URL-Based Phishing Website Detection using Deep Learning Methods

Sam Evans

School of Computing and Communications

Lancaster University

B.Sc. (Hons) Computer Science

March, 2023

URL-Based Phishing Website Detection using Deep Learning Methods

Sam Evans,

School of Computing and Communications, Lancaster University.

March, 2023.

Abstract

This report discusses the creation of a Deep Learning model capable of discerning whether a given URL leads to a phishing (malicious) website or a benign (legitimate) one. The project aims to determine whether models with no internet-based information are able to accurately discern the legitimacy of any given URL, and how these models might compare to industry-standard internet-enabled models. Different feature extraction methods are explored and evaluated for their effectiveness given the limitations of modern datasets as well as threat actor techniques. Three different prediction models are proposed each of which are capable of classification of an unseen URL using both subsets of industry standard methods and novel strategies. Each proposed model builds upon the previous such that the resulting performance of each model takes advantage of the techniques used in the previous model as well as its novel approach. The models serve as a proof-of-concept for a privacy-centric approach to digital safety and security. The models are tested against multiple different industry-standard datasets each of which is intended to test the model's perfomance under different conditions. The aims of this are to determine whether the model over-fits to a specific type of URL approach preferred by a given dataset or a specific time period from which the training data comes from. It may also serve as a visualisation of the shifting naming techniques used by threat-actors today in an attempt to conceal phishing websites from always-improving

automated filtering systems. In the project, Google Colaboratory is used to run the projects with higher computational power than local machines, Python and, within it, the library TensorFlow is used to create, train, test and run the models.

A full working repository of source code and data can be found at: https://github.com/samevans77/TYP

Declaration

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Sam Evans,

24th March, 2023

Contents

List of Figures

Chapter 1

Introduction

Phishing is a type of cyberattack where a threat actor poses as a legitimate entity to lure individuals into trusting them. A victim may then provide sensitive personal information such as passwords or banking details, or may be tricked into downloading malicious files which could be used in future attacks. Phishing is the most common type of cyberattack in the UK representing 83% of successfully identified attacks against UK businesses in 2022 [11]. Phishing is so prominent for a number of reasons. Firstly, it can be automated such that thousands of threat vectors, i.e. messages, emails, phone calls, can be sent to as many potential victims as possible to increase the chances that the threat actors are able to find any victim at all. Secondly, phishing is often the path of least resistance for a threat actor. Instead of trying to break into a victim's system from the outside finding vulnerabilities and exploiting them in complex ways, a threat actor can instead simply manipulate a victim into giving them the information which they are seeking to steal, or into installing a program which the threat actor wishes to use for future attacks. Attacks like this being relatively low-skill means that they can be launched by almost anyone with understanding of website design and the inclination to steal user information. Furthermore, phishing attacks generally make use of techniques which cannot be outright stopped by product manufacturers since they do not exploit vulnerabilities in computer systems but instead exploit vulnerabilities in human nature. This

means that phishing attacks are unlikely to go away and, as vulnerability detection becomes more sophisticated, it is likely that even more emphasis will be placed by threat actors on phishing in order to perform cyberattacks. This is because phishing is often used as a gateway to other more sophisticated cyberattacks- since phishing exploits the authority which users have to manipulate a system or download more malicious payloads. It is for these reasons that the mitigation of phishing attacks has been researched for as long as the cyberattack has existed, this project aims to contribute to this research.

Generally, phishing involves the use of a website to trick users into believing the legitimacy of the malicious party. Therefore, the classification of such websites as phishing or benign is performed in order to prevent users from being misled. It is, of course, in the interests of threat actors to prevent accurate classification of their phishing websites for as long as possible in order to acquire as many victims as possible. This has created a cat-and-mouse game between threat actors and internet providers where threat actors try to bypass securities constructed by internet providers and internet providers then attempt to refine their classification systems to prevent this bypass. Classification techniques have varied over the years, and a brief overview of some notable techniques used for classification will be given in the background section. Modern-day techniques take advantage of Machine Learning (ML) to create highly accurate classification systems. These classification systems generally use large numbers of features extracted from the website itself, as well as external information such as the DNS record and SSL certificate of the domain in order to create a high-accuracy model.
There are limitations to current state-of-the-art approaches, the reliance on external information in classification may lead to unpredictable results if threat actors learn to manipulate these external sources. Furthermore, feature extraction using so many external features can be a computationally expensive process.

Another issue raised by external-information-based feature extraction is that it lacks the privacy and security which some individuals will require, as information

which is queried to remote services is no longer within the control of the user. In an age where an individual's information is increasingly becoming the property of large corporations there is a need for a privacy-centric approach to security, and therefore to the classification of phishing URLs, which currently does not exist in any sophisticated way. It is from these limitations that this paper contributes the following:

1.1 Contribution Statement

This paper explores whether it is possible to create a model which is capable of accurately classifying a URL as phishing or benign given no access to the internet. This paper explores methods of feature extraction using no external sources, relying on the URL and features which could be extracted solely from the URL to train and test Deep Learning (DL) models. This paper explores this in the creation of three separate models, each of which builds on the approaches of the previous with the aim of creating a proof-of-concept that a high-accuracy model can be created given the limitations of the project. This paper then evaluates the performance of such approaches against industry standard approaches. While internet-enabled approaches are the industry-standard and have high accuracy for good reason, the creation of a model which conforms to these limitations does fulfill the requirement for a privacy-centric and computationally-light approach to phishing website classification, and has the capability to fulfill a societal niche for privacy-focused scenarios.

1.2 Aims

1. Create deep learning models capable of detecting phishing websites using only URLs;

2. Make use of exclusively URL-based feature extraction methods which are based

on current state-of-the-art approaches;

3. Evaluate the performance of newly created DL models against highly accurate industry-standard approaches.

1.3 Organisation

The following chapters are developed as follows:

chapter 2 - Background A review of the current state-of-the-art revolving around the report area. An evaluation of the limitations of current approaches.

chapter 3 - Technical Preliminary An identification and definition of key theoretical topics throughout the project.

chapter 4 - Design An analysis of the design decisions made throughout development of the project, as well as an exploration of potential avenues from which to pursue the project.

chapter 5 - Implementation An analysis of the techniques used to complete the project, and how the design decisions were implemented in practice.

chapter 6 - Results Evaluating the implemented project's performance on a number of defined metrics against industry-standard methods

chapter 7 - Conclusions Discussion of results as well as future research directions.

Chapter 2

Background

2.1 Behaviour of Phishing Attacks

Mohammad, Thabtah and McCluskey (2015) [20] found that phishing attacks generally fall under three primary categories:

1. Mimicking attack: A type of attack where a website is carefully tailored to match the design of a legitimate website. Normally an unsuspecting user is sent an email/message/phone call asking them to update/set/validate personal details and by clicking the link are sent to the fake web-page to enter their details. These are then harvested by threat actors and used or sold for profit.

2. Forwarding attack: Similar to mimicking, a phishing link redirects victims to a web-page which asks submission of personal user information before redirecting the victim to the real website. This reduces overall detection since according to a victim, they followed the website's instructions and the website then behaved exactly as expected.

3. Pop-up attack: A carefully designed pop-up window mimicking a legitimate service requests users submit their information or potentially download malicious software. Threat actors manipulate the victims into believing that the use of a pop-up window is more authoritative or requires more urgent

attention. Sometimes the pop-up windows use scripting techniques to prevent easy removal of the pop-up window as a form of scare-ware. [17]

Mohammad, Thabtah and McCluskey note that all classic phishing attacks have a fraudulent web-page as part of their attack cycle.

Once an attacker has successfully stolen the victim's details, they use their victim's data for profit. This is done either by impersonating the victim and committing fraud or sometimes their information is then sold on digital black-markets. Alternatively, if the victim has unintentionally downloaded malicious software it may lay undetected for some time silently stealing the victim's information and sending it back to a threat actor.

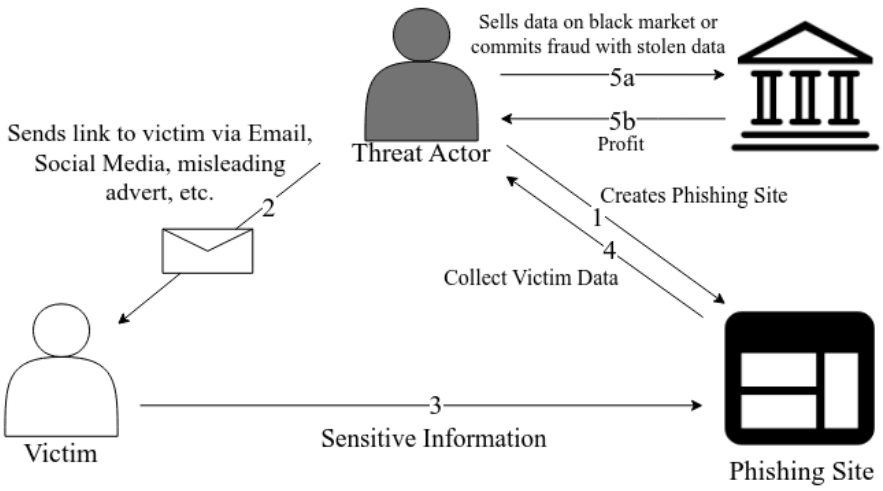


Figure 2.1: Taxonomy of a phishing attack.

2.2 Phishing Website Detection Approaches

2.2.1 Modern Browser Approaches

The current widely adopted approach modern browsers use to detect phishing attacks is by using denunciation platforms. These behave as an external repository of blacklists used to inform which websites should be blocked by any specific browser. These can be fed through community denunciations [25], and through automated systems [15]. The most popular of the denunciation platforms is SafeBrowsing [13] which is maintained by Google. While denunciation platforms are efficient and simple, they have some significant limitations. The main limitation is that the platforms are ineffective at detecting new phishing URLs, an attack known as zero-day phishing. (AlEroud, Zhou, 2017, Srinivasa, Alwyn, Pais, 2019) [3] [23].

Another limitation of denunciation platforms is the amount of resources and data which they require to maintain an accurate representation of current threat landscapes. Since phishing websites are active for a very short amount of time [21] denunciation platform repositories end up containing huge numbers of dead phishing URLs and very few active ones.

Finally, one way threat actors can circumnavigate denunciation platforms is by changing their URLs by just one character- which would make it an entirely new URL to the platform and therefore no longer be caught by the system. While this strategy would increase the chances that an unsuspecting user may notice the phishing website isn't what it is mimicking- the letter change could help the user notice the URL is not what they believed it to be- it's unlikely that it has any significant affect on the number of victims.

It is as a result of these limitations that predictive solutions have been created to take a given URL and determine its legitimacy to be used in combination with a denunciation platform, thus increasing the probability that a phishing website is caught before it is served to any potential victims.

2.2.2 Types of Prediction Approaches

For the purposes of this paper, there are two major groups of prediction approaches [8]:

1. Content-Based Approach: The predictive model takes content from the web-page itself such as images, JavaScript code, any HTML elements and uses them in order to create a more real accurate picture of the true intentions of the web-page. While this requires considerable processing power and time in order to query the relevant parts of the page, this approach is likely to produce highly accurate results.

2. Non-content-based Approach: The predictive model is only given the URL to work with. In this case, often blacklists or lexical analysis is used in order to discern whether the URL is phishing or benign. This approach is likely to be quicker than content-based approaches because no details need to be extracted from the web-page. However, these approaches could use external services in order to discern additional information about the domain given only the URL. Such queries to remote sources raise potential security or privacy concerns, as these systems would be active across any web-page visited by an end user queries could leak private information about unsuspecting users. It is likely that the limited nature of non-content-based approaches means that the results are less accurate than content-based approaches.

Generally, phishing web-page detection falls under two categories: Rules-Based detection and Machine Learning (ML) detection:

2.2.3 Rules-based detection

A rules-based system relies on the notable differences between phishing and benign web-pages. Research revolves around synthesis of effective rule sets to distinguish

web-pages. Should a web-page's features when applied to the rules not fall between expected values for benign web-pages it would be labeled as suspicious or phishing depending on the severity from which the web-page diverts from the expected values. Zhang et al. (2007) [26] propose a content-based approach for web-page classification. The authors make use of the Term Frequency/Inverse Document Frequency (TF-IDF) algorithm to extract key terms from any given web page. From these terms, a lexical signature for the page is generated using the five terms with the highest TF-IDF weighting. These are then queried on a search engine (of which the authors use Google). Should the domain of the page being analysed appear within the top X results (X because the authors varied the number of top results to balance for true and false positives,) then the page is deemed to be legitimate. CANTINA also makes use of some heuristic features established by Fette et al. (2007) [12] and Chou et al. (2004) [24] which are used throughout the field of phishing web-page detection. These include:

"At" symbol Phishing sites' URLs have no relation to the site which they are attempting to mimic. It is as a result of this that they often use the "at" (@) symbol which makes the browser disregard anything written to the left of the @ symbol. This can be used in combination with the fact that address bars have limited space in order to hide suspicious elements of the URL after legitimate elements. For example, `paypal.com/account@not-phishy.com/abcd` would redirect the user to `not-phishy.com/abcd`.

"Dash" symbol Dashes (-) are symbols which are rarely used in the URLs of legitimate services. Malicious services can use dashes to use legitimate and recognisable service names in their URLs while still being able to use the URL for their own purposes. Consider the examples `paypal-payment-help.com/account` and `paypalpaymenthelp.com/account`. In the first example, threat actors are able to separate a recognisable word to a potential victim which may make them more likely to believe the web-page is representing the service it is mimicking.

IP in URL Attacks can simply be hosted on PCs which may not have specific DNS entries. The easiest way to access a page hosted on a PC would be by their IP address. This would also allow threat actors to append whatever they wished to the remainder of the address. For example, `192.168.0.1/paypal.com?account`. Less technically literate victims are less likely to question the IP address and would instead recognise what follows it and the mimicked service.

Number of dots One method by which threat actors are able to construct URLs which may look legitimate is to use subdomains. This would allow a threat actor to separate words of legitimate services into separate domains as in this example: `payp.al.help.com/account`. This approach has gained credibility in recent years in the eyes of an average consumer as it is practice of some legitimate services to split their service name into separate domains for example, the legitimate URL: `youtu.be/dQw4w9WgXcQ`.

Rami et al. (2014) [19] identified 17 features which could be used to classify benign and phishing web-pages. Some of these features are the as listed above, features which are found in the URL itself. Some of these features were content-based such as whether the web-page was using JavaScript to disable the right-click function. If using right-click was completely disabled this feature would be classified as phishy, if an alert was shown when a right-click was performed it would be considered suspicious and otherwise for this feature the page would be considered legitimate. Some of these features were domain-based, and included querying external databases for information relating to website traffic or the age of the domain. For example, if the web-page ranked in the top 100,000 websites of a known top 1,000,000 websites it would be considered legitimate, if the web-page appears outside of the top 100,000 websites it would be considered suspicious and if the website does not appear at all (as it may not be alive anymore and thus not in the database, or may not have been visited enough to register in the top million websites) it would be considered phishy. These features were then placed into a range of rules-based classification algorithms

such as C4.5 and RIPPER.

2.2.4 Machine Learning-based detection

In Abu-Nimeh et al. (2007) [2] multiple previous studies using Machine Learning (ML) techniques are evaluated. Using the techniques used in previous studies as well as emerging ML techniques the authors set out to evaluate the methods against a new dataset of 2889 emails. Features are extracted through identifying the frequency of words used in the emails and creating a dataset consisting of 43 binary values. In addition, TF-IDF identifies the most common terms and uses this to weight the binary values. Logistic Regression, Classification and Regression Trees, Random Forests, Neural Networks, Support Vector Machines and Bayesian Additive Regression Trees are all analysed against this dataset. Across all experiments it was found that there was a trade-off between the False Positive (FP) and False Negative (FN) rates between any given algorithm. For example, Random Forests achieved 8.29% FP rate, and 11.12% FN while Logistic Regression achieved 4.89% and 17.04% respectively.

Waleed and Adlen (2019) [4] perform weighted feature selection on the UCI Machine Learning Repository Website Phishing Data Set using Genetic Algorithms [10]. The authors evaluated a number of different ML techniques including SVM, C4.5, kNN, and DNN. The authors found that DNN outperformed all other ML techniques using both the Genetic Algorithm-weighted data as well as without any feature selection and weighting adjustment, achieving an 88.77% accuracy without features selection and weighting and 90.39% with.

Do et al. (2021) [9] evaluated the findings of previous research into different DL techniques like DNN, CNN, LSTM. The authors use the University of California Irvine Machine Learning Repository of 11055 URLs (4898 Phishing, and 6157 Legitimate) [10] to evaluate the performance of different DL techniques under optimal conditions. The authors found the optimal structures for each of DNN, CNN, SLTM and GRU respectively and gave their parameters when being used

with the UCI dataset. The authors found that DNN models were most accurate under optimal conditions, achieving a 97.29% accuracy, however concluded that no model was the best in all performance metrics and that researchers should select their model approach according to their specific needs.

2.3 Limitations of current approaches

Current detection approaches revolve around two main approaches each with their own advantages and limitations. Content-based detection revolves around parsing through the content of a web-page and extracting key features. These approaches are likely to extract features most representative of the web-page and therefore carries the highest chances of finding features which can accurately classify the web-page. There are some limitations to this approach; firstly, scanning through the page, extracting key features and using these to classify the web-page is a computationally intensive approach. Furthermore, because of the short-lived nature of phishing websites there remain issues to create full datasets. Because some catalogued URLs will not lead to a web-page, it's unlikely that datasets will be sufficient to create highly accurate classification models.

Non-content-based detection revolves around taking as much information as possible from the URL. The most accurate approaches revolve around querying additional data from the URLs. This could include information regarding the URLs SSL certificate, finding website traffic information or finding URL registry information using WHOIS. This approach holds a number of limitations; firstly, as previously, the short-lived nature of phishing URLs means that extensive datasets are difficult to create given simple raw URL data. Secondly, querying data from URLs to different services is both time-intensive and brings privacy concerns. Sending URL data to an external service could leak user data.

2.4 Proposal

This project proposes the use of a DL model using exclusively the URL and features which can be extracted solely from the URL text itself. This approach resolves any privacy issues since URL input, model processing and prediction output would be able to be run from just a local machine. This approach also alleviates potential security concerns since the feature extraction process would otherwise involve querying data from known potentially malicious websites. Furthermore, limited feature extraction is less computationally intensive than other approaches which increases the performance of the approach- making it more suited for industry application. Finally, the approach by taking only a URL as input means that virtually all available datasets can be used in training and testing of the model regardless of whether the domain is still active or has been taken down since the dataset was created. This means that extensive training data can be used to prevent URL overfitting to any specific dataset or time period- furthermore, data can be tested in order to discern whether modern URL detection practices works on old phishing URL creation strategies and vice-versa. The increase in data available for the model to use is also likely to improve model accuracy overall.

The project proposes three separate models, each of their approaches building on top of the previous model's approach. This means that the models can build upon the accuracy of previous models by using novel approaches. The project acts as a proof-of-concept that privacy-centric approaches are viable in human security- that is, protecting people from being misled, since often high-security approaches require an exchange of privacy to achieve high accuracy.

Chapter 3

Technical Preliminary

The following sections contain brief introductions and explanations to key technical areas which the remainder of this report will revolve around. These sections are important to understand the report and its conclusions.

3.1 URLs

A Uniform Resource Locator, more commonly known as simply URL, is a string of characters and symbols which always follows a specific structure. A URL is used as a textual representation of an IP address stored in the Domain Name System (DNS). The DNS takes a URL and returns its associated IP address, this is then used by browsers in order to access remote content by making requests directed towards the received IP addresses. A URL is structured to achieve high performance with

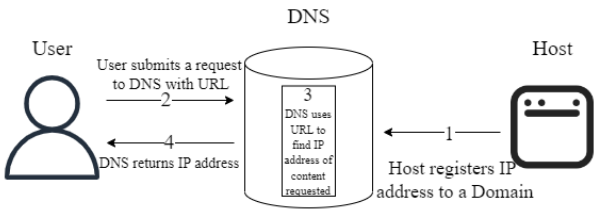


Figure 3.1: Simplified DNS example

the DNS. This is by having different components as follows. Each element of the

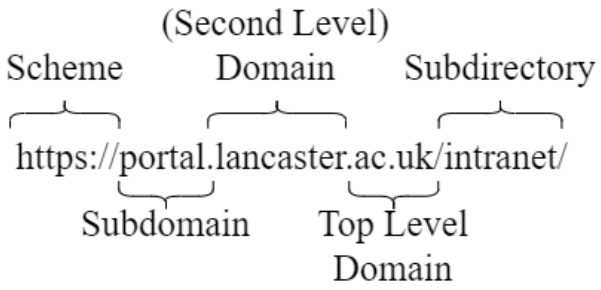


Figure 3.2: The components of a URL

Domain, in particular the Top Level and Second Level inform the DNS as to what specific server to search in order to find the associated IP address. For example, there may be a specific sever as part of the DNS which contains the IP addresses associated with `.ac.uk` Top Level Domains. The systematic nature of distribution of URL-IP address pairs allows the DNS to continue functioning at high performance even when handling higher quantities of internet traffic.

3.2 Deep Learning

Deep Learning (DL) is a subset of the Machine Learning (ML) field. The distinction between DL and ML is in data pre-processing. For example, a generic ML model may need specific features in a picture to be extracted. In ML, a feature is a distinct data-point used as an input to a model. If we were distinguishing an image of a stop sign and an image of a give-way sign this could be the specific location of the center of each sign. A DL model would be able to achieve comparable accuracy with only the image as a raw input. This is to say, a DL model derives its own features through self-correcting against feature-label pairs in training. In ML, a label is the respective output of a set of one or more features. In the previous example, this might be a binary value - `[0]` if the sign is a stop sign and `[1]` if the sign is give-way.

3.3 Deep Neural Networks

Deep Learning is implemented through Deep Neural Networks (DNN). They derive their name from the function of the brain, which DNNs attempt to emulate in a far more simplified form. DNNs are collections of interconnected nodes in a network - these nodes are known as neurons or perceptrons. These perceptrons are grouped into layers which are split into the input layer, the output layer and a number of hidden layers. Perceptrons take input data from the layer previous to it, process it and then output a result. In a fully connected layer each perceptron in the previous layer would output its result to the perceptrons in the next layer.

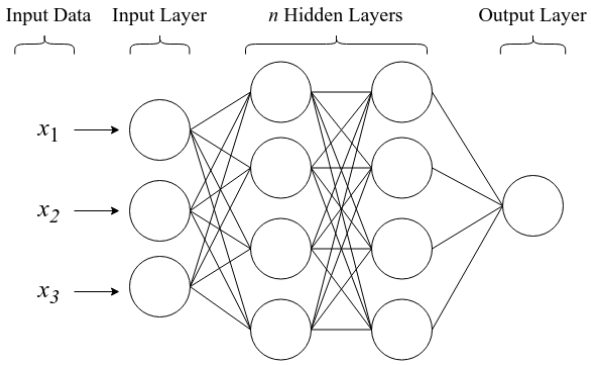


Figure 3.3: Example of a DNN with n = 2 hidden layers

3.4 Perceptrons

A Perceptron is one "node" in a DNN. It collects weighted inputs from the previous layer, performs a summation of the inputs and uses this as the input to an activation function. Where the output of the previous layer is $x_1, x_2, ..., x_n$ and the weights of each respective edge connecting the previous layer to the current perceptron is $w_1, w_2, ..., w_n$ the summation of the inputs is defined by:

$$\sum_{k=1}^{k=n} x_k \cdot w_k$$

An activation function takes the summation and uses a function to create an output.

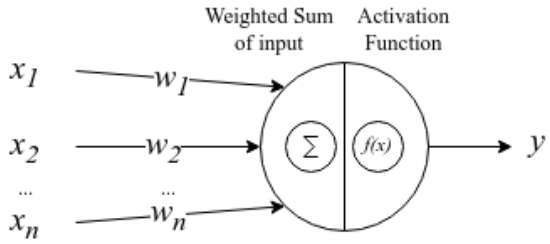
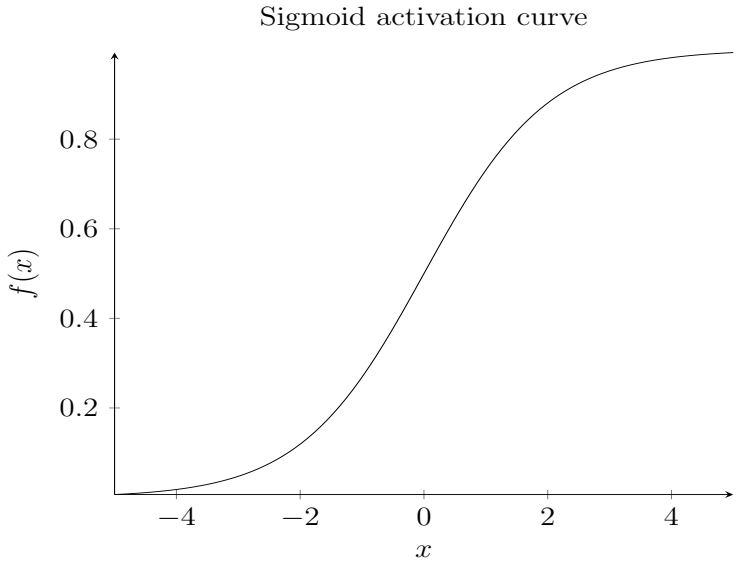


Figure 3.4: A diagram of a Perceptron

One popular activation function is the Sigmoid activation function which is defined by:

$$f(x) = \frac{1}{1 + e^{-x}}$$

This takes any real input and returns a number between 1 and 0. The larger the number is, the closer to 1 it is. The smaller the number is the closer to 0 it is:



Throughout the training process, input data is processed by the DNN and the output layer's result - the model's predicted output- is compared to the input data's corresponding label - the true output. The DNN then optimises its weights in order to minimise the difference between its predicted output and the true output of each input in the training data. Over multiple "epochs" (Separate runs through the training data, performing these operations and optimising the weights), this

approach is able to produce highly accurate models capable of classifying their inputs in the correct label space and eventually classify novel features into their labels.

Chapter 4

Design

The design philosophy when approaching the project was a privacy-centered high-accuracy DNN model. This means that there would be no external querying of information, as it would raise concerns as to how the queries would be safely made without leaking user information. This section includes all considered approaches and an evaluation as to their viability in conformity with the aforementioned limitations. It is as such that the design aims of the project are as follows:

1. Create a Machine Learning model using a Deep Neural Network capable of classifying a website as phishing or benign;

2. The agent's performance is comparable to industry-standard model performance as observed in the background section;

3. The model's data preprocessing and feature extraction is privacy conscious and doesn't risk leaking user visited pages and,

4. The model's data preprocessing is capable of taking a single input and computing the remainder of the data (if any) for itself.

4.1 Model Choice

All models proposed in Do et al. [9] were explored to determine the most effective model in terms of overall performance. It was determined that accuracy was the most important metric to determine. Given authors in Do et al. [9] and Waleed et al. [4] determined DNN to be the type of DL model capable of producing the most accurate results this was determined to be the approach to implement in the project.

4.2 Content-Based Detection

One way of extracting additional features from any given URL would be to find the web-page which it links to and extract features from the raw HTML. Such features are likely to be high quality for the purposes of the model because it is challenging for threat actors to conceal information which could give away that the web-page is malicious. One proposed approach would be to use a web crawler to derive additional URLs to scan from one URL by scanning the HTML. All of these URLs would then be put through a DNN model and the outputs would be weighted to create a prediction. One potential limitation of this approach would be that it is in a threat actor's best interests to pack their website with legitimate links in order to fool a user into believing it is legitimate. For example, using Facebook "share" links or embedding a Twitter feed of the company which the phishing website is attempting to pose as. This can be partially circumnavigated through proper weighting of outputs, however such approaches are likely to produce higher numbers of false positives. This is because URLs which have been predicted to be malicious will have a higher weighting to compensate for the aforementioned threat actor behaviour. Another potential limitation is the computation requirements of scanning through HTML data at the scales which would be useful to a Deep Learning model. With datasets in the tens of thousands even one second on each query for the HTML data of a URL would take upwards of nearly three hours for an appropriate

dataset to be synthesised.

One major issue noted with content-based detection is the acquisition of appropriate datasets. While datasets which include HTML data and URLs do exist, they are significantly outnumbered by simple URL data. To the author's knowledge, all major, regularly-updated phishing website repositories do not contain HTML data of the kind which would be used in the aforementioned approach. Since phishing websites have such a short lifespan, even up-to-date repositories are unlikely to hold active URLs not least because by being listed it means they have been detected as phishing and are thus of no further use to threat actors.

It is as a result of these reasons that while a content-based approach to detection was explored, it was ultimately deemed not to be viable if the resultant model was to conform with the limitations placed on the project.

4.3 Non-content-based Detection

If the content of the web-page is ignored, what other data might be available for analysis? The foremost piece of information is the URL. Any other data which would be available would have to be queried using the input URL as a parameter. Other information which could be derived from the URL could be internet traffic to the URL, WhoIs database searches for data regarding the URL's domain age or location of registry or could be security certificate information from the registered URL.

One limitation of feature extraction in this way is data completeness. In many cases, querying these services using a domain which is no longer active would return no data at all. As aforementioned, the time-sensitive nature of feature extraction on active phishing URLs means that it is highly unlikely that a model relying on tens of thousands of data points would be able to acquire a complete dataset. Another

potential limitation of this form of feature extraction would be that in order to acquire it, tens of thousands of queries into remote services using URL data must be made. In a real-time implementation of the model this would involve sending live user data to a remote server in order to acquire suitable input data for a model. This raises significant privacy concerns, and while it is likely possible to create a data preprocessing process which is capable of safely extracting the additional features this would fall outside of the scope of the project. This, therefore, leaves the URL as the primary feature.

During the initial literature review it was noted that the UCI Machine Learning Repository held a phishing URL dataset. This held 30 features, some of which were identified as being capable of extraction without external requests or the URL needing to still be live, these were:

IP address Does the URL contain an IP address?

URL Length Is the URL excessively long, a way which threat actors use to hide the suspicious elements of the URL?

Shortening Service Is the URL that of a shortening service such as `bit.ly`?

@ Symbol Does the URL contain an @ symbol?

"//" Redirection Does the URL contain "//" within its path for redirecting?

Prefix suffix Is there a prefix or suffix separated by a dash (-) in the domain?

Sub Domain Is there a subdomain, how many subdomains are there?

These features were noted and retained for use in the project implementation. The idea of using a subset of the model proposed by Do et al.[9] as a high-accuracy method of generating additional features was also retained for future investigation. Once any additional features have been extracted the URL data has to be processed to be usable to a DL model. A number of approaches were explored

when determining how effectively to format URL data to be processed by a DL model. Literature has explored language processing approaches to feature extraction through algorithms like TF-IDF or through term similarity.[26]
This project's novel approach relies on the inherent nature of all characters in all languages which is that they have a Unicode representation as well. These Unicode representations are then used as the model input.

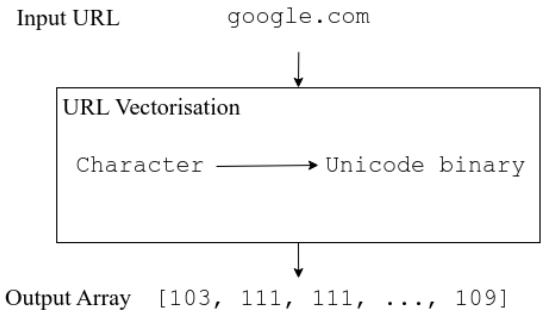


Figure 4.1: URL Vectorisation

In order to check the effectiveness of the methods proposed, three experiments have been created with the goal of discerning whether exclusively URL-based approaches are viable in phishing website detection. Each experiment builds on the approach introduced in the previous experiment - and increases the complexity of the inputs. These experiments are as follows:

4.3.1 Experiment One: URL Vectorisation

The first experiment involves acquiring a dataset split equally into phishing and benign data-points, the URLs are extracted and processed such that just the relevant text remains. The HTTPS/HTTP is removed as this feature is not longer relevant to discern phishing from benign web-pages, this is because threat actors have adapted to be able to acquire SSL certificates for any domain they may need. After this, all symbols are removed and the remaining text is concatenated. The text is vectorised and this is split into training and testing data. This data is then used as the input

to a deep learning model of similar design to the DNN model proposed in Do et al. [9] This approach is described in figure 3.2. After training, the model is evaluated by comparing the predictions outputted from the DNN model when provided with the testing data with the true values of the testing data. The purpose of this experiment is to evaluate whether the proposed approach using URL vectorisation is viable for phishing URL detection at all, assuming it is this approach is built on in experiments two and three in order to extract further features from one URL.

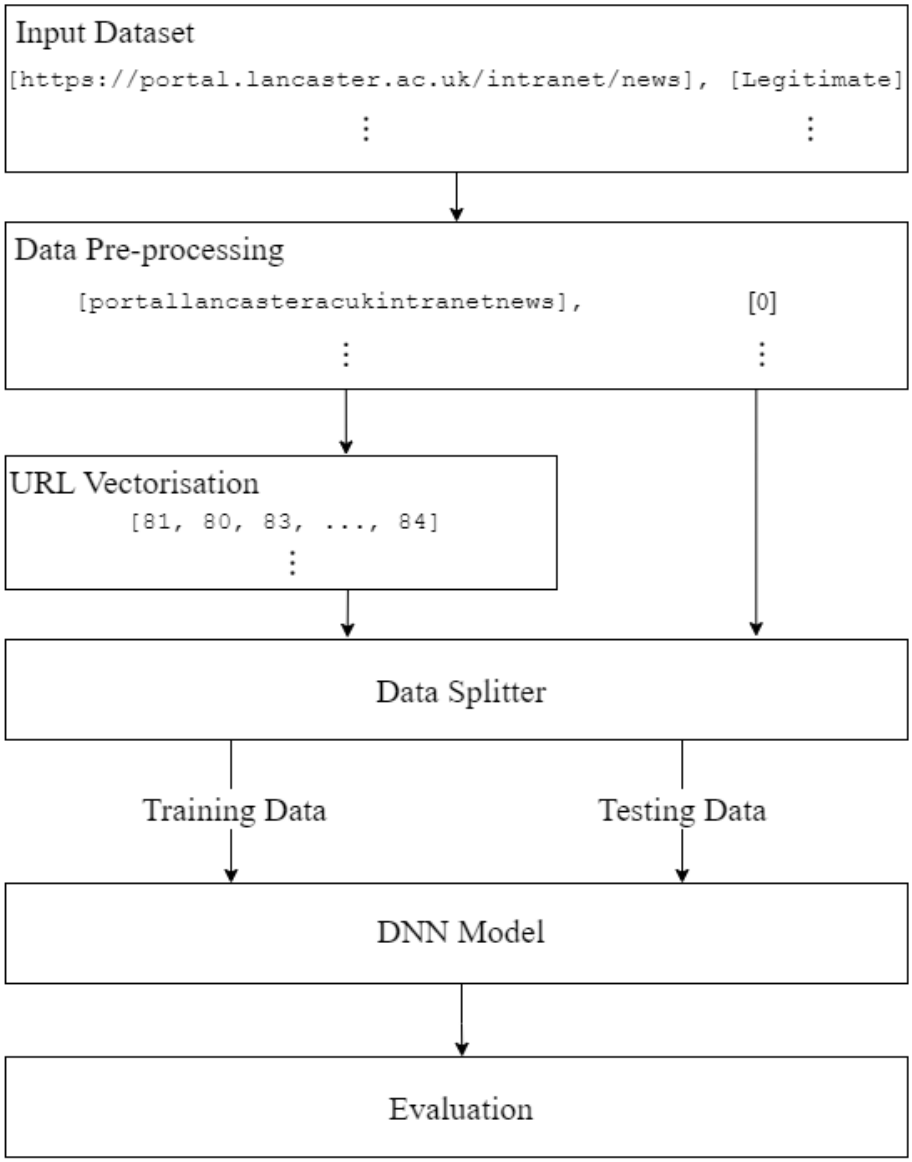


Figure 4.2: Experiment One Design

4.3.2 Experiment Two: Splitting URLs

The second experiment's purpose is to determine whether the explicit splitting of the URL data into its individual components as outlined in figure 3.2, "The components of a URL" would improve the performance of a DNN model. This is done by parsing the contents of each URL and extracting each component. Each individual component is then vectorised separately. The DNN model is designed in this approach with a different input node for each URL component, thus explicitly passing them in as separate features to the model. For example, in input of `blog.google.com` would result in the splitting of features as `[blog],[google],[com],[]`, the empty set because the example URL's subdirectory is empty. This approach can be represented as follows:

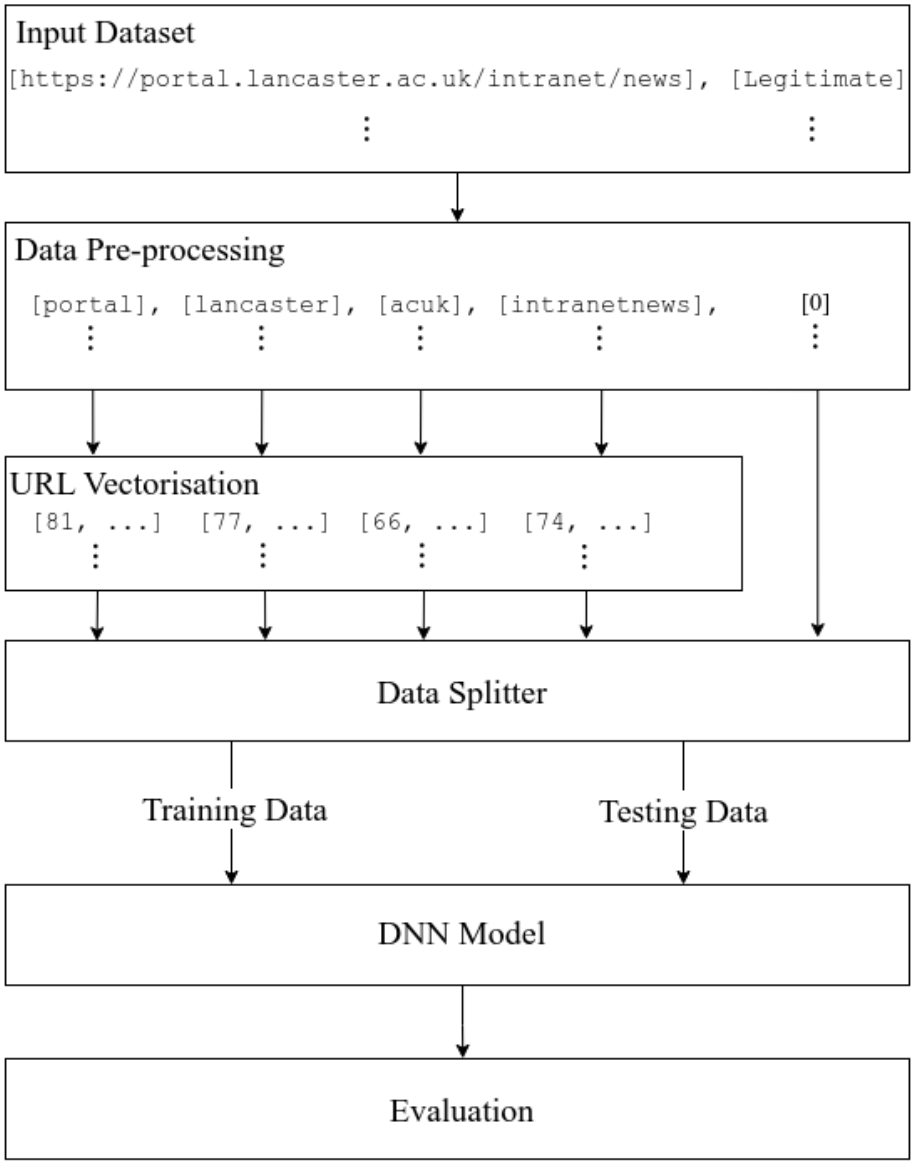


Figure 4.3: Experiment Two Design

4.3.3 Experiment Three: Hybrid Approach

The third experiment's purpose is to determine whether additional feature extraction using purely URL-based features improves the DNN model's performance. The features extracted are the URL-based features outlined in the UCI Machine Learning Repository Phishing Dataset [10]. In the dataset as given, -1 is a feature consistent with phishing URLs and 1 is a feature consistent with benign URLs, with 0 being feature consistent with a suspicious URL. These are as follows:

1. IP in URL; [No IP present → 1, IP present → -1]
2. URL Length; [Length > 74 → -1, 74 > Length > 54 → 0, Length > 54 → 1]
3. URL shortening service; [URL Shortening service → -1, Otherwise → 1]
4. "@" in URL; ["@" present → -1, Otherwise → 1]
5. "//" in URL; ["//" Present → -1, Otherwise → 1]
6. Prefix or suffix separated by "-", ["-" Present in Subdomain, Top Level Domain or Second Level Domain → -1, Otherwise → 1] and,
7. Number of dots (subdomains) [Dots > 3 → -1, Dots = 3 → 0, Dots < 3 → 1]

With this dataset two approaches are possible. Firstly, a subset of the model proposed in Do et al. [9] could be created using only the seven features outlined above. It could be trained and tested using a subset of the UCI Machine Learning Repository Phishing Dataset, then its prediction to the extracted features from the original dataset would be used as one high quality feature. Assuming high accuracy of the subset model, this feature would be high quality and would significantly improve the model's accuracy as a whole.

Alternatively, the extracted features could be used as a new input to the model much like the separate components of the URL were.

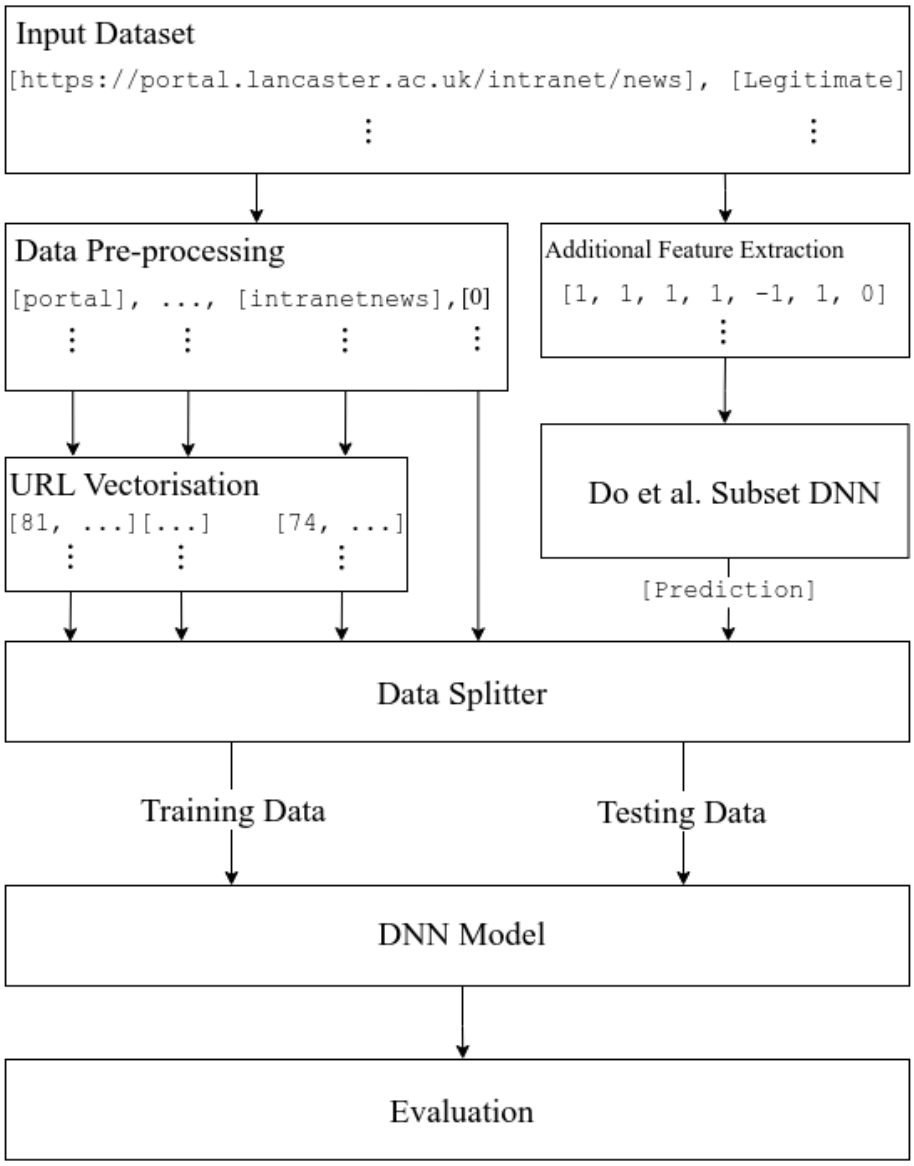


Figure 4.4: Experiment Three First Design

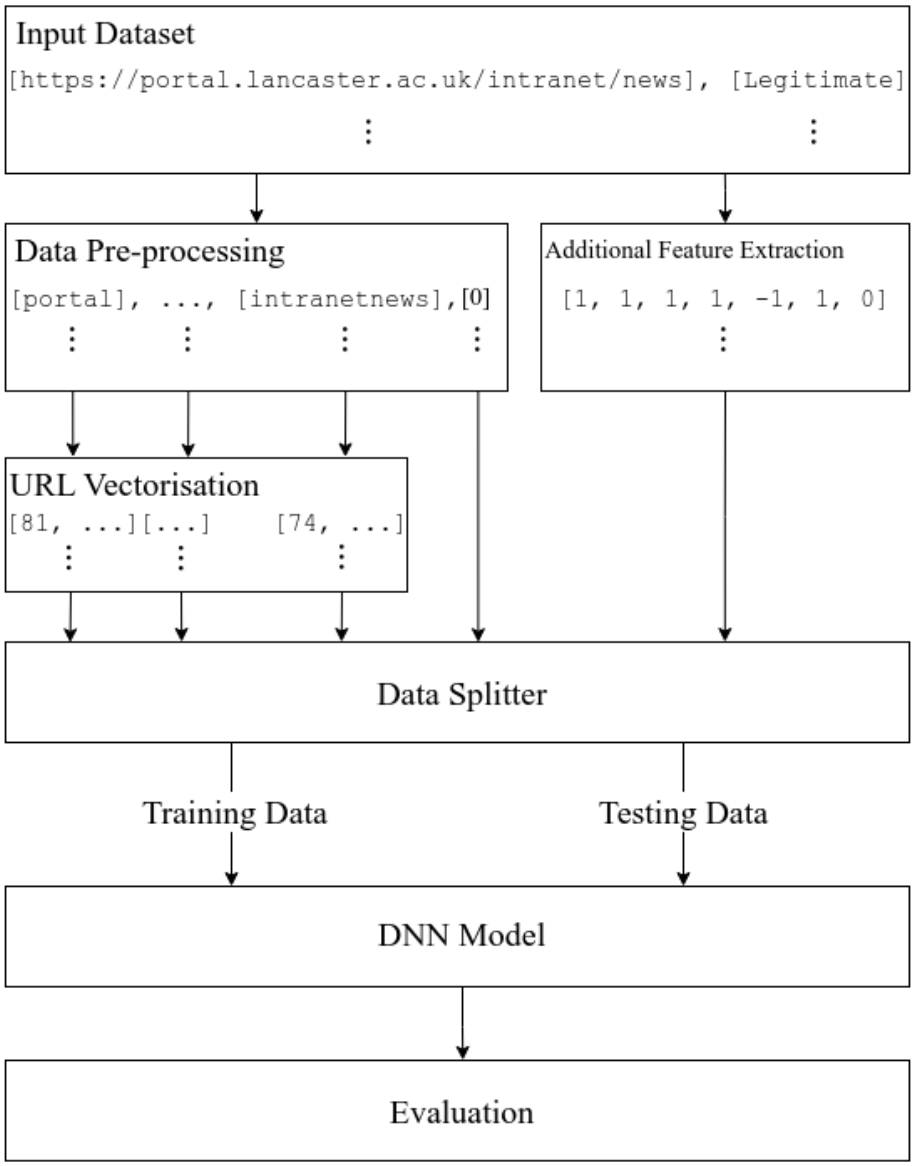


Figure 4.5: Experiment Three Second Design

Chapter 5

Implementation

In this chapter, the report discusses the mechanisms by which the results of this report were synthesised. This includes an explanation of the environments used to produce and run the models, as well as how the data was acquired, prepared, and finally how the models were created.

5.1 Environment

5.1.1 Software

The project is written in Python, a programming language which the vast majority of AI research is conducted in. This is largely because the two main ML libraries, TensorFlow [1] and PyTorch [22], were written expressly for Python. Another benefit of Python is that it produces highly readable code, useful for new programmers as well as for demonstrating code. TensorFlow was selected as the ML library with which to implement the project. This decision was mostly because of the number of resources available for learning TensorFlow from scratch. Furthermore, since TensorFlow is written by Google it is likely to make fewer issues with regards to compatibility in Google Colaboratory [14]. Within TensorFlow, the Keras [7] was used to build, compile, train and test the models.

5.1.2 Hardware

As aforementioned, the hardware used is a cloud-based code-running service known as Google Colaboratory [14]. It allows users to upload Python code for it to run in its own hardware. This allows for significant performance increases over using a personal computer and the service is free. This facilitated the execution of more computationally-intensive code, such as more extensive training operations on larger datasets which would otherwise exceed traditional memory capacity rules in TensorFlow. The service provides a GPU runtime with Intel Xeon CPU @2.20GHz, 13 GB RAM, and 12 GB GDDR5 VRAM. All formal results synthesised were done so using this hardware configuration.

5.2 Data Preparation

5.2.1 Experiment One

The dataset used for initial training and testing was the dataset synthesised in Marchal et al. (2014) [18]. This is because it contains an extremely high number of data points for both Phishing and Benign web-pages (48,009 for both), this allows for early testing regarding feature extraction performance. Furthermore, the dataset held additional information such as page ranking score on the Alexa top million websites [6] which in the planning stages of the process was considered potentially useful. The relevant URL and label data was extracted through CSV file handling.

URL vectorisation was completed through the following code:

```
1 def convert_to_unicode(plaintext, total_length):
2     output = ' '.join(format(ord(i), 'b') for i in plaintext)  #
    convert to bytes first. (7 bits)
3
4     out_array = []
5     for i in output.split(" "):
```

```
6         out_array.append(int(i, 2))  # convert to an array of
    integers.
7     for j in range(len(out_array), total_length):
8         out_array.append(0)  # adds zeroes on for as many as
    necessary
9
10    return out_array
```

Listing 5.1: URL Vectorisation

The for loop beginning on line 7 is necessary because Deep Learning models programmed in TensorFlow expect input tensors (meaning, for the purposes of this paper, arrays) which are identical in size for each individual input node. For example, `[81, 82, 81, 102, 101]` and `[65, 73, 56, 90, 105]` would be compatible however `[76, 48, 104, 101]` would not be compatible without a `0` appended to the end of it: `[76, 48, 104, 101, 0]`.

Feature-label pairs were separated with a 70:30 ratio between Training and Testing data, and then the model is trained using the simple Python command:

```
1 model.fit(train_features, train_labels, epochs=x, batch_size=y)
```

Listing 5.2: DNN training

This model could then be evaluated with the testing data to extract relevant statistics such as accuracy to determine the model's performance.

5.2.2 Experiment Two

Experiment two's approach differed primarily in the method by which data was fed into the DNN, this involved splitting URL data into its component parts as given in figure 3.2 however the scheme was not used as it is established this is not a classifying feature. This is completed by parsing the input URL to extract the individual components as given in the following code:

```
1 def remove_special_characters(input_string):
```

```
    # takes any given string, returns it without any punctuation in
    special_characters = list(string.punctuation)
    new_string = ""
    for character in input_string:
        if character not in special_characters:
            new_string += character

    return new_string


def format_string(input_string):
    # take any given string, format it (removing https, www.)
    if input_string.startswith("'"):
        input_string = input_string[1:]  # removes the ' which some
    parts of the dataset includes.

    if input_string.endswith("'"):
        input_string = input_string[:-1]

    # split by subdomain, second-level domain and top-level domain
    split_string = input_string.split("/")[0]
    subdirectory = ""

    for j in range(1, len(input_string.split("/"))):
        subdirectory += input_string.split("/")[j]

    split_string = split_string.split(".")
    # need to scan for subdomain.
    subdomain = ""
    startpoint = 1

    # to make work for example www.users.waitrose.com (found in url
    set)
    if len(split_string) > 3:
        subdomain = split_string[1]
```

```
        second_level_domain = split_string[2]
        startpoint = 3
    elif len(split_string) == 3:
        subdomain = split_string[0]
        second_level_domain = split_string[1]
        startpoint = 2
    else:
        second_level_domain = split_string[0]
        startpoint = 1

    top_level_domain = ""

    for i in range(startpoint, len(split_string)):
        top_level_domain += split_string[i]

    return remove_special_characters(subdomain),
   remove_special_characters(second_level_domain),
   remove_special_characters(top_level_domain),
   remove_special_characters(subdirectory)
```

Listing 5.3: URL splitting

Data which is extracted here is then placed into a model which has a separate input node for each element of the URL.

5.2.3 Experiment Three

Additional feature extraction was completed through text analysis of the URL. The algorithm was written to be a simple set of text analysis operations. Later in the development process, this code was modified to allow parameterisation of the specific features such as URL length in order to tune the additional feature extraction to a user's needs. The algorithm to complete this was as follows:

```
dictionaryURLService = {"bit": "ly", "tinyurl": "com", "ow": "ly",
   "srt": "gy"}
ip_pattern = r"\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b"
```

```
3
4  # 1 denotes benign
5  # 0 denotes suspicious
6  # -1 denotes Phishing
7
8
9  def extract_additional_features(input_url, label, verbose):
10     output_array = []
11
12     if verbose:
13         print(input_url)
14
15     # feature 1: having ip address in url
16     ip_match = re.search(ip_pattern, input_url)
17     if ip_match:
18         ip_address = ip_match.group(0)
19         if verbose:
20             print("IP In URL -", ip_address, "- phishing")
21         output_array.append(-1)
22     else:
23         if verbose:
24             print("No IP in URL - benign")
25         output_array.append(1)
26
27     # feature 2: url length
28     if len(input_url) > 74:
29         if verbose:
30             print("Long url - phishing")
31         output_array.append(-1)
32     elif len(input_url) < 54:
33         if verbose:
34             print("Short url - benign")
35         output_array.append(1)
36     else:
37         if verbose:
```

```
38              print("Medium-sized url - suspicious")
39          output_array.append(0)
40
41      # feature 3: shortening url service
42      subdomain, secondlevel, toplevel, subdirectory =
    format_string_wsc(input_url)
43
44      if secondlevel in dictionaryURLService.keys():
45          if toplevel == dictionaryURLService[secondlevel]:
46              if verbose:
47                  print("IS a shortened url - phishing")
48              output_array.append(-1)
49          else:
50              output_array.append(1)
51      else:
52          if verbose:
53              print("Not a shortened url (that we know of) - benign")
54          output_array.append(1)
55
56      # feature 4: having @ symbol
57      if "@" in input_url:
58          if verbose:
59              print("Contains @ - phishing")
60          output_array.append(-1)
61      else:
62          if verbose:
63              print("Does not contain @ - benign")
64          output_array.append(1)
65
66      # feature 5: // for redirection
67      if "//" in input_url:
68          if verbose:
69              print("Contains // - suspicious")
70          output_array.append(-1)
71      else:
```

```
72          if verbose:
73              print("Does not contain // - benign")
74          output_array.append(1)
75
76      # feature 6: adding a prefix or suffix separated by - to the
    domain
77      if "-" in subdomain or "-" in toplevel or "-" in secondlevel:
78          if verbose:
79              print("Contains dash - suspicious")
80          output_array.append(-1)
81      else:
82          if verbose:
83              print("Does not contain dash - benign")
84          output_array.append(1)
85
86      # feature 7: sub-domain and multi sub-domains
87      dot_count = 0
88      for character in input_url:
89          if character == ".":
90              dot_count += 1
91
92      if dot_count < 3:
93          if verbose:
94              print("Contains less than 3 dots - benign")
95          output_array.append(1)
96      elif dot_count == 3:
97          if verbose:
98              print("Contains 3 dots - suspicious")
99          output_array.append(0)
100     else:
101         if verbose:
102             print("Contains more than 3 dots - phishing")
103         output_array.append(-1)
104
105     output_array.append(label)
```

```
106
107     if len(output_array) == 8:
108         if verbose:
109             print("Output array of correct length")
110         return output_array
111     else:
112         print("[ERROR]: Output array in additional feature
    extraction not of recognised length")
113         print(output_array)
114         print(input_url)
115         return 0
```

Listing 5.4: Additional Feature Extraction

5.3 DNN Model

5.3.1 Input Layer

The shape of the input layer depended on the experiment. This is because in order to treat the separate elements of the input as distinct different input nodes would have to be used. In the first experiment, the input node was solitary - accepting a fixed-size input vector. This was tuned in order to maximise model performance and was held at a maximum fixed size of 64.

For the second experiment, separate input nodes had to be created for each element of the URL as defined by figure 3.2. For this experiment, the size of of vector which the input nodes accepted could be kept at a fixed size each time or be extended such that no input element would be truncated. This approach was taken and repeated across different test cases for consistency.

For the third experiment, similarly to the second experiment a separate input node was used for each feature of the URL however an additional feature space was given for the additional feature extraction. This final input node varied in size depending on whether the feature was the set of extracted features seen in listing

5.4 or the assigned label from the additional DNN.

5.3.2 Hidden Layers

The hidden layers of each of the three experimental models followed the same structure in each of them. They are in a similar structure to that given in Do et al.'s [9] approach. Each of the hidden layers follow the 'Rectified Linear Unit' activation function, which is to say they output directly what their input was if positive, and otherwise output zero. This is a commonly used output functions and was used almost exclusively by Do et al. in their hidden layers in their DNN implementation. Six dense hidden layers were used in a configuration `128, 64, 32, 16, 8, 4` in order to balance processing time and model effectiveness, this was established in Do et al.'s work. A simple implementation of the above is as follows:

```
1 hidden_1 = tf.keras.layers.Dense(128, activation=tf.nn.relu)(
    input_tensor)
2 hidden_2 = tf.keras.layers.Dense(64, activation=tf.nn.relu)(
    hidden_1)
3 hidden_3 = tf.keras.layers.Dense(32, activation=tf.nn.relu)(
    hidden_2)
4 hidden_4 = tf.keras.layers.Dense(16, activation=tf.nn.relu)(
    hidden_3)
5 hidden_5 = tf.keras.layers.Dense(8, activation=tf.nn.relu)(hidden_4
    )
6 hidden_6 = tf.keras.layers.Dense(4, activation=tf.nn.relu)(hidden_5
    )
```

Listing 5.5: Hidden Layers Implementation

5.3.3 Output Layer

The output layer was the same for each model, a single node with a sigmoid activation function, giving a number between -1 and 1. 1 Discerning a benign input

URL and -1 discerning a phishing input URL. Following on from the above example the code that implements this layer is given as:

```
output_layer = tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)(
    hidden_4)
```

Listing 5.6: Output Layer Implementation

As such, a complete program for the model, in the following example for experiment two, could look like the following:

```
# defining input tensor sizes
input_tensor_subdomain = tf.keras.layers.Input(shape=(sd_shape,))
input_tensor_second_level = tf.keras.layers.Input(shape=(sl_shape,)
    )
input_tensor_top_level = tf.keras.layers.Input(shape=(tl_shape,))
input_tensor_subdirectory = tf.keras.layers.Input(shape=(
    subdir_shape,))

input_tensor = tf.keras.layers.concatenate([input_tensor_subdomain,
     input_tensor_second_level, input_tensor_top_level,
    input_tensor_subdirectory])

hidden_1 = tf.keras.layers.Dense(64, activation=tf.nn.relu)(
    input_tensor)
hidden_2 = tf.keras.layers.Dense(32, activation=tf.nn.relu)(
    hidden_1)
hidden_3 = tf.keras.layers.Dense(16, activation=tf.nn.relu)(
    hidden_2)
hidden_4 = tf.keras.layers.Dense(8, activation=tf.nn.relu)(hidden_3
    )
output_layer = tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)(
    hidden_4)

model = tf.keras.models.Model(inputs=[input_tensor_subdomain,
    input_tensor_second_level, input_tensor_top_level,
    input_tensor_subdirectory], outputs=output_layer)
```

```
16
17 model.compile(optimizer='rmsprop', loss=tf.keras.losses.
       MeanSquaredError(), metrics=['accuracy'])
```

Listing 5.7: A complete DNN Implementation

In the `model.compile` function, inputs are given to change the way the model completes it's training and testing. These are adjusted to produce the most effective results as follows: [continue this when more details about results]

5.4 Experiment Three - Hybrid Model

Experiment three's approach involves extracting additional features, which has been explored in detail earlier, and using these extracted features as the input to a subset model of Do et al.'s [9] highly accurate DNN - in an attempt to extract a high-quality feature. This additional feature extraction would be performed on an already trained model. The training and testing data for only the features extracted in this project's implementation would be extracted from the UCI Machine Learning Repository Phishing Dataset, and then the model's predicted labels of the extracted features would be fed as an additional input tensor into the main model similar to what is defined above. The complete implementation of this model is as follows:

```
1 from prepareUCIAdjustedData import prepare_data
2 import tensorflow as tf
3
4
5 training_features, training_labels, testing_features,
       testing_labels = prepare_data()
6
7 # defining input tensor sizes
8 input_tensor = tf.keras.layers.Input(shape=(7,))
9 hidden_1 = tf.keras.layers.Dense(64, activation=tf.nn.relu)(
       input_tensor)
```

```
hidden_2 = tf.keras.layers.Dense(32, activation=tf.nn.relu)(
    hidden_1)
hidden_3 = tf.keras.layers.Dense(16, activation=tf.nn.relu)(
    hidden_2)
hidden_4 = tf.keras.layers.Dense(8, activation=tf.nn.relu)(hidden_3
    )
output_layer = tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)(
    hidden_4)

model = tf.keras.models.Model(inputs=input_tensor, outputs=
    output_layer)

model.compile(optimizer=tf.keras.optimizers.SGD(), loss=tf.keras.
    losses.MeanSquaredError(), metrics=['accuracy'])

model.fit(training_features, training_labels, epochs=100,
    batch_size=10)

test_loss, test_acc = model.evaluate(testing_features,
    testing_labels, batch_size=10)

print("Test accuracy:", test_acc)
```

Listing 5.8: Hybrid Model Subset DNN Implementation

As the model was trained, however, the accuracy of the model was discovered to be exceptionally low- 30% after 100 epochs. No amount of optimisation was able to increase this to acceptable bounds and as such it was determined that the subset dataset extracted from the UCI Machine Learning Repository was insufficient to exclusively classify the extracted features as phishing or benign. As such the hybrid approach was deemed infeasible for accurate classification of these features without additional features which would not be able to be extracted from any given URL while still conforming to the limitations on this implementation. Therefore, the extracted features were used in the model as a separate input and given simply as

an tensor of 7 digits as given in the data processing stage for experiment three.

As such the input tensor for experiment three is as follows:

```
# defining input tensor sizes
input_tensor_subdomain = tf.keras.layers.Input(shape=(sd_shape,))
input_tensor_second_level = tf.keras.layers.Input(shape=(sl_shape,)
    )
input_tensor_top_level = tf.keras.layers.Input(shape=(tl_shape,))
input_tensor_subdirectory = tf.keras.layers.Input(shape=(
    subdir_shape,))
input_tensor_additional = tf.keras.layers.Input(shape=(add_shape,))

input_tensor = tf.keras.layers.concatenate([input_tensor_subdomain,
     input_tensor_second_level, input_tensor_top_level,
    input_tensor_subdirectory, input_tensor_additional])
```

Listing 5.9: Experiment Three Input Tensor Implementation

Chapter 6

Results

This chapter will discuss the results acquired when running the models as defined above with varying datasets and in as close to identical conditions as possible. This is to make comparisons between the models as accurate as possible.

Each model has the same number of identical hidden layers, and have an identical output layer which is a single node with a sigmoid activation function. The models differ in their input layer, which is different for the shapes of inputs which each model is expected to receive. Each model will operate under the same dataset with the same split of training and testing data (70:30)- however this data will be in a randomised order each time (so it will not be the same data in training and testing datasets. Each model was run for 100 epochs (training runs), with a batch size of 32.

All models were compiled with the Adam optimiser with a learning rate of 0.001, and the loss function for each of the models was binary cross-entropy. This is because such optimisers and loss functions are suited to the type of functionality which this specific model requires. Adam is used because it is a popular reliable optimiser which is well-suited to large datasets. Binary cross-entropy is used because it is well-suited to binary classification problems such as the one this project is attempting to solve. Once the model has completed its training, it will be tested on the remaining 30% of data and that result will be used to generate the metrics. Training data will also

be made available. Separate statistical measures will be extracted and presented from the testing data. These statistical measures will first be defined:

6.1 Metrics

The most common method by which metrics are extracted from a DL model is through a confusion matrix. These contain four basic measures, True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). It is implemented in figure 6.1 and shall be implemented for each of the DNN models created for this project.

		Predicted Label	
		Phishing	Benign
Actual Label	Phishing	TP	FN
	Benign	FP	TN

Figure 6.1: A Confusion Matrix

True Positive (TP) A true positive value counts the number of instances where the model has correctly positively classified an input, in this case that would be the number of times the model correctly classifies a phishing website as phishing.

False Positive (FP) A false positive value counts the number of instances where the model has classified an input as positive but when compared to the true value has been found to be incorrect. In this case, that would be the number of times a model incorrectly classifies a benign website as phishing.

True Negative (TN) A true negative value counts the number of instances where the model has correctly negatively classified an input, in this case that would

be the number of times the model correctly classifies a benign website as benign.

False Negative (FN) A false negative value counts the number of instances where the model has classified an input as negative but when compared to the true value has been found to be incorrect. In this case, that would be the number of times a model incorrectly classifies a phishing website as benign.

There are a number of statistical measures which can be extracted from these values. These statistical features are able to characterise the performance of the model. These are:

Precision (PR) Precision is the number of correctly predicted positive inputs as a proportion of the number of positive predictions. This is able to indicate the confidence a model has in its positive predictions, so in this case how trustworthy a 'phishing' detection might be. It is defined as:

$$PR = \frac{TP}{TP + FP} \tag{6.1}$$

False Positive Rate (FPR) The false positive rate is the proportion of false positive classifications as compared to the total number of predicted negative samples - in this case the number of benign inputs falsely classified as a phishing website over the total number of benign samples. It is defined as:

$$FPR = \frac{FP}{FP + TN} \tag{6.2}$$

False Negative Rate (FNR) The false negative rate is the proportion of false negative classifications as a proportion of the total number of predicted positive samples. In this case it would be the number phishing inputs incorrectly classified as benign as a proportion of the total number of phishing samples. It is defined as :

$$FNR = \frac{FN}{FN + TP} \tag{6.3}$$

Sensitivity (SEN) Sensitivity is the percentage of correct positive predictions as a proportion of the total number of predicted positive samples. In this case, it would be the number of correctly classified phishing URLs as a proportion of the total number of URLs classified as phishing by the model. It is defined as:

$$SEN = \frac{TP}{TP + FN} \tag{6.4}$$

Accuracy (ACC) Possibly the most popular metric, it gives the proportion of correctly classified inputs as compared to the number of total inputs. In this case, it shows how efficiently a model is capable of accurately classifying a phishing or benign input. It is defined as:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.5}$$

6.2 Dataset 1

The first dataset which the models are processed against is the dataset used for basic testing and creation of the model as discussed in Chapter 5 - Implementation, which was synthesised by Marchal et al. (2014) [18]. To the author's knowledge there are no other DNN models which use this dataset directly, as such, any comparisons made will be made through the statistics alone. The processed dataset consists of 47,902 phishing URLs and 48,009 benign URLs. Which is split at a 70:30 ratio of training and testing data to 67,138 training URLs and 28,773 testing URLs.

6.2.1 Training

Each model achieved a higher accuracy during training than in testing. This could be a result of a number of factors, firstly, it's possible that some slight overfitting was taking place (though measures were taken to prevent this from getting too bad, for example limiting the Epochs to 100). This would result in a model which is very capable of classifying its own training data but very poor at classifying any other,

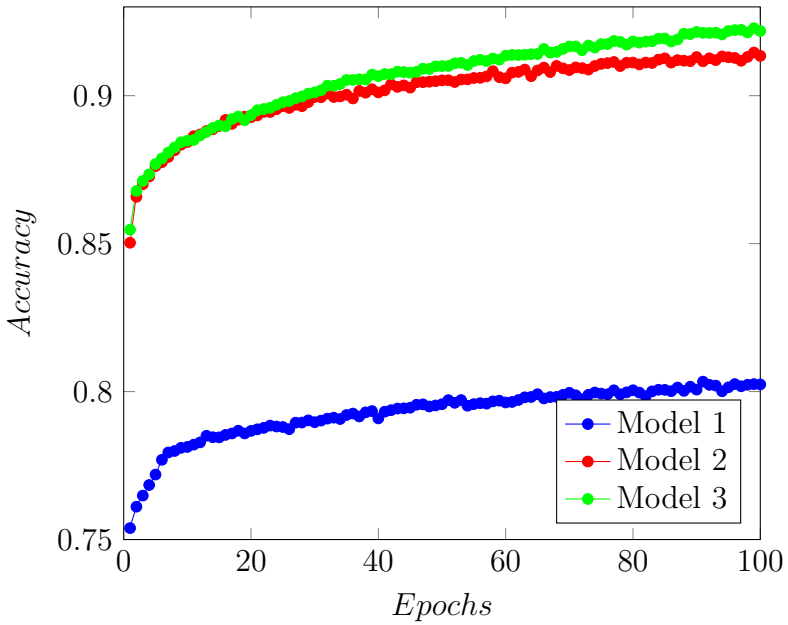


Figure 6.2: Dataset1, all model training accuracy over 100 epochs

previously unseen, data. Furthermore, due to the data shuffling methods used to ensure data heterogeneity, it's possible that there was an imbalance in data causing the training data to be a different distribution of phishing URLs to benign URLs - causing inaccuracies later. A sharp increase within the first 20 epochs followed by a decreasing positive gradient is followed by all graphs. It appears that as the models increase in complexity, the rate at which the positive gradients decrease is lower, that is to say model 3 improves faster than model 2 and will improve for longer. The problem with allowing the models to run for as many epochs as possible would be preventing overfitting from making the models inaccurate with testing data. Models two and three have significantly better accuracy than model 1 throughout training.

6.2.2 Performance

Below are the confusion matrices for each of the models, as well as the model statistics:

		Predicted	
		Phishing	Benign
Actual	Phishing	10081	4293
	Benign	1567	12832

Figure 6.3: Dataset 1, Model 1, Confusion Matrix

		Predicted	
		Phishing	Benign
Actual	Phishing	12467	1926
	Benign	1088	13292

Figure 6.4: Dataset 1, Model 2, Confusion Matrix

		Predicted	
		Phishing	Benign
Actual	Phishing	12678	1284
	Benign	1672	13139

Figure 6.5: Dataset 1, Model 3, Confusion Matrix

	Model 1	Model 2	Model 3
Precision	86.5%	86.6%	88.3%
False Positive Rate	10.9%	7.6%	11.3%
False Negative Rate	29.9%	13.4%	9.2%
Sensitivity	70.1%	86.6%	90.8%
Accuracy	79.6%	89.5%	89.7%

Table 6.1: Dataset 1 All Model Statistics

Throughout each of the models, the accuracy increases between each additional approach. However, the main increases in statistics are between the first and second models. The second model's sensitivity and accuracy increase by significant margins, and the false positive rate decreases significantly. The different between model two and three is minor and the false positive rate even increases.

The difference in the statistics between Models two and three are as a result of the additional features extracted. The additional features are, on their own, not significant enough to classify a URL as phishing or benign as discovered when attempting to create a hybrid model training using the UCI Machine Learning Repository dataset. The features are, however, able to improve the performance of a model, and increase its potential for further performance as discussed above. It can be hypothesised that additional features extracted here would be able further increase the performance of the model. Furthermore, model three has a higher false-positive rate when compared to model two, as all else is equal this could be due to threat actors manipulating extracted features to ensure that their URLs aren't caught- or benign URLs fulfilling conditions which are normally attributed to phishing URLs. Again, additional features extracted here would likely reduce the false positive rate.

6.3 Dataset 2

The following dataset, created by Ariyadasa et al. (2021) [5] consists of 50,000 URLs taken from legitimate websites, and 30,000 URLs taken from phishing websites. It was chosen as it provides a very high number of modern URLs- thus testing the model against threat actor techniques which could be up to 7 years newer than the dataset used previously. The models performed similarly to how the models performed in the previous models. Model 1 performs noticeably worse on modern threat actor techniques which may demonstrate the requirement for additional features. Furthermore the distinction that models two and three have between each other are

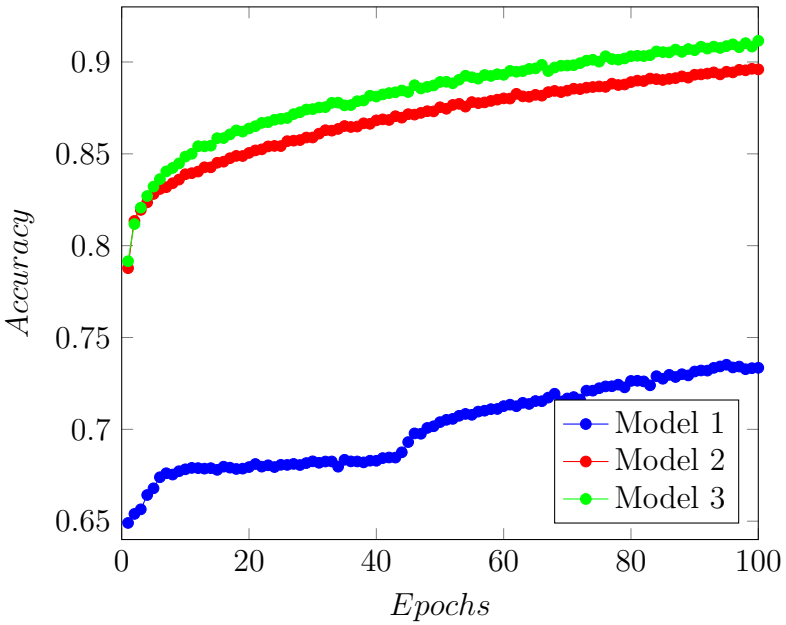


Figure 6.6: Dataset 2, all model training accuracy over 100 epochs

more pronounced which could be an indicator that the additional extracted features have a more pronounced impact on modern threat actor techniques.

		Predicted	
		Phishing	Benign
Actual	Phishing	3274	5643
	Benign	990	14089

Figure 6.7: Dataset 2, Model 1, Confusion Matrix

Model 1 makes noticeably fewer positive predictions than any other model in the dataset and proportionally fewer positive predictions than any of the models thus far. Instead the model is heavily over-predicting negatively, which affects its true negative value. Models two and three have similar prediction numbers, with model two more accurately predicting in all sides.

		Predicted	
		Phishing	Benign
Actual	Phishing	7121	1830
	Benign	1884	13161

Figure 6.8: Dataset 2, Model 2, Confusion Matrix

		Predicted	
		Phishing	Benign
Actual	Phishing	7264	1803
	Benign	1573	13356

Figure 6.9: Dataset 2, Model 3, Confusion Matrix

	Model 1	Model 2	Model 3
Precision	76.8%	79.1%	82.2%
False Positive Rate	6.6%	12.5%	10.5%
False Negative Rate	63.3%	20.4%	19.9%
Sensitivity	36.7%	79.6%	80.1%
Accuracy	72.4%	84.5%	85.9%

Table 6.2: Dataset 2 All Model Statistics

6.4 Dataset 3

Dataset three, created by Hannousse et al. (2021) [16] was selected in order to test the models under more challenging conditions. The dataset consists of 11430 URLs, in equal parts phishing and legitimate. The dataset consists of considerably fewer URLs than in previous approaches.

The training performance is noticeably less consistent than previous models as a result of the fewer training data-points. Furthermore, a similar behaviour- though to a lesser extent- is exhibited by model one on this data set when compared with the previous dataset. That is to say, model one over-predicts benign, at a significant cost to the model's accuracy. This could be, again, as a result of the newer threat actor techniques which are used in more modern datasets. Models two and three again perform comparably, with model three improving marginally by taking advantage of the additionally extracted features.

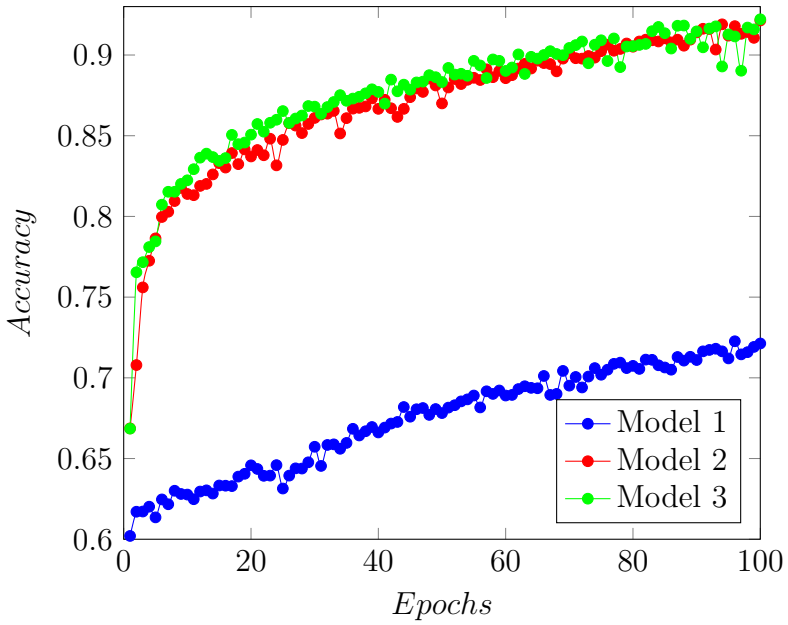


Figure 6.10: Dataset 3, all model training accuracy over 100 epochs

		Predicted	
		Phishing	Benign
Actual	Phishing	825	849
	Benign	263	1492

Figure 6.11: Dataset 3, Model 1, Confusion Matrix

		Predicted	
		Phishing	Benign
Actual	Phishing	1402	311
	Benign	297	1419

Figure 6.12: Dataset 3, Model 2, Confusion Matrix

		Predicted	
		Phishing	Benign
Actual	Phishing	1414	317
	Benign	276	1422

Figure 6.13: Dataset 3, Model 3, Confusion Matrix

	Model 1	Model 2	Model 3
Precision	75.8%	82.3%	82.7%
False Positive Rate	15.0%	17.3%	16.3%
False Negative Rate	50.7%	18.2%	18.3%
Sensitivity	49.3%	81.8%	81.7%
Accuracy	67.6%	82.5%	83.7%

Table 6.3: Dataset 3 All model Statistics

6.5 Discussion

The different models perform similarly to each other in most cases. Notably, model three achieves the highest overall performance in the majority of cases however struggles with False Positive rate. However, in all datasets it exceeds model one's false positive rate. The behaviour of model one to over-predict benign data points will significantly affect the model's overall false positive rate. When taking this into account model three outperforms model two in false positive rate on both dataset two and three - the more modern datasets. It can be hypothesised, therefore, that model three is more adept at detecting benign data points in modern datasets than model two because the additionally extracted features from modern URLs are effective at assisting in classification in these datasets. In dataset three, each of the models has a higher false positive rate, this is likely because of the reduced amount of training data available to the models.

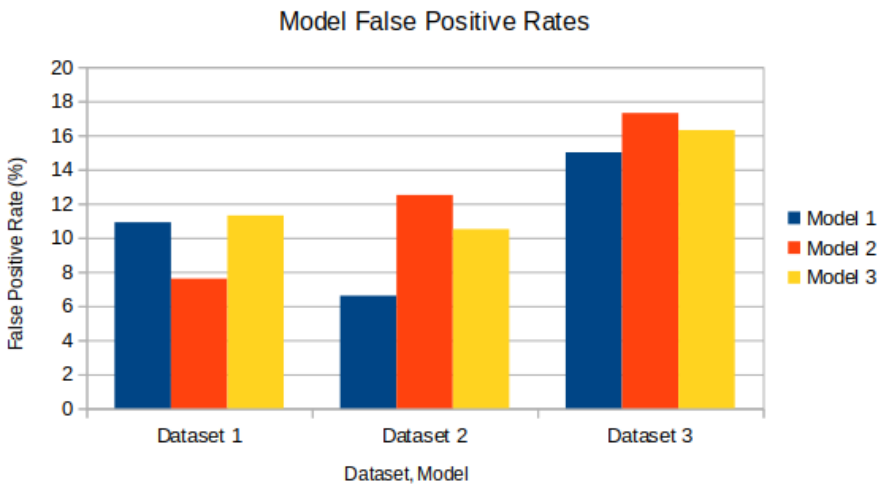


Figure 6.14: False Positives Across All Models and Datasets

The false negative rates pictured below clearly demonstrate the behaviour observed by model one across all datasets. That is that the model incorrectly predicts large portions of phishing URLs as benign - this results in very high false-negatives, which is not observed in either models two or three. Model three generally

outperforms model two in this metric as well - especially when the size and quality of a dataset increases. It can be observed from this graph in particular that dataset one is the highest quality dataset for predicting phishing URLs correctly.

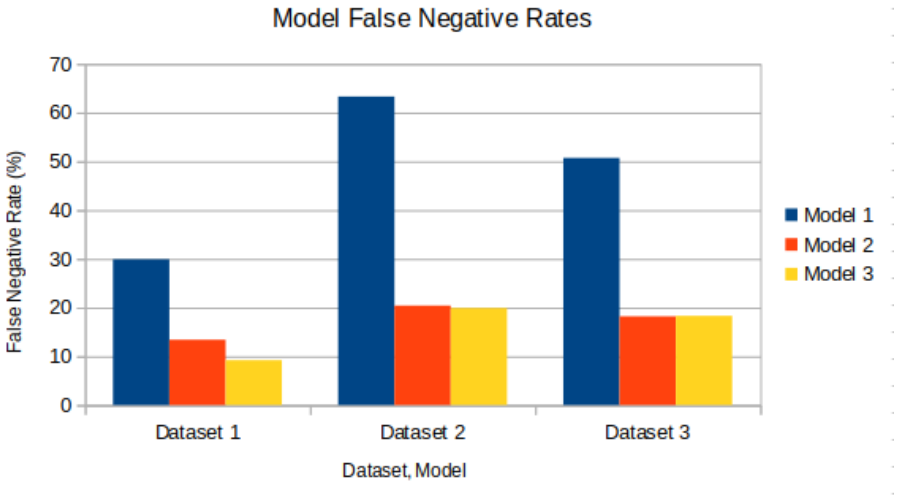


Figure 6.15: False Negatives Across All Models and Datasets

When compared with high quality approaches in the state-of-the-art, model three's performance is closest to what might be expected of an internet-enabled model. Of the DNNs identified in the background section as being used for phishing identification model three performs comparably - especially when in the highest quality dataset, dataset one. Figure 6.16 is a comparison between the Accuracy and Sensitivity values- which are the statistics of the DNN models given which were in common with this report- between Do et al. [9], Ali et al. [4] and Model 3.

Model 3 in dataset 1 performs better in both statistics than the DNN model proposed by Ali et al. [4], which uses seven internet-enabled feature extraction techniques and two URL-based features to classify URLs. Ali et al. make use of the UCI machine learning repository as well as external sources to acquire testing and training data. Do et al.'s model outperforms Model 3's accuracy and sensitivity, their model makes use of 30 features of which 7 are URL-based and 23 are internet-enabled.

Model 3's performance, while less effective than Do et al's highly effective DNN

Figure 6.16: Performance of literature DNN models against Model 3

model, is still capable of producing results which are promising for the future of URL-based detection methods. Realistically, further novel features need to be extracted from the URL, and more information might be needed- such as blacklists- order to more effectively classify URLs in a privacy-centric manor.

Chapter 7

Conclusion

7.1 Review of Project Aims

1. **Create deep learning models capable of detecting phishing websites using only URLs;**

   The creation of three deep learning models capable of classifying a given URL as phishing or legitimate by using only URLs was achieved and proven to be effective across three different datasets, even performing well given challenging training conditions. Three separate models using different feature extraction techniques and inputs were used and had varied results, however all were capable of high-performance phishing URL detection. None of the models used any information other than the URL or that which could be derived from the URL.

2. **Make use of exclusively URL-based feature extraction methods which are based on current state-of-the-art approaches;**

   The state-of-the-art was examined in Chapter 2 - Background and key feature extraction methods were identified and evaluated for use. These were clarified in Chapter 4 - Design to conform with the UCI machine learning repository dataset. These were then used as part of model 3's implementation to improve the performance of my models against various datasets.

3. **Evaluate the performance of newly created DL models against highly accurate industry-standard approaches.**

   The models proposed hold certain advantage over denunciation platforms with regards to the ability to stop zero-day phishing attacks with more effectiveness. Furthermore, the most promising models trained in this report are capable of performing at standards which are comparable to the state-of-the-art. The models created do, however, have a long way to go in terms of improvement if they are to compete with the best-performing internet-enabled models still.

7.2 Revisions to Design and Implementation

The models which were created conform almost exactly to the initially proposed designs with one exception. Model 3's hybrid approach was ineffective because the seven features which were extracted were deemed to be insignificant in terms of classification of a single URL, and using the seven features through a subset model of Do et al.'s approach [9] proved to be unfeasible due to the inaccuracy of the model. However, the model's alternative approach using the extracted features as a separate input was able to make noticeable improvements to the model's performance, as such this approach is ultimately deemed successful.

7.3 Future Work

Further analysis should to be made into the feasibility of privacy-based phishing detection techniques. In specifics, more features which can be extracted from the URL itself. This element of the proposed models has repeatedly been shown to be somewhat effective in its current state however holds potential to be a strong classifying factor given more extracted features.

Threat actor techniques are always shifting but mimicking remains a prominent approach because it exploits the main weakness in many secure systems which is the

humans who operate them. A feature could make use of a variable list of legitimate services and identify similar terms in input URLs in an attempt to catch mimicking attacks more effectively than current methods. Such a similarity approach could also be relevant for known phishing websites, as mentioned some threat actors will change single characters in their URLs to make them completely different to denunciation platforms. An approach identifying similar website names to phishing websites could be useful for mitigating this type of behaviour.

7.4 Closing Remarks

The work in this project acts as a proof-of-concept that privacy-centric approaches to human security are, while currently less effective than internet and content-based approaches, viable options which need further investigation. The importance of privacy on the internet cannot be overstated. In a world where user data is increasingly becoming the property of corporations and governments, private options should be available for anyone who chooses - current approaches abandon this philosophy. This risks privacy-centric approaches being widely considered as ineffective, which this project has provided evidence against. I believe this project plays a small part in a greater issue of privacy- but is evidence that with modern technology, privacy does not need to be abandoned.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. In *Proceedings of the Anti-Phishing Working Groups 2nd Annual ECrime Researchers Summit*, eCrime '07, page 60–69, New York, NY, USA, 2007. Association for Computing Machinery.

[3] Ahmed Aleroud and Lina Zhou. Phishing environments, techniques, and countermeasures: A survey. *Computers and Security*, 68:160–196, 2017.

[4] Waleed Ali and Adel A. Ahmed. Hybrid intelligent phishing website prediction using deep neural networks with genetic algorithm-based feature selection and weighting. *IET Information Security*, 13(6):659–669, 2019.

[5] Subhash Ariyadasa, Shantha Fernando, and Subha Fernando. Phishing websites dataset. doi: 10.17632/n96ncsr5g4.1, 2021.

[6] Amazon AWS. Alexa's top one million websites, 2023. `https://s3.amazonaws.com/alexa-static/top-1m.csv.zip`.

[7] François Chollet et al. Keras. `https://keras.io`, 2015.

[8] Carlo Marcelo Revoredo da Silva, Eduardo Luzeiro Feitosa, and Vinicius Cardoso Garcia. Heuristic-based strategy for phishing prediction: A survey of url-based approach. *Computers and Security*, 88:101613, 2020.

[9] Nguyet Quang Do, Ali Selamat, Ondrej Krejcar, Takeru Yokoi, and Hamido Fujita. Phishing webpage classification via deep learning-based algorithms: An empirical study. *Applied sciences*, 11(19):9210, 2021.

[10] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[11] Maddy Ell. Cyber security breaches survey, 2022.

[12] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656, 2007.

[13] Google. Google safebrowsing. `https://safebrowsing.google.com/`, 2006.

[14] Google. Google colaboratory. `https://colab.research.google.com/`, 2017.

[15] R. Gowtham and Ilango Krishnamurthi. A comprehensive and efficacious architecture for detecting phishing webpages. *Computers and Security*, 40:23–37, 2014.

[16] Abdelhakim Hannousse and Salima Yahiouche. Web page phishing detection. doi: 10.17632/c2gw7fy2j4.3, 2021.

[17] Kaspersky. What is scareware? definition and explanation, 2023.

[18] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel. Phishstorm: Detecting phishing with streaming analytics. *IEEE Transactions on Network and Service Management*, 11(4):458–471, 2014.

[19] Rami M. Mohammad, Fadi Thabtah, and Lee McCluskey. Intelligent rule-based phishing websites classification. *IET Information Security*, 8(3):153–160, 2014.

[20] Rami M. Mohammad, Fadi Thabtah, and Lee McCluskey. Tutorial and critical analysis of phishing websites methods. *Computer Science Review*, 17:1–24, 2015.

[21] Tyler Moore and Richard Clayton. Examining the impact of website take-down on phishing. In *Proceedings of the Anti-Phishing Working Groups 2nd Annual ECrime Researchers Summit*, eCrime '07, page 1–13, New York, NY, USA, 2007. Association for Computing Machinery.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[23] Routhu Srinivasa Rao and Alwyn Roshan Pais. Jail-phish: An improved search engine based phishing detection system. *Computers and Security*, 83:246–267, 2019.

[24] NCRLY Teraguchi and John C Mitchell. Client-side defense against web-based identity theft. *Computer Science Department, Stanford University. Available: http://crypto. stanford. edu/SpoofGuard/webspoof. pdf*, 2004.

[25] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-scale automatic classification of phishing pages. In *NDSS '10*, 2010.

[26] Yue Zhang, Jason I. Hong, and Lorrie F. Cranor. Cantina: A content-based approach to detecting phishing web sites. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 639–648, New York, NY, USA, 2007. Association for Computing Machinery.

Appendix A

Initial Proposal

Final Year Project Proposal - Sam Evans 37973983

Phishing website detection using deep learning techniques

Abstract

The project proposed in this paper aims to develop an AI (Artificial Intelligence) agent using deep learning mechanisms which is capable of phishing website detection. This would become a tool that could be used to improve the efficacy of web-filtering tools to protect and inform end-users.
The project will identify appropriate inputs for an AI powered by DL (Deep Learning) techniques, while balancing accuracy and performance to ensure that the agent could be embedded in programs in a real-time scenario.
The project is expected to conclude with the development of a high-performance, accurate agent capable of providing either a percent chance of a website being malicious in nature- or simply classifying the website as malicious or safe in nature. It is expected that the agent would be able classify websites which are malicious in nature by way of aiming to make users mistake it for a legitimate website but with hidden functionality (for example, a website with URL amazpn.com which aims to masquerade as the legitimate amazon.com and lure users into giving away payment details[1]). It is also expected that the agent would be able to classify websites which are malicious by way of misleading users into downloading malware (for example, a website with URL nortonanti-safepcscan.io aiming to get users to download a piece of malicious software pretending to be a pc scanner only to be spyware or software which requires payment to use) however is able to classify the website in manor which is safe to the end user.

Introduction

Phishing attacks were the most common cyber attack in the UK in 2021, with 83% of businesses and 79% of charities reporting being the targets of phishing attacks [2]. Because phishing exploits human error, conventional anti-malware systems tend to be ineffective. Furthermore, due to the short-lived nature of many phishing websites (Kaspersky found that just 12.3% of phishing websites identified were active after 30 days [3]) creating databases of

potential phishing threats is generally ineffective as it would become outdated very quickly and would otherwise require continuous maintenance which would be expensive. AI-enabled solutions are recognised as having potential due to their ability to classify novel data based on old training data.
The proposed project would develop an agent trained using Deep Learning methods to accurately identify phishing websites from safe websites. The model will be trained by having the model classify websites which are new to it as malicious or safe based on data scanned from the URLs of the website. The training data will be acquired by taking websites previously identified as being phishing websites and commonly accessed websites and combining them in a uniform and random way.
This project proposal will go over the background of the project area itself, the project objectives, project methodology and analysis and overall workplan. All writing will be referenced.

Background

Current research shows that phishing websites can be detected within data sets with high accuracy, Hota et al. (2018) [4] show models can achieve accuracy up to 99.11%, but take time and cannot be seen as real-time. However, Hannousse et al. (2021) [5] point out problems in identifying accuracy of models through percentage as there is no benchmark dataset and accidental bias can easily be introduced into datasets to affect the accuracy.
Basit et al. (2021) [6] propose that there is a need for a high-performance, low-false positive and accurate model which can be enabled in applications such as web-browsers. Sameen et al. (2020) successfully developed a tool known as “PhishHaven”, a browser plug-in which uses an AI machine learning model to achieve a 98% accuracy. Sameen et al. state that their research could further be improved through the use of deep learning and unsupervised learning, This leaves a gap for my research to be completed by using deep learning on similar data-sets to see if unsupervised-learning based models are as effective in detection of phishing websites.

Project Objectives

The primary aim of this project is to develop an AI agent capable of distinguishing between a phishing website and a legitimate website by using deep learning techniques. In order to achieve this I have outlined the following objectives:

Final Year Project Proposal - Sam Evans 37973983

1) Acquisition, formatting and organisation of appropriate training data - both with legitimate websites and malicious ones and formatting them in a way which does not introduce bias to the agent's operations.
2) Training of an agent using deep learning techniques and using heuristics to a limited degree to develop an accurate and high performance agent through iterative development and analysis of results - adjusting training data where relevant
3) Some implementation of the agent in a program or plug-in whereby URLs can be tested in real-time to determine the agent's assessment of a novel website.

Project Methodology and Analysis

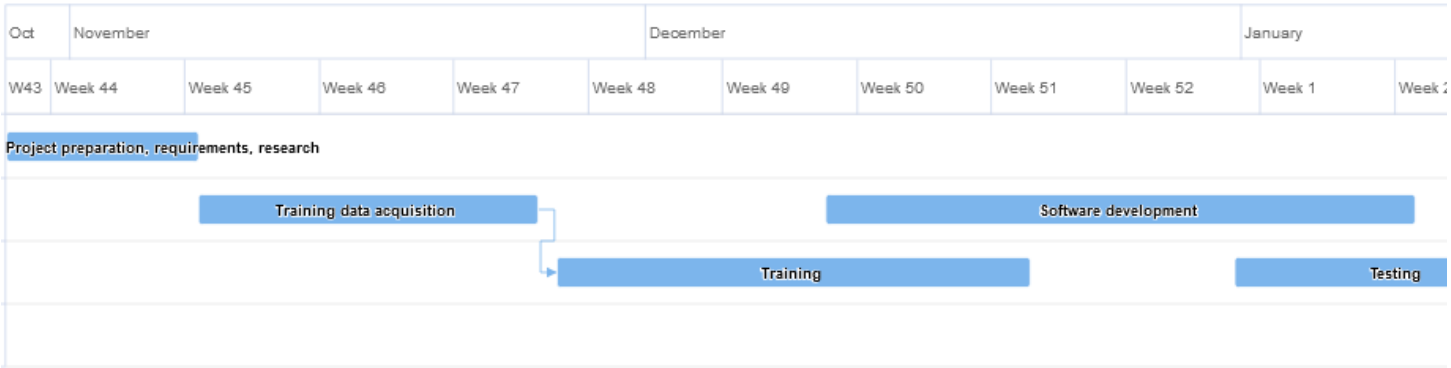
Training data will be acquired in a linear manner, both website URLs and website data will be extracted, extra care will be taken when extracting any information from known phishing websites as they are potentially dangerous to access and could contain malware.
The agent will be trained in an agile style, with iterative changes being made to the configuration of the training data and the agent in order to achieve the best results.
To ensure that the agent is accurate it will be thoroughly tested on novel data once it has been trained to the best of my abilities.
I will develop some kind of software or plug-in which will test URLs provided to it, this will be done following agile methodology and will be iterated upon given the changing needs that the agent may have (for example, different data is required for accurate results.) This software will be thoroughly tested to ensure that any machine running it will not suffer significantly in performance while the software is running. Furthermore, it is important to users that they do not see their privacy as being compromised by the system, as it will have the ability to read URLs and website data. This means that I will ensure as little user data as possible if not no user data is stored at all, and that no website data or URLs are stored longer than is absolutely necessary.
The final component of the project will be a detailed review of the software and final agent in order to evaluate my performance in this project, the performance and accuracy of the software as a whole and agent respectively will be taken most closely into account. It is possible that a usability study will be performed to identify how usable the system would be to a standard user- as the system itself would be used by almost exclusively non-tech savvy people in the real world.

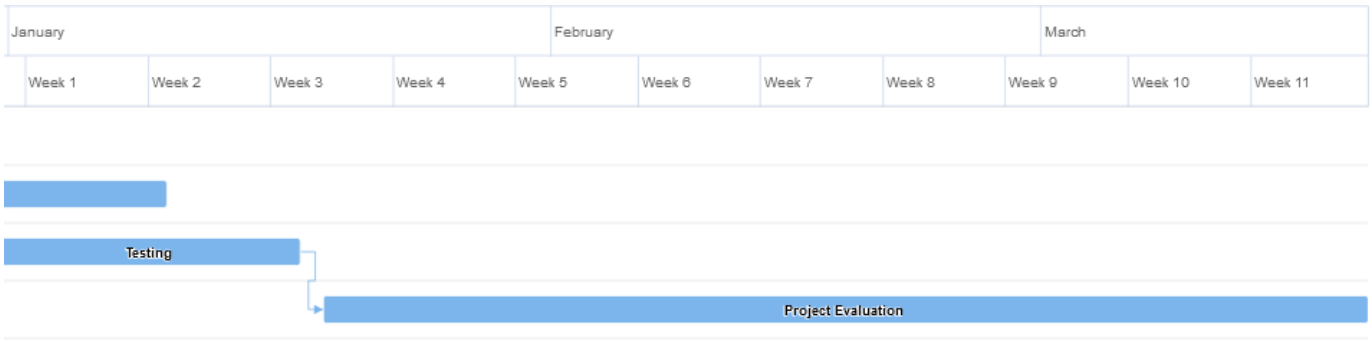
Final Year Project Proposal - Sam Evans 37973983

Project Workplan

The project will be completed from October 2022 through until March 2023. I have broken it down into different phases:

1) Project preparation, software requirements gathering, AI research: I will gather requirements for the final piece of software, as well as determining my most effective approach for developing my agent system. I estimate this will take 1-2 weeks.
2) Training data acquisition: I will gather training data from reliable sources for both legitimate websites and website data and malicious websites with their data in a safe way. I will then format all of the data and organise it to be processed by my training system. I estimate this will take 3-4 weeks.
3) Training: Partially in parallel to the previous phase, I will begin to develop my agent-training system, adjusting my requirements for data acquisition as I proceed through the development process. I estimate this will take around 3 weeks.
4) Software development: I will develop user-facing software which will process the completed agent and provide it with new information before presenting its output to the user. I will develop this after I am content with the agent system I have developed. I estimate that this will take 2 weeks
5) Testing: I will test all portions of the system and conduct systems usability tests on my software to ensure that they are usable by general public and non-tech-savvy individuals. I estimate this will take 2 weeks
6) Project Evaluation: I will evaluate the performance of my project as a whole and account for the usability study in my evaluation. Both the performance of the software (with the goal of creating a real-time system) and the accuracy of the software will be analysed. A report will be written on the project as a whole. I estimate this will take 6 weeks.





Bibliography

1) Types of phishing (pandasecurity, 2021)
https://www.pandasecurity.com/en/mediacenter/tips/types-of-phishing/
Accessed 16th October 2022
2) Cyber Security Breaches Survey 2021 (gov.uk, 2021):
https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2021/cyber-security-breaches-survey-2021
Accessed 16th October 2022
3) An analysis of the life cycle of phishing pages (Securelist, 2021)
https://securelist.com/phishing-page-life-cycle/105171/
Accessed 17th October 2022
4) Hota, H., Shrivas, A., & Hota, R. (2018). An ensemble model for detecting phishing attack with proposed remove-replace feature selection technique. Procedia Computer Science, 132, 900–907
Accessed 17th October 2022
5) Abdelhakim Hannousse, Salima Yahiouche (2021), Towards benchmark datasets for machine learning based website phishing detection: An experimental study, Engineering Applications of Artificial Intelligence, Volume 104, 2021, 104347, ISSN 0952-1976, https://doi.org/10.1016/j.engappai.2021.104347.
Accessed 18th October 2022

6) Basit, A., Zafar, M., Liu, X. *et al.* A comprehensive survey of AI-enabled phishing attacks detection techniques. *Telecommun Syst* **76**, 139–154 (2021). https://doi-org.ezproxy.lancs.ac.uk/10.1007/s11235-020-00733-2
7) M. Sameen, K. Han and S. O. Hwang, "PhishHaven—An Efficient Real-Time AI Phishing URLs Detection System," in *IEEE Access*, vol. 8, pp. 83425-83443, 2020, doi: 10.1109/ACCESS.2020.2991403.