Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Aksel Baardsen

Phishing and Social Engineering Attack Detection by Applying Intention Detection Methods

Phishing Detection using Intent Detection Methods

Master's thesis in Information Security
Supervisor: Sule Yildirim Yayilgan
Co-supervisor: Sarang Shaikh
June 2022

NTNU
Norwegian University of Science and Technology

Aksel Baardsen

Phishing and Social Engineering Attack Detection by Applying Intention Detection Methods

Phishing Detection using Intent Detection Methods

Master's thesis in Information Security
Supervisor: Sule Yildirim Yayilgan
Co-supervisor: Sarang Shaikh
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Abstract

Nowadays, email and SMS are used daily by almost everyone in the developed part of the world. A threat which follows this trend is phishing. Phishing and social engineering attacks are among the most common form of cyberattacks experienced, as they are not complex for an attacker to put together. In this thesis we attack the phishing problem using models gathered from the domain of intention detection. Intent detection is a sub-genre of natural language processing, and is very important in computer systems revolving around human interaction. Applications of intent detection are present in chat-bots, network intrusion, e-commerce, and more. Concretely, we test three of the state of the art intention detection models against a dataset consisting of both phishing and benign emails. By analyzing the text in emails using these models we correctly identify the vast majority of malicious emails; the intention detection models performed as well as the current state of the art in email phishing detection. Through the experiments in this thesis it was also found that not analyzing phishing emails using these models did not degrade the classification quality of the models.

Sammendrag

Nå til dags brukes e-post og SMS daglig av nesten alle i verden. En trussel som følger denne trenden er phishing. Phishing og manipulasjonsangrep er blant de vanligste formene for nettangrep som oppleves nå til dags, siden de ikke er vanskelige for en angriper å utføre. I denne oppgaven angriper vi phishing-problemet ved å bruke modeller samlet fra intensjonsdeteksjonsdomenet. Intensjonsdeteksjon er en undersjanger av naturlig språkbehandling, og er svært viktig i datasystemer som dreier seg om menneskelig interaksjon. Intensjonsdeteksjon er allerede tilstede i chat-roboter, nettverksinntrenging, e-handel og mer. Konkret tester vi tre moderne intensjonsdeteksjonsmodeller mot et datasett som består av både phishing og godartede e-poster. Ved å analysere teksten i e-poster med disse modellene klarer vi å identifisere de aller fleste ondsinnede e-poster; modellene for intensjonsdeteksjon gjorde det like bra som de nåværende toppmodellene innen phishing-deteksjon av e-poster. Gjennom eksperimentene i denne oppgaven ble det også oppdaget at det å ikke analysere nettlenker i phishing e-poster ikke forværret klassifikasjonskvaliteten til modellene.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

In this day and age, people frequently communicate using Internet based services - whether it be for professional or personal uses. Although this is often described as a positive direction for society, as it allows for easier communication, it has opened up a door for social engineering which was previously limited to phone calls and SMS's. Social engineering related crimes such as online phishing has soared in popularity in the past years, to the point where even 'call-centers' have been established to systematically lure trusting people to give up assets. This thesis explores methods to detect such social engineering attacks, and attempts to apply existing techniques from another field to enhance the already existing methods.

1.1 Topic covered by the project

This thesis will delve into the technology behind understanding the intentions that lay within human language, or more precisely, natural language. The majority of methodologies explored within the realm of natural language processing (NLP) are computational in nature, hence this master's thesis will focus on the computational methods employed in intent detection and apply these onto the phishing problem area in an appropriate manner. Intent detection is used in a multitude of applications, as it gives a machine a more thorough understanding of human language. This increased understanding is what advances areas of applicability of NLP, such as personal assistants which can, with time, be used for increasingly more tasks. As intent detection require proper semantic understanding of natural language, simple machine learning methods will not be evaluated in this thesis. Instead, the focus of this master's thesis will be on deep learning methods using mainly neural networks and transformers.

1.2 Keywords

Natural Language Processing, Digital Forensics, Cybersecurity, Phishing attacks, Machine Learning, Deep Learning

1.3 Problem description

As it stands now, a lot of emails with malicious intent are not picked up by automatic spam filters on the way to the recipient. Services such as Outlook does its job partly by flagging emails that originate outside the organization of the recipient. However, this is far from enough to eradicate the phishing problem. A whole bunch of malicious emails still pass through regularly employed filters, and even though they may be marked as 'External,' they are often still opened and seen by the recipient. By just opening such an email, we, as employees or private people, have made the first stage of the attack a success. Then, if the email is enticing enough, the recipient might reply to continue the conversation, or they might click a link taking them to a fabricated website of someone else's intellectual property.

To combat malicious emails, the commonly used filters employ a variety of methods. These methods include analysis of links in the email body, as well as scanning files for malicious hashes. Often times this is enough to detect attacks and prevent the recipient from ever seeing the mail. However, there are bound to be false negatives from such filters - mails marked as safe when they indeed are not. Therefore, it is important to shift the attention of the phishing detection task more towards the implicit meaning behind the text written in email. Of course the URLs' and files' validity and origin should always be questioned, but if phishing can be detected from just the intent behind the email body, it will be possible to more accurately detect targeted emails in attacks such as spear-phishing.

1.4 Justification, motivation and benefits

To fight the issue of having malicious mail floating around in one's inbox it is important to have a rigorous real-time system in place which does not let as many malicious mails as possible through. The product of this thesis will, if positive results are achieved, advance the field of phishing detection and effectively merge the intent detection and phishing detection problem together. This means that the advancement in one of the field most likely will also be an advancement in its counterpart.

This thesis will also advance the community's understanding of which methods might be best suited for the stakeholder's purpose, because the thesis will, in addition to traditional metrics used to evaluate performance of classifiers, use program runtime as a relevant metric. Stakeholders which will benefit from this thesis are any corporation who deal with email server hosting and/or email

filtering services, as state of the art methods will be evaluated on their efficiency time-wise. Evaluating the method as thus will help potential stakeholders in making a decision on which methods are the best, which might reduce cost used on filtering-related hardware.

But, the most important reason for even doing this thesis is simple; further improve upon guarding the public from malicious third-parties. Everyone who use messaging-services or email will benefit from the product of this thesis if the service they use chose to employ some change in their filtering based on this thesis' product.

1.5 Research questions

In order to solve the problem posed, we need to find out:

Q1. What is the state of the art in phishing detection of emails?
Q2. What is the state of the art in intent detection?
Q3. Can phishing emails be detected reliably without examining URL's or file attachments?
Q4. Will applying intent detection methodologies to the email phishing detection problem increase accuracy, precision and recall compared to the state of the art found in Q1?
Q5. Does applying intent detection methodologies to the email phishing detection problem decrease classification time compared to the state of the art in phishing detection?

1.6 Planned contributions

The main contributions of this thesis will be testing and comparing performance metrics of phishing detection methods which have not been performed before. The main goal and hypothesis for this thesis is that formulating the phishing detection problem into an intention detection problem and then applying the state of the art intent detection methods onto it will increase the performance metrics used to evaluate successful phishing detection, including program run-time. Also, the thesis will conclude on whether analyzing URLs in phishing emails provide benefits for classification results, or whether they can be excluded.

As a result of this thesis, intent detection methodologies will be examined and the most relevant framework for intent detection of social media posts will be used in the METICOS project as a step in user feedback analysis of automated border control systems.

Chapter 2

Related Works

There is a lot of work done related to classification of written phishing and social engineering attempts. A simple search using Google Scholar lands several tens of thousands of hits for 'phishing detection', and it is a hard task to prioritize which ones are the best. This chapter will prepare the reader with knowledge of the existing phishing and social engineering detection methods, establish what phishing is, and contents which are needed to understand the works in this thesis. In addition, the topic of intent detection and natural language processing (NLP) will also be introduced, as these are main components of the thesis.

The first part will introduce the reader to the problem of intent detection. Then, the next section will consist of a rundown of what the most popular computational methods used today to uncover intent in texts. Thereafter, the next part will introduce the reader to the topic of phishing and social engineering and its prevalence in the world. Following this introduction, the next section will elaborate on both the history of phishing detection and the state of the art methods used today to uncover phishing in written texts. Lastly, the importance of datasets in phishing detection will be discussed alongside a presentation of widely used datasets collected from other articles.

2.1 Intent detection

Intention detection, intention classification and intention mining are all widely used and similar names representing extracting intention out of a string of text. There are of course other uses for the terms, such as intent detection in videos to uncover crimes [1], but in this thesis the focus will be on intent detection in short texts. Furthermore, in this thesis, all of the three terms presented at the start of this paragraph will be referred to and presented as 'intent detection' in order to avoid confusion down the line.

To start off, it is important to establish a level understanding of what an intention itself is when it comes to text written by normal humans. According to the Merriam-Webster dictionary intention is "*a determination to act in a*

certain way" [2], and this is used as the be-all end-all definition when intention or intent is written in this thesis. This vague definition of intention is an umbrella-term of all explicit or underlying meanings of text.

Intent detection is a hot topic within natural language processing (NLP) these days. It is used in social media platforms to detect intents such as harm and hate speech [3, 4], in the commerce industry to help predict what customers want to buy [5–7], in chat-bots to unravel which response is desired from the bot [8–11], and the list goes on. In concise terms, intent detection in the practice of computers determining the purpose of a text segment using a determined methodology. This methodology can of course vary from the selection of features which represent the text, to the computational method employed to ultimately classify the intent of the given text.

There are two main categories of intent [12]: explicit intent and implicit intent. Explicit intent is when a string of text has no ulterior motives or goals, such as 'I am going to eat ice cream in five minutes,' whereas implicit intent is when the intent of a string is not directly conveyed in a single, isolated, string. To use the previous example, the explicit intent is to eat ice cream in five minutes, but there might be an implicit intent hidden beneath that statement, such as 'I am hungry, but not for a proper meal'. This thesis will focus entirely on explicit intent detection. In this thesis we will also only deal with single intent detection where a string of text is only assigned one label, as opposed to multi-intent detection [12] where multiple intents are assigned to a string of text.

2.1.1 Computational methods

Machine learning is a subset of computational methods, in which algorithms, or models, form their decision making based on the data presented to it [13]. Basic machine learning algorithms, such as support-vector machine (SVM), Naive Bayes and Random Forest (RF) are generally not suited for intent detection due to them not having the capability to properly understand the semantics in a given text [12]. This statement is represented well by Akulick and Mahmoud [14] who in 2017 attempted to classify intents in their dataset using n-grams, Parts-of-Speech and SVM. Their findings show that SVM's performance is abysmal, achieving around a 40% accuracy rate. However, in 2019, Larson et. al. [15] also performed intent classification on the CLINIC dataset, but insead of using n-grams as a feature, they instead opted for Bag of Words (BoW). This resulted in around 90% accuracy for in-scope intents, but below 20% accuracy for out-of-scope intents. Nonetheless, SVM fell behind all of the deep-learning methods considerably both in in-scope and out-of-scope intent detection problems.

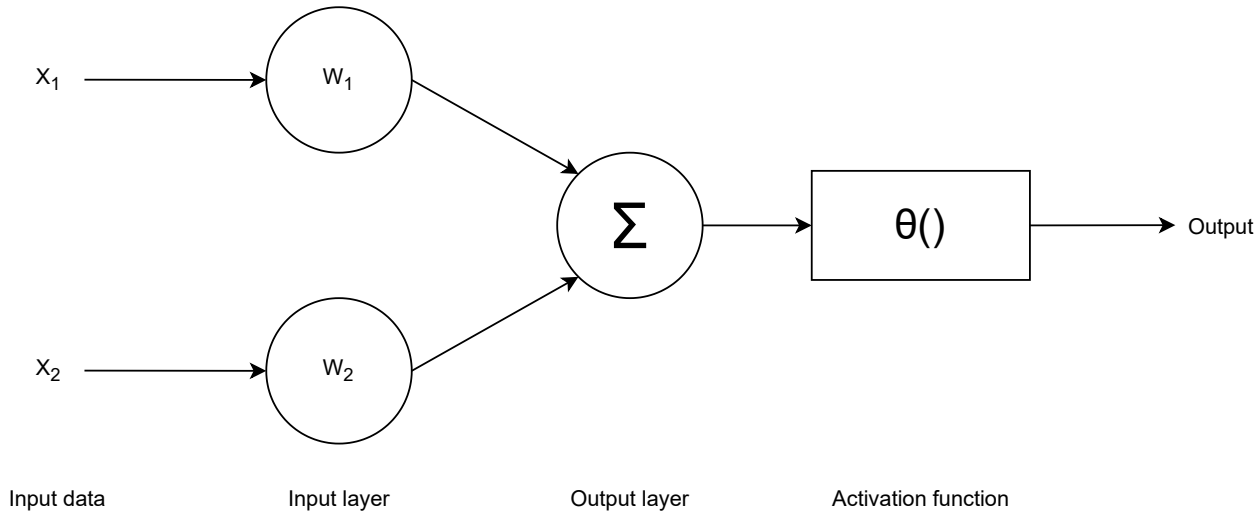


Figure 2.1: Illustration of a simple perceptron

2.1.2 Deep Neural Networks

Deep Neural Networks (DNN) are a subset of Machine Learning methods, which is based on the concept of using neurons in computing [13]. The concept of neurons themselves stem from biology, as our brain contain neurons which are thought to dominate the function of the brain [16]. This section will go through the popular methods of tackling the intent detection problem through three different variations of DNNs.

A simple perceptron is a neural network which consist of a set of input vectors, neurons, a bias, and an activation function [13, 16], as illustrated in figure 2.1. A DNN can be thought of a simple perceptron which consist of many layers of neurons connected to each other by form of weights. In figure 2.1 there is only one layer (the input layer), but in a DNN there are many *hidden* layers between the input layer and output layer.

An example of a technique used to format text into formats which can be input to neural networks is FastText [15]. Larson et. al. [15] achieved an accuracy comparable to SVM using FastText. Balodis et. al. [17] combined FastText's embedding with different types of DNNs across different languages, which resulted in generally good accuracies compared to its counterpart, which implies that FastText is best used alongside other computational methods to achieve optimal results.

Convolutional neural networks

Convolutional neural networks (CNNs) are a variation of deep neural networks which were originally used in image processing [12, 13]. These differ from regular neural networks in the fashion of having convolutional layers, meaning convolution operations are applied on the input data between each connected

convolutional filter in the model [13]. This computational method has been used for multiple-intent detection when used on syntactic features and for feature engineering, but has limitations when it comes to representation of results [12]. Larson et. al. [15] tested a CNN on their CLINIC dataset, and achieved good results compared to the other models tested, where the accuracy was relatively high for both in-scope and out-of-scope intents (where scope refers to whether the model has been trained on the classification material). Wang et. al. [18] proposed to combine CNN with bidirectional gated recurrent unit (BGRU) to achieve better results in intent detection. The results state that the model performed well with different parameters for layers, but the model was only tested on a Chinese dataset, not an English one.

Recurrent neural networks

Recurrent neural networks (RNNs) vary from other neural networks by implementing one or several feedback-connections [13]. Recurrent neural networks have been used in intent detection, but the most basic implementation of an RNN has several problems such as Gradient explosions [12]. Therefore, the field has opted to use a version of RNNs which solves the problems associated with RNNs: Long short-term memory.

Long Short-Term Memory

Long short-term memory (LSTM) improves upon traditional RNNs by including a module for retaining (and forgetting) memory, making it better suited for intent detection [12]. The network is better suited for text classification, because it more accurately is able to figure out which words (or features) are important in a sequence by utilizing the improved memory.

Ting He et. al. [19] tested an LSTM on both the ATIS and SNIPS datasets. LSTM performed among the worst of the methods tested, only besting attention-based RNN in one instance. Wang et. al. [18] tested LSTM against several other methods, where LSTM performed the worst out of the deep-learning methods tested - only performing better than the traditional non-deep machine-learning methods Naive Bayes and SVM. Gennaro et. al. [20] showed, by experimentation, that using more hidden dimensions resulted in better accuracy when using traditional LSTM.

Bi-LSTM

Bidirectional Long Short-Term Memory (Bi-LSTM) differ from the 'traditional' LSTM model by not entirely depending on previous results, but also on future iterations [21]. This has resulted in general performance increase in intent detection [12]. Recently, in 2021, Qin et. al. [22] incorporated Bi-LSTM into their proposed model, utilizing its functionality for producing a slot-aware representation. The results from this methodology indicate better performance metrics

for intent classification on the MixATIS and MixSNIPS datasets compared to other state of the art methods. Miyazaki et. al. [23] combined BERT for embedding, Bi-LSTM, and a feed-forward neural network in order to create a model which outperformed others on the Incident Streams dataset, which is a dataset sourced from Twitter used in a competition in 2018. Their accuracy was better than all of the other contestants, but the precision and recall of the method were remarkably low.

GRU

Liu et. al. [12] stated that Gated Recurrent Unit (GRU) is an improvement of the standard LSTM model. The reason why is due to GRU being able to retain longer pieces of information, as well as the ability to learn semantics of a given text. Wang et. al. [18] discussed the debate of Bi-LSTM versus Bi-GRU, where they mention that a concrete conclusion about which method is better is not reachable, but they show that CNN-Bi-GRU performed the best in their tests compared to CNN-Bi-LSTM. Firdaus et. al. [24] compared Bi-LSTM and Bi-GRU across the ATIS, TRAINS and FRAMES datasets. The result from that article shows Bi-LSTM and Bi-GRU perform very similarly, firmly affirming that concretely stating which is better is a difficult problem.

2.1.3 BERT & Transformers

In 2018, Google released a paper introducing the new state of the art in natural language processing, Bidirectional Encoder Representations from Transformers (BERT) [25]. BERT is a transformer, and ever since its publication it has been widely adopted and tested within the realm of intent detection. It is frequently used in conjunction with other deep learning methods such as CNNs and RNNs in order to achieve better performance than by using them in a stand-alone fashion.

In 2020, Balodis et. al. [17] used BERT's embedding together with a CNN, LSTM and a Bi-LSTM network to judge its performance against FastText embeddings in intent detection problems across several datasets. As expected, BERT performed the best, i.e. had the best accuracy, most of the time.

As mentioned earlier, Miyazaki et. al. [23] also used BERT for embedding the text for use with other deep-learning methods (without fine-tuning it to the dataset used), and achieved the best accuracies of detecting intents compared to the other methods.

In the paper which introduce the CLINIC dataset, Larson et. al. [15] also use BERT for classifying intent in their dataset. BERT performed the best, with a significantly higher intent detection accuracy and f1-score than other methods tested.

He et. al. [26] presented a novel method for intent detection in which a CNN was combined with BERT. The results show that their method achieved significantly higher intent detection accuracy on both an English dataset (ATIS) as

well as on a Chinese dataset (YTBD) compared to the other state of the art.

2.2 Phishing and Social Engineering

Phishing and social engineering attacks are some of the most frequent cyber-attacks nowadays. According to IBM Security's X-Force Threat Intelligence Index published in early 2022, phishing attacks are the most common vector by which attacks are successful by [27]. In addition, phishing is also identified as being the most widespread attack vector in cyberattacks [28]. This is a strong indicator that both organizations and individuals need to put more effort into successfully detecting and preventing phishing and social engineering attacks in order to prevent potentially massive disasters. A survey conducted in 2021 asked participants to rank a list of emails on a range of 'safe' to 'suspicious' [29]. The surprising finding from this survey is that the mean correctness of the participant's choices in the survey only amount to 62%. An accuracy of just above fifty-fifty is definitely not good, meaning there is a good chance many of the participants would have fallen for some of the phishing attempts. One option to decrease successful phishing attacks, which will be the focus of this thesis, is prevention, or blockage, of such phishing attempts before they even reach the target.

In this thesis, henceforth, the term 'phishing' will be used as a collective term for both 'phishing and social engineering', as the two terms are often interchanged. The term 'phishing' will be used in a manner which falls under the definition given by Bowcut [30], in that phishing is a technique by which an attacker entice the victim to perform a harmful action while not having a clue that the action is harmful to either oneself or an organization. Although this definition is vague in defining the contents of the attack, it best sums up the attack method without leaving obscure instances out of the definition. The main feature of the phishing attack is that its target is not computer systems themselves, but instead the users controlling the computer systems. A prime example of such formulation can be observed in figure 2.2, wherein a cybercriminal impersonated the 'Wells Fargo Team' to make the victim send them debit card information using a custom software supplied in the email.

2.2.1 Phishing types

Phishing is not just a straight-forward collection of attack vectors. Different pieces of literature disagree on exactly which categories of phishing there are, but this section aims to summarize which categories exist, and establish which ones are included in the scope of this thesis.

First, there is the regular phishing attack. According to Bowcut [30], this type is what people most commonly think of when hearing the word 'phishing'. The regular phishing attack targets the everyday person, and the attack (often in the form of a fraudulent email) is often mass produced. Mass producing a phishing attacks refer to the attacker using the same template for every single victim without altering the contents of the message. A popular example of such an attack is the 'Nigerian Prince' scam [32], which is an example of an

Figure 2.2: Example of a phishing email. Extracted from the Nazario Phishing Corpus [31].

advance-fee scam where an attacker would pretend to be a wealthy distant relative, but required a 'small sum' of money transferred to their account first before they could give the victim their 'gift'.

Secondly, Bowcut [30] highlights the first category of specialized phishing attacks: whale phishing. In this category, the phishing email is especially crafted for an intended recipient. These mails are not mass-sent out, as this would increase the risk of the mail being caught in a spam-filter or other filters. Since these emails are carefully created to hook the victim, it may take the attacker some time to actually gather intelligence on the victim in order to make the email appear benign. Since the emails are so meticulously crafted, the victims are much more likely to take the bait. Reiterating the most important aspect: spear phishing is a targeted attack.

The third category of phishing attacks is called 'spear phishing'. It shares a lot of similarities with whale phishing, but the main difference is that spear phishing is aimed at a circle of individuals, not a singular individual [30]. For example, a spear phishing attack is made to target the IT administrators of an organization, while a whale phishing attack instead would just target the CIO (Chief Information Officer). Whale phishing often require a lot more effort and research of the individual person and their social circuit than the other forms of phishing.

Lastly, Bowcut [30] lists 'smishing' as the final category of phishing. The only feature separating this category from regular phishing is that smishing happens via SMS, not email. This can perhaps be a more effective method to hook a victim, seeing that there is a significantly higher response rate for SMS's

than emails [33]. However, for the sake of simplicity, in this thesis smishing will be a part of the general phishing category, as the attack vector too similar to phishing done via email.

In this thesis, the focus will be on the former categories which are text-oriented. This includes traditional phishing via Email and SMS services targeting any victim - be they a commoner or a high ranking employee of a large organization.

2.2.2 URL Analysis

One of, if not the, the most popular methods for detecting phishing is to simply analyze URLs or domains. This technique is quite useful, as the methodologies of detecting illegitimate domains are well established and are known to work well. Researchers and companies employ these methods both in scanning emails for malicious content, but also for blocking domains from service.

Barraclough and Woodward [34] combined popular methods, such as blacklisting and heuristic based approaches, for phishing URL detection to create a phishing URL detection model. By combining some of the most popular methods, their model managed to achieve performance-metrics on par with the state of the art in malicious URL detection. The researchers also employed run-time as a metric to prove that the model was efficient, which is not common for researchers to supply.

To detect phishing URLs, Ozcan et. al. [35] employed deep neural networks (DNN), inspired by the works in the field of natural language processing (NLP). Through the experiments performed, the results show that using DNNs closely resembling the state of the art in NLP perform much better in exposing phishing URLs than traditional machine learning methods, as their hybrid DNN-BiLSTM model perform similarly to the best algorithms in the field.

Lansley et. al. [36] combined URL analysis with text analysis in order to improve detection rates of social engineering (phishing) attacks in the new and improved 'SEADer++ v2'. By first analyzing a URL's reputation on a scale from 1-100 and then analyzing text on features such as spelling quality, stop-words, and basic topic modeling the authors achieved decent results in classifying phishing emails and messages. However, this novel method of combining different algorithms fall short of the current state of the art, where the average accuracy, precision and recall are significantly higher. This might be due to the authors using traditional machine learning algorithms for classification instead of using transformers or DNNs, but the authors concluded that fuzzy logic is the best next step to improve the algorithm. In PhiBoost [37], the authors used AdaBoost, adaptive boosting, alongside custom features extracted from the URLs in order to classify a URL as malicious or not. They achieved performance metrics very similar to the state of the art, but the authors suggest testing the classification model in a real-time environment to further evaluate its performance.

While purely analyzing URLs can be sufficient in many cases, Vijayalakshmi et. al. [28] notes that there are methods employed where the websites themselves are scanned (web content analysis) and compared to legitimate sites in order to determine whether the website is used for phishing or not, referred to page similarity evaluation. There is a clear disadvantage in using this approach, as it is much more resource demanding than simply analyzing a URL. However, in cases where the URL analysis method does not suffice, employing web-page similarity evaluation methods might increase the classification accuracy.

Another novel approach to detecting phishing attacks is by analyzing network packets themselves. Rendall et. al. [38] proposed a model which analyzes DNS packets using supervised machine learning. Although novel, the approach did not achieve accuracies nearing the state of the art, topping off at 84% using SVM for classification.

2.2.3 Text Analysis

A more relevant methodology is the approach of text analysis. These methods try to mainly analyze the text in the phishing scams instead of just the link, as just analyzing a URL or attachments can cause false positives especially in communication between two close people.

Linguistics Inspired Methods

Some researchers have tested if linguistic analysis of emails is a viable method for detecting phishing emails. One example is Valecha et. al. [39] who in early 2022 published a paper in which they attempt to use *persuasion cues* to accurately identify phishing attempts. The essence in their methodology is to identify whether the sender is attempting to convince the receiver into performing an action, such as transfer money or give up a password. Using Word2Vector, a neural network model, alongside five machine learning classifiers they achieved high accuracies. However, the accuracies presented fall short of the state of the art by several percentage points.

Natural Language Processing Techniques

Verma et. al [40] proposed a model for combining text analysis, link analysis, and email header analysis in 2012 named PhishNet-NLP. For the text analysis part of the model, parts of speech tagging, stop word removal and character normalization algorithms were used. By combining the scores from these separate parts, the model achieved close to state of the art results, with above 97% accuracy when tested on the Nazario phishing corpus [31].

Smadi et. al. [41] created a highly complex model for phishing email detection using a dynamically evolving deep neural network using a reinforcement learning approach. Their model, which consists of dynamic feature evaluation

and extraction, an ever changing deep neural network, and a reinforcement learning agent, managed to completely outperform the state of the art when the paper was published in 2017. However, this model did not purely analyze text. In fact, the features selected (or rather shown) in the paper mostly revolve around other data found in the email, such as HTML scripts, headers and URLs. Only 9 of the 50 proposed features were extracted from the text in the email body.

Sonowal et. al. [33] performed smishing detection using traditional machine learning techniques. By using mostly text related features such as spelling, parts of speech, and character count. Employing the Random Forest classifier, they managed to achieve 98.66% accuracy with an F1-score of 94.72, which was at the time a better performance than the state of the art in smishing.

Combining deep neural networks with traditional machine learning methods have been successfully shown to be a reliable method for phishing email detection. Bountakas et. al. [42] used the state of the art in feature extraction such as Word2Vec and BERT for the purpose of extracting features from the emails. Then, by employing popular traditional machine learning algorithms such as decision trees for classification of emails, they were able to achieve performance metrics close the state of the art. However, the false positive and false negative rates fell short of what is expected of the state of the art.

Jonker et. al. [43] recently published an article closely related to this thesis, in that their goal was to transfer NLP methods onto the problem of phishing detection. By using deep-learning methods such as LSTM, other deep neural networks, and BERT they achieved substantially good results. However, although they achieved good results, they note that since they got better results than anticipated using standard models they did not implement proper state-of-the-art modifications to those models like is done in this thesis.

Semantic analysis of text is a prominent subcategory of Natural Language Processing, and is closely related to intent detection. Peng et. al. [44] combined semantic analysis approaches with URL analysis in order to produce a model used for phishing detection of emails which performed very well compared to only using NetCraft - a link analysis tool used to detect phishing emails. Due to the good performance of the model, the researchers concluded that semantic analysis techniques are highly proficient in detecting social engineering. By performing a semantic analysis of text, Khan et. al. [45] were able to detect phishing emails fairly accurately. Their algorithm compared verb-object pairs found in sentences with a blacklist of such pairs, alongside finding if the verb-object pair was a question or an order. By then using traditional machine learning techniques, they crafted a decision tree from the blacklist, and detection of malicious mails were done based on this. Although the results do not come close to the accuracies from transformers and deep neural networks, it was a novel methodology in the phishing domain.

2.2.4 Datasets

Table 2.1 shows a listing of different datasets which have been utilized in the literature reviewed. It is important to note their existence and contents, as some of these will be selected for use in the experiments in this thesis. The most common data by far is plaintext emails, not organized any further, meaning content extraction is necessary to use them. A common theme for the reviewed literature is that most use a balanced dataset, i.e. there are a similar amount of phishing emails as there are benign emails. Since this is definitely not representative of a real scenario, one must keep in mind that the results might be skewed.

Dataset	Label examples	Description
Enron [46]	{Subject, Body}	The Enron dataset is the largest dataset comprised of emails written by real employees of Enron. It also contains the employees' inbox, some emails of which contain content from personal use outside of the organization. Emails from this dataset is often used as the 'benign' mails in phishing email detection, as the emails are the most popular and well-known to be an accurate representation of real emails written by the average Joe or Joanne.
HoneyTrap [47]	{Subject, Body}	MillerSmiles.co.uk is a website dedicated to archiving phishing and email spoofing scams, and carry a database of over 2.6 million entries long full of phishing scams dating as far back as to 2004. Accessing the archive, however, is not free. The owners charge 180 British pounds per month for a commercial license to use it. On the website itself there is an 'browse archive' feature, but the entries do not showcase the full bodies of the emails.
UCI SMS Spam [48]	{spam, ham}	The UCI SMS Spam Collection consists of a plethora of both benign (ham) and spam SMS messages. Several studies have used this dataset for phishing detection, as most spam email and SMS are either in the category of phishing or advertisement. Of course, the advertisement SMS's have been removed from the dataset when used in phishing classification.
Nazario [31]	{To, Body}	The Nazario Phishing Corpus is a collection of phishing emails collected from the period of 2015 to 2021, last updated in February in 2022. It is freely available online, but the data is not cleaned, and only exist in the form of plaintext and mbox files.
Nigerian Scams [49]	{Message-Id, To}	The Fradulent E-mail Corpus is a collection of 'Nigerian scam' emails dating from 1998 to 2007 - they are old. Hence, this dataset represents techniques used in the early dates of the Internet. It is available on Kaggle, but the data itself only exist in the form of a single text file.

Table 2.1: Datasets used for phishing detection in the literature.

Chapter 3

Choice of Methods

This section aims to inform the reader of the methods used in this paper to answer the research problem. It will firstly go over the different criteria for selecting computational methods which will be evaluated in this thesis, and thereafter the criteria for dataset selection will be presented. Lastly, the methods by which the computational algorithms will be evaluated is explained.

3.1 Computational methods

This thesis' aim is to apply the state-of-the-art (SOTA) in the field of intent detection on to the field of phishing detection in order to determine whether Natural Language Processing is sufficient for protecting against phishing attacks. In order to achieve this, the SOTA in both intent detection and phishing detection have been identified in section 2 and understood. This section will present the computational methods which have been selected for use in the experiments. The main criterion for any computational methodology to be considered for evaluation is that it fits into the definition of SOTA for intent detection and is not currently a part of the state of the art in phishing detection. The implementation of these methods will be described in detail in section 4.

3.1.1 Bidirectional Long Short-Term Memory (Bi-LSTM)

The first deep neural network chosen for classification was Bidirectional Long Short-Term Memory (Bi-LSTM), which is a specific implementation of a Recurrent Neural Network (RNN). LSTM aims to reduce the problem of exploding and vanishing gradient problems of a bare-bones RNN [50]. Exploding and vanishing gradients are a problem in which the earliest inputs into the model influence the network less the longer the total input string is. Essentially, this means that RNNs have short-term memory, making the model less useful for longer inputs. The following is an explanation of the LSTM model using formulae from Sak et. al. [50] and Olah [51]:

Firstly, the LSTM cell calculates what should be forgotten from the previous cell by performing the sigmoid activation function (σ) on the information from the previous hidden state (h_{t-1}) and the current input (x_t), which returns an integer between 0 and 1 for each input. The closer the value to 1 means to keep the information, while the closer the output value to 0 means the opposite. Note that b denotes bias vectors, and W represent weights between gates in the network.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.1}$$

Next is the first step of the input gate. A similar action to the first equation is performed, although it uses different weights and a different bias for the calculation. This ensures a different result in the computation, hence why the output of the first equation cannot be used for both the forget and input gate. The output of this equation dictates which values will be changed.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.2}$$

The second step of the input gate produce a vector C_t^a containing new values which will affect the updating of the old cell state by performing the tanh activation function with similar parameters that the aforementioned sigmoid functions got, with its own weights and bias.

$$C_t^a = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.3}$$

Then, the old cell state is updated by first multiplying the output of the forget gate with the old cell state, which decide which values are kept or forgotten. This result is then added upon by the result of the input gate, producing the new cell state representing this particular cell C_t.

$$C_t = f_t \times C_{t-1} + i_t \times C_t^a \tag{3.4}$$

Again, to decide which cell state values will be output o_t a sigmoid activation function is performed is performed on the previous hidden state and the current input.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.5}$$

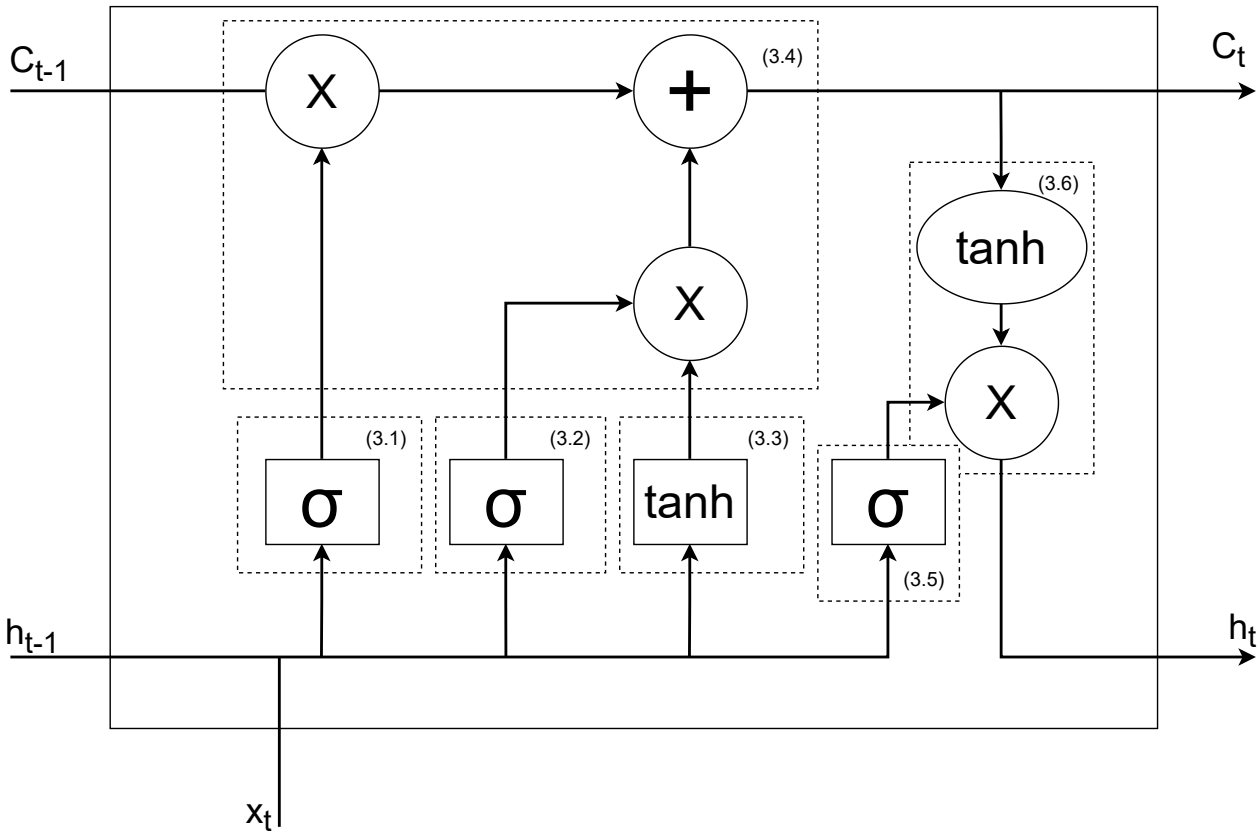


Figure 3.1: Simple visualization of an LSTM cell.

Lastly, the model produces the new hidden state h_t by first performing the tanh activation function on the current cell state to get values in the range of $+-1$, and then multiplying the result with the output achieved in the previous formula.

$$h_t = o_t \times \tanh(C_t) \tag{3.6}$$

The LSTM model is further illustrated in figure 3.1, in which the different equations are annotated where they are performed using dotted boxes.

The bidirectionality of the Bi-LSTM model simply describe that the process described above is also performed in the other direction on the input, i.e. starting the processing from the end of the input string. We consider the bidirectional model for this thesis instead of the standard model due to its general performance being better in NLP tasks, as found in section 2.

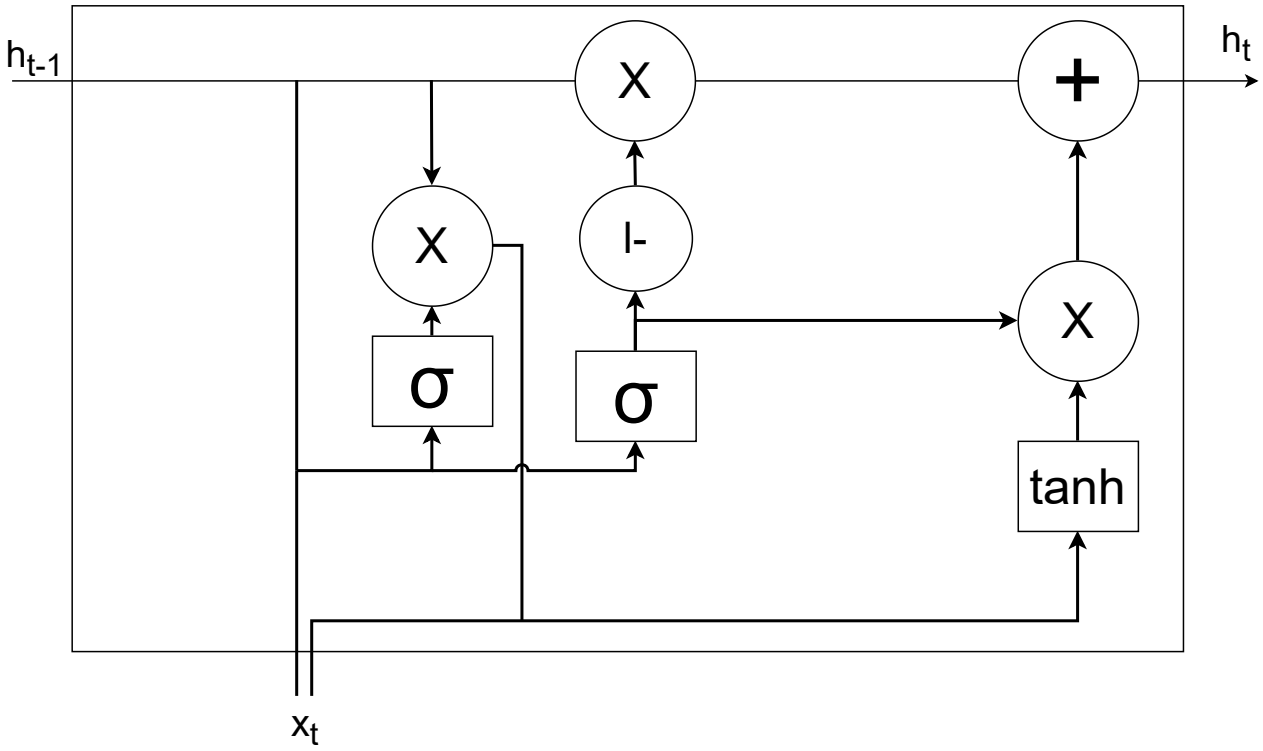


Figure 3.2: Simple illustration of a GRU cell.

3.1.2 Gated Recurrent Unit (GRU)

A Gated Recurrent Unit (GRU) is a variation of the LSTM model which combines the forget and input calculations into one unit [51]. This model was first proposed by Cho et. al. [52], and also merges cell states and hidden states into one singular entity. Due to these combinations, the model is less complex, requiring less tensor operations to function, making it more light-weight. Being light-weight compared to LSTM means that running the model will either (1) require less computational resources, (2) complete program execution quicker than its LSTM counterpart or (3) both. The removal of one activation function (square box) which makes the algorithm quicker can be seen in the illustration in figure 3.2.

3.1.3 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a specialized deep feed-forward neural network which was primarily used for image classification since its conception [13]. Since the model was designed for processing imagery, most implementations support 2-dimensional input data and also the application of

2-dimensional convolution operations. However, many implementations also support 1-dimensional and 3-dimensional convolutions [53]. Using the implementation which allows for one-dimensional input data is beneficial for users which want to use CNNs for classifying text data, as the network theoretically can receive an input of a raw string or a tokenized string.

Figure 3.3 depicts a simple CNN with just one convolutional layer. Firstly, the input data of length n is inserted into the model. This input data can be manually extracted features from a sequence of words (sentence or paragraph), word embeddings from that same sequence or tokenized words from the input data. Then, the convolutional layer processes the input using a set of convolution filters, whose job it is to select important features. These convolutional filters contain both biases and weights which are modified using the process of back-propagation, similar to a standard feed-forward neural network [13]. After the convolution filters have been applied, the selected features from all of the inputs in the input vector are combined into a single vector once again, which is then fed into a classifier. A commonly used algorithm for classification is softmax [54]. When softmax has finished the classification we have the final output from the CNN for input x.

3.1.4 BERT

Bidirectional Encoder Representations from Transformers, or BERT for short, have been used in some of the best performing models in the literature reviewed back in section 2. As the name implies, the model consists of multiple transformers, which themselves are deep neural networks [25]. BERTs implementation of the transformers architecture is heavily derived from the model proposed in [55], and use multiple such transformers stacked together in order to achieve state-of-the-art performance [25]. Specifically, in the version 'BERT-base-uncased' which is the most commonly used, BERT is implemented with 12 transformer blocks (layers), a hidden size of 768, and 12 self-attention heads [25].

There are two main selling points to BERT: pre-training and bidirectional self-attention. Firstly, as the model's self attention mechanism is bidirectional it is able to better understand the meaning of a string of text. This is because, as explained in the other bidirectional models, the model is able to understand the context of words or tokens in the sentence as a whole instead of understanding its meaning contextual only to its position from the start of the input. Secondly, the pre-training of the model shorten training times of derivative models, as only fine-tuning of parameters are required [56]. Turc et. al. [56] provided several other sized models of BERT on GitHub in 2019, hoping the smaller pre-trained models can be used in environments where resources are scarce. Since the BERT authors recommend using BERT for feature extraction [56], the model will also be tested with a standard feed-forward neural network similar to the simple perceptron depicted in figure 2.1

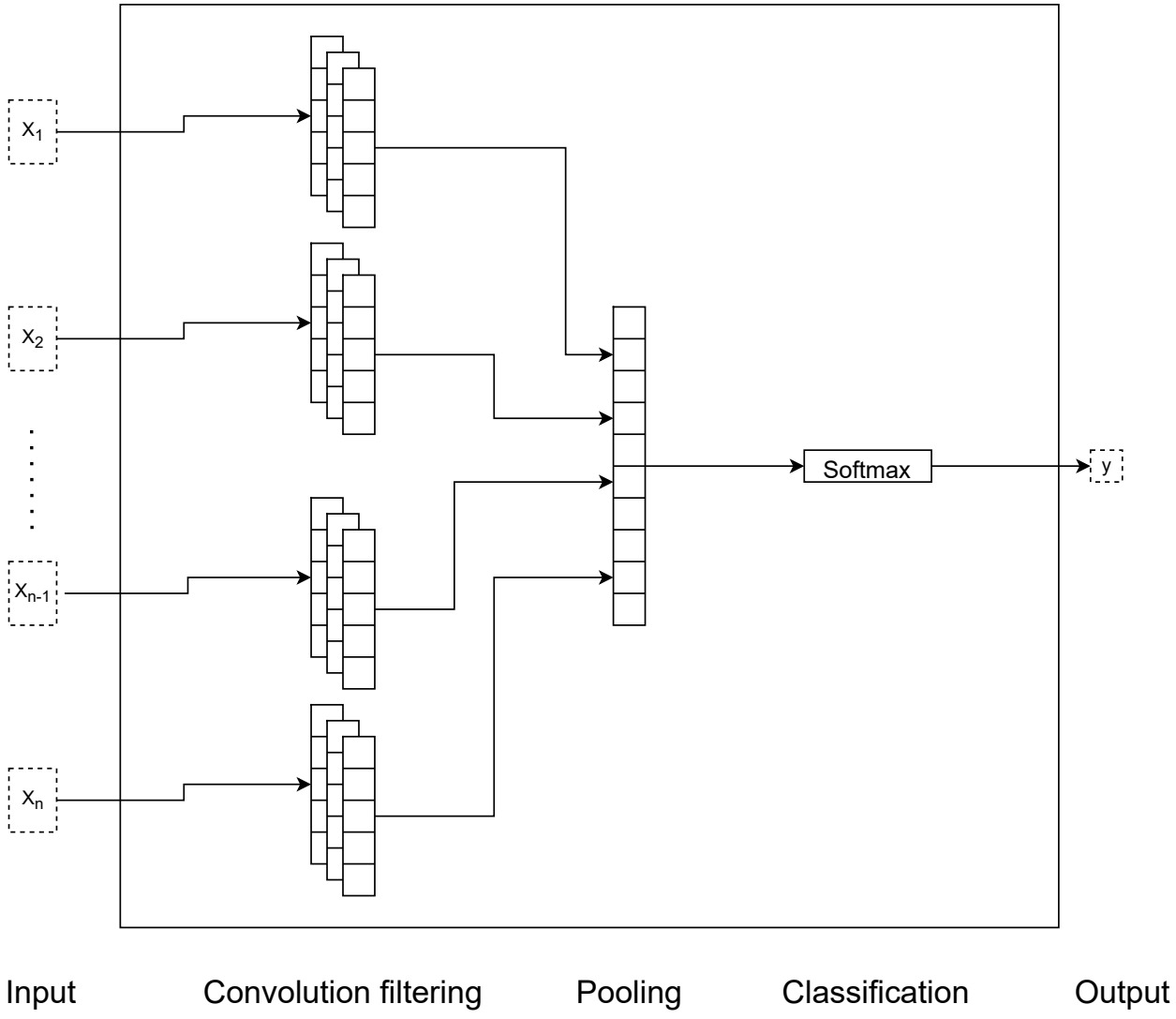


Figure 3.3: Illustration of a CNN with one layer.

Tokenization and Embedding

Tokenization refers to the process of interpreting strings as numbers. This method is often used in deep learning tasks, as the implemented models often cannot handle raw strings. BERT can be used for both character/word embeddings which are fed into another classifier, or it can be used as a classifier itself. In this thesis, BERT has been selected for testing in experiments as a word embedder for other neural networks, namely Bi-LSTM, Bi-GRU, CNN and FFNN. The reasoning behind this decision is derived from section 2, where BERT was identified as the best performing word tokenizer tool for use in conjunction with other deep neural network architectures.

3.2 Dataset selection

In this thesis it is important to select the correct datasets on which to estimate the performance of the selected computational methodologies. The datasets to be used have been selected from table 2.1, which present datasets used in the literature reviewed.

From the table identifying the existing datasets in section 2.2.4 the phishing dataset from Nazario [31] was chosen to be used in the experiments due to its popularity in the community of phishing detection [43]. Other phishing datasets such as MillerSmiles was not chosen due to it not being readily available without a paywall, or simply due to lack of unique examples.

The ever popular Enron dataset was chosen to represent the benign portion of emails in the final dataset, as it is one of the few datasets recognized for containing real emails collected by the CALO project [46]. To ensure the emails used are indeed benign, only emails sent by Enron employees will be considered as benign, while the rest will be discarded.

- Nazario Phishing Corpus
- The Enron dataset

The chosen datasets will in the end be concatenated into one large dataset, with each entry having a label identifying which dataset it belongs to. Keeping the origin of every entry in the dataset allows for showing statistics in the end related to the datasets themselves, and is expected to result in a decent discussion about the qualities of each dataset and models. The concatenated dataset will then also be duplicated, and all URLs and emails will be removed from all message bodies in the duplicate dataset in order to be able to answer research question $Q3$..

3.3 Performance evaluation

The most used, and most useful, metrics for evaluating performance in machine learning algorithms are accuracy, precision, recall, and the f1-score (also

referred to as f-measure) [16]. These scores are popular due to the information they convey. They are based on the concept of negatives and positives in a binary classification scenario, and their 'false' variants. A true positive (TP) is when a classification algorithm has classified an unclassified item correctly. A false positive (FP), however, is when a computational method predicts a class as positive, but its real label was negative. The same logic applies to true negative (TN), where the classifier predicts negative correctly, and to false negatives (FN) where the classifier labels data as negative, when it was indeed positive. Below, these terms are used in order to define the metrics mentioned at the beginning of this paragraph:

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

Figure 3.4: Formula for accuracy

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

Figure 3.5: Formula for recall

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

Figure 3.6: Formula for precision

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Figure 3.7: Formula for F1-score

In addition, classification time will also be considered a metric in order to facilitate answering research question $Q5$. After every computational method has been evaluated using these metrics, for every metric, for every computational method on every dataset, the mean performance metric will be presented as the resulted performance of the model. The result will be a table (or multiple) which contain every computational method evaluated, with the mean of their performance metrics, i.e. two columns for each of the performance metrics presented above for each dataset. This is the main method by which we will attempt to answer research question $Q3.$, $Q4.$, and $Q5.$, as this experiment approach is estimated to contain enough information about each computational method to draw proper conclusions. In formal terms, this will be quantitative

research project. At the same time, a literature review in the form of Related Works (section 2) will also be present, in which $Q1$. will be answered, as well as parts of the rest of the questions which cannot be answered by this experiment.

Chapter 4

Experiments

This chapter will walk the reader through the specifics of the implementation of the experiments performed. The overall description of the methodology of the experiments can be found in chapter 3, while this chapter covers specific steps and parameters not walked through in the previous chapter. First, the chapter outline and experimental design is illustrated and explained. Thereafter, the chapter goes through the preprocessing steps performed on the datasets used in the experiments, including the merging of the two total datasets which each represent a class each. Then, there is a section dedicated to providing information on how the tokenization and embedding was performed on the dataset. Lastly, the specific details of the implemented intent detection methods will be explored and explained.

4.1 Experimental design

This section is meant to provide the reader with an overview of the design of the experiments performed in an attempt to answer the research questions posed in section 1. Figure 4.1 depicts the design of the experiments using arrows depicting flow in the process. As with any machine learning and deep learning task, the process can be divided into three phases: dataset preparation, feature extraction and data classification. The premise of the design is to merge two datasets, preprocess them so that they fit into the feature extraction model, and then perform classification on those extracted features using methods found in the state-of-the-art in intent detection.

The Nazario [31] and Enron [46] datasets serve as the base of the experiments performed in this thesis. Enron represents actual emails written by Enron employees, as opposed to any other artificial email dataset. Nazario, however, represents phishing emails, and is composed of emails that Jose Nazario collected from their personal inbox ever since 2004. The Enron dataset is extracted from a Kaggle project [57] as it comes in a neat .CSV file, while the Nazario dataset is directly collected from their website [31].

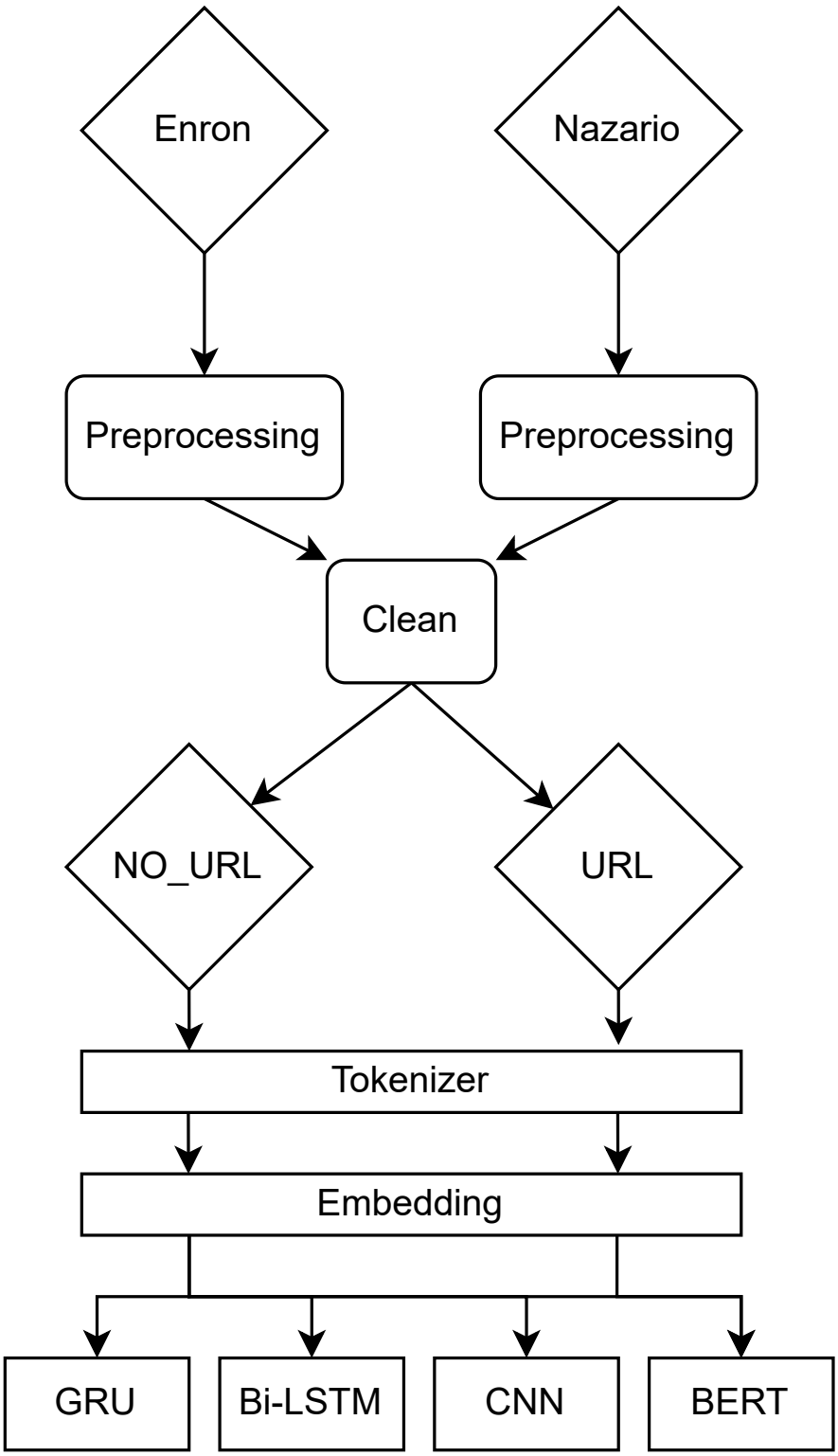


Figure 4.1: Experimental design.

After first fetching the two datasets from their respective sources, preprocessing is performed on each of them. The purpose of preprocessing the datasets is to only output relevant data from the original dataset for further cleaning down the pipeline. Since the results of the classification will depend on the information removed from this step in the process, it is important to be careful to only process the data in a way which preserve the data we want from the emails - words. Dataset preprocessing was performed in parallel on both dataset, with 12 steps used on the Enron dataset and 11 steps for the Nazario dataset.

When both datasets have been preprocessed by themselves they are then merged together to form a 'complete' dataset. However, this combined dataset also needs some additional cleaning as it will be split into two separate datasets wherein one has all the URLs in the email's bodies replaced by the string *'URL'* (including quotation marks), and the other dataset has all URLs outright removed from all email bodies (these datasets are named 'URL' and 'NO_URL' respectively in figure 4.1).

Then, for each dataset we split them into a training and testing subset. In these experiments the datasets are split at a 80/20 ratio for the training and testing subsets respectively. The following steps are performed on both datasets' training and test subsets, but are written as performed on just one dataset for simplicity's sake: First, both of the subsets are passed through a tokenizer each. This ensures that the data is transformed into a format which the embedder in the next step can interpret, including padding entries to ensure same length for each email. Second, the tokenized version of the subsets are passed into the BERT transformer. The output from which will serve as the features from the emails. For each token extracted from the emails the embedder creates 768 features for. That means e.g. for a subset with 100 samples which have a length of 50 there will be a feature vector of size [100, 50, 768], effectively meaning 38'400 features for every email.

Finally, the subset's features are fed into the four models used for classification. Here, the metrics on time for training the model, final classification time, accuracy, recall, and precision are extracted. These are the results, and are presented in chapter 5.

4.2 Experimental setup

The planned experiments require high computational power in order to run in a timely manner. Training each model for each dataset is what takes the most time, not the actual classification, due to the large scale of the models and the heaps of data which need to be processed. All experiments were performed on a relatively powerful virtual machine provided by NTNU, the most important specifications of which can be seen in table 4.1.

Note that the GPU in table 4.1 is shared across multiple users in the cloud, and therefore only 8GB VRAM was allocated to this machine as opposed to the

Ubuntu LTS 20.04 Virtual Machine	
CPU	Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz, 8-cores
Memory	88.4GB
GPU	Nvidia Tesla A100 8GB VRAM

Table 4.1: Specifications of the cloud testing platform.

total 24GB available on the graphics card. Also, there are only eight CPU-cores present, as this was also a limitation posed by the available resources at the time they were requested. Due to the limited VRAM on the GPU, the size of the models trained are also limited. The deep neural networks implemented in this thesis are derived from existing implementations in PyTorch, documentation for which can be found in [58]. All of the code used in this thesis is Python code. For timing code runtime '`time.perf_counter()`' is used, as it is the most accurate for timing code implemented in the Python 'time' library [59].

4.3 Datasets

As mentioned in section 3, we will use two datasets for experimentation on different intent detection methodologies. First, the Enron corpus, which was chosen to represent benign data due to its credibility and widely adopted use, is preprocessed in a manner which results in only text being left behind for the computational methods to process. Then, the same is done to the Nazario corpus, which was chosen as the de-facto phishing corpus due to its wide-spread use in the phishing detection problem. Afterwards, both datasets were combined into a singular dataset before being processed further with tokenization and embedding algorithms.

4.3.1 Enron Corpus

Firstly, before any processing can commence, the dataset is downloaded from Kaggle [57]. It is the 2015 version of the dataset, and is the dataset which is recommended for use in research and other projects [46]. Enron dataset comes in a comma-separated values (.csv) file. This file separates the folder in which the emails were located in from the entire email. The info contained in the email-column is all the metadata from the email, including, but not limited to, the sender, recipient, content-type, and the actual email body. Since we only desire the email body, one of the first preprocessing steps we do, although not listed in table 4.2, is extract the email body from the other data deemed useless in the experiments. Table 4.2 enumerates the preprocessing work, and this section is dedicated to explaining every step in the process.

Starting off with over half a million email samples in the corpus, we first remove all emails which were not sent by an Enron employee. This is to ensure that no spam or phishing emails which might have been present inside

SN#	Preprocessing Action	Emails
1	Starting corpus size	517'401
2	Remove all emails which was not located in a folder containing the string 'sent'.	126'846
3	Remove emails which are forwarded or redirected, indicated by the subject of the email containing one of the strings {'fw:', 'fwd:', 're:'}. Also remove emails which contain inline attachments or any of the strings {'to:', 'cc:', 'subject:'}.	33'061
4	Remove emails which contain the string '_____'.	31'109
5	Remove emails which were sent by 'no.address@enron'.	31'096
6	Remove duplicate emails.	31'096
7	Remove parts of emails which are not written by the sender. The email body is split on one of the strings in {'— Original Message', '— Original Message', '—Forwarded by', '— Forwarded by'}.	31'096
8	Remove all emails starting with '>', twice.	31'083
9	Remove emails which contain more than 2'000 characters in the email body.	30'395
10	Replace character codes: • '=01' with '' (apostrophe) • '=20\n' with ' ' • '=20\r\n' with ' ' • '\n' with ' ' • '\r\n' with ' ' • '=09' with ' ' • '\t' with ' ' • ' ' with ' '	30'395
11	Remove emails containing less than 21 words.	21'067
12	Remove duplicate emails.	16'442

Table 4.2: Preprocessing steps performed on the Enron corpus before exporting as .csv file.

employee's inboxes. Furthermore, this step also attempts at ensuring that no duplicate emails are present in the corpus.

Then, the emails whose subject or bodies contain the strings 'fw:', 'fwd:', and 're:' were removed from the dataset. Removing emails which contain these strings removes the vast majority of emails containing text not written by the sender, making it better represent a real email, as we attempt to detect phishing emails when they first are sent.

Preprocessing step 4 delete all string which contain seven dashes in a row in the email body. This is because when the experiments were performed and the dataset manually explored, several emails which contained this string had an inline attachment, possibly not written by the author of the email.

Step 5 is removing emails sent by the address 'no.address@enron.com'. By performing this step we ensure that the entire corpus represent mails in the personal 'sent' mailboxes of Enron employees, making the dataset more representative of benign emails.

In step 6 we simply remove all emails which are duplicate. This step does not produce any results, but it is important to ensure that all instances in the dataset are unique to prevent over-representation of one message.

Then, in step 7, all emails are checked if they contain pre-determined strings which indicate that the message was forwarded and contain the message of some other sender. If these strings are detected in an email, that string and everything following that string is removed from the email body, resulting in an email body which only consist of the sender's written text.

The eighth step removed all instances of the character '>'. Since some emails which are forwarded might have the original message's content represented with that character, so removing emails which start with that character further ensures that the dataset only contain text from the respective senders.

Step 9 removes all emails which contain 2'000 or more characters. This step removes some outliers in the dataset which could make the subsequent tokenization, embedding and classification take a lot more resources to successfully process.

In the tenth step, all character codes commonly found in the emails with an appropriate representative. Since formatting of the email body is not important in this thesis, all escaped characters and character codes which represent different formatting were replaced with a single space character (' ').

The second to last step removed emails which contained 20 words or less, making sure that emails containing just a few words (or in some cases just a letter) are deleted. Lastly, but most importantly, all emails whose bodies were equal were removed and replaced with the first instance. This is the final step, and prevents over-representation of single email messages. The remainder of the dataset were written to a CSV file for future use.

4.3.2 Nazario Corpus

The Nazario dataset was retrieved from Jose Nazario's own website [31], wherein one can find several files containing phishing emails from the years 2015-2021.

For the experiments in this thesis, the file 'private-phishing4.mbox' is chosen to represent the phishing emails in the final dataset. However, as these emails come in the format of '.mbox', they first have to be extracted. Before extracting the dataset, the bash command '`!iconv -f utf-8 -t utf-8 -c private-phishing4.mbox > pp4`' is applied on the file, as the email parser was not able to decode some of the content. This command ensures that the file is readable with a UTF-8 typeset. Unlike with the format of the Enron dataset, all emails in the mbox file are not separated and need to be separated. Extracting the emails from the mbox file is the first step presented in table 4.3.

The second step of preprocessing the Nazario corpus is one of the most important steps. It replaces all the HTML link tags with the actual link they are pointing to. This is the main method by which URLs are extracted, as image sources are not included in the links extracted. However, if the HTML link tag does not point anywhere the tag is simply replaced by an empty string.

Since a lot of the phishing emails contain HTML and CSS code, the Python library 'BeautifulSoup' is used to strip away all of such code from all emails' bodies in the dataset. Doing this ensures that most of the code in the phishing emails are removed.

Fourth, all escape characters which were found are removed from the email bodies. This is again to ensure that the email bodies contain mostly scammer-written messages.

In the fifth step all emails which contain the '@'-sign are removed from the corpus. All emails could be interpreted and removed from email bodies to ensure no emails are in the corpus, but since a lot of emails remain after this step, the emails containing '@' were simply removed.

Sixth, all emails sent from the address 'service@paypal.com' were removed. These emails were identified to be spam-emails instead of phishing emails, and were therefore not wanted in the final corpus.

A problem caused by converting the entire mbox file to UTF-8 is that some emails were not properly interpreted by the parser. Such mails always contained byte-codes which all started with '0x'. Therefore, all emails whose bodies contain such code were removed from the dataset in step seven.

In step 8, similarly to the Enron dataset, all emails which contained less than 21 words were removed. This was done for the same purpose as with the Enron dataset - to ensure a proper representation of emails. This also removes outliers whose bodies might be empty after other preprocessing steps.

Step 9 removes all emails whose bodies contain 'doctype html'. As this string is indicative of there being HTML code in the email which was not properly removed in step 3, all emails containing this string were removed.

Then, as with the Enron corpus, all emails which contain more than 2'000 characters were removed from the dataset. A find here is that there are a lot more emails which contain more than 2'000 characters than in the Enron dataset.

Lastly, phishing emails ending with '=' were removed from the corpus.

SN#	Preprocessing Action	Emails
1	Extract mails from private-phishing4.mbox	3'534
2	Replace HTML <a>-tags with the href's inside them (if the href field is not empty).	3'534
3	Remove HTML tags and CSS code from the email's body.	3'534
4	Replace escape characters: • '=01' with '" (apostrophe) • '=20\n' with ' ' (space) • '=20\r\n' with ' ' • '\n' with ' ' • '\r\n' with ' ' • '=09' with ' ' • '\t' with ' ' • ' ' with ' ' • '\xa0' with ' ' • '=A9' with ' ' • '=C3' with ' '	3'534
5	Remove all messages containing at least one instance of the character '@'.	3'259
6	Remove all emails from PayPal ('service@paypal.com').	3'226
7	Remove all emails containing byte codes (' 0x').	3'099
8	Remove all emails which contain less than 21 words.	2'967
9	Remove all emails containing 'doctype html'.	2'963
10	Remove all emails whose body contains more than 2'000 characters.	2'565
11	Remove all emails whose body ends in '='.	2'435

Table 4.3: Preprocessing steps performed on the Nazario corpus before exporting as .csv file.

Upon manual inspection of the dataset, emails ending with an equals-sign were broken by either the preprocessing, parsing, or they were broken upon arrival to the mbox. Therefore all such emails were deleted from the dataset. The remainder of the dataset were written to a CSV file for future use.

4.3.3 Merging and Splitting Datasets

After preprocessing the two datasets, both datasets had to be merged in order to perform the final cleaning. The cleaning steps performed on the remaining datasets have been listed in table 4.4. The purpose of this section is to explain the preprocessing steps described in the table.

SN#	Preprocessing Action	Emails
1	Remove all email bodies with more than 120 words.	—
2	Extract 300 emails from each corpus	600
3	Duplicate dataset	600
4	Replace all URL instances with either a blank string or 'URL'.	600
5	Replace all characters which are not used for punctuation or in the alphabet with ' '.	600
6	Remove all stopwords from all emails.	600
7	Remove all emails with less than 21 words in the body.	474

Table 4.4: Preprocessing steps performed on the combination of Enron and Nazario.

In step 1 all email bodies containing more than 120 words were removed. The main driver for this decision was that, during experimentation the feature dimensions became very large and therefore it became a lot more demanding to process the models. Therefore we eliminate emails with a larger email body.

Step 2 also has the same reasoning as the preceding step. Here, 300 samples from each of the Nazario and Enron datasets are extracted and combined into a single dataset. During the experimental phase it was observed that the GPU on which the computations were run ran out of memory when running the classifiers on larger sample sizes.

Thereafter, this single copy was duplicated. These two datasets are the premise for the two versions of the same dataset, one in which all URLs are replaced with an empty string. The other version of the dataset will have all its URLs replaced by 'URL'. Replacing those URLs are done in step 4.

Step 5 then replaces all characters which are not in a predefined range with a single space. To do this a regular expression was used. The regular expression removes all characters from the emails' bodies which are not in the alphabet, an apostrophe, or a punctuation character. Doing this helps reducing feature dimensionality substantially.

Then, in step 6 all stopwords are removed from the text. Stopwords are words commonly found in text, and are not necessary in most text classification scenarios as they do not add any value. The stopwords used in this thesis was

gotten from the NLTK Python library using the code: '`from nltk.corpus import stopwords`'. The NLTK stopword corpus consist of 179 words, some of which are compound, and some of the stopwords included in the list are '["I", "a", "is"]'. Removing such stopwords also contribute to reducing the dimensionality of the features later on.

Lastly, all emails whose bodies contain less than 21 words are removed from the corpus. Although this is an arbitrary line to draw, it represent emails which contain more than a simple request or response. As the dataset cleaning is finished each of the two dataset are left with 290 emails representing the phishing emails and 184 emails representing the benign emails, for a total of 474 emails.

4.4 Tokenization and Embedding

For tokenization and embedding of the preprocessed and cleaned datasets we use 'bert-base-uncased', as described in section 3.1.4. To use the model we simply import it in Python which downloads the model automatically, as seen in listing 4.1. The data is first ran through the BERT Tokenizer, which we are using the Fast version of. This produces tokens for every word in every email. Since the BERT model requires all data to be of the same length when embedding, the tokenizer is equipped with '`padding=True`' to pad all emails which contain less tokens than the rest.

Code listing 4.1: Text tokenization and embedding implementation

```
from transformers import BertTokenizerFast, BertModel
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained("bert-base-uncased")

# Create Tokens and Embeddings for every word/token in a set of data
def TokenEmbed(data):
    t1 = time.perf_counter()
    with torch.no_grad():
        encoded_input = tokenizer(data.tolist(), return_tensors='pt',
                                  padding=True, truncation=True)
        output = model(**encoded_input)
    t2 = time.perf_counter()
    return encoded_input, output, t2-t1  # Tokens, mail_embedding, runtime
```

4.5 Implementation

This section will go over the specifics of the implemented models and showcase the essential part of the code which represent the model design. All code listings depict Python code, as all experiments were run using Python 3 using Jupyter Notebook and Visual Studio Code. The parameters of the models were standardized as much as possible. Therefore, all models ran for 15 epochs dur-

ing testing. All of the models also used the 'BCEWithLogitsLoss()' function for calculating loss in every epoch.

4.5.1 BiLSTM

The bidirectional LSTM used for experiments can be seen in listing 4.2. Here, the LSTM network consist of a single bidirectional LSTM layer fully connected to a linear layer, which in turn produce the output. The input size of the LSTM is equal to the number of features produced for each word embedding (768). The size of the hidden layer in the LSTM was set to 256, as this was comfortable to run on the graphics card. The learning rate of the model was set to 0.005.

Code listing 4.2: BiLSTM implementation

```
# LSTM Class
class LSTM1(nn.Module):
    def __init__(self, num_classes, input_size, hidden_size, num_layers, bi, drop):
        super(LSTM1, self).__init__()
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                        num_layers=num_layers, batch_first=True, bidirectional=bi,
                        dropout=drop)
        self.fc_1 =  nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()
```

4.5.2 BiGRU

The implementation of the bidirectional GRU network in listing 4.3 is based on the implementation of the BiLSTM model, hence why the code is very similar. The only difference from the description provided in the BiLSTM section, is that this network use the 'torch.nn.GRU' layer instead of a LSTM layer. Other than that the parameters and design are the same, with 768 input features and 256 features in the hidden layer of the GRU. The learning rate of the model was set to 0.005.

Code listing 4.3: BiGRU implementation

```
# GRU Class
class LSTM1(nn.Module):
    def __init__(self, num_classes, input_size, hidden_size, num_layers, bi, drop):
        super(LSTM1, self).__init__()
        self.lstm = nn.GRU(input_size=input_size, hidden_size=hidden_size,
                           num_layers=num_layers, batch_first=True, bidirectional=bi,
                           dropout=drop)
        self.fc_1 =  nn.Linear(hidden_size, num_classes) # fully connected 1
        self.relu = nn.ReLU()
```

4.5.3 CNN

The convolutional neural network was implemented much in the same manner as the two previous models using 'torch.nn.Conv1d', as seen in listing 4.4. However, as the model is not related to recurrent neural networks, it has different input parameters. The input dimension remain the same at 768, as the embeddings never change. But, the model is equipped with three different instances of the aforementioned module, with filter sizes of 3, 5 and 7 respectively with 100 filters per instance. The learning rate of this model was set to 0.001.

Code listing 4.4: CNN implementation

```
# Convolutional Neural Network with varying filter sizes
class CNN(nn.Module):
    def __init__(self, embedding_dim, n_filters, filter_sizes, output_dim,
                 dropout_rate):
        super().__init__()
        self.convs = nn.ModuleList([nn.Conv1d(embedding_dim,
                                              n_filters,
                                              filter_size)
                                    for filter_size in filter_sizes])
        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)
        self.dropout = nn.Dropout(dropout_rate)
```

4.5.4 BERT using FFNN

Lastly, the FFNN classifier using BERT is implemented using a simple dense layer (torch.nn.Linear) which takes 768 features and outputs just one neuron containing one array of integers, representing the predicted class for each input email. The implementation of the classifier can be observed in listing 4.5. Since this classifier is much less 'smart', it was equipped with a learning rate of 0.1 instead of 0.001 or 0.005, as it had trouble with learning fast in the predetermined epoch space of 15.

Code listing 4.5: BERT-FFNN implementation

```
# Feed-forward neural network for classification with BERT
class BERTClass(nn.Module):
    def __init__(self, embedding_dim, output_dim, dropout):
        super(BERTClass, self).__init__()
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(embedding_dim, output_dim)
```

Chapter 5

Results

This chapter presents and comments on all of the results from the experiments performed. The experimental design which lead to these results are described in chapter 4. Firstly, all of the classification results are presented on a per-algorithm basis using confusion matrices for both datasets the algorithms were trained and tested on. Afterwards, all of the results are pooled together in a table to make comparisons between algorithms easier. This table contains all of the relevant metrics for measuring performance, which were outlined in chapter 3.

5.1 Confusion Matrices per Algorithm

Before diving into the results it is important to understand that the confusion matrices do not show the count of objects classified as one or the other, but rather the percentage of the entire dataset used that was classified in every cell. Also, rows indicate the actual classification labels for the data, while the columns represent the classifiers prediction. Lastly, the captions of each figure represent which algorithm used on which dataset, where 'NO_URL' and 'URL' refer to the two almost-identical datasets created following the steps in chapter 4.

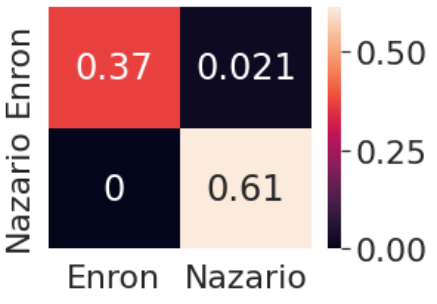


Figure 5.1: BiLSTM on NO_URL

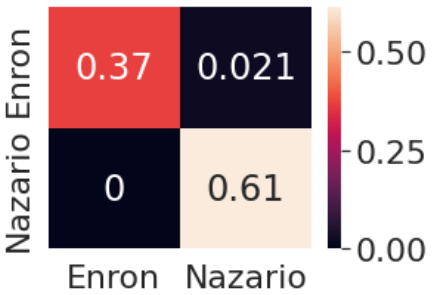


Figure 5.2: BiLSTM on URL

Starting off with the results achieved from BiLSTM, we observe that the performance is identical on both versions of the dataset. 37% of all emails were correctly classified as benign, whereas 2.1% of the corpus were incorrectly classified as phishing emails, meaning for both datasets there was 2.1% false positives. All phishing emails were classified correctly, with 61% of all the emails in the corpus being classified as phishing emails (Nazario).

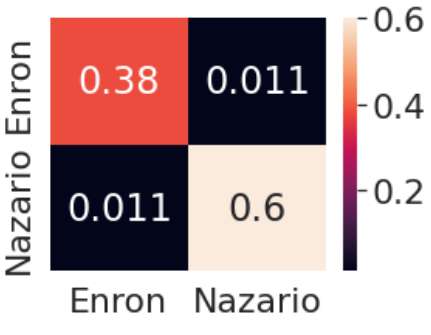


Figure 5.3: BiGRU on NO_URL

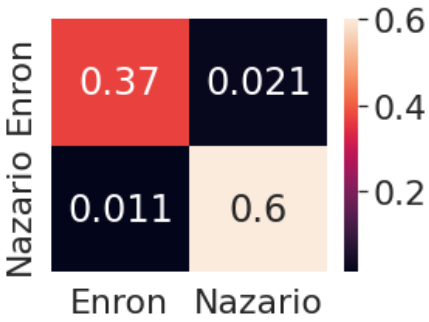


Figure 5.4: BiGRU on URL

Moving on to the BiGRU model, we actually observe some difference between the classifications on the datasets. In the 'NO_URL' dataset 38% of all emails were correctly classified as benign, with 1.1% of all emails being incorrectly classified as phishing emails. Adding to this, 1.1% of all emails were incorrectly classified as benign (false negatives) while 60% of all emails were correctly classified as phishing emails. On the other hand, with the 'URL' dataset we observe a change in the classification of benign emails, where the correct classification of the entire corpus is 37%, while 2.1% of all emails were incorrectly classified as phishing emails. However, the phishing emails were classified with the same accuracy as with the 'NO_URL' dataset.

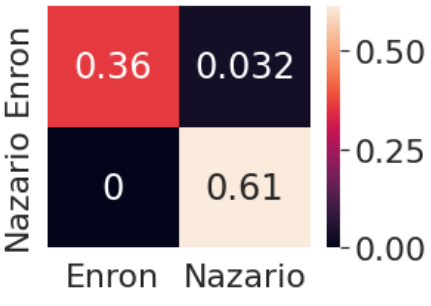


Figure 5.5: CNN on NO_URL

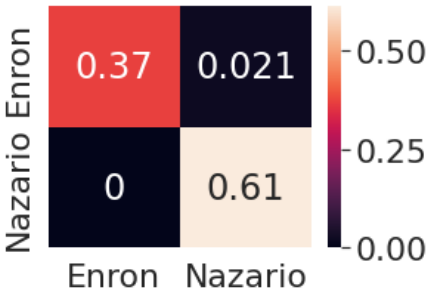


Figure 5.6: CNN on URL

The CNN model was able to achieve good results as well. For both datasets the model managed to correctly identify all the phishing emails from the Nazario corpus as phishing emails, but there are some false positives among the

benign emails. On the 'NO_URL' dataset, the model classified 3.2% of benign emails incorrectly as phishing emails, compared to on the 'URL' dataset where only 2.1% of benign emails were incorrectly classified.

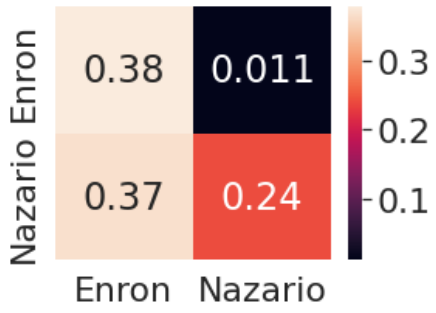


Figure 5.7: BERT-FFNN on NO_URL

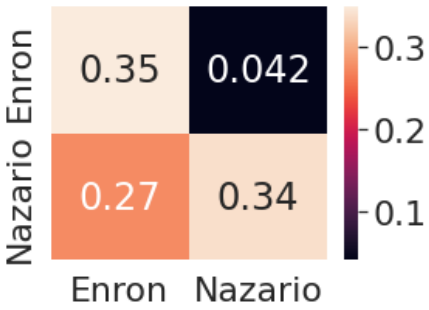


Figure 5.8: BERT-FFNN

Lastly, the BERT-FFNN model achieved interesting results. We observe that for the 'NO_URL' dataset, 38% of all emails were correctly classified as benign, with 1.1% of the corpus being incorrectly classified as phishing. However, the model classified 37% of the dataset incorrectly as benign, meaning 37% of classification results were false negatives. For the 'URL' the results are similar, but an increase in false negatives with 4.2% of the dataset being falsely identified as phishing emails. There is also a decrease, compared to the results achieved on the 'NO_URL' dataset, in false negatives, in which 'only' 27% of the emails were incorrectly identified as benign.

5.2 Classification Results

The summary of all the confusion matrices above can be found in table 5.1. In addition to the results found in the confusion matrices, this table also supply the calculated accuracy, precision, recall, and f1-score for each of the models on both datasets. A brand new feature addition are the 'training time' and 'test time'. These metrics were supplied in addition to the aforementioned ones, as they are required for answering research question $Q5$. The title of the most accurate method is tied between BiLSTM, BiGRU and CNN, on both datasets, as all of them achieved 97.9% accuracy. The same models also compete for the highest f1-score, as all three also ends up with at least one instance of an f-1 score of 0.989. In classification time, however, we observe that there is a clear difference in the tested methods. The method with both the highest training time and test time is BiLSTM, while the fastest algorithm implemented was BERT-FFNN by a large margin. All of these results will be discussed up against their relevant research questions in chapter 5.

Method	**Dataset**	**Accuracy**	**Precision**	**Recall**	**f1**	**Training Time**	**Test Time**
BiLSTM	URL	0.979	1	0.979	0.989	1.668s	0.013s
BiGRU	URL	0.968	0.989	0.979	0.984	1.261s	0.010s
CNN	URL	0.979	1	0.979	0.989	1.026s	0.006s
BERT-FFNN	URL	0.684	0.714	0.942	0.812	0.034s	0.001s
BiLSTM	NO_URL	0.979	1	0.979	0.989	1.653s	0.010s
BiGRU	NO_URL	0.979	0.989	0.989	0.989	1.273s	0.010s
CNN	NO_URL	0.968	1	0.968	0.984	0.995s	0.007s
BERT-FFNN	NO_URL	0.621	0.628	0.983	0.766	0.033s	0.002s

Table 5.1: Final results of all experiments.

Chapter 6

Discussion

This chapter will discuss the results found in chapter 5. Firstly, the goals of the thesis will be explained once again. Then, the main points of discussion will be comparing the results of the different models for every dataset, and hypothesizing why some models performed better than the others. These comparisons also include the two time metrics, as that is where the most variance has been observed across the different models. Lastly, some caveats and possible pitfalls with the experiments performed in this thesis will be discussed.

We planned to utilize the state of the art in the intent detection domain on the phishing email detection problem to test whether it would improve classification metrics compared to the state of the art in the phishing detection domain. To do this we first surveyed the state of the art in both the intent detection and phishing detection domains in chapter 2. The state of the art in the intent detection domain were then tested in the experiments section on the phishing detection problem. We also intended to quantify whether phishing emails could be reliably detected without examining file attachments and URLs, hence why two separate datasets were used in the experiments. The last research question posed is directed at whether the intent detection methods could be faster than the methods used in the current phishing detection state of the art, which resulted in us recording the metrics for classification training and testing runtime.

The performance of the intent detection models implemented in the experiments section perform comparably well to the already existing state of the art in phishing detection. For example, Jonker et. al. [43] applied similar methods to the phishing detection problem, using both LSTM and CNN. But, instead of using BERT for embeddings they used Word2Vec, which is also a popular embedding algorithm. With the LSTM and CNN models they achieved accuracies of 98.5% and 98.4% respectively, which is almost a percent higher than the best results documented in our experiments. In addition they tested BERT purely for classification, which achieved an accuracy of 98.9%. However, the highest f1-score achieved in their paper was 98.94 for BERT, while our tested models also achieved an f1-score of 98.9 consistently. Even though their results have

higher accuracies, the f1-scores for their results and the results in this thesis are the same. This can be explained by the discrepancy in dataset size, since Jonker et. al. used a dataset size of over 20 times that of which is used in this thesis. Having such a small dataset increases the sensitivity in the accuracy metric for every misclassification, hence why the accuracy decreases very much with only one false positive or false negative. While Jonker et. al. achieved state of the art results classifying phishing emails on text, Barraclough et. al. [34] performed phishing detection on phishing URLs using standard feature extraction methods and machine learning methods such as Naive Bayes and J48. Using those methods they achieved accuracies of 99%-99.3% . Pure URL classification is expected to have higher accuracies, as unknown domains can be ruled out mostly using white- and blacklisting techniques in addition to machine learning.

From the results, however, we do observe that whether an URL is present or not in the email body does not play a large role in classification, as seen in the confusion matrices in chapter 5, the results are very similar for every model on both versions of the dataset. This is probably due to the wording used in phishing emails, as the scammers can e.g. be more assertive or begging in the phishing email text. The methods proposed by Valecha et. al. [39] reinforce this finding, as they also achieved good classification accuracies upwards of 95%-95.9% by only analyzing text based behavioral features. Since BERT is excellent at extracting relevant features during the embedding stage, our hypothesis is that the emails, after preprocessing, contain vastly different language when comparing benign to phishing emails, which make classification easier for the models. This is especially true when stopwords are removed, as they represent the most frequent occurring words in the English language.

Whether the models tested in this thesis were slower or faster than other models is hard to quantify, as the exact hardware as used in the literature is used for our experiments. In this thesis we used cuda (GPU) acceleration for classification, as it is a lot faster than classifying using a CPU. However, in literature such as [34, 60, 61], a CPU is used instead for classification. It should be mentioned that those papers use a lot less complex models for classification than the models used in this thesis. Mostly, only models whose purpose is to be used not only for research supply classification times, as resource consumption is an important factor when deploying such models in a production environment. The models in [34] only take 0.006 seconds to classify their dataset, while the models in [60] take 0.4-23.9 seconds for classification depending on the model complexity. Comparatively, the state of the art models in our thesis use 0.013-0.006 seconds for classification, which is almost equal to the best classification times in [34].

The results also show that using a less complex neural network for classification is not suitable for phishing detection. the BERT-FFNN model consistently performed worse the other models, and the state of the art, with accuracies ranging from 62.1% to 68.4%. Such poor accuracies were expected, as the model

is simply a larger version of the simple perceptron illustrated back in figure 2.1. However, it should be noted that due to the lack of model complexity, the BERT-FFNN model performs classifications around ten times faster than the other models. If any of the models used in this thesis were to be considered for use in a production environment, the best choice would most likely be the CNN model. Although the CNN model scored worse than BiGRU and BiLSTM models on the 'NO_URL' dataset, both the CNNs training and testing times were much lower than its competitors.

The two main drawbacks of the models used in this thesis are: i) small imbalanced dataset for both training and classification, and ii) lack of proper result validation. Since the models were trained on small datasests, it is likely that not many different email topics are used for training and testing. This lack of uniqueness in emails might have artificially increased the accuracy of the models, as the benign and phishing emails might use entirely different words and terms. Also, the dataset was imbalanced in the favor of the phishing dataset. We believe that is the reason behind there being a lot more false positives than false negatives when testing, as the model has been fed more phishing emails and thus has a bigger context around phishing emails than benign emails. Due to the limited resources (VRAM) on the GPU we were not able to perform classification on larger datasets, but leave this as future work. As for the lack of result validation, we were originally going to use stratified K-Fold validation as it is the most common validation technique used [16] for machine learning problems, but the implementation kept leaking the GPU's VRAM, causing the models to crash during training due to running out of available VRAM. To mitigate the problem of not using any formal validation methods on the models, the models were run from scratch more than five times each. The median performance of those runs were used and presented in chapter 5. Although this is not a complete remedy, it ensures to a large degree that the possible outliers are not presented as the results.

Nevertheless, the results from the experiments performed in this thesis show that intent detection methods perform equally as well as the pre-existing state of the art in phishing detection. The results also show that these models are not dependent on whether a phishing email contains a URL, making these models well suited for detecting more targeted attacks such as spear- and whale-phishing. Lastly, it is also demonstrated that the classification times of the models are on equal footing with the state of the art in URL phishing detection.

Chapter 7

Conclusion

There were three main goals of this master's thesis related to the phishing detection problem. These goals were to i) identify whether applying models from the intention detection domain onto the problem of phishing detection increased performance metrics compared to the pre-existing state of the art in phishing detection, ii) quantify how much the presence of URLs in emails contributed to the classification performance metrics, iii) and to figure out how the classification time of the applied models were in comparison to existing phishing detection models.

In order to meet these goals, we first selected two datasets to represent the phishing problem. The Enron dataset [46] was used to represent benign emails, while the Nazario dataset [31] was chosen to represent malicious phishing emails. These two datasets went through a thorough preprocessing before being sent to BERT for the tokenization and embedding process. The embeddings from the emails were then fed into four different models (BiLSTM, BiGRU, CNN and BERT-FFNN) which classified the emails as either benign or malicious (phishing).

Through the results achieved in this thesis we have answered all research questions posed in chapter 1. We show that the models inspired by intention detection methods (BiLSTM, BiGRU and CNN) all perform equal to the existing state of the art in the phishing detection domain in terms of accuracy, precision and recall. Adding on to those metrics, we also show that whether an email contains a URL or not does not affect the models' classification performance in any meaningful manner. Lastly, we show that the classification times of the models also are comparable to the state of the art in phishing detection. In conclusion, this thesis has answered the five research questions that were posed and show that models from the intention detection problem area can effectively be applied onto the phishing detection problem.

Chapter 8

Future work

For future work, we propose testing the models used in this thesis using the full dataset from the dataset preprocessing phase. As discussed earlier, the dataset used in this thesis was small and imbalanced. This might have resulted in biased results, so therefore the models should be tested on a balanced version as well - with more email samples from each original corpus. In addition, the models should also be run using the same methodology in this thesis, but with stratified K-Fold validation. This was also discussed before, but something went wrong during the implementation making stratified K-Fold validation near impossible. Although some remedies were put in place, the models should be tested again to ensure as un-biased results as possible from the models. These models could also be tested on entirely different datasets, as there are a lot of phishing emails out there and the Nazario corpus does not guarantee that every possible phishing scam is included.

If a phishing detection model based on natural language processing ideas is to be deployed in the real world, more metrics than classification time would be needed to quantify how much resources the model use. We therefore propose that in future iterations researchers should also include peak memory (RAM or VRAM) usage for the model. Gathering these data will make it easier for potential stakeholders to decide on which models to deploy, as there most likely will be a cost/benefit trade-off. All classification run-times should also be tested on both CPUs and GPUs for comparison.

The intent detection problem, from which the models came, mostly deal with multiple classes. It might be useful for researchers to attempt at classifying emails into different sub-classes such as 'financial fraud' and 'identity theft'. An implementation as such could enhance the field by allowing for more nuanced analysis of emails to prevent specific security breaches. Also, this could help with analyzing email phishing scam trends.

Furthermore, it is rumored that Google is going to release BERTs successor, MUM [62]. The release seems to be just speculation, but if it is released in the near future, it is probably going to become the new state of the art in many natural language processing domains. Therefore, it should also be tested on

the phishing detection problem, as we did with BERT.

Bibliography

[1] U. V. Navalgund and K. Priyadharshini, 'Crime intention detection system using deep learning,' in *2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET)*, IEEE, 2018, pp. 1–6.

[2] Merriam-Webster, *Intention*. Merriam-Webster.com Dictionary. [Online]. Available: https://www.merriam-webster.com/dictionary/intention (visited on 23/03/2022).

[3] H. Purohit and R. Pandey, 'Intent mining for the good, bad, and ugly use of social web: Concepts, methods, and challenges,' in *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*, N. Agarwal, N. Dokoohaki and S. Tokdemir, Eds. Cham: Springer International Publishing, 2019, pp. 3–18. DOI: 10.1007/978-3-319-94105-9_1. [Online]. Available: https://doi.org/10.1007/978-3-319-94105-9_1.

[4] Y. Senarath and H. Purohit, 'Evaluating semantic feature representations to efficiently detect hate intent on social media,' in *2020 IEEE 14th International Conference on Semantic Computing (ICSC)*, IEEE, 2020, pp. 199–202.

[5] M. Koniew, 'Classification of the user's intent detection in ecommerce systems-survey and recommendations.,' *International Journal of Information Engineering & Electronic Business*, vol. 12, no. 6, 2020.

[6] L. Li, L. Sun, C. Weng, C. Huo and W. Ren, 'Spending money wisely: Online electronic coupon allocation based on real-time user intent detection,' in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2597–2604.

[7] X. W. Zhao, Y. Guo, Y. He, H. Jiang, Y. Wu and X. Li, 'We know what you want to buy: A demographic-based system for product recommendation on microblogs,' in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14, New York, New York, USA: Association for Computing Machinery, 2014, pp. 1935–1944, ISBN: 9781450329569. DOI: 10.1145/2623330.2623351. [Online]. Available: https://doi.org/10.1145/2623330.2623351.

[8] A. Arora, A. Shrivastava, M. Mohit, L. S.-M. Lecanda and A. Aly, 'Cross-lingual transfer learning for intent detection of covid-19 utterances,' *Facebook*, 2020.

[9] C. Abbet, M. M'hamdi, A. Giannakopoulos, R. West, A. Hossmann, M. Baeriswyl and C. Musat, 'Churn intent detection in multilingual chatbot conversations and social media,' *arXiv preprint arXiv:1808.08432*, 2018.

[10] A. Nigam, P. Sahare and K. Pandya, 'Intent detection and slots prompt in a closed-domain chatbot,' in *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, IEEE, 2019, pp. 340–343.

[11] B. Gamage, R. Pushpananda and R. Weerasinghe, 'The impact of using pre-trained word embeddings in sinhala chatbots,' in *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, IEEE, 2020, pp. 161–165.

[12] J. Liu, Y. Li and M. Lin, 'Review of intent detection methods in the human-machine dialogue system,' in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1267, 2019, p. 012 059.

[13] N. Anantrasirichai and D. Bull, 'Artificial intelligence in the creative industries: A review,' *Artificial Intelligence Review*, pp. 589–656, 2021.

[14] S. Akulick, E. S. Mahmoud *et al.*, 'Intent detection through text mining and analysis,' in *Proceedings of the Future Technologies Conference (FTC), Vancouver, Canada*, 2017, pp. 29–30.

[15] S. Larson, A. Mahendran, J. J. Peper, C. Clarke, A. Lee, P. Hill, J. K. Kummerfeld, K. Leach, M. A. Laurenzano, L. Tang and J. Mars, 'An evaluation dataset for intent classification and out-of-scope prediction,' in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. [Online]. Available: https://www.aclweb.org/anthology/D19-1131.

[16] I. Kononenko, *Machine learning and data mining : introduction to principles and algorithms*. Chichester: Horwood, 2007, ISBN: 9781904275213.

[17] K. Balodis, J. Kapočiūtė-Dzikienė and R. Skadiņš, 'Intent detection problem solving via automatic dnn hyperparameter optimization,' *Applied Sciences*, vol. 10, no. 21, 2020, ISSN: 2076-3417. DOI: 10.3390/app10217426. [Online]. Available: https://www.mdpi.com/2076-3417/10/21/7426.

[18] Y. Wang, J. Huang, T. He and X. Tu, 'Dialogue intent classification with character-cnn-bgru networks,' *Multimedia Tools and Applications*, vol. 79, no. 7, pp. 4553–4572, 2020.

[19] T. He, X. Xu, Y. Wu, H. Wang and J. Chen, 'Multitask learning with knowledge base for joint intent detection and slot filling,' *Applied Sciences*, vol. 11, no. 11, 2021, ISSN: 2076-3417. DOI: 10.3390/app11114887. [Online]. Available: https://www.mdpi.com/2076-3417/11/11/4887.

[20] G. Di Gennaro, A. Buonanno, A. Di Girolamo, A. Ospedale and F. A. N. Palmieri, 'Intent classification in question-answering using lstm-architectures,' in *Progresses in Artificial Intelligence and Neural Systems*, A. Esposito, M. Faundez-Zanuy, F. C. Morabito and E. Pasero, Eds. Singapore: Springer Singapore, 2021, pp. 115–124. DOI: 10.1007/978-981-15-5093-5_11. [Online]. Available: https://doi.org/10.1007/978-981-15-5093-5_11.

[21] W. Ullah, A. Ullah, I. U. Haq, K. Muhammad, M. Sajjad and S. W. Baik, 'Cnn features with bi-directional lstm for real-time anomaly detection in surveillance networks,' *Multimedia tools and applications*, vol. 80, no. 11, pp. 16 979–16 995, 2021, ISSN: 1380-7501.

[22] L. Qin, F. Wei, T. Xie, X. Xu, W. Che and T. Liu, 'Gl-gin: Fast and accurate non-autoregressive model for joint multiple intent detection and slot filling,' *arXiv preprint arXiv:2106.01925*, 2021.

[23] T. Miyazaki, K. Makino, Y. Takei, H. Okamoto and J. Goto, 'Label embedding using hierarchical structure of labels for Twitter classification,' in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 6317–6322. DOI: 10.18653/v1/D19-1660. [Online]. Available: https://aclanthology.org/D19-1660.

[24] M. Firdaus, H. Golchha, A. Ekbal and P. Bhattacharyya, 'A deep multitask model for dialogue act classification, intent detection and slot filling,' *Cognitive Computation*, vol. 13, no. 3, pp. 626–645, 2021.

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

[26] C. He, S. Chen, S. Huang, J. Zhang and X. Song, 'Using convolutional neural network with bert for intent determination,' in *2019 International Conference on Asian Language Processing (IALP)*, 2019, pp. 65–70. DOI: 10.1109/IALP48816.2019.9037668.

[27] IBM, 'X-force threat intelligence index 2022,' IBM Corporation, New Orchard Road, Armonk, NY 10504, Tech. Rep., Feb. 2022. [Online]. Available: https://www.ibm.com/downloads/cas/ADLMYLAZ (visited on 25/04/2022).

[28] M. Vijayalakshmi, S. Mercy Shalinie, M. H. Yang and R. M. U, 'Web phishing detection techniques: A survey on the state-of-the-art, taxonomy and future directions,' *Iet Networks*, vol. 9, no. 5, pp. 235–246, 2020.

[29] Z. M. Hakim, N. C. Ebner, D. S. Oliveira, S. J. Getz, B. E. Levin, T. Lin, K. Lloyd, V. T. Lai, M. D. Grilli and R. C. Wilson, 'The phishing email suspicion test (pest) a lab-based task for evaluating the cognitive mechanisms of phishing detection,' *Behavior research methods*, vol. 53, no. 3, pp. 1342–1352, 2021.

[30] S. Bowcut, *Phishing attacks: A complete guide*, `https://cybersecurityguide.org/resources/phishing/`, Accessed: 2022-04-26, 11th Feb. 2022.

[31] J. Nazario, *Phishing email dataset*, `https://monkey.org/~jose/phishing/`, Accessed: 2022-05-02, 12th Feb. 2022.

[32] AARP, *Nigerian scams*, `https://www.aarp.org/money/scams-fraud/info-2019/nigerian.html`, Accessed: 2022-04-26, 28th Oct. 2020.

[33] G. Sonowal, 'Detecting phishing sms based on multiple correlation algorithms,' *SN Computer Science*, vol. 1, no. 6, pp. 1–9, 2020.

[34] P. Barraclough, G. Fehringer and J. Woodward, 'Intelligent cyber-phishing detection for online,' *Computers & Security*, vol. 104, p. 102 123, 2021, ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2020.102123`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167404820303965`.

[35] A. Ozcan, C. Catal, E. Donmez and B. Senturk, 'A hybrid dnn–lstm model for detecting phishing urls,' *Neural Computing and Applications*, pp. 1–17, 2021.

[36] M. Lansley, S. Kapetanakis and N. Polatidis, 'Seader++ v2: Detecting social engineering attacks using natural language processing and machine learning,' in *2020 International Conference on Innovations in Intelligent Systems and Applications (INISTA)*, IEEE, 2020, pp. 1–6.

[37] A. Odeh, I. Keshta and E. Abdelfattah, 'Phiboost-anovel phishing detection model using adaptive boosting approach,' *Jordanian Journal of Computers and Information Technology (JJCIT)*, vol. 7, no. 01, 2021.

[38] K. Rendall, A. Nisioti and A. Mylonas, 'Towards a multi-layered phishing detection,' *Sensors*, vol. 20, no. 16, p. 4540, 2020.

[39] R. Valecha, P. Mandaokar and H. R. Rao, 'Phishing email detection using persuasion cues,' *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 747–756, 2022. DOI: `10.1109/TDSC.2021.3118931`.

[40] R. Verma, N. Shashidhar and N. Hossain, 'Detecting phishing emails the natural language way,' in *European Symposium on Research in Computer Security*, Springer, 2012, pp. 824–841.

[41] S. Smadi, N. Aslam and L. Zhang, 'Detection of online phishing email using dynamic evolving neural network based on reinforcement learning,' *Decision Support Systems*, vol. 107, pp. 88–102, 2018.

[42] P. Bountakas, K. Koutroumpouchos and C. Xenakis, 'A comparison of natural language processing and machine learning methods for phishing email detection,' in *16th International Conference on Availability, Reliability and Security*, ser. ARES 2021: The 16th International Conference on Availability, Reliability and Security, Association for Computing Machinery, pp. 8887–12, ISBN: 1-4503-9051-X.

[43] R. A. A. Jonker, R. Poudel, T. Pedrosa and R. P. Lopes, 'Using natural language processing for phishing detection,' in *Optimization, Learning Algorithms and Applications*, A. I. Pereira, F. P. Fernandes, J. P. Coelho, J. P. Teixeira, M. F. Pacheco, P. Alves and R. P. Lopes, Eds., Cham: Springer International Publishing, 2021, pp. 540–552, ISBN: 978-3-030-91885-9.

[44] T. Peng, I. Harris and Y. Sawa, 'Detecting phishing attacks using natural language processing and machine learning,' in *2018 IEEE 12th international conference on semantic computing (icsc)*, IEEE, 2018, pp. 300–301.

[45] M. T. F. Khan *et al.*, 'Detecting phishing attacks using nlp,' *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 2, pp. 369–372, 2021.

[46] William W. Cohen, *Enron email dataset*, `https://www.cs.cmu.edu/~enron/`, Accessed: 2022-04-26, 8th May 2015.

[47] MillerSmiles, *Phishing scams and spoof emails at millersmiles.co.uk*, `http://www.millersmiles.co.uk/index.php`, Accessed: 2022-04-26, 2022.

[48] T. A. Almeida, J. M. G. Hidalgo and A. Yamakami, 'Contributions to the study of sms spam filtering: New collection and results,' in *Proceedings of the 11th ACM symposium on Document engineering*, 2011, pp. 259–262.

[49] R. Tatman, *Fraudulent e-mail corpus*, `https://www.kaggle.com/datasets/rtatman/fraudulent-email-corpus`, Accessed: 2022-05-02.

[50] H. Sak, A. Senior and F. Beaufays, 'Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition,' *arXiv preprint arXiv:1402.1128*, 2014.

[51] C. Olah, *Understanding lstm networks*, `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, Accessed: 2022-05-18, 27th Aug. 2015.

[52] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, 'Learning phrase representations using rnn encoder-decoder for statistical machine translation,' *arXiv preprint arXiv:1406.1078*, 2014.

[53] Torch Contributors, *Convolution layers*, https://pytorch.org/docs/stable/nn.html#convolution-layers, Accessed: 2022-05-30, 2019.

[54] MathWorks, *What is a convolutional neural network?* https://se.mathworks.com/discovery/convolutional-neural-network-matlab.html#how-they-work, Accessed: 2022-05-30.

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

[56] I. Turc, M.-W. Chang, K. Lee and K. Toutanova, 'Well-read students learn better: On the importance of pre-training compact models,' *arXiv preprint arXiv:1908.08962v2*, 2019.

[57] W. Cukierski, *The enron email dataset*, https://www.kaggle.com/datasets/wcukierski/enron-email-dataset, Accessed: 2022-05-15, 2016.

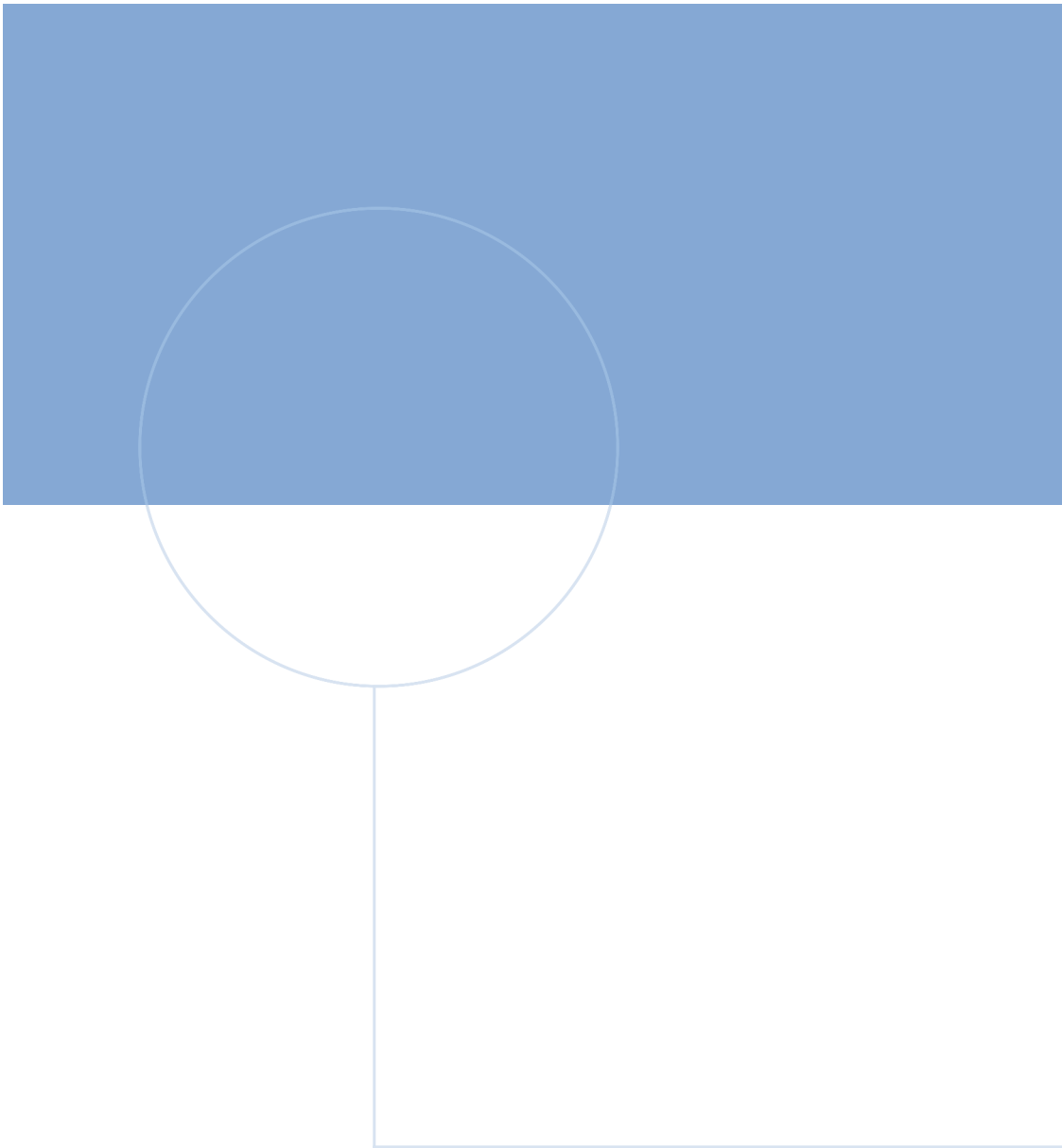
[58] Torch Contributors, *Torch.nn*, https://pytorch.org/docs/stable/nn.html, Accessed: 2022-05-15, 2019.

[59] Python Software Foundation, *Time — time access and conversions*, https://docs.python.org/3/library/time.html#time.perf_counter, Accessed: 2022-05-15, 2019.

[60] M. Almseidin, A. A. Zuraiq, M. Al-Kasassbeh and N. Alnidami, 'Phishing detection based on machine learning and feature selection methods,' *iJIM*, vol. 13, no. 12, 2019.

[61] Y. Sawa, R. Bhakta, I. G. Harris and C. Hadnagy, 'Detection of social engineering attacks through natural language processing of conversations,' in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, IEEE, 2016, pp. 262–265.

[62] H. Marius, *Rip bert: Google's mum is coming*, https://towardsdatascience.com/rip-bert-googles-mum-is-coming-cb3becd9670f, Accessed: 2022-05-15, 2022.



NTNU

Norwegian University of
Science and Technology