [==============================] - 636s 18s/step - loss: 0.4804 - accuracy: 0.8554 - precision: 0.8528 - recall: 0.8589

Epoch 4/5

35/35 [==============================] - 666s 19s/step - loss: 0.4292 - accuracy: 0.8562 - precision: 0.8387 - recall: 0.8821

Epoch 5/5

35/35 [==============================] - 627s 18s/step - loss: 0.3978 - accuracy: 0.8670 - precision: 0.8513 - recall: 0.8893

**Evaluation of the Model**

This line evaluates the model's performance on the testing data and displays the evaluation metrics to get an estimate of how the model is performing.

*# Evaluating performance*

model.evaluate(X\_test,y\_test)

12/12 [==============================] - 210s 18s/step - loss: 0.3877 - accuracy: 0.8690 - precision: 0.9423 - recall: 0.7861

Out[20]:

[0.38766226172447205,

0.8689839839935303,

0.942307710647583,

0.7860962748527527]

The function below takes an email message as input and predicts whether it is spam or not using the trained model. The model's output is post-processed to assign a label ("phishing" or "not phishing").

def spam\_filter(email):

test\_results = model.predict([email]) *# Assuming model.predict() accepts a list of emails*

output = np.where(test\_results > 0.5, 'phishing', 'not phishing')

return output[0][0]

**User Interface with Gradio**

Gradio is used to create a web-based interface for the spam filter. Users can input email text, and the interface will display the predicted label.

*# Define the interface*

demo = gr.Interface(

fn=spam\_filter,

inputs=gr.Textbox(lines=2, placeholder="Email Here..."),

outputs=[gr.Textbox(label="Prediction")],

)

*# Launch the interface*

demo.launch(share=True)

# EVALUATION METRICS

For this report, we had a dataset of 5, 572 emails, including 4825 legitimate emails and 747 phishing emails. Tests were carried out by inputting the data through the graphical interface.

# CHALLENGES

1. The team attempted to differentiate between spam and marketing emails which was a part of our scope of work, however, we were unable to find a suitable dataset as most spam/phishing emails are also considered marketing emails are considered as spam.
2. It took some time to find an appropriate open-source LLM to fulfill all the functions of our email filtering system.
3. Training the model was rather time-consuming because we wanted to reduce the number of false positives.
4. Compiling and training the model using the history = model.fit(X\_train, y\_train, epochs = 10) command took a very long time to complete and impacted significantly impacted the time taken to finalize the system.

# CONCLUSION

The escalating threat posed by phishing attacks in today's digital landscape demands a proactive and not reactive response from cybersecurity experts. There has been a huge surge in advanced phishing methods used by cyber criminals which has led to numerous cybersecurity incidents. As cybersecurity professionals our aim to safeguard online interactions led to the development of an innovative AI-driven Anti-phishing detection system, using the powerful BERT large language model through the Hugging Face API.

By taking advantage of the capabilities of Artificial Intelligence, we have created a sophisticated defense mechanism that examines the tone and language of incoming emails. Through thorough training on a diverse range of email content, encompassing legitimate, spam, and phishing emails we were able to successfully identify and block phishing emails.

# REFERENCES

Horev, R. (2018). BERT Explained: State of the art language model for NLP. Medium. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Hugging Face. (2001). bert-base-uncased. https://huggingface.co/bert-base-uncased

Lutkevich, B. (2020). BERT language model. Enterprise AI. https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model

Mfaisalqureshi. (2022). Email spam detection 98% accuracy. *Kaggle*. https://www.kaggle.com/code/mfaisalqureshi/email-spam-detection-98-accuracy/notebook

Muller. B. 2022. BERT 101 - State Of The Art NLP Model Explained. https://huggingface.co/blog/bert-101

Prateekjoshi. (n.d.). *Fine-Tuning-BERT/Fine\_Tuning\_BERT\_for\_Spam\_Classification.ipynb at master · prateekjoshi565/Fine-Tuning-BERT*. GitHub. https://github.com/prateekjoshi565/Fine-Tuning-BERT/blob/master/Fine\_Tuning\_BERT\_for\_Spam\_Classification.ipynb

Rotaru, M. Kok, K. Textkernel. n.d. Seven Limitations of Large Language Models (LLMs) in recruitment technology. Textkernel. https://www.textkernel.com/technology/seven-limitations-of-large-language-models-in-recruitment-technology/#:~:text=A%20major%20limitation%20of%20LLMs,behave%20the%20way%20they%20do.

Shukla, P. (2021). Performing email spam detection using BERT in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/performing-email-spam-detection-using-bert-in-python/

Shukla, P. (2021). Performing email spam detection using BERT in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/performing-email-spam-detection-using-bert-in-python/