Evaluating the Effectiveness of AI in Detecting Malicious Emails with Advanced Prompt-Engineering Techniques

Rafferty Ireland (Pupil)

This dissertation is submitted for the degree of Computer Science

August 2024

School of Computing and Communications

Declaration of Originality

I submit this work as wholly my own, with all resources properly referenced where relevant. All external material created has also been sourced. I consent to this work being stored electronically for the use of Lancaster University assessment purposes and plagiarism checks.

Acknowledgements

Thank you to Antonios Gouglidis who has been an immense help to me throughout the lengthy duration of this project. Through his advice, I have been able to push this project and myself to our limits. I hope you enjoy reading.

Abstract

Spear phishing is one of the most vicious forms of cyber-attacks that currently plague the web. Affecting 39% of UK businesses in 2022 alone, they account for an immense amount of monetary loss and exposure of private data. In this paper, the aim was to create a set of AI models that compete with current state-of-the-art phishing detection systems such as Barracuda and Outlooks spoof intelligence and put them through a series of experiments to evaluate and learn from their performance. Using carefully curated fine-tuning, the models have been optimised for the task of identifying malicious emails. Advanced prompt engineering techniques, such as Chain-of-Thought prompting, have been employed to create complex tests for the models to not only evaluate their performance, but to see if they can make up for where modern tools have not. The results gathered from these experiments shows that these tools have a real place among the current frontrunners, mayhap not to replace them, but to work alongside them to create a near-impenetrable system. The models we create have varying levels of performance, however the strongest manages to compete with the 99% accuracy we have come to expect from these tools today. Not only has a model of this level been created, interesting observations on how the data chosen for fine-tuning have been made. A correlation between the variety of data and the quality of the model has been proven through the results, with further insights into which sort of variety is good and which is detrimental to the models. There are always going to be new developments in the spear-phishing world, but with the help of these tools it's possible we can hold back the tide and save people on a personal and grander scale.

Table of Contents

1. INTRODUCTION- Page 8

2. BACKGROUND- Page 11

2.1 Case Studies- Page 11

2.1.1 Texas Medical Liability Trust- Page 11

2.1.2 Jack Todey- Page 12

2.2 Related Works- Page 13

2.2.1 Email Anti-Phishing Detection Application- Page 13

2.2.2 A Spam Email Detection Mechanism for English Language Text Emails Using Deep Learning Approach- Page 13

2.2.3 Traditional Machine Learning Models and Bidirectional Encoder Representations from Transformer (BERT)–Based Automatic Classification of Tweets About Eating Disorders: Algorithm Development and Validation Study- Page 14

2.3 Bidirectional Encoder Representation from Transformers- Page 15

3. METHODOLOGY- Page 17

3.1 Datasets- Page 17

3.2 Data Preparation- Page 18

3.3 Preprocessing- Page 21

3.4 Model Creation- Page 22

3.5 Training/Fine-Tuning Phase- Page 28

3.6 Testing Phase- Page 30

3.7 Prompt Engineering and Advanced Prompt Techniques- Page 30

4. RESULTS AND EVALUATION- Page 33

4.1 Dataset Testing Results and Evaluation- Page 33

4.2 Prompt Testing Results and Evaluation- Page 36

4.3 Additional Experiments- Page 37

4.3.1 Work Emails- Page 37

4.3.2 Manually Created Emails- Page 38

4.3.3 The Perfect Phishing Email- Page 39

5. Discussion and Conclusion- Page 41

5.1 Discussion- Page 41

5.1.1 System Architecture Application- Page 41

5.1.2 Future Applications- Page 42

5.1.3 Assumptions- Page 42

5.2 Conclusion- Page 43

6. APPENDICES- Page 44

6.1 Dataset Formatting- Page 44

6.2 Training Photos- Page 46

6.3 Heatmaps- Page 47
6.4 Old System Architecture Diagram- Page 49

1. INTRODUCTION

In recent years, AI has become one of the most innovative and exciting parts of the tech industry. With the advent of GPT's and transformers such as ChatGPT (ChatGPT, N/A), the widespread adoption of AI and their various uses has become commonplace throughout society. With the ever-looming threat of cyber-attacks, AI has begun to be deployed in combatting these attacks, by spotting the various signs associated with them to keep users safe without human intervention.

The aim of this project is to train a group of AI models on carefully selected datasets, to then evaluate each model and compare how efficient each one is based on how accurately they predict whether an email is maliciously altered. A range of experiments have been conducted to evaluate the models, revealing a deeper understanding of what makes the models tick. By fine-tuning each of the BERT (Bidirectional Encoder Representations from Transformers) models we aim to identify even the hardest to detect spear phishing emails that may come from an unexpected source. Even an email from a work colleague gone rogue trying to get back at their organisation with a crooked revenge email. We aim to test these fine-tuned models on a variety of scenarios pushing the models to their limits, employing advanced prompt engineering techniques alongside our other experiments to create a set of "puzzles" for the models to solve in the form of different safe and unsafe emails with different twists.

Businesses are victims to various types of cyber-attacks every year, causing them on average $4.35 million in 2022 from data breaches and exposing around 1 in 5 users' emails (Griffiths, 2024). 39% of UK businesses were affected by cyber-attacks in 2022 with Cyber-attacks globally increasing 125% from 2020 to 2021 (Griffiths, 2024). The below graph shows the drastic effects spear phishing attacks alone had on business globally in 2022. (Published by Petrosyan Ani Petrosyan & 18, 2023) The case studies below show the drastic effects these attacks can have on both a personal and national level, with a hospital being exploited for $407,000, proving the mercilessness of these emails and their senders. It proves that even with these tools in place there is a need for more protection (Medical Liability Trust, 2023).

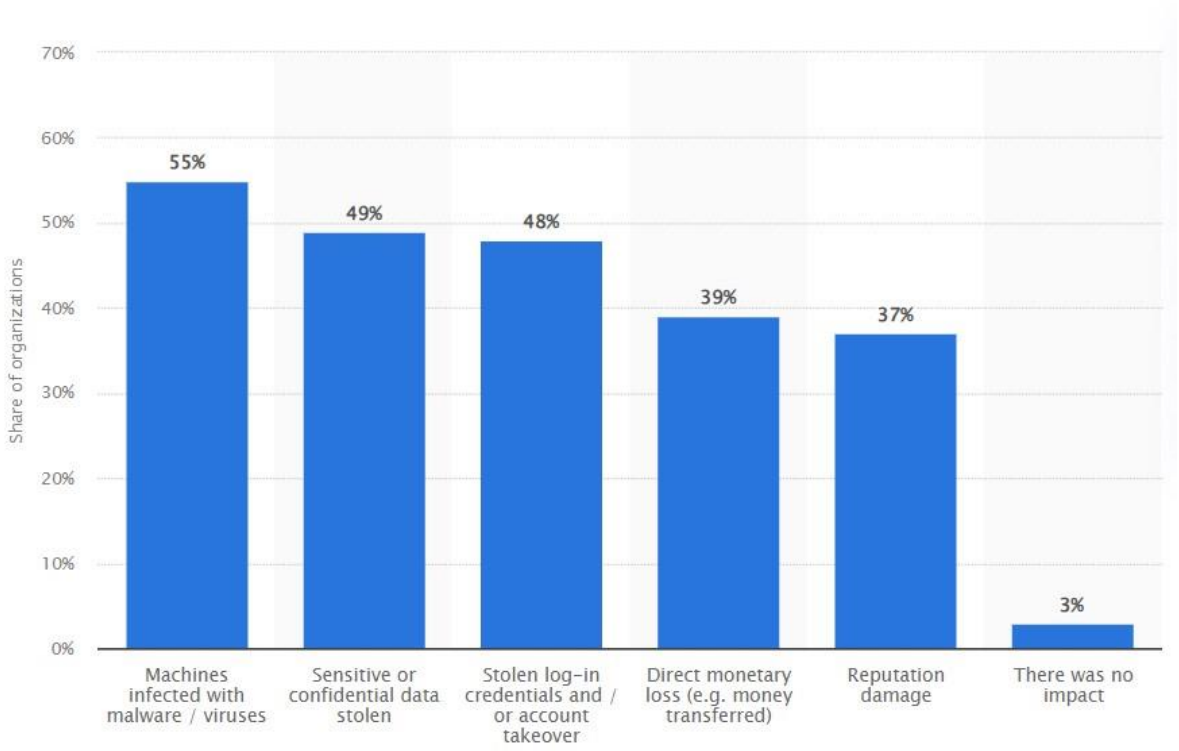

Figure 1: Presenting the affects spear phishing had on business in 2022(Published by Petrosyan Ani Petrosyan & 18, 2023)

Spear phishing attacks are focused on the human aspect of organisational systems, which today is unfortunately one of the weakest. There is no firewall that can be applied to human behaviour. Emails are the leading style of spear phishing attacks today being one of the simplest and easiest attacks to create, with systems that can spam thousands of messages out to recipients at the hope of catching some unlucky people off guard (Chin, 2024). Since there is a focus on attacking the human element of a system it creates a problem, businesses must go through the process of individually training each of their employees on the danger signs to look out for, which can still leave room for error if one of these emails came from a 'trusted' address such as a colleague or regular customer, something even modern day AI tools have trouble detecting.

The proposed system will combat a range of malicious emails including spear-phishing and spam using existing datasets gathered online. The datasets have compiled a collection of both phishing and spam emails from real world settings, as well as safe emails gathered from real businesses such as Enron. The models will focus on detecting the differences between bodies of text and how subtle differences in tone, urgency or spelling can expose these malicious emails. The BERT language model is a leading AI model which uses encoders to classify bodies of text like the emails mentioned above and spot these differences with its vast knowledge of the English language and how words relate to each other. Through additional fine-tuning we can even "teach" it to perform classification tasks like the ones below (Devlin et al, 2019).

The goal is for the model to compete with current state of the art email detection systems such as Barracuda (Barracuda, 2024). If successful and employed in the real world, a system like this would ensure businesses have less pressure to account for human error. This would save them money and resources as well as

protecting their clients and employees from confidential data leaks. Not only would a system like this benefit businesses, but it could also provide a general shield to everyday users and their personal emails that contain private information. To set out the aims in a quick list:

- Create a group of pre-trained BERT AI models
- Fine tune the BERT models using carefully selected datasets to shape them to detect phishing and spam emails
- Test that the BERT models on a collective of data from all the datasets
- Put the BERT models created through a series of experiments to test their limits using advanced prompt engineering techniques and specially curated emails
- Evaluate and compare each model and their effectiveness to answer the question set out by this paper

2. BACKGROUND

This section aims to delve into the systems and studies that are closely related to this project. Firstly, we delve deep into some case studies to understand the claims made in the introduction, showing the adverse effects these attacks have on both businesses and the common person. We then explore some related works that use various AI tools like neural networks and LLMs (Large Language Models) that relate to the topic at hand in this project, learning from the experience of the past. The final subsection in this part of the report is some higher-level info about BERT.

2.1 Case Studies

In this section, I have chosen two case studies that stood out to me among the sea of millions. They show the exact reasons why tools like the models created in this paper need to exist, and how even those with experience can fall victim to the targeted attacks tailored by professional teams today. Whether a simple oversight or a rushed mistake, the aftereffects of these attacks can be devastating.

2.1.1 Texas Medical Liability Trust

This case talks about a small rural hospital that was contracted with an emergency medical group for its emergency department coverage. The hospital would pay a monthly amount of around $200,000 through an electronic funds transfer which would be sent over on an email invoice. When making one of their regular payments the hospital was requested by the emergency medical group to send the funds to a new account. They sent this money to the account which ended up getting frozen on the first try, red flag number one. After this, the medical group requested the funds be sent to another new account which this time went through successfully. This process was then repeated another time, with the emergency group requesting another $200,000 be sent to another new account the next month.

A **total of $407,000** was sent to the new accounts over a two-month period before it was discovered that the requests were not made by the emergency group at all. All the requests for money to be sent to new accounts were **fraudulent** and led to all the money being lost. Due to this, the hospital ended up renewing their policies to ensure a mistake like this never occurred again. This case shows us why there is a need for more advanced and readily available anti-phishing software, malicious users can access information and mimic transactions as seen within this study. We can assume that whoever was making the transactions did not know the classic signs of a phishing email, such as an altered sender address or urgent language, which is the case of most modern-day businesses.

With more powerful technology to combat these attacks it is possible this could have been avoided without the need to train staff on the dangers of phishing (Medical Liability Trust, 2023). It also shows the lengths people are willing to go to when committing these acts, not even hospitals emergency department funding is a concern so long as they aim to make a profit.

2.1.2 Jack Todey

"Jack Todey is a businessman and considers himself to be financially savvy" is the beginning of this case study and as you can imagine since it is included in this paper, that did not help when it came to a phishing attack directed at him. The study says it caused him "considerable stress" and left him "out of pocket" showing the personal effects attacks like these can have, not just being a business statistic. The study documents how Jack received a phishing email that looked "absolutely genuine" sent from his bank.

The email had the same logo and mimicked the style of email his bank would genuinely send if the request they were making was real. Jack talks about how the email uses urgent language saying they "put in a note of fear" so it's "urgent for you to act". It shows the genuine manipulation tactics used by these people, with empty threats that force unknowing users to give up their details for the attackers' personal gain. After he had given up his details, Jacks identity was stolen and around £1000 was taken from his bank account before he realised what was going on and he got his account frozen.

Although this isn't an amount like $407,000, it must be recognised that this is just a single man that was victim to this attack and he lost money on a personal level, as opposed to a hospital which most likely has government funding and wasn't at any high risk from losing said money, £1000 being taken from an individual could mean missing rent payments or not being able to afford food. It highlights that there is a need for this type of protection on an individual level, as no one is entirely safe, and attacks can occur outside of just businesses (Conner, 2023). Globally, 323,972 people fell victim to phishing attacks in 2021 with an upwards trend, 29% increase in 2022 (Micro et al., 2023), and it's important to understand the impact this has on people and why these studies are necessary (Griffiths, 2024).

2.2 Related Works

This section covers some interesting papers I managed to find whilst researching this project relating to the topics at hand. Each of these papers provides a lesson that I applied to my own paper, steering me in the direction of using BERT and providing some information on how it compares to other models and neural networks.

2.2.1 Email Anti-Phishing Detection Application (Helmi,2019)

In this paper (Helmi, 2019) details the creation of an application that can attach to an email service and automatically scan and vet any incoming emails to determine whether they are phishing emails. An interesting statistic gathered from the paper is the fact that in a study of 22 people, when presented with 20 websites that are either safe or phishing sites, 23% of the users were not aware of the cues associated with phishing attacks; leading to 40% of decisions made being incorrect. The program they created focused on a decision-tree based algorithm to identify phishing emails, which is a sequential algorithm opposed to BERT and its transformer architecture. There is very little focus in this paper on the effect different datasets can have on the model that is produced at the end, and how variations between models can affect results. There is also more of a focus on the metadata of emails instead of the semantics of language used within the emails themselves. This paper picks up on the areas missed out by Helmi with more of a focus on the text and language used in emails as opposed to the data they are made up of.

2.2.2 A Spam Email Detection Mechanism for English Language Text Emails Using Deep Learning Approach (Kaddoura et al, 2020)

In this paper S.Kaddoura et al set out to create a mechanism that can accurately predict spam emails using neural networks. They set out to test two separate neural networks, FFNN (Feed Forward Neural Networks) and BERT. Feed forward neural networks are unidirectional, as opposed to BERT being bidirectional, that works by passing data straight from the input nodes, through any hidden layers and then through the output layers (Tyrell, 2005). They train both models on an Enron dataset, similar to the one used in this paper, containing 16544 spam emails and 16094 ham(safe) emails. While the main focus is on testing the FFNN through a series of experiments, they do compare the two models with the FFNN having an F1 score of 99.154 and BERT having an F1 score of 99.18. The F1 score of a model is a combination of its precision and recall when testing (Kundu,2022).

The rest of the paper features a series of experiments using the FFNN with varying layers and neurons that evaluate its performance. The paper provides evidence that BERT is one of the leading models that is readily available, pre-trained and effective at solving text classification problems such as the one this paper attempts to solve. The paper focused more on the limits of the models themselves as opposed to this paper which focuses on how different datasets

effect F1 score and how a single model compares when trained on these different datasets.

2.2.3 Traditional Machine Learning Models and Bidirectional Encoder Representations from Transformer (BERT)–Based Automatic Classification of Tweets About Eating Disorders: Algorithm Development and Validation Study (BenítezAndrades1* et al., 2022)

Here we have a study conducted by Jose Benitez-Andrades et al that focuses on using a variety of modern machine learning models to detect tweets about eating disorders, focusing on four different factors which include:

1. Messages written by someone suffering from an eating disorder or not
2. Messages promoting suffering from eating disorders or not
3. Informative messages or not
4. Scientific or non-scientific messages

They compared the F1, accuracy and computational time of each model based on the above tasks. A fascinating point brought up in the literature section of this paper is the use of social media informatics and how that can be used to monitor and predict the health of communities to improve a range of medical, environmental, and social factors relevant to the health of communities using social media (Gamache et al., 2018). These informatics can be combined with machine learning models like BERT to process live information en masse, leading to even more effective use of social media in the prediction and improvement of health within said communities.

The implications of further developing this system using BERT could create a world of difference by predicting outbreaks in real time, classifying a user's symptoms, or identifying patterns that could spell danger for a community. Jose et al collected over a million tweets and compressed these into subsets of 2000 posts containing n=286 tweets that contained keywords related to eating but did not fit into any of the four categories above, with all 2000 of these tweets then being manually labelled based on whether they fit a category or not.

The models chosen for the experiment were random forest, recurrent neural networks, bidirectional long short-term memory networks and pretrained BERT. The BERT models and bidirectional long short-term memory models were chosen as they seem to be best suited for natural language processing tasks such as the one this paper is undertaking, with the random forest being chosen to compare to as a legacy method for NLP (natural language processing). From the BERT models the 3 most notable and relevant to this paper are BERT base, RoBERTa and DistilBERT. RoBERTa is a modified version of BERT trained on a larger corpus with a larger encoding vocabulary of 50k as opposed to BERTs 30k, which in Jose et al experiment outperformed BERT in every test, whilst also having a shorter

implementation time (Kumari, 2023). The DistilBERT model is a stripped-down version of BERT base featuring half the number of transformers of its parent model, that being six transformers opposed to BERTs twelve (Holt-Nguyen, 2023). This model surprisingly took the longest in its implementation time across all the different models and it slightly outperformed BERT in scenarios one and two, with BERT being more accurate in scenarios three and four. The BERT model itself had an implementation time that sat in the middle of both models and even though it was not as powerful as RoBERTa it still outperformed the traditional methods in terms of F1 score and accuracy.

The BERT models took approximately 7-10 times longer to implement than the traditional methods with similar accuracies in scenario one and no larger a disparity between models than 9.79% in terms of accuracy in other scenarios, proving that in a quick and simple run of things the simple models may be a better choice. Since the experiment conducted in this paper focuses on the accuracy and F1 score of a model applied to different datasets the time-based advantage of the traditional methods is not a priority and so the BERT model has been chosen.

2.3 Bidirectional Encoder Representation from Transformers (Devlin et al, 2019)

The BERT model is an open-source machine learning framework focused on natural language processing (Lutkevich, 2020). The model focuses on using context and surrounding information to identify the semantics of the body of text it is working with. The model "understands" the meaning of language having been pre-trained on the entire English Wikipedia and the Toronto BookCorpus to help interpret a variety of different texts and their specific format (Raj et al, 2019). The BERT model is based on the transformer architecture which consists of tokenizers, embedding layers, transformer layers and an un-embedding layer which are used for NLP (natural language processing) tasks. The transformer architecture was first introduced by Google researchers in 2017 through the paper "Attention is all you need" (Toiziz et al, 2019). The focus of the paper is to move from sequentially processing data to using attention mechanisms, in which every input is mapped to every output, to reduce sequential computation (Vaswani et al, 2017).

This architecture has now become the leading design for most modern-day large language models including BERT, GPT-4, Claude and ChatGPT (Toiziz et al, 2019). The BERT model uses the "encoder-only" transformer architecture meaning it has a stack of encoders within its transformers that encode data passed to it into numerical representations that can be used in further layers or as an output. Based on the results presented in the related works and the tasks performed using BERT I believe this model best suits the task at hand today. With the 110 million parameters that can be fine-tuned, it allows accurate control over how we want the model to process a piece of text and is perfect for identifying the acute

semantics we have set out to identify in this paper. The model is not only perfect for text classification tasks like the email classification in this paper, it is one of the most readily available models with pre-trained versions ready to be fine-tuned out the box. Through the altering of these parameters, we hope to achieve the aims set out above.

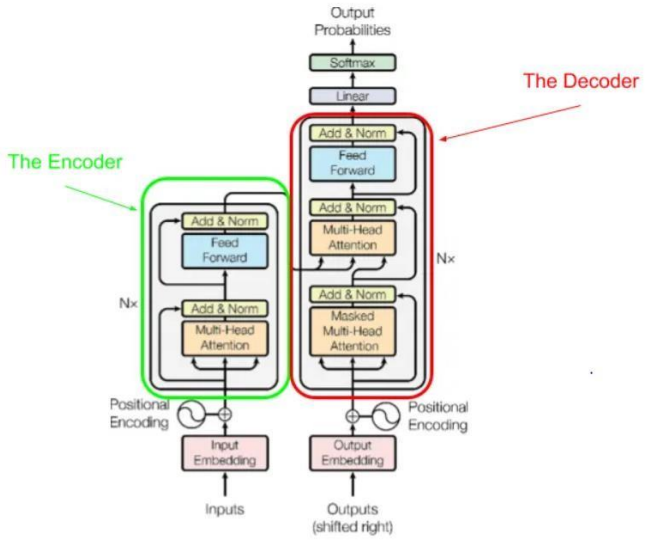

Figure 2: The transformer architecture as specified by the “attention is all you need” paper (Beast, 2023)

3. Methodology

3.1 Datasets

Before getting into the meat of the BERT model there are a few steps we must take to ensure we are ready to begin experimentation, starting with gathering our data. Datasets are large collections of data that have been gathered for the purpose of natural language processing tasks, there are thousands of datasets available online for the purposes of text classification. Since there were limited computational resources for this experiment and the aim is to train multiple models on multiple datasets, we have gathered our data from publicly available sources.

The focus was to obtain datasets that covered both spear-phishing and spam, as well as emails collected from an actual business, as to train our models on information as close to a real-world email inbox as possible. Creating our data entirely through prompt-engineering was a possibility, however this would have taken away from the credibility of the training and testing data, with no emails taken from real world scenarios. It also would have taken much longer to generate than the already available datasets mentioned below. Prompt engineering has instead been employed in our experiments later to generate a range of emails that are more intentional tests for the models, with each one giving us an insight into their performance.

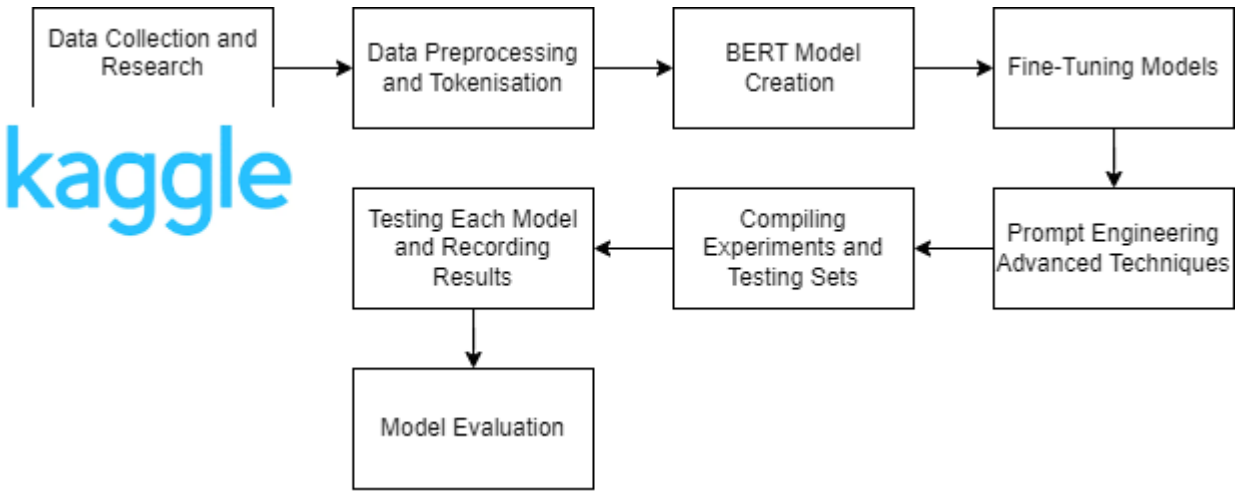

Figure 3: An illustration of the workflow from the research stage to results gathering

Kaggle is a website focused on and around machine learning and has a plethora of datasets that are all open source and easy to use (Goldbloom, 2010). After searching through the site three datasets were selected:

- The Enron Dataset- A dataset containing 500000 emails taken from the Enron corporation after its collapse. It is one of the only datasets of "real"

emails out there and fits our need for a business corpus perfectly (Cukierski, 2016). These emails will need labelling in later steps as this dataset is not specifically created for spam/phishing classification.

- The Spam Email Classification Dataset- A dataset containing 83446 emails taken from a subset of Enron emails called the Enron Spam dataset and the 2007 TREC Public Spam Corpus. It contains a mix of emails labelled as either "ham" (safe) or "spam" (Singhvi, 2023). Singhvi used this dataset to create a spam classification tool of his own, showing that it is the perfect dataset for our own spam section.
- The Phishing Email Detection Dataset- A dataset containing 18650 spear-phishing emails, it is not stated where they have been gathered from. The dataset contains the body of text from these spear phishing emails with a label of either "Safe Email" or "Phishing Email" (Cop, 2023).

Using the above datasets, four BERT models have been created. Each model uses the data to fine-tune themselves to be suitable for combatting malicious emails. The sets above were specifically chosen as they provide coverage of the wide range of malicious emails one could expect to find in their inbox, instead of just focusing on a single avenue e.g. only spam. This helps in testing, where we can use a combination of these datasets to simulate an inbox and classify each email to see whether they are safe or not. Two of the datasets take from the Enron corpus, giving us the genuine emails that have already managed to sneak through state-of-the-art tools. The perfect test for the models!

3.2 Data Preparation

To conduct this experiment, we first had to decide on a platform to utilise the above datasets and create models to be fine-tuned and tested. For this, we chose the notebook editor within Kaggle, as it allowed us to easily import data from the site without having to export, download or upload any additional files (Goldbloom, 2010). The Kaggle editor is python-based, providing a full virtual environment to experiment with using the Kaggle servers to run NLP tasks like the experiments below. After adding the datasets to the notebook through the editor, there were a few steps that took place to alter the datasets into a form that BERT can understand. The dataset that needed the most work was the Phishing Email Detection Set as not only did the text itself need preprocessing but there were columns within the .csv file that needed altering.

The aim here was to make all the datasets have a set format: the body of text from the email and a label of either 0 (safe) or 1 (unsafe). This label would indicate whether an entry in these datasets was spam or phishing in their respective cases. Most of the datasets included additional columns that needed removing and labels that needed changing from text (e.g. "Safe", "Spam") to the 0 or 1 format. This was done through the use of the pandas library in python (NumFOCUS, 2009). The pandas library allows for the easy manipulation of data frames such as the datasets used in this paper. To ensure the datasets were in the correct format for preprocessing the below steps were performed:

- Read dataset from input folder that has dataset stored:

  .read_csv("dataset.csv")- from the pandas library allows us to read a csv file and convert it into a data frame for us to manipulate

- Drop unnecessary columns:

  dataset.drop(column)- This command allows us to drop the specified column from out data frame previously created

- Separate safe emails and unsafe emails:

  datasetSafe = phishset[dataset['Email Type'] == 'Safe Email']- In this code we take an existing dataset with all the phishing emails and create a subset from it containing only the emails labelled 'safe email'

- From the larger set of emails (varies between datasets) create a new set of emails that matches the smaller set in size:

  datasetSafeBalanced = datasetSafe.sample(datasetFish.shape[0])- this code creates another subset. In this example, we take the datasetSafe we just created and shape it to match a set that contains only phishing emails. This is done to ensure we have an equal number of safe and unsafe emails (Balanced)

- Concatenate these two sets together to create a balanced set of safe and unsafe emails:

  balancedData = pd.concat([datasetSafeBalanced, datasetFish])- This pandas command creates a concatenated data frame made up of our subsets.

- Apply a function to the new dataset that applies a 1 if the corresponding data label is unsafe ('Phishing Email' in the above example) and a 0 in any other case

  balancedData['phish'] = balancedData['Email Type'].apply(lambda x:1 if x=='Phishing Email' else 0)- This line of code applies to the 'phish' column of our balanced dataset, our label. We check the label to see if it is equal to 'Phishing Email'. We the convert the entry to a 1 if the condition is met or a 0 if it isn't met. This will then give us a dataset that is ready to be processed by the models.

- A sub step here is the removing of any null entries which only applied to the phishing dataset

The above steps were applied to each of the datasets used in this experiment. Two steps that require more explanation are the balancing of the data and the filling of null values.

Due to how the BERT model processes inputs, we must ensure that none of the entries we pass to it have NULL values, the model cannot interpret and associate tokens to an empty value. If any of the fields are NULL the model will reject them, the entries can be "empty" as long as they are not NULL. An empty string token or a 0 can be assigned to an entry that is empty but not NULL. However, an empty string or label provides no value to our model, so we remove all entries containing a null value to purify the quality of our data.

The balancing of the data is a necessary step in ensuring our fine-tuning and testing is accurate and representative of the data we are using. To avoid model bias, we balance the datasets to ensure that there are an equal number of entries in both safe and unsafe categories. Model bias occurs when there is a vast difference between the categories within a dataset, the model will always favour the class with more entries as it will appear more in the testing and fine-tuning, thus giving the model the inclination to choose said category over the other in most cases (Hvilshøj, 2022). All the models in this paper were trained on balanced sets, even the final model where all the datasets were combined, ensuring the most accuracy within the datasets.

7]:

	Unnamed: 0	Email Text	Email Type
0	0	re : 6 . 1100 , disc : uniformitarianism , re ...	Safe Email
1	1	the other side of * galicismos * * galicismo *...	Safe Email
2	2	re : equistar deal tickets are you still avail...	Safe Email

Figure 4: Example excerpt from phishing dataset pre-alteration (Goldbloom, 2010)

After the data has been balanced, the next step is to split the data. When traditionally training AI models we have testing, training and validation sets. Since BERT is already available pre-trained on the resources mentioned previously, the training set is instead used for fine-tuning. The split used for this paper is 70% fine-tuning, 20% testing and 10% validation. We perform the split using the below method from the sklearn module:

train_test_split(balancedData['Email Text'], balancedData['phish'],test_size = 0.3, stratify = balancedData['phish'])

The above method splits the balanced dataset we have created with a test size of 0.3 or 30% and the other 70% going to fine-tuning. We then repeat this method on the testing set we created with a 0.33% split to get our 10% overall validation set.

The fine-tuning set is the first set utilised by the model, as it is the data that the model learns from to be able to discern whether an email is safe or malicious. The models make predictions on the labels of the training data and then evaluates these predictions against the actual label, learning from each correct guess and mistake. During each epoch (loop) the currently trained model is tested against a validation set, which helps correct the model in case of overfitting or misclassification and provides a rough progress check on performance. The fine-

tuning and validation sets affect hyperparameters such as hidden layers, layer width and layer count.

We create two variables for fine-tuning, Training1 and Training2, with one containing the bodies of text from the emails and the other containing the corresponding label. The validation and training sets follow this trend with their 1-counterpart containing the bodies of text and their 2-counterpart containing the labels. After the final epoch, the final model is produced which is then tested using the testing set and our experiments, more details on this process will be provided below (Wikipedia Contributors [29], 2024).

3.3 Preprocessing

Preprocessing is the process of converting a raw data input into a format that can be understood by an LLM such as BERT. In this case, text preprocessing is needed to convert the bodies of text from the emails into an understandable format to be inputted into the model's encoder. Preprocessing involves the following steps when handling text:

- Tokenisation- Splitting up bodies of text into tokens, a token can be a word, phrase, special character or whitespace and other reserved special tokens such as CLS (Classify Token). These tokens are easier for the model to understand, and BERT can bidirectionally understand the relation between these tokens.

- Normalisation- In normalization we aim to create a standard for all the text within the model, usually by converting all text to upper or lower case to reduce inconsistencies within the texts.

- Stemming- The process of converting words to their base form by removing any suffixes. For example, "buying" would be reduced to "buy". This helps simplify the language used in the text so that different words with the same meaning can be grouped together, reducing the size of the data overall. Base BERT has a dictionary size of 30k which means this process of stemming is important to fit as much information as possible into that set size.

- Lemmatisation- Like stemming, except words are converted to their dictionary format depending on the context within the text.

- Stopword and Punctuation Removal- These steps are fairly self-explanatory, stopword removal takes words that do not provide much meaning or value to the model and removes them. Punctuation removal takes all punctuation out of the text. This is done to simplify the data and ensure more focus is placed upon words with actual meaning.

- Spelling Correction- Spelling correction ensures all words within the data are spelt correctly so that the text is more readable and so words that are misspelt aren't considered separate from their correct spelling. (Germec, 2023)

Preprocessing is the final step in data preparation before we can finally take the processed data and input it into the encoder. The TensorFlow module in python allows us to create a preprocessing object that can convert text it is given into a form understandable by the BERT models in this paper (Abadi et al, 2015). This allowed us to quickly preprocess the text from each dataset. After this step we pass this pre-processed data to a BERT encoder also from the TensorFlow module. This encoder converts the pre-processed data passed to it into a contextualised representation with embeddings that can be read and understood by the BERT models. We create the preprocessor object by retrieving the appropriate BERT preprocessor version using:

bertPreProcess=hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3")

This fetches the BERT preprocessor which matches the pre-trained BERT encoder object we retrieve later. We must create the preprocessor using this method as otherwise it would not be compatible with our encode.

3.4 Model Creation

After all the dataset formatting had been handled, the next step was creating the models for fine-tuning and experimenting. To create interesting dynamics between the models using the datasets provided, the four following models were created:

- Phish Model- The first model created was trained purely on the phishing email detection dataset. This model theoretically should be able to predict spear phishing emails with high accuracy and precision, whilst not being as accurate at predicting spam emails.
- Spam Model- This model is like the one above except trained on the spam dataset with the inverse results, higher accuracy and precision for spam with a lower score for phishing emails.
- Phishron Model- The Phishron model is a combination of the phishing email detection set and the Enron email dataset. The goal was to combine a real-world influence with a mix of spear phishing emails, hopefully allowing the model to differentiate between spear phishing emails and emails sent within a business even when following a similar format. The focus when testing this model is to see whether a spear phishing email formatted in an Enron style will be detected or not, testing the subtlety of the BERT model.
- Final Model- The final model is a combination of all the above models, with appropriately balanced data. By combining all the datasets, the expectation is that the model should have the highest accuracy overall when predicting spam and phishing emails, but maybe a lower accuracy than specifically predicting spam or phishing emails.

After planning out the four above models the next step was to create them within the Kaggle notebooks. A BERT model is made up of layers, with each layer serving a different purpose. Each model created in this experiment has the layers shown in the excerpt below:

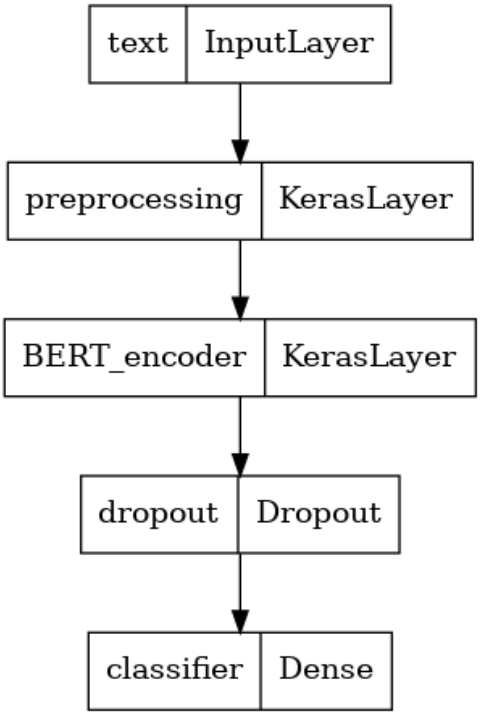

Figure 5: An excerpt taken from Kaggle showing a summary of the models created (Goldbloom, 2010)

The first layer we create is the input layer. Using the TensorFlow module, we can create input with:

Input = tenflow.keras.layers.Input(shape=(), dtype=tenflow.string, name='text')

With the input layer produced by this function, we can now add it to the very 'top' of our model. It is the layer that we pass our raw text from the datasets we have reformatted. The layer connects to the first Keras layer, which is the preprocessing object we created above. The input layer ensures all data passed into the model is of the correct format and is ready to be pre-processed and tokenised. Once the input layer has checked the type of the data passed to it and ensured it is text, it then passes the input to the layer below for it to begin preprocessing.

The Preprocessing-KerasLayer in the figure above is the layer that takes the raw text input from the bodies of our emails and performs the preprocessing steps mentioned above. The layer takes the inputted text and produces the input_word_ids, input_type_ids and input_mask which are passed down to the next KerasLayer. These objects are the tokens the BERT model uses when understanding language (Chollet et al, 2015). input_word_ids is an int32 object with the token_ids of the packed input sequence. The token_ids correspond to the words position in the vocabulary so a word at position 4567 in the vocabulary would have the token_id 4567, we also have

token_ids for the start and end of a sentence or word (Tinkerd, 2023). input_type_ids is another int32 object which help identify different sequences in our code. If we have two sentences that are related, with one being the context and one that follows this context, we use type_ids to determine which part of the overall text belongs to the context and which part belongs to what follows the context. This is done by taking the input segment and setting index 0 of this segment to include a start of segment and end of segment token, with any following segments only containing their end of segment tokens, allowing the model to know determine where specific sentences begin and end (HuggingFace, Tensorflow, n/a, 2024). The input_mask is a mask applied to the input with a 1 where text is present and a 0 where any padding is present (TensorFlow, 2024). After these onjects have been produced using all the inputted data, they are passed down to the next KerasLayer for embedding.

Bert_encoder-KerasLayer takes the pre-processed outputs produced by the previous layer and produces three new outputs being sequence_output, pooled_output and default. This is the encoder object we mentioned in the preprocessing section, we retrieve the encoder using the following:

bertEncode=hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4")

The encoder is the pretrained BERT object that contains the actual values and understanding taken from the Wikipedia and Toronto corpus. The encoder object produces the embeddings needed for the dense layer to classify our emails. The BERT_encoder layer was created when we passed our pre-processed input object to our encoder object. The sequence_output object is a float32 type tensor of shape [batch_size, seq_length, dim] which contains the embeddings of each token of every input sequence. pooled_output is another float32 tensor of shape [batch_size, dim] which contains the embeddings of each input sequence as a whole, which is derived from sequence_output. default is a float 32 tensor of shape [batch_size, dim] which is used as the output for our dense layer, containing the embeddings of each input sequence, it is essentially an alias for the pooled_output (Tensorflow, 2024). To get the pooled_output from sequence_output we must perform a pooling function, the default being average pooling performed on the second to last hidden layer on all tokens in a sentence (Xiao, 2018). The average pooling operation takes the average from a “patch” in a set of values and uses this average alongside other patches to create a new overall representation of the data (PaperWithMe, N/A).

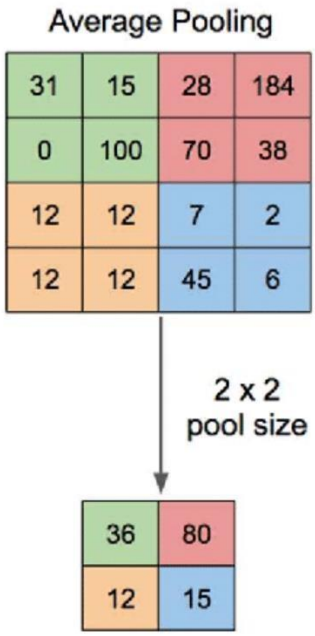

Figure 6: Average pooling operation mapped out visually (ResearchGate, N/A)

The embedding layer also contains some parameters within itself. These are the actual individual output from the 12 BERT transformer layers, with each layer containing 768 hidden features applied to these outputs (Dhami, 2020).

Before we can pass our data to the model we must create a few more layers. Firstly, we create a dropout layer. Dropout in NLP is when we drop nodes for every forward iteration in the model training. By dropping nodes, we avoid overfitting and create a new network from the parent architecture of our previous iteration (Yadav, 2022). The dropout rate in this experiment is 0.1 meaning a tenth of the nodes are dropped each epoch. Overfitting is when a model learns to make accurate predictions on the training data but not on any new data, essentially by predicting the same things repeatedly it may start to pick up features that relate to that specific email instead of the idea of a phishing or spam email.

For example, if a set of spam emails all start with a certain address that has been spoofed, then the model would predict this as spam even if the body of text may not necessarily be a spam email. This is because the address itself is now associated with spam emails, meaning the model will not be able to make accurate predictions based on other factors (Jassy et al, 2024). This is also why we separate training and testing sets, so that when it comes to testing the data the model is seeing is completely new and cannot be 100% accurately predicted based on the training data.

The next layer we create is a dense layer, a layer with densely connected neurons where we apply an activation function. The activation function used in this paper is the sigmoid function (Chollet et al, 2015). The sigmoid function is a mathematical formula that is applied to the output of each neuron when training or testing our models. The function essentially applies a numerical value between 0 or 1, which can then be used, almost like a probability, to predict the result of the output. For example, if the sigmoid function was applied to an output and

returned a value of 0.723 then this is most likely a phishing email, if it returned a value less than 0.5 then its most likely a safe email. A problem with the sigmoid function is that it can return a value of 0.5, in this case the model cannot accurately predict what type the email is which is one of the causes of inaccurate predictions (Vishwakarma, 2023).

The dense layer is where the classification occurs and decisions are made based on the value returned from the sigmoid function, these values are then passed to the dropout layer where 10% are dropped and the cycle repeats. Each model is then configured using three metrics to evaluate the information presented from the output of the above layers.

The metrics used in these models are accuracy, precision and recall. These metrics are all taken from the Keras module. The accuracy metric measures how often the prediction made by the model matches the label of the email. In this case we are using the binary accuracy class seeing as the emails can either be labelled with 0 or 1 as mentioned above. The precision metric measures the correctness of the predictions made with respect to the label, considering true and false positives. A false positive is when the model incorrectly predicts a safe email as a phishing or spam email, precision is calculated by dividing the true positives by the sum of the true and false positives. The recall metric is the same as precision, except it takes the true positives and false negatives. A false negative is when the model predicts an email as safe when it is a spam or phishing email. With these measurements, we can get an accurate representation of each models' performance throughout the training and testing phases (Chollet et al, 2015).

The next step is compiling the model, we first take the metrics from above and pass these to the Model.compile() method from the keras module (Chollet et al, 2015). In this method we introduce two new parameters, the optimiser and loss functions. Machine learning optimisation is the process of improving the accuracy of a model's predictions, whilst minimising the loss or error. The aim of an optimisation function is to match the predicted output to the true output and to iteratively work toward that state. Each epoch, during fine-tuning, the optimisation function will be applied, fine tuning the models hyperparameters to understand the relation between the labels and bodies of text the model is predicting. An example of a hyperparameter is the learning rate, this is the rate at which the model "steps" towards the gradient of the loss function.

The optimiser fine tunes the learning rate to ensure we reach the gradient without stepping too far past the curve (overstepping) or stepping so slowly we never reach the curve (Wikipedia, 2023). Other examples are the size and topology of the neural network itself, by altering all these parameters the optimisation function gets as close as possible to minimising the gap between the results and the loss function. The optimisation function used in this paper is the Adam (Adaptive Moment Estimation) optimiser, "an algorithm for first-order gradient-based optimisation of scholastic objective functions based on adaptive estimates of low-order moments" (Kingma et al, 2017). The Adam optimiser stores two variables, the exponentially decaying average of past **squared**

gradients vt and the exponentially decaying average of past gradients mt. mt and vt are calculated as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Figure 7: The Adam gradient calculation rule (Ruder, 2020)

mt and vt are the mean and uncentered variance of the gradients. mt and vt are initialised as vectors of zeros which gives them an inherent bias towards this value, it uses biascorrected estimates to fix this using the following formula:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Figure 8: The Adam bias correction rule (Ruder, 2020)

The standard value for Beta1 is 0.9 and Beta2 is 0.999 according to the author Adam as this allows for optimal learning. With the above bias corrected values of mt and vt we finally update our parameters using the Adam update rule (Ruder, 2020):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Figure 9: The Adam update rule (Ruder,2020)

The loss function is how we measure the gap between the models' predictions and the expected output, it is the measurement which the optimisation algorithm works to minimise. The loss function is applied each epoch to evaluate the performance of the model and how it has changed compared to previous iterations (Alake, 2023). The loss function used in this paper is the binary cross entropy or log loss function. The process of the cross-entropy function first involves taking the predicted probabilities given by our model that a prediction is going to match the correct label. We take these probabilities and get the corrected probabilities for any that are predicted incorrectly, if x is has a probability of 0.67 being safe but it is correct then its corrected probability would be 0.33. We then get the log values of all these corrected probabilities and get the negative average for the loss. This process can be simplified into the following formula:

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} -\left(y_i * \log(p_i) + (1-y_i) * \log(1-p_i) \right)$$

Figure 10: Binary Cross-Entropy formula (Saxena,2023)

In the above figure pi is the probability of an email being a phishing or spam email and 1-pi is the probability of it being a safe email, with yi being the actual value of the label. We loop for each probability and depending on whether the probability is safe or unsafe we only use the corresponding part of the function, removing the other half for the current iteration (Saxena, 2023). Using this in tandem with the optimisation function above allows us to create a more efficient and accurate model every epoch until we eventually use it for testing. With these two functions and the metrics above, we can finally compile our model for training.

3.5 Fine-Tuning Phase

To explain fine tuning in more detail, BERT models all come with a base understanding of the English language due to their training on the Wikipedia corpus and Toronto book corpus. This understanding of the English language has no specific purpose until we fine-tune it to perform a task. By finding or creating datasets like the ones in this paper, we can further train BERT to 'fit' to the task we want it to perform. Using fine-tuning with labelled data, we can 'teach' BERT text classification. BERT doesn't inherently know how to do text classification which is why we must iteratively change its parameters to suit the task. By providing data relevant to the task, BERT will change its hyperparameters every single prediction it makes to better suit the task. BERT adapts from its mistakes until it starts to make accurate predictions that satisfy the task. After fine-tuning has finished, the model is now ready to be used on any excerpt of text to predict whether it thinks it is a malicious email or not. Due to how available and fine-tuneable BERT is, it has become one of the most popular models for text classification.

The fine-tuning of the models takes place using the two training sets and two validation sets we created in previous steps (Training1, Training2). The fine-tuning is done using the model.fit(fine-tune_set) method from the Keras module (Chollet et al, 2015).

```
[==============================] - 2433s 8s/step - loss: 0.6071 - accuracy: 0.6699 - precision: 0.6689 - recall: 0.6729 - val_loss: 0.5371
val_accuracy: 0.7472 - val_precision: 0.6823 - val_recall: 0.9120

Epoch 2/10

  1/321 [..............................] - ETA: 36:20 - loss: 0.5564 - accuracy: 0.6562 - precision: 0.5833 - recall:
0.9333                                                                                                          2/321
```

Figure 11: An example output from training a model in the Kaggle notebook (Goldbloom, 2010)

In the above figure we can see the raw output from the Kaggle notebook during runtime. It is a little messy, but it is showing the end of the first epoch and the

beginning of the second epoch. We have our metrics specified in the previous steps as well as the validation equivalent of these metrics and an ETA. At the end of the first epoch, we see the total time for getting through each email in the training set (2433 seconds in this case), with a prediction of the label made for each. These predictions are then evaluated using the validation sets and the actual labels of the training data, with the metrics changing each iteration through the set in an upward trend.

Each epoch contains 321 pieces of data with predictions to be made on each, with all data being balanced equally in every model regardless of the amount of data originally in each dataset. Each model took approximately 10-11 hours to train, ideally there would have been more time for training with larger datasets however there is a 12-hour limit to cloud sessions on services such as Kaggle and Google Colab (Google, N/A) (Goldbloom, 2010). Due to this limitation we had to ensure the training fit within this 12-hour limit, which meant reducing the number of epochs and minimising the size of training sets. On average, each epoch would take around 2800-2900≈ seconds to complete with a steady increase over each incrementation, finishing at around 2500≈ seconds per epoch. Since the metric values start low and rise per epoch as expected the results below just include the final values after fine-tuning each model.

Final Fine-Tuning Scores

MODEL	LOSS	ACCURACY	PRECISION	RECALL	VALIDATION LOSS	VALIDATION ACCURACY	VALIDATION PRECISION	VALIDATION RECALL
PHISH	0.3287	0.8693	0.8655	0.8745	0.3060	0.8933	0.9024	0.8785
SPAM	0.3128	0.8798	0.8801	0.8792	0.2746	0.9022	0.9098	0.8976
PHISHRON	0.0785	0.9810	0.9813	0.9807	0.0692	0.9869	0.9878	0.9865
FINAL	0.1325	0.9570	0.9458	0.9690	0.1125	0.9614	0.9503	0.9739

It is worth noting that the PhishRon set took considerably less time to train starting with a time of 1900≈ seconds per epoch and maintaining this time all the way through to the final epoch, even with the same amount of training data as the other sets. It is interesting to see that the time did not decrease between epochs, this could suggest that the emails within this model are more distinct, allowing the model to adapt to the task quicker and make more accurate decisions.

Since this is not the results section, I will not go too in depth on what we see above, but we can see clearly that the models with combined datasets have a much higher score in all metrics as predicted above. The Phish and Spam model have surprisingly low metrics compared to the combined models which indicates they could have done with more training time and data, which was unfortunately limited by the constraints mentioned previously.

I will add that the validation results above are essentially a testing set on exclusively the datasets they were trained on, as opposed to the combined dataset

testing that takes place below. Based on these intermediate results we can already see the PhishRon model is at least on par with modern day tools.

3.6 Testing Phase

All testing and experimenting took place using the **model.predict()** method from the keras module (Chollet et al, 2015). In this phase we take the two testing sets created before (Testing1, Testing2) and the experiment sets we create below, using **model.predict(data)** to classify each entry within the sets. The sets contain the bodies of text as mentioned above and when passed to the predict method, the model will make a prediction of the label for each body of text. We store these predictions in a finalPredicted variable and then flatten this using **flatten()** from the numpy module so that we have a list of the predictions in row order. We compare the predictions to Testing2 which contains the actual labels for the data we have predicted on. Since some values may not come out as 1 or 0, we round each value so that they match the format of the labels in Testing2. In the testing phase, each model is passed 93 pieces of data which it makes label predictions on. Ideally, we would also have liked the testing sets to be larger, but they were limited due to computational constraints. Since the experiment sets do not have labels, we just pass the model the data and then take the raw prediction and evaluate the scores manually. This allows us to look even further into the performance as we take the raw output from the dense layer instead of a 0 or 1, meaning we can measure the exact percentage the model believes the data to be.

3.7 Prompt Engineering and Advanced Prompt Techniques

Prompt Engineering is the process of creating an informative request to a generative AI to receive a suitable output. In the context of this paper, we will use Prompt Engineering to create a set of emails to test our models on. By artificially creating this data, we create an unbiased and unknown source for the testing of our models. Using advanced prompt engineering techniques, we can create detailed prompts that create adverse tests for our models. AI can only mimic what it has been trained on, so these emails may not be one to one with real world examples. However, they provide a baseline to test on that will still provide insight into the performance of the models. We are using a combination of both **ChatGPT** and Microsoft **CoPilot** (Microsoft Copilot, 2024) in this paper, generating the emails using ChatGPT and validating them through CoPilot. At both stages of the creation of most of the emails, advanced prompt engineering techniques such as **Few-Shot prompting** will be employed to guarantee the quality of our data. ChatGPT (ChatGPT, N/A) and CoPilot must adhere to ethics protocols, which is why there were certain limitations in this stage. Emails could not contain any specific information on people or businesses, so some of these fields have been manually filled to mimic recipients and the like. Despite this, the aim was to create a group of small datasets as per the following:

- Spam and phishing emails created using Zero-Shot prompting
- The same types of emails using few-shot prompting
- The same types of emails with Chain-of-Thought (COT) prompting

The aim of using prompt engineering is to create unbiased and challenging tests for our models. Advanced prompt engineering techniques are methods that have been devised by AI experts that allow expansive manipulation of generative AI and the output they generate. There has been a huge increase in these techniques due to the recent AI explosion, and they have been fully taken advantage of in this paper. The techniques used to create the emails are as follows:

- **Zero-Shot prompting**- This technique is the simplest form of prompting. It involves giving the task in whole to the generative LLM without any context or demonstration. It is essentially just asking the LLM to do the task and taking whatever output is received. This is the type of prompting used in the most basic spam and phishing emails created through prompt engineering techniques.

- **Few-Shot prompting**- This technique is the step-up of the previous. Instead of just giving the task to the LLM we instead provide context and examples where needed. By providing a clearer path for the model it creates a "stronger" output that more closely matches the desired output. This has been employed to create a mid-level set of spam and phishing emails.

- **Chain of thought prompting**- COT prompting involves taking intermediate steps with the LLM, to ensure each detail of the final "goal" has been considered. Instead of just asking the LLM to create an email we instead begin the task of creating the email and detail the steps to get to the desired goal. Combining this with few-shot prompting allows us to not only give context and examples of what we want, but also walk through each individual part altering aspects considered undesirable. This technique has been used with few-shot prompting to create a set of emails that will provide a real challenge to the BERT models.

- **Prompt Chaining**- Prompt chaining is the process of breaking tasks down into subtasks and using the output of one task to feed the next. Since this technique is complimentary to COT and Few-Shot prompting they have been used in tandem for this experiment. It helps avoid creating a lengthy overcomplicated prompt that tries to solve the problem all at once (DAIR.AI, 2024).

The process of creating an email using advanced techniques and validation through CoPilot went like so:

- Create a prompt plan tree- Set a goal to work towards and the necessary steps that need to be taken to achieve it. List all the 'intermediate steps' involved in the COT process.
- Begin the prompting sequence by starting on a small task, in the context of this paper this would be getting the LLM into a spam or spear phishing-oriented headspace and setting up a 'persona' for the emails writing style
- Follow the COT technique using chain prompting, breaking down tasks into subtasks and chaining them to reach a desired output.
- Once a satisfactory output is reached the email is then passed to CoPilot to check if any improvements or alterations can be made to ensure the email suits are demands.
- Pass the emails to the model for predictions.

Using these prompts 18 emails were created, each one following some of the techniques above. Below are examples of some of the emails:

Zero-Shot Spam:

Dear Customer, Congratulations!

You have been selected to get the chance at our exclusive VIP membership.

To activate your membership, please confirm your email with the link below.

http://website.com/claim-ur-vip

There is a limited number of memberships available, so make sure to confirm your details as soon as possible. We can't wait to have you on board!

Thank you!

COT Phish:

Your Future, Our Jobs:

Master the skills you require for your next big move!

Skill development is crucial for career growth and personal transitions. It will not only enhance your job prospects but also pave the way for long-term success. We are offering professional certificates to help you grow your skills to get you ready for the high-growth fields you desire.

We offer a list of course to help you grow on your journey:

Software Development Course: https://website.co/softdev

Graphic Design Course: http://website.com/graphdes

Please get in touch with us if you have any questions!

4. RESULTS AND EVALUATION

In this section we will cover the results from testing each of the models on both the publicly gathered datasets and the experiments detailed below. We gather the results for each experiment separately and investigate any interesting discoveries we have found. After we have inferred what we can, we will then evaluate the performance of the models against each other. The aim of this section is not only to determine which model is the greatest, but also to see if there are any additional insights we can garner from the cross comparison. For each model we gathered the metrics specified in the model creation section and the F1 score. The heatmaps of each model have also been gathered and placed in the appendix. All emails gathered from external sources have been anonymised to adhere to ethics guidelines. All experiments were run multiple times to ensure the final result did not contain any outliers that misrepresented our findings.

4.1 Dataset Testing Results and Evaluation

In this experiment the idea was to take a random piece of data from each dataset, whether safe or unsafe, and use the BERT models to predict on their class. This is the traditional form of testing that most AI models are put through after their training, it is the simplest form of test to ensure the models we have trained have not fallen victim to overfitting and can at least serve the purpose of correctly identifying a malicious email. Each piece of data was taken from the testing set created in previous steps to ensure it is new to each model. The reason we have chosen random pieces of data for each test instead of testing against just safe or just unsafe is to show the benefits a system like this would have in a real system architecture, which could have both safe and unsafe emails at random. The testing sets contain data from each dataset, to provide a simulation of a real-world inbox instead of just the data the models have been specifically trained against.

Table of Results for models

MODEL	ACCURACY	PRECISION	RECALL	F1SCORE
PHISH	0.8978	0.9121	0.8819	0.90
SPAM	0.9049	0.9015	0.9065	0.90
PHISHRON	0.9844	0.9796	0.9890	0.98
FINAL	0.9626	0.9523	0.9723	0.96

We can deduce a few interesting discoveries from the results above. The first two models are expectedly the worst in terms of overall performance, trained on a limited amount of data they both seem to perform at sub state-of-the-art levels. With scores floating around **90%,** they do not compete with the 99%+ scores that most modern tools today have. This underwhelming performance is clearly due to the **lack in variety of data**. Since each model is trained purely on a single type of malicious email, they struggle when it comes to discerning other types of emails that may or may not be malicious. This is backed up by the fact that the next two models outperform the Phish and Spam models in every way. Although this may not be the most surprising discovery, it shows us the necessity for future models to be trained on data that covers all aspects of the task at hand and how this exponentially increases performance.

The **PhishRon model** is astonishingly the most efficient of all the models, achieving results close to modern day tools without fine-tuning. Outperforming even the final model, the **PhishRon model** stands proud with scores ranging from **98-99% in accuracy**, precision, recall and F1 score. This shows how diverse data can greatly assist in model performance with a **10%~ jump in accuracy** from the Phish and Spam models. This model has also clearly benefitted from the influence of true "real-world" data in terms of the Enron dataset. Since the Enron dataset contains the emails of real people instead of just unverified data, like the phishing dataset, the model has a stronger sense in identifying the semantic differences between a genuine email between colleagues and a phishing email.

The **PhishRon model** outperforming the final model indicates that there is an optimum level of mixture when it comes to the data passed into the model during training. Since the final model is trained on three different datasets, the results point towards an overcomplication in the signs the model looks for when classifying emails. The final model classified **267 emails** as malicious when they were safe, and only **128** as safe when they were malicious. Compared to the Phish and Spam models, trained on only a single dataset which have a difference of around **10-20%** in the disparity between there false negatives and positives, the **Final model** has double the false positives to false negatives. The results show the **Final model** has been given too many "flags" to raise when it comes to scanning the emails it's been provided, which has led to lower scores than the **PhishRon model.**

The results at this stage have already shown us some concrete evidence to expected trends that were mentioned above. The variety of data, and influence of real-world data are paramount in creating tools that compete with state-of-the-art AI spoof protection services. Perhaps with access to larger datasets of real-world information and the resources to train models on such data these models could achieve 99%+ scores.

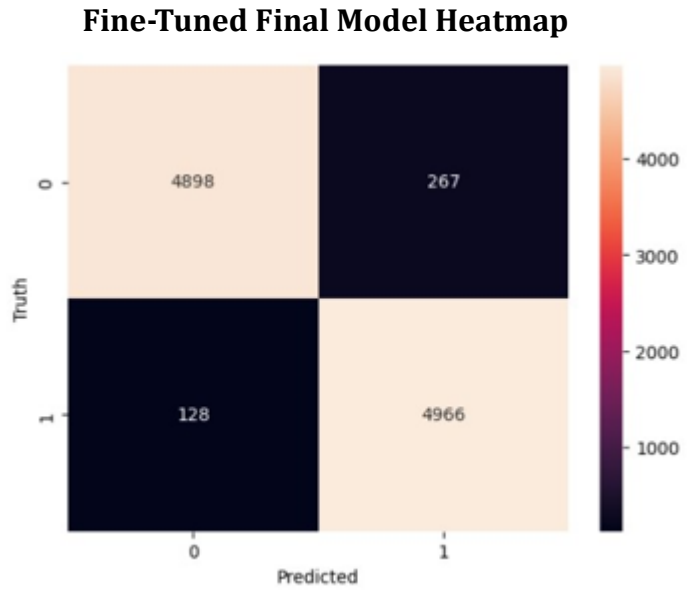

Figure 12: Results and heatmap from dataset testing for Final Model
(Goldbloom, 2010)

The above figure is a heatmap of the results from testing the final model. It shows the results mentioned in my evaluation, the map follows the format:

Top Left- False predictions when actually false (True Negative)

Top Right- True predictions when actually false (False Positive)

Bottom Left- False predictions when actually true (False Negative)

Bottom Right- True predictions when actually true (True Positive)

The heatmaps for the other models are contained in the appendix section for results referencing.

4.2 Prompt Testing Results and Evaluation

Spam Prompt Results

MODEL	ZERO-SHOT SPAM	FEW-SHOT SPAM	COT SPAM
PHISH	0.77	0.70	0.69
SPAM	0.67	0.62	0.60
PHISHRON	0.98	0.96	0.98
FINAL	0.90	0.87	0.84

The above results show the score each model achieved when being tested against the spam sets created using prompt engineering. This is different to the testing above as instead of just outputting a 0 or 1, the models give a score ranging from 0-1 based on how likely they think the email they are processing is malicious or not, with the higher the score being the more likely. All emails in these sets were malicious so no low scores below 0.5 were expected. In the spam section, most scores were above 0.5 with very few dipping below. Since the sets are small, getting one "wrong" technically means there is an accuracy of 66%. Seeing as this isn't the most insightful when it comes to analysing model performance, the scores above show an average score of how close they were to the classification threshold (0.5).

The results presented seem to fall in-line with what we would expect based on the previous testing and original predictions. The spam model achieved the lowest scores of all the models for the entire spam section, an unexpected result considering it was fine-tuned on the spam dataset. The result is most likely due to the diversity of spam attacks. Since spam emails are random in terms of content and language, it points to the model having been trained to point out too many signals perhaps confusing it in the long run. This supports claims made about the final model and its own issues with performance, showing the reason it underperforms compared to the PhishRon model is due to it being trained on the spam set.

Although the style of spam and phishing emails are entirely different, the malicious aspect is usually the same being a link or untrustworthy request. Since the phishing dataset includes more tailored emails with sneakily hidden malicious content, it trains the models to pick up on this subtlety, while the spam set contains a range of emails that may not try as hard to hide their malicious intent. Due to this, the Spam and Final models seem to be trained on classifying the obvious, but perhaps the more elusive emails of both the spear phishing and spam variety pass under the radar. In future, it may be worth considering foregoing fine-tuning such models using spam emails, as it seems to have left our models worse

off in the end. Instead, we should focus purely on spear phishing to train the models to pick up even the most subtle signs of malicious tampering. By training on real-world data and spear phishing emails as per the PhishRon model, it seems we could create an effective warning tool for use in a real-world scenario that compliments modern day tools. The PhishRon model correctly classified every email in both the spam and phishing sets, giving it a 100% accuracy score. Although the testing in this paper is limited, based on what we can gather, the model does compete with state-of-the-art tools.

Phish Prompt Results

MODEL	ZERO-SHOT PHISH	FEW-SHOT PHISH	COT PHISH
PHISH	0.72	0.65	0.61
SPAM	0.61	0.57	0.55
PHISHRON	0.98	0.95	0.96
FINAL	0.93	0.77	0.86

The table above shows the results on the phishing side of the emails created using prompt engineering. These results reflect what we have seen from the previous testing stage with the same rankings between the models. The models seem to have all achieved lower scores for most of this section, showing that spear phishing emails are harder to catch out than the average spam email. This is to be expected considering spear phishing emails are targeted and expertly curated whereas spam are sent out en masse. There isn't much in terms of insights into the models to take that hasn't been mentioned in the spam section other than this small dip in results.

4.3 Additional Experiments

This section is dedicated to some experiments that I found interesting and useful to the purpose of this paper. Although the main focus of the results section was the two previous experiments, the ideas explored here prove the models may have the potential for real-world use.

4.3.1 Work Emails

This experiment is the perfect test to see if the models we have created can compete with modern day tools. I have gathered these emails from my personal inbox at work, two malicious emails that managed to slip through the spoof detection service employed by Microsoft Outlook (Chrisda, 2024). The Outlook security service either marks malicious emails as junk or entirely removes them from users' inboxes, however no system is invincible. Most modern-day tools have a 99.9% accuracy when it comes to predicting these emails, the 0.1% that does manage to slip through can still have monumental security risks proven by the

case studies covered previously. I have tasked the models with predicting how likely they think the emails are to be malicious.

Work Email Results

MODEL	EMAIL 1	EMAIL 2
PHISH	0.78	0.83
SPAM	0.54	0.68
PHISHRON	0.95	0.96
FINAL	0.78	0.88

Both the emails I gathered were phishing emails, so it is good to see that all the models have correctly predicted them as such. The scores themselves seem to match up to what we have seen previously, with the spam model getting dangerously close to misclassifying one of the emails. These models are obviously not superior to modern day security services and still require a lot of work, however the results clearly show the use these models could have as a complementary attachment to email services such as Outlook. The service could take these scores and use them as warning signs when a user clicks on a phishing email. Additional human training is usually the last line of defence when it comes to emails that have slipped through the cracks like the ones above, so additional support such as these tools could help tremendously.

4.3.2 Manually Created Emails

These are a set of emails that I have manually created for this experiment. I essentially just mimicked my own style of emails at work and created four safe emails, with one spear-phishing email mixed in between. I mimicked the style of myself at work as that is where tools like these would be put in place and where most phishing attack are received. The aim here was to see if the model has picked up on the tone and semantic meanings of text and whether they can identify a malicious email when it may follow a similar style to some safe emails.

Manual Email Results

MODEL	MANUALLY MALICIOUS EMAIL SCORES

PHISH	0.76
SPAM	0.63
PHISHRON	0.98
FINAL	0.93

I have not included the results from my emails that were “safe” as they were all classified correctly and past that point do not really provide much for us to learn from. The results presented in the table above prove the models have the capabilities to discern phishing emails even when presented with a set of email in identical style, language and format. The BERT models deep understanding of the semantics of the English language have been put on full display in this experiment, showing us exactly why it was the correct LLM for this experiment. All the models managed to correctly classify the odd email of the bunch, even if the levels vary between them. As expected, the PhishRon model comes out on top yet again with a score of 0.98. If businesses had access to additional resources like these it could help prevent interorganisational attacks and provide an extra layer to employee inboxes when dealing with any phishing emails that come their way.

4.3.3 The Perfect Phishing Email

This was the final experiment I decided to include in the paper. While all the other experiments so far have been to evaluate the performance of the model this one is a little different. The aim here was to create an email that could entirely bypass the models, giving us a visual representation of what it takes to get past these systems. In the end, this became a trial-and-error process of reworking the same email until it could fool the models. I used a combination of all the techniques above incorporating manual work and advanced prompt engineering techniques to create as complex an email as possible. The results are below:

Perfect Email Results

MODEL	PERFECT EMAIL SCORE
PHISH	0.61
SPAM	0.56
PHISHRON	0.78
FINAL	0.60

Based on the results, you would not think the email I created was challenging at all. Even using all the techniques and manipulation I could muster; I still could not manage to “break” the models after multiple attempts. The results above are what the models achieved after I had spent the better half of a day altering a single email to make it as undetectable as possible. I ran out of ideas after passing it through Copilot and ChatGPT too many times to count.

Although this experiment was technically a failure, it is promising to see the models stand up to the best of my spear phishing attempts. I guess this means I don't have a promising future career as a phisherman.

5. DISCUSSION AND CONCLUSION

5.1 Discussion

5.1.1 System Architecture Application

To fully understand the potential of these models I have created a simple system architecture diagram to illustrate how these tools can be used in tandem with existing systems to make a real impact. Please see the below figure:

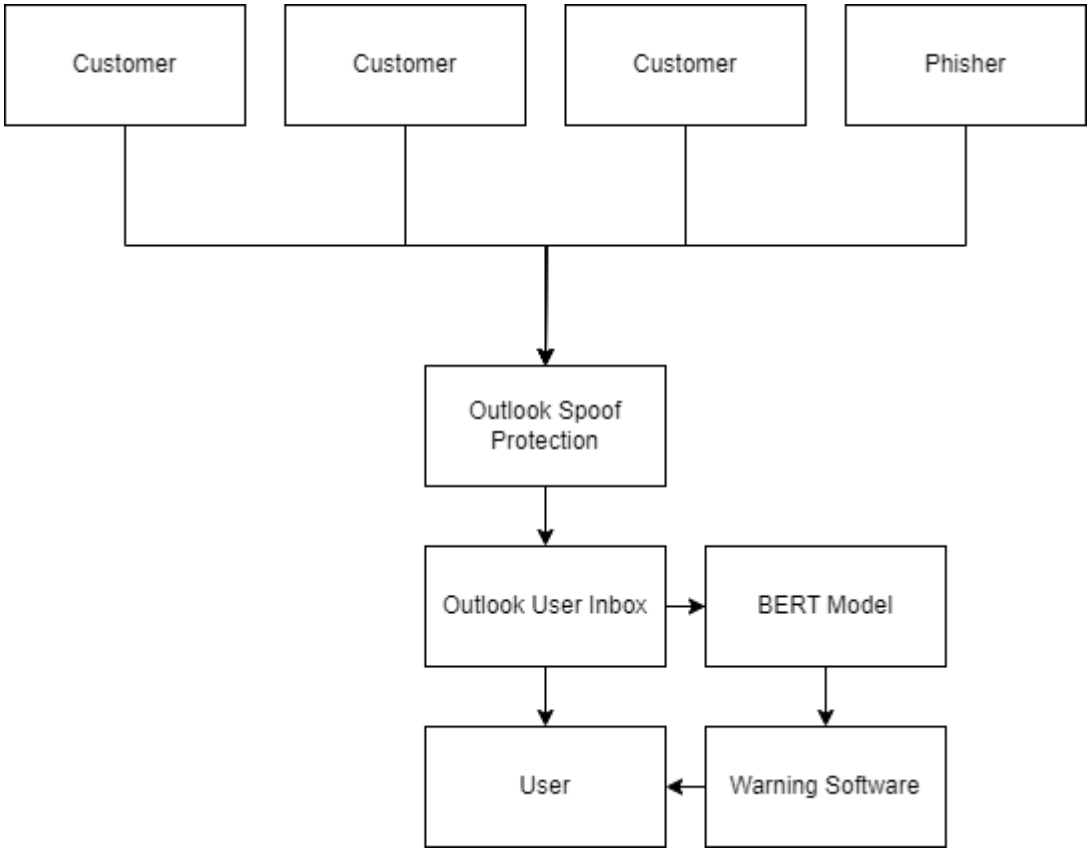

Figure 13: A system architecture diagram demonstrating the ideal real-world use for the BERT models

In the above diagram, we have a range of components in a simple workplace setting. These closely emulate my experience of office work and how emails are an integral part of customer communication and inter organisational communication. Users/Employees will receive upwards of a hundred emails a day and while most of these emails will have been vetted by the Microsoft Outlook spoof detection it is possible for a malicious email to slip through, just like the email seen in the experiments section of this paper. As demonstrated through our results, by having the models we have created today act as an additional tool on top of this, we can provide users potential warnings and guidance when it comes

to the email in their inbox. The models could be used in tandem with a small program as follows:

1. A scanner program that takes any new emails received to the User inbox and passes them to the BERT model
2. The input is then pre-processed using the steps previously mentioned
3. The pre-processed input is then encoded as above
4. The BERT model generates an output from 0-1 like the scores in the experiments section
5. Pass the models output to a new program which warns the user based on how close the value is to 0 or 1 with varying levels of severity.
6. User receives warning and makes informed decision on what to do with email

Using the raw score output from BERT, we can allocate thresholds on how we should warn the user, the higher the score the more severe the warning. By developing the system further, we could even present the specific parts of the email that present the perceived threat for the users' further analysis. The user would ultimately make the decisions in a system like this, with the models providing advice, as at the end of the day we can never 100% guarantee an email is safe or unsafe using just the models.

5.1.2 Future Applications

Moving on from this project and into the future, there are several ways the discoveries in this paper can be utilised. In a future study, if given more time to dive deeper into these topics, it could be possible to create an AI that perfectly complements modern day tools, like the proposed system above. Since the time and resources in this project were limited to free environments and public data, it would be interesting to see how models would perform when given adequate computational power and expertly curated data. The results from our experiments can be utilised as we have shown both areas of weakness and strength in the performance of these AI models.

5.1.3 Assumptions

Since the majority of the aims in this paper have been achieved there isn't much to say in terms of assumptions made. However, there were many drawbacks in terms of resources that may have held back some of the models from becoming as accurate as they could potentially be. Due to the project taking place entirely using the Kaggle notebook environment, the time for fine-tuning the models was incredibly limited. Ideally, the aim would be to set up all the fine-tuning data in larger sets and have all the data used instead of just portions. Due to the runtime limit on Kaggle the fine-tuning sets had to be cut down to a fraction of their original sizes. This could have been circumvented using personal hardware, however the devices available to fine-tune all the models to the desired extent were not sufficient for the task. In future, if these models were given access to the resources they required, they could meet the 99%+ barrier we have tried to pass. Another assumption made is that the publicly gathered datasets are entirely

valid. Checking was performed to ensure the quality of the data; however, the datasets all contain at minimum tens of thousands of emails making it impossible to check each one. Since some of the models have achieved high scores in the end it is assumed that they are valid.

5.2 Conclusion

In conclusion, the aims set out at the beginning of this paper to create a model that can detect the semantic undertones of human language and behaviour, in-line with modern day tools, have been achieved. The PhishRon model not only achieves scores that would be expected of modern-day tools, but it also manages to identify emails that have slipped through the tight-knit protection of such services. Using fine-tuning on the rigorously selected datasets, we have shown that the influence of real-world data is paramount to creating models as efficient as state-of-the-art security services. The correlation between the increase in performance and the variety and quality of the data used to fine-tune has been clearly shown through the results in both the fine-tuning stage, dataset testing and experiments. The experiments conducted in this paper prove that BERT is a leading LLM when it comes to text classification, with its versatility to be fine-tuned to any text classification task and its ease of use for quickly building up new models for testing. Although a perfect defence against spear phishing attacks will never exist, it is comforting to know that tools can be developed and expanded upon just like the ones in this paper. Through the combination of these models and existing tools, it seems possible to create a shield that can protect users in personal and organisational respects. The ramifications of such tools could be monumental in saving future Jack Todey's. At the end of the day, these tools only serve as a veil that protects the end user, and ultimately there is no way to completely prevent phishing attacks. Considering the results of the paper, it is still not safe to say users should not be made painfully aware of both the signs and consequences of these attacks. The end user will always remain the final firewall, the trust we put in them should surpass those we put in the models. However, combining these tools with an expected level of personnel training on the signs of phishing emails, the models seem more than capable in effectively identifying these attacks and providing users with the resources to report and remove them.

6. APPENDICES

6.1 Dataset Formatting

- Read dataset from input folder that has dataset stored

:	Unnamed: 0	Email Text	Email Type
0	0	re : 6 . 1100 , disc : uniformitarianism , re ...	Safe Email
1	1	the other side of * galicismos * * galicismo *...	Safe Email
2	2	re : equistar deal tickets are you still avail...	Safe Email
3	3	\nHello I am your hot lil horny toy.\n I am...	Phishing Email
4	4	software at incredibly low prices (86 % lower...	Phishing Email
5	5	global risk management operations sally congra...	Safe Email
6	6	On Sun, Aug 11, 2002 at 11:17:47AM +0100, wint...	Safe Email
7	7	entourage , stockmogul newsletter ralph velez ...	Phishing Email
8	8	we owe you lots of money dear applicant , afte...	Phishing Email
9	9	re : coastal deal - with exxon participation u...	Safe Email

Figure 4: Unaltered first 10 entries from the Phishing dataset, labelled either safe or phishing (Goldbloom, 2010)

- Drop unnecessary columns

	Email Text	Email Type
0	re : 6 . 1100 , disc : uniformitarianism , re ...	Safe Email
1	the other side of * galicismos * * galicismo *...	Safe Email
2	re : equistar deal tickets are you still avail...	Safe Email
3	\nHello I am your hot lil horny toy.\n I am...	Phishing Email
4	software at incredibly low prices (86 % lower...	Phishing Email
5	global risk management operations sally congra...	Safe Email
6	On Sun, Aug 11, 2002 at 11:17:47AM +0100, wint...	Safe Email
7	entourage , stockmogul newsletter ralph velez ...	Phishing Email
8	we owe you lots of money dear applicant , afte...	Phishing Email
9	re : coastal deal - with exxon participation u...	Safe Email

Figure 5: The same 10 rows from the phishing table with the “Unnamed” column removed

(Goldbloom, 2010)

- Separate safe emails and unsafe emails

	Email Text	Email Type
0	re : 6 . 1100 , disc : uniformitarianism , re ...	Safe Email
1	the other side of * galicismos * * galicismo *...	Safe Email
2	re : equistar deal tickets are you still avail...	Safe Email
5	global risk management operations sally congra...	Safe Email
6	On Sun, Aug 11, 2002 at 11:17:47AM +0100, wint...	Safe Email

Figure 6: The first 5 rows of the newly created set of the safe emails from the Phishing set (Goldbloom, 2010)

- From the larger set of emails (varies between datasets) create a new set of emails that matches the smaller set in size

```
phishsetSafe.shape
```

```
(11322, 2)
```

```
phishsetSafeBalanced.shape
```

```
(7328, 2)
```

Figure 7: The first image depicts the size of the safe phishing set before being scaled down in the second image to match the size of the unsafe phishing set (Goldbloom, 2010)

- Concatenate these two sets together to create a balanced set of safe and unsafe emails

```
balancedFish.info
```

```
<bound method DataFrame.info of
15156  re : revised contact list . note changes to ex...      Safe Email
1866   On Thu, 12 Sep 2002, Tom wrote:> Yep, that ws ...      Safe Email
15760  Dave Long wrote:>>\n>>    \n>>\n>>>"The United...      Safe Email
7141                                                empty      Safe Email
7447   Once upon a time, Brian wrote :> \n>     I hav...      Safe Email
...                                                  ...             ...
18634  congratulations you have won ! ! ! pls contact...  Phishing Email
18637                                               empty  Phishing Email
18638  strong buy alert : monthly newsletter topstock...  Phishing Email
18645  date a lonely housewife always wanted to date ...  Phishing Email
18649                                               empty  Phishing Email

       phish
15156      0
1866       0
15760      0
7141       0
7447       0
...      ...
18634      1
18637      1
18638      1
18645      1
18649      1

[14656 rows x 3 columns]>
```

Figure 8: The .info of the newly created balanced set, with the same number of safe and phishing emails (Goldbloom, 2010)

- Apply a function to the new dataset that applies a 1 if the corresponding data label is unsafe ('Phishing Email' in the above example) and a 0 in any other case • Finally fill any null values with an empty string for the AI model

7]:

	Unnamed: 0	Email Text	Email Type
0	0	re : 6 . 1100 , disc : uniformitarianism , re ...	Safe Email
1	1	the other side of * galicismos * * galicismo *...	Safe Email
2	2	re : equistar deal tickets are you still avail...	Safe Email

Figure 9: Example excerpt from phishing dataset pre-alteration (Goldbloom, 2010)

6.2 Training Photos

Phish Model:

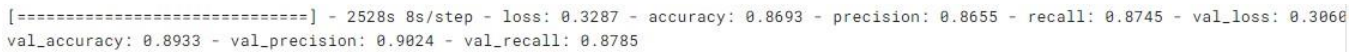

Figure 16: Results from training Phish Model in Kaggle notebook (Goldbloom, 2010)

Spam Model:

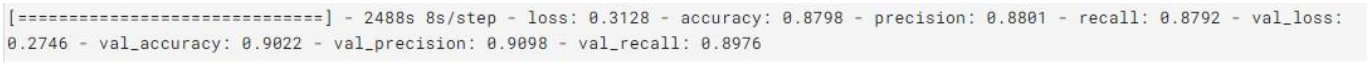

Figure 17: Results from training Spam Model in Kaggle notebook (Goldbloom, 2010)

PhishRon Model:

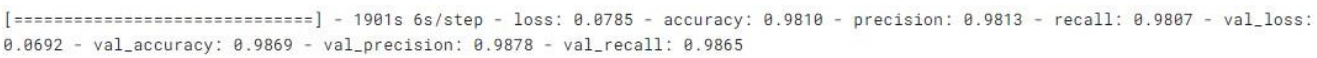

Figure 18: Results from training PhishRon Model in Kaggle notebook (Goldbloom, 2010)

Final Model:

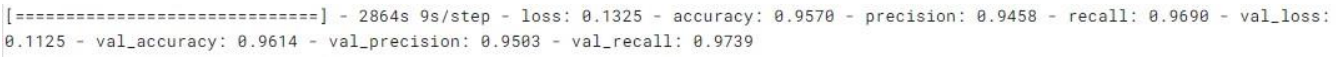

Figure 19: Results from training Final Model in Kaggle notebook (Goldbloom, 2010)

6.3 Heatmaps

Phish Model

```
93/93 [==============================] - 631s 7s/step - loss: 0.2990 - accuracy: 0.8978 - precision: 0.9121 - recall: 0.8819
321/321 [==============================] - 2221s 7s/step
              precision    recall  f1-score   support

           0       0.89      0.90      0.90      5129
           1       0.90      0.89      0.90      5130

    accuracy                           0.90     10259
   macro avg       0.90      0.90      0.90     10259
weighted avg       0.90      0.90      0.90     10259
```

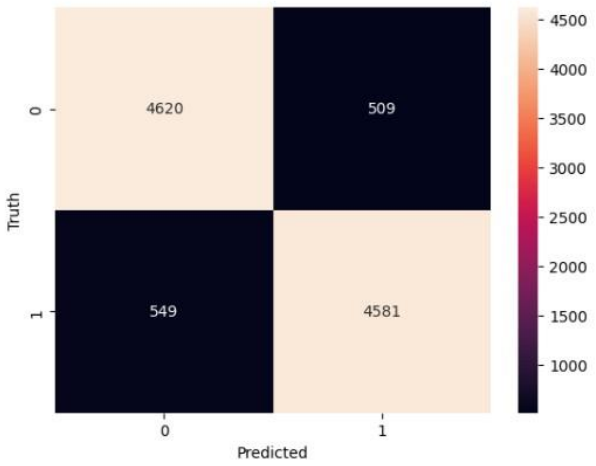

Figure 20: Results and heatmap from dataset testing for Phish Model
(Goldbloom, 2010)

Spam Model

```
93/93 [==============================] - 628s 7s/step - loss: 0.2859 - accuracy: 0.9049 - precision: 0.9015 - recall: 0.9065
321/321 [==============================] - 2144s 7s/step
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      5135
           1       0.89      0.91      0.90      5124

    accuracy                           0.90     10259
   macro avg       0.90      0.90      0.90     10259
weighted avg       0.90      0.90      0.90     10259
```

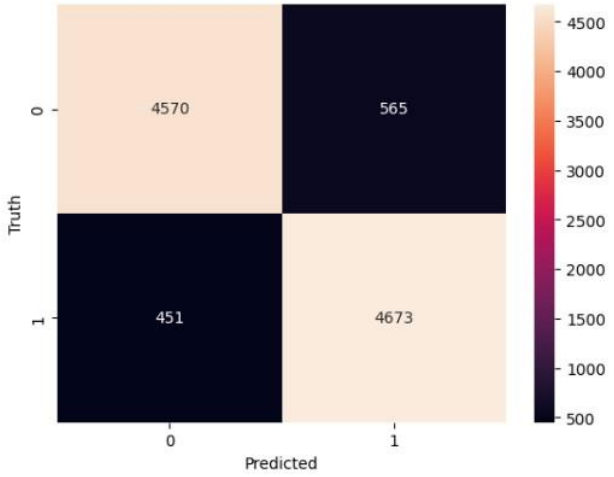

Figure 21: Results and heatmap from dataset testing for Spam Model (Goldbloom, 2010)

PhishRon Model

```
93/93 [==============================] - 480s 5s/step - loss: 0.0668 - accuracy: 0.9844 - precision: 0.9796 - recall: 0.9890
321/321 [==============================] - 1642s 5s/step
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      5130
           1       0.98      0.99      0.98      5129

    accuracy                           0.98     10259
   macro avg       0.98      0.98      0.98     10259
weighted avg       0.98      0.98      0.98     10259
```

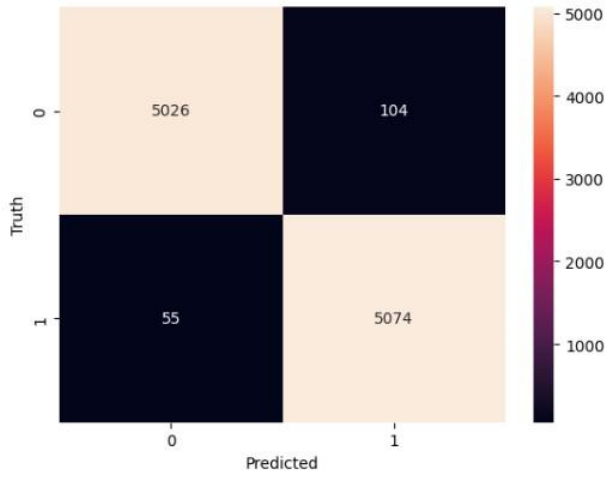

Figure 22: Results and heatmap from dataset testing for PhishRon Model (Goldbloom, 2010)

Final Model

```
93/93 [==============================] - 712s 8s/step - loss: 0.1107 - accuracy: 0.9626 -
precision: 0.9523 - recall: 0.9732
321/321 [==============================] - 2477s 8s/step
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      5165
           1       0.95      0.97      0.96      5094

    accuracy                           0.96     10259
   macro avg       0.96      0.96      0.96     10259
weighted avg       0.96      0.96      0.96     10259
```

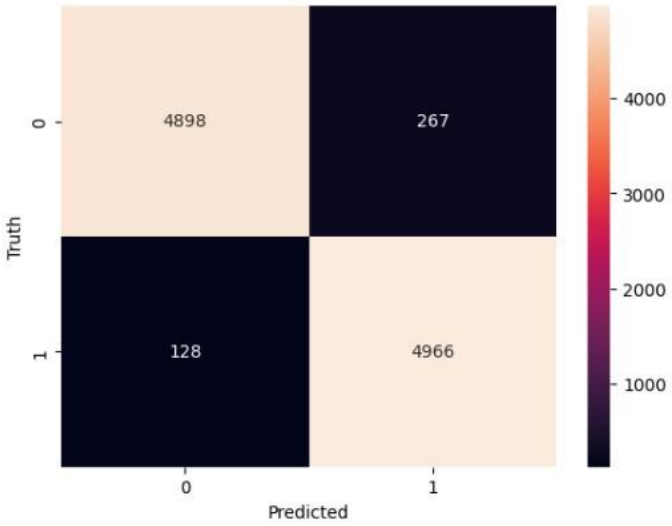

Figure 23: Results and heatmap from dataset testing for Final Model (Goldbloom, 2010)

6.4 Old System architecture diagram

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 text (InputLayer)           [(None,)]                 0         []

 keras_layer (KerasLayer)    {'input_word_ids': (None,  0         ['text[0][0]']
                             128),
                              'input_mask': (None, 128)
                             , 'input_type_ids': (None,
                              128)}

 keras_layer_1 (KerasLayer)  {'sequence_output': (None,  109482241  ['keras_layer[0][0]',
                              128, 768),                             'keras_layer[0][1]',
                              'encoder_outputs': [(None              'keras_layer[0][2]']
                             , 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768),
                              (None, 128, 768)],
                              'default': (None, 768),
                              'pooled_output': (None, 7
                             68)}

 dropout (Dropout)           (None, 768)               0         ['keras_layer_1[0][13]']

 output (Dense)              (None, 1)                 769       ['dropout[0][0]']

=================================================================================================
Total params: 109483010 (417.64 MB)
Trainable params: 769 (3.00 KB)
Non-trainable params: 109482241 (417.64 MB)
_________________________________________________________________________________________________
```

REFERENCES

1. Griffiths, C. (2024) *The latest Cyber Crime Statistics (updated January 2024): Aag it support*, *AAG IT Services*. Available at: https://aag-it.com/the-latest-cyber-crimestatistics/#:~:text=Data%20breaches%20cost%20businesses%20an,a%20cyber%20att ack%20in%202022. (Accessed: 15 January 2024).
2. Chin, K. (2024) *19 most common types of phishing attacks in 2024: Upguard*, *RSS*. Available at: https://www.upguard.com/blog/types-of-phishing-attacks (Accessed: 23 January 2024).
3. Published by Petrosyan Ani Petrosyan, A. and 18, A. (2023) *Spear-phishing attacks impact in global firms 2022*, *Statista*. Available at: https://www.statista.com/statistics/1406109/companies-worldwide-impact-spearphishing/ (Accessed: 23 January 2024).
4. Lutkevich, B. (2020) *What is Bert (language model) and how does it work?*, *Enterprise AI*. Available at: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model (Accessed: 12 February 2024).
5. Toiziz (2019) *Transformer (Deep Learning Architecture)*, *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture) (Accessed: 12 February 2024).
6. Vaswani, A et al. (2017) *Attention is all you need - neurips*. Available at: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aaPaper.pdf (Accessed: 12 February 2024).
7. Raj, B et al. (2019) *Bert (Language Model)*, *Wikipedia*. Available at: https://en.wikipedia.org/wiki/BERT_(language_model) (Accessed: 12 February 2024).
8. R. A. A. Helmi, C. S. Ren, A. Jamal and M. I. Abdullah, "Email Anti-Phishing Detection Application," 2019 IEEE 9th International Conference on System Engineering and Technology (ICSET), Shah Alam, Malaysia, 2019, pp. 264-267, doi: 10.1109/ICSEngT.2019.8906316. keywords: {Phishing;Electronic mail;Testing;Decision trees;Software;Information science;phishing;phishing detection;email phishing;information security},
9. S. Kaddoura, O. Alfandi and N. Dahmani, "A Spam Email Detection Mechanism for English Language Text Emails Using Deep Learning Approach," 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, France, 2020, pp. 193-198, doi: 10.1109/WETICE49692.2020.00045. keywords: {Deep learning;Unsolicited email;Phishing;Neural networks;Information filters;Malware;Optimization;spam detection;text-based analysis;email spammers;deep learning;machine learning},

10. Tyrell (2005) *Feedforward Neural Network*, *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Feedforward_neural_network (Accessed: 14 February 2024).
11. Kundu, R. (2022) *F1 score in Machine Learning: Intro & Calculation*, *V7*. Available at: https://www.v7labs.com/blog/f1-scoreguide#:~:text=F1%20score%20is%20a%20machine%20learning%20evaluation%20metric%20that%20measures,prediction%20across%20the%20entire%20dataset. (Accessed: 14 February 2024).
12. Benítez-Andrades1*, J.A. *et al.* (2022) *Traditional machine learning models and bidirectional encoder representations from transformer (bert)–based automatic classification of tweets about eating disorders: Algorithm development and Validation Study*, *JMIR Medical Informatics*. Available at: https://medinform.jmir.org/2022/2/e34492 (Accessed: 20 February 2024).
13. Gamache, R., Kharrazi, H. and Weiner, J.P. (2018) *Public and Population Health Informatics: The bridging of big data to benefit communities*, *Yearbook of Medical Informatics*. Available at: https://www.thieme-connect.com/products/ejournals/html/10.1055/s-0038-1667081 (Accessed: 20 February 2024).
14. Kumari, K. (2023) *Roberta: A modified Bert Model for NLP*, *Comet*. Available at: https://www.comet.com/site/blog/roberta-a-modified-bert-model-fornlp/#:~:text=Differences%20between%20RoBERTa%20and%20BERT&text=Training%20Corpus%3A%20RoBERTa%20is%20trained,masked%20in%20each%20training%20example. (Accessed: 20 February 2024).
15. Holt-Nguyen, C. (2023) *Battle of the transformers: Fine-tune Roberta and Distilbert for Sota sentiment analysis using Hugging face*, *LinkedIn*. Available at: https://www.linkedin.com/pulse/battle-transformers-fine-tune-roberta-distilbert-sotaholt-nguyen/ (Accessed: 20 February 2024).
16. Tam, A. (2023) *A brief introduction to bert*, *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/a-brief-introduction-to-bert/ (Accessed: 21 February 2024). –
17. Goldbloom, A. (2010) *Your machine learning and Data Science Community*, *Kaggle*. Available at: https://www.kaggle.com/ (Accessed: 21 February 2024).
18. Cukierski, W. (2016) *The Enron email dataset*, *Kaggle*. Available at: https://www.kaggle.com/datasets/wcukierski/enron-email-dataset/data (Accessed: 21 February 2024).
19. Singhvi, P. (2023) *Spam Email Classification Dataset*, *Kaggle*. Available at: https://www.kaggle.com/datasets/purusinghvi/email-spam-classification-dataset (Accessed: 21 February 2024).
20. Cop, C. (2023) *Phishing email detection*, *Kaggle*. Available at: https://www.kaggle.com/datasets/subhajournal/phishingemails?rvi=1 (Accessed: 21 February 2024).
21. Nielsen, F. (2021) *Prompt engineering*, *Wikipedia*. Available at:

https://en.wikipedia.org/wiki/Prompt_engineering (Accessed: 21 February 2024).
22. Beast, T.D. (2023) *Attention is all you need:: Summary & important points*, *Medium*. Available at: https://medium.com/@thedatabeast/attention-is-all-you-need-summaryimportant-points-40769b99d6f8 (Accessed: 22 February 2024).
23. Medical Liability Trust, T. (2023) *Cyber fraud case study: Failure to recognize phishing email*, *Resource Hub*. Available at: https://hub.tmlt.org/case-studies/cyberfraud-case-study-failure-to-recognize-phishing-email (Accessed: 23 February 2024).
24. Conner, L. (2023b) *I Fell for a Phishing Scam: A Case Study*, *about identity theft*. Available at: https://www.aboutidentitytheft.co.uk/i-fell-for-phishing-scam.html (Accessed: 23 February 2024).
25. NumFOCUS (2009) *Pandas*, *pandas*. Available at: https://pandas.pydata.org/ (Accessed: 28 February 2024).
26. Hvilshøj, F. (2022) *Introduction to balanced and imbalanced datasets in Machine*
Learning, *Balanced and Imbalanced Datasets in Machine Learning [Full Introduction]*. Available at: https://encord.com/blog/an-introduction-to-balanced-andimbalanced-datasets-in-machine-learning/ (Accessed: 28 February 2024).
27. Germec, M. (2023) *Text preprocessing with Natural Language Processing (NLP)*, *LinkedIn*. Available at: https://www.linkedin.com/pulse/text-preprocessing-naturallanguage-processing-nlp-germec-phd/ (Accessed: 28 February 2024).
28. Abadi.M et al TensorFlow: Large-scale machine learning on heterogeneous systems,
2015. Software available from tensorflow.org.(Accessed 28 Febuary 2024)
29. Wikipedia contributors. (2024, January 26). Training, validation, and test data sets. In *Wikipedia, The Free Encyclopedia*. Retrieved 16:02, February 28, 2024, from https://en.wikipedia.org/w/index.php?title=Training,_validation,_and_test_data_sets&oldid=1199288619
30. Yadav, H. (2022) *Dropout in neural networks*, *Medium*. Available at: https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9 (Accessed: 29 February 2024).
31. Jassy et al, A. (2024) *What is overfitting? - overfitting in machine learning explained - AWS*, *aws*. Available at: https://aws.amazon.com/what-is/overfitting/ (Accessed: 29 February 2024).
32. Chollet, F. and Others (2015) *Keras Dense Layer*. Available at: https://keras.io/ api/layers/core_layers/dense
33. Vishwakarma, S. (2023) *How to understand sigmoid function in artificial neural networks?*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2023/01/why-is-sigmoid-function-importantin-artificial-neural-networks/ (Accessed: 01 March 2024).
34. Chollet, F. and Others (2015) *Keras Embedding Layer*. Available at: https://keras.io/ api/layers/core_layers/embedding

35. Tinkerd (2023) *Bert Tokenization*, *Understanding BERT | Tokenization*. Available at: https://tinkerd.net/blog/machine-learning/berttokenization/#:~:text=IDs%20101%20and%20102%20are, automatically%20into%20t he%20tokenizer%20output. (Accessed: 01 March 2024).
36. HuggingFace *Glossary*. Available at: https://huggingface.co/docs/transformers/en/glossary (Accessed: 01 March 2024).
37. TensorFlow (2024) *Common savedmodel apis for text tasks : tensorflow hub*, *TensorFlow*. Available at: https://www.tensorflow.org/hub/common_saved_model_apis/text#transf ormerencoders (Accessed: 01 March 2024).
38. Xiao, H. (2018) *Frequently asked questions¶*, *bert*. Available at: https://bert-asservice.readthedocs.io/en/latest/section/faq.html (Accessed: 18 March 2024).
39. PapersWithCode (no date) *Papers with code - average pooling explained*, *Explained | Papers With Code*. Available at: https://paperswithcode.com/method/averagepooling#:~:text=Average% 20Pooling%20is%20a%20pooling,used%20after%20a%20 convolutional%20layer. (Accessed: 18 March 2024).
40. Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-AveragePooling-Figure-2-above-shows-an-example-of-max_fig2_333593451 [accessed 18 Mar, 2024]
41. Dhami, D. (2020) *Understanding Bert-word embeddings*, *Medium*. Available at: https://medium.com/@dhartidhami/understanding-bert-word-embeddings7dc4d2ea54ca (Accessed: 18 March 2024).
42. Chollet, F. and Others (2015) *Keras Accuracy Metrics*. Available at: https://keras.io/api/metrics/accuracy_metrics/
43. Chollet, F. and Others (2015) *Keras Classification Metrics*. Available at: https://keras.io/api/metrics/classification_metrics/
44. Wikipedia contributors. (2023, June 6). Learning rate. In *Wikipedia, The Free Encyclopedia*. Retrieved 11:40, March 19, 2024, from https://en.wikipedia.org/w/index.php?title=Learning_rate&oldid=1158838577
45. Kingma, D. P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization', *arXiv [cs.LG]*. Available at: http://arxiv.org/abs/1412.6980.
46. Alake, R. (2023) *Loss functions in machine learning explained*, *DataCamp*. Available at: https://www.datacamp.com/tutorial/loss-function-in-machine-learning (Accessed: 19 March 2024).
47. Ruder, S. (2020) *An overview of gradient descent optimization algorithms*, *ruder.io*. Available at: https://www.ruder.io/optimizing-gradientdescent/#stochasticgradientdescent (Accessed: 19 March 2024).
48. Saxena, S. (2023) *Binary cross entropy/log loss for binary classification*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/03/binary-crossentropy-log-loss-for-binary-classification/ (Accessed: 20 March 2024).

49. Chollet, F. and Others (2015) *Keras Model training APIs*. Available at: https://keras.io/api/models/model_training_apis/
50. Barracuda (2024) *Phishing and impersonation protection*, *Barracuda Networks*. Available at: https://www.barracuda.com/products/email-protection/phishingprotection (Accessed: 26 March 2024).
51. *Chatgpt* (no date) *ChatGPT*. Available at: https://chat.openai.com (Accessed: 02 April 2024).
52. Google (no date) *Colab.google*, *colab.google*. Available at: https://colab.google/ (Accessed: 04 April 2024).
53. Devlin, J. *et al.* (2019) 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', *arXiv [cs.CL]*. Available at: http://arxiv.org/abs/1810.04805.
54. DAIR.AI (2024) *About – nextra*, *Prompt Engineering Guide*. Available at: https://www.promptingguide.ai/about (Accessed: 14 August 2024).
55. Microsoft Copilot (2024) 'Conversational AI Service', 15 August. Available at: https://www.microsoft.com/copilot.
56. Chrisda (no date) *Spoof intelligence insight - microsoft defender for office 365*, *Spoof intelligence insight - Microsoft Defender for Office 365 | Microsoft Learn*. Available at: https://learn.microsoft.com/en-us/defender-office-365/anti-spoofing-spoof-intelligence (Accessed: 15 August 2024).

LIST OF CHANGES

1. Added to the introduction, background and methodology
2. Changed testing results and evaluation by adding all results into table and having more coherent comparison
3. Improved the styling of the report with consistent fonts and easier reading through the use of spacings and structure
4. Improved the flow of the report through moving around sections, especially in the earlier half.
5. Removed some lengthy sections such as the dataset instructions and moved them to the appendix, keeping enough information to summarise the topic in the report itself.
6. Added a discussion and assumption section separate to the conclusion, redone conclusion itself to add in new info from experiments and general rework
7. Talked about how the model would work in a modern-day system architecture
8. Changed punctuation errors and wording throughout paper
9. Added more in-depth coverage of both the fine-tuning process and advanced prompt engineering techniques, Better explained fine-tuning and how it is essentially another term for training
10. Added an experiments section detailing different tests I put the models through to test the fine tuning of the models
11. Changed the testing from testing models on datasets only from their own emails to testing them on a mix of email from all the datasets, makes the testing results different from training results and provides more useful results in terms of real-world applications
12. Added additional explanations in methods used for both preprocessing and fine-tuning
13. Added a detailed explanation of the new advanced techniques used and a step by step guide on how I generated the COT variety of email