Computer Science

COC251

F032072

Phishing Prevention Application

by

Shrey Shah

Supervisor: Magda Zajaczkowska

Department of Computer Science
Loughborough University

April/June 2025

Abstract

Phishing remains a persistent and damaging cybersecurity threat. It exploits human error to bypass increasingly sophisticated technical measures. This project aims to develop a layered approach, combining user education and robust detection to mitigate these effects. This project will deliver an extension that combines the natural language capabilities of a chatbot in order to educate, with a machine learning approach to detect threats. This report discusses the system design, implementation challenges and evaluation of its effectiveness. The project demonstrates the effectiveness of combining automated detection with human-centered education in mitigating phishing risks.

Acknowledgements

Many thanks to my supervisor, Magda Zajaczkowska, for her continued support throughout the duration of this project.

Contents

Chapter 1

Introduction

1.1 Project Description

The aim of this project is to create an AI-based phishing prevention extension. It will use a machine-learning model to analyse URLs and evaluate if they are malicious by using known phishing patterns. However, this approach alone is insufficient to properly protect against attacks as the security of the extension relies on the robustness of the model alone. A program could always incorrectly classify URLs, in this case the final responsibility lies on the user. As such, an AI-based chatbot will also be employed to educate users on common phishing tactics and other aspects to be cognizant of. Through the combination of filtering and training,a higher level of phishing prevention can be achieved.

Current phishing detection applications on the market purely block phishing attacks and highlight them to the user, however this method always relies purely on the robustness of the detection algorithm and no attention is directed towards the user side. Social engineering is used in 98% of cyberattacks, and as such there will be an exceedingly small amount, if any, of systems that can prevent direct action from users. Due to this, the implementation of a teaching system, which learns from user actions using AI is deemed necessary.

1.2 Aims and Objectives

The goal for this software is to create a lightweight, phishing detection software that can continuously educate the user. This would reduce the human weak link in cybersecurity and provide an efficient detection system to mitigate the threat of phishing to a greater extent.

The objectives for this project are to:

1. Perform thorough technological research to find the best methodology for this project.
2. Examine competitors and evaluate missing features and areas of improvement.
3. Create a suitable design plan for the final application.
4. Develop a suitable phishing detection system supported by research.
5. Explore the most suitable deployment method for the application.
6. Create a chatbot that is capable of answering phishing queries and educating users.
7. Accurately test the final application.
8. Explore future expansions and deployment options.

Chapter 2

2.1 Background

This project aims to mitigate the rise of phishing attacks over time, and as such there will be preliminary research into the landscape of current phishing attacks and detection systems that will be conducted throughout this section.

Social engineering is a type of cyber attack that involves manipulating users into personally compromising either their own, or their organisation's safety. In a report detailing the state of cybersecurity (ISACA, 2022) social engineering was shown to be the most common type of cyber attack and has the highest success rate.

Phishing is a type of social engineering attack that deceives users into sending critical information, such as passwords, money or downloading malware (ICO, no date). It entails sending fraudulent messages, usually through text or email, that encourage the user to follow links to fraudulent websites. The method relies on exploiting the weakest link in a cybersecurity system, the user (Sebastian and Kolluru, 2021). However, as further highlighted in this article, the weakest link could also be shown as an organisation's, or lack of, cybersecurity culture and awareness training. This is what this project aims to tackle, strengthening the weak link whilst bolstering the detection systems already in place.

A common type of phishing is known as bulk phishing, which involves sending emails to millions of users, needing only to deceive a few. These emails will typically contain a malicious link that will take users to a fake website, allowing them to steal details such as passwords and credit card information. Another common type of phishing is known as spear phishing, which targets a specific individual which may be known to have privileged access to information the attacker may want. It consists of researching the target by looking at their digital footprint, perhaps from social media websites or otherwise, and creating an attack that is individualised.

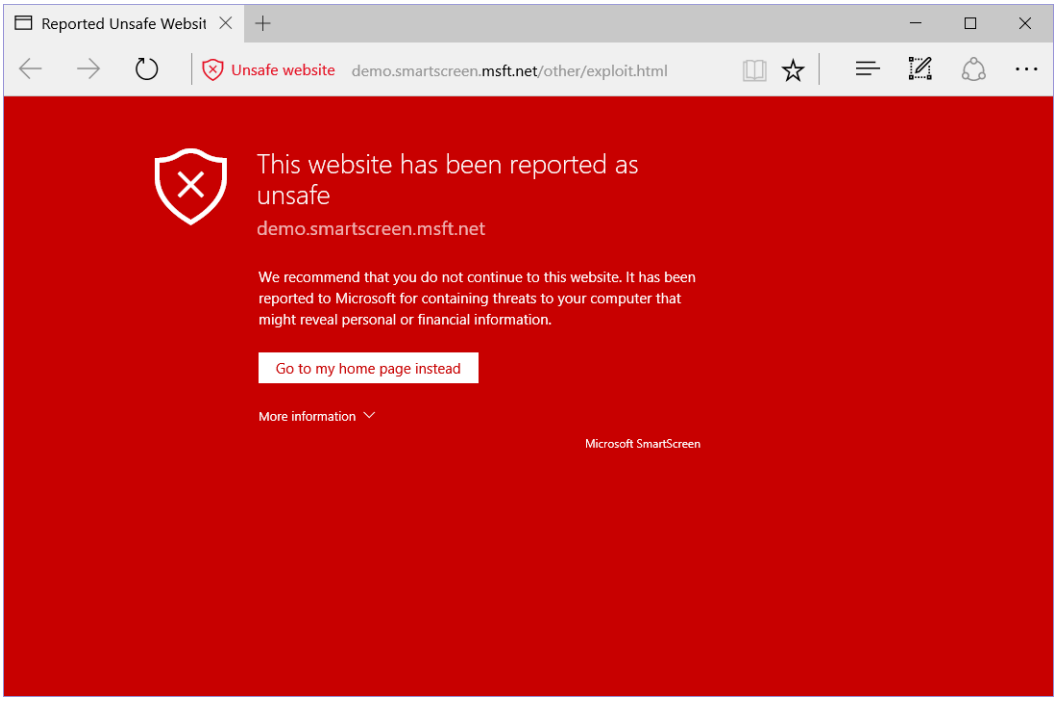
The reason for the frequency of phishing attacks lies in their success. In 2021, the average click rate for a standard phishing campaign was 17.8%, with that number climbing to 53.2% for attacks that were more targeted and sophisticated *(Imber, 2025)*. This statistic alone shows the importance of anti-phishing software and 'digital hygiene' by end-users who are subject to these attacks, however researchers at Trend Micro discovered that over 90% of attacks started with spear phishing emails *(Trendmicro.com, 2025)*. Further insight from Deloitte states that users are failing to adhere to 'digital hygiene' principles and are increasingly putting security at risk *(Deloitte, 2020).*

This project will aim to mitigate the two factors that allow phishing attacks to have this level of success. Whilst detection is an important factor, it alone does not provide an effective solution. As seen before, 17.8% of phishing attacks succeed despite preventative measures, a success rate which nearly triples if the attack is targeted. As such it is just as important to instil strong cyber safety principles amongst end users, which can be achieved by training them practically, coaching provided in real time using LLMs that allow users to question and learn.

2.2 Gap Analysis

Numerous phishing detection systems already exist and are widely used. Within this section, these solutions will be examined, weighing positives and negatives to examine if any gaps are found and what advantages this project offers.

2.2.1 Windows Defender



A common anti-phishing solution is Microsoft Defender. It allows for phishing protection by screening emails and boasts high levels of learning due to access to Microsoft's security stack. In addition to this it works in tandem with other microsoft security products to accurately respond to many threats the user may face.

Advantages:

- Easy to integrate for Windows users.
- Robust machine learning model trained on Microsoft data
- Free software

Disadvantages:

- No user training options, relying on the strength of the detection system alone
- Cannot be used for users or organisations that do not utilise Windows

2.2.2 Barracuda Email Protection

Barracuda Email Protection offers a comprehensive amount of protection in regards to emails, with examples being phishing, spam, malware and ransomware. A cloud-based solution which analyses incoming email traffic and detects threats. Barracuda also boasts an AI model that learns how an organisation communicates and uses those patterns to determine if new emails are malicious. This application also offers employee training resources.

Advantages:

- Comprehensive protection for email attacks
- Offers training services
- Detects zero hour phishing attacks

Disadvantages:

- Training services are generalised
- Cost of product is high, being difficult for a user or small-level organisations to implement
- Training services incur an extra cost, which could dissuade some organisations from utilising them
- Restricted to email attacks

2.2.3 Ironscales

Ironscales utilises AI and machine learning techniques in order to gauge threats. It utilises AI modules that process natural language and evaluate incoming messages for abnormal behaviour. They also tailor their approach per user, establishing normal inbox activity per user and determining if new messages disrupt this. Employee training is also offered, resulting in a score that the organisation can use to determine how to proceed effectively to cover gaps in training.

Advantages:

- Robust detection system utilising multiple layers
- Extensive employee training and feedback

Disadvantages:

- Cost of product may alienate some organisations and cannot be used at a user level
- Training modules lack reminder systems
- Cannot flag false positives

2.2.4 Conclusion

Upon examining current solutions, many share the same issues and benefits. Many programs do not offer user-level protection and instead are only applicable to organisations. In addition to this, whilst some options offer training in phishing detection, the burden lays on the organisation to provide training at effective intervals, and if this is omitted then no reliable results will be shown. It was decided that there was a place for this project, as a software for users to utilise for free whilst also gaining cybersecurity awareness from repeated interactions with the chatbot.

2.3 Technological Research

This section will conduct research into certain technological aspects of the project, this would allow an informed approach to requirements and design.

2.3.1 Phishing Detection Systems

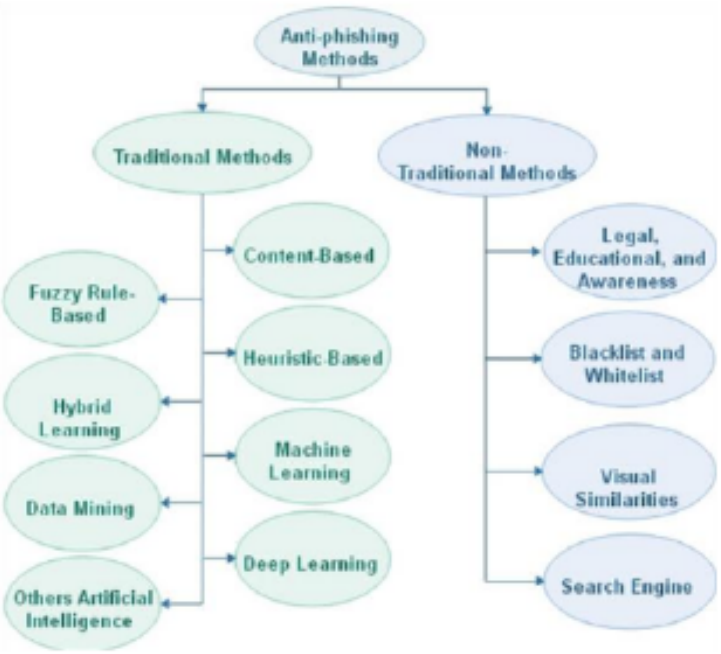


Figure 1 : A taxonomy of anti-phishing detection (Alanezi, 2021)

In this section, the usage and success rates of these methods will be explored and compared in order to select a suitable approach for this project.

Non-traditional methods include increasing user awareness for cyber-security threats, this is a core component of this project and the reasoning behind the chatbot implementation. This is considered a proactive approach to cybersecurity and does not rely on incoming attacks. With humans often being identified as the weakest link in a cybersecurity program, raising awareness with a ‘hands-on’ approach is deemed to be one of the best methods for adult learners (Davies, 2021). The chatbot in this project aims to integrate seamlessly with the user’s workflow, and allow for the user to continuously learn through interacting with it on detected phishing sites and compound knowledge.

Other non-traditional methods include blacklisting and whitelisting, a simple approach as lists of known phishing websites increase continuously. However, it is unsuitable as a singular approach, not being able to detect zero-hour attacks and often being reactive, this functionality could be utilised to add a layered approach to security.

A common traditional method is the usage of heuristic-based technologies, this method involves extracted features from the target website and scanning several attributes such as the DNS, website traffic and digital certificates. Whilst this application can detect zero-hour attacks (Jain & Gupta, 2017), and has shown to be well implemented, the ability to create a robust model would be time-consuming and outside of the current scope of this project, requiring singular focus on this aspect alone.

An often used, modern approach for phishing detection is the usage of machine learning models (Safi & Singh, 2023). Common identifiers are collected in a dataset and classifiers are trained on these identifiers. As the number of phishing attacks grow, datasets also inevitably increase and allow for greater precision. According to the previous study, the machine learning classifiers achieved greater than 99% accuracy, in turn being the most effective method of detection. As a result, this method would be very suitable for this project, as several datasets such as Phishtank and Openphish are readily available to train classifiers on.

Deep learning methods are recently becoming prominent, with speculation that they should perform better than machine learning methods (Basit et al., 2020). Whilst it was considered for this project, it was ultimately deemed unnecessary due to the current scope of this project and the need for interpretable results.

For the detection aspect of this project, it is vital that a robust method is selected. As such, machine learning would be the most suitable, it provides high detection rates and readily available datasets that are apt for training. This project must maintain a high rate of detection alongside its training to truly minimise phishing attacks as much as possible.

2.3.2 Deployment Platform

Many current solutions offer an application-based implementation for phishing detection, these work as an additional layer on top of the user's browser activity. For example, Ironscales provides a mailbox-level solution that works in tandem with the user's activity. Whilst this can offer greater freedom for development, it ultimately can stagnate the user experience due to additional processing requirements. On the other hand, an extension-based solution is integrated into the user flow, allowing for a low-friction integration that ultimately lessens the load on the user to utilise and engage with the application, potentially making real-world adoption more feasible.

Continuing on this, with the current focus of this project being on phishing websites, it is vital that any detection system can act in real-time and intercept any attacks. Browser extensions offer privileged access to user browsing in real time, being able to analyse the website's URL and DOM elements as they appear. This can be achieved with well-documented and readily available APIs that have been developed for extension environments, simplifying the process.

Taking these factors into account, an extension-based solution was decided upon for this project, offering greater harmony with the user's browsing flow, whilst also providing practical

tools to enable development and meet the required result, an integrated phishing detection and teaching system.

2.4 Requirements

2.4.1 Extension

System Requirement a1:
The user should be able to access all features through the UI
Description: The final software should have a functioning UI
Priority: Must
Rationale: Allows the user to utilise the software effectively and easily
Fit Criteria: Determine if a suitable UI is implemented and all features are accessible

System Requirement a2:
The extension should automatically scan the user's current page
Description: When the user loads a new page, the URL should be processed immediately
Priority: Must
Rationale: Allows for real-time phishing detection
Fit Criteria: Log scans and that they occur for newly loaded pages

System Requirement a3:
The extension should have a whitelist for false positives
Description: A whitelisted site will have no predictions run on it
Priority: Must
Rationale: Allows for false-positive logging and improves user quality-of-life
Fit Criteria: Sites can be added to a whitelist and cease having predictions run on them

System Requirement a4:
Appropriate phishing alerts should be shown to the user
Description: For a predicted phishing website, an alert will be shown to the user
Priority: Must
Rationale: Displays warning to user to notify about potential threat
Fit Criteria: Ensure alerts are displayed on potentially malicious pages

2.4.2 Detecting Threats

System Requirement b1:
Machine learning model will predict website status
Description: The model will predict the status of a website
Priority: Must
Rationale: Detection system needs to be functional
Fit Criteria: Log results of predictions and ensure they are appropriate

System Requirement b2 :
Extract features from URL
Description: All features for machine learning should be extracted upon visiting a new page
Priority: Must
Rationale: Feature extraction is necessary to receive model response

Fit Criteria: Log results of extraction and ensure they match standard query

System Requirement b3 :

Return interpretable values for prediction

Description: Feature importance values should be returned to explain to user the rationale behind a decision

Priority: Could

Rationale: Allows for greater cohesiveness between chatbot and extension, allowing for greater explainability.

Fit Criteria: Ensure feature values are transferred to the chatbot

2.4.3 AI Chatbot

System Requirement c1 :

Fine-tune bot to respond to cybersecurity situations

Description: Bot should be adjusted to be able to respond to relevant cybersecurity questions

Priority: Must

Rationale: The training bot should be able to respond to user questions accurately

Fit Criteria: Test bot by asking questions and gauging validity of answers

System Requirement c2 :

Bot should store memory of previous conversations with user

Description: Bot should remember previous conversations and build upon them

Priority: Should

Rationale: Using previous user information the bot should customise the advice given

Fit Criteria: Test using several accounts if memory is retained and how it affects future answers

System Requirement c3 :

Chatbot should only respond to cybersecurity related queries

Description: Questions should be answered appropriately in a cybersecurity sense

Priority: Must

Rationale: Bot should stay on topic and only provide relevant advice

Fit Criteria: Ask unrelated questions and related questions, judge if responses are appropriate

System Requirement c4 :

Bot should be able to speak during real-time and explain if a website is phishing

Description: If an websitel is a phishing threat, it will be flagged and the bot will explain why

Priority: Could

Rationale: Allows for the user to learn more effectively what to be aware of in phishing attacks

Fit Criteria: Test with phishing websites if they are detected and if the bot can accurately explain why

System Requirement c5 :
Bot should be able to give training exercises to user
Description: Training exercises should be given to user on request
Priority: Could
Rationale: Allows for more formal training
Fit Criteria: Design courses, upon user request test if they are successfully delivered

2.4.4 Non-functional Requirements

System Requirement d1 :
Communication should be done over secure channels
Description: API requests should be done over HTTPS channels
Priority: Must
Rationale: Allows for greater security and prevents man in the middle attacks
Fit Criteria: Ensure all API calls are completed over HTTPS

System Requirement d2 :
System components should be designed modularly
Description: Components such as the model, chatbot and frontend should be developed modularly
Priority: Should
Rationale: Allows for greater maintainability and allows for future upgrades to be implemented easily.
Fit Criteria: Design components separately and interconnect them

Chapter 3

3.1 Machine Learning Design

Due to the information gleaned from chapter 2, this project will use a traditional, supervised machine learning model for the purposes of detection. It will be trained on features derived from the domain and URL.

To select an appropriate model to deploy, extensive research into different models and their efficacy will be undertaken. The ever-changing nature of phishing attacks may cause a year-old model, that performs excellently on its trained data, to no longer function to an acceptable standard. This highlights the need for a model that is not only strong on the data on which it is trained, but also performs well on unseen data. To assess this accurately, the method of external validation was employed, two datasets were selected by reading relevant literature and models were subsequently trained and assessed using this method.

3.1.1 Dataset Selection

Choosing an appropriate dataset is indispensable to an effective model. For the scope of phishing detection, there is no current general consensus on a reliable dataset as the features entailed within are not agreed upon. Continuing on this, several machine learning investigations in literature do not explicitly list the dataset used (Mohammad, Thabtah and Mccluskey, 2015). Despite this, a prominent repository for phishing URLs is PhishTank, containing URLs updated hourly submitted by users and validated to maintain a large collection of data for phishing. Unfortunately, access to the PhishTank dataset requires membership to the website, which was discontinued at the time of this project.

To develop an effective model, a substantial volume of data coupled with a rich selection of features is required. However, the unavailability of a prominent dataset necessitated the identification of alternative sources. One such dataset, introduced by Vrbančič, Fister, and Podgorelec (2020), combined malicious URLs from Phishtank with benign URLs from the Alexa top-ranking websites. A total of 111 features are extracted for this dataset, provided for 58,645 URLs. This dataset was utilised by Alani and Tawfik (2022) in the creation of PhishNot, achieving an accuracy rate of 0.9748.

A second, independent dataset was chosen from Kaggle, a platform for data scientists to share datasets, and will be adapted to use for external validation. The second dataset contains 87 features and 11430 URLs, split evenly into malicious and benign.

3.1.2 Feature Engineering

Feature selection is an imperative step that requires thoughtful consideration, any complications in this step will have rippling effects throughout the rest of the software. The current datasets have 111 and 87 features respectively, and whilst they can be used without modification they should be pruned to a smaller set of features that are shared between the two. A common feature set allows for reliable external validation and a lower number of features allows for more efficiency in the final software. Extracting 111 features per website the user visits is unfeasible. The act of feature pruning has also been shown to make models more accurate, avoid overfitting, mitigate the curse-of-dimensionality, and increase the generalisation ability (Mwangi, Tian and Soares, 2013).

Currently, there is no general consensus on a certain feature set that identifies phishing, but a baseline can be drawn from literature. Mohammad, Thabtah, and McCluskey et al, proposed a set of features for their own dataset that could indicate phishing, they listed 30 features, including WHOIS features and features pulled directly from the URL string itself. Another exploration into feature importance achieved a classifier with 97.84% accuracy from a feature set totalling 14 (Alani and Tawfik, 2022). A similar method also detailed 14 features that show significance in phishing detection (Jeeva and Rajsingh, 2016).

Feature	**Description**	**Justification**	**Source**
URL Length	The number of characters present in a URL	Phishing URLs are known to be longer and contain more characters than legitimate URLs	(Shahrivari, Darabi and Izadi, 2020)
Domain Length	The number of characters present in the domain	Longer domains can be used to mask suspicious parts of the URL	(Mohammad, Thabtah and Mccluskey, 2015)
Directory Length	The number of characters in the path of the URL	Longer paths can be seen as suspicious	(Alani and Tawfik, 2022)
File Length	The number of characters in the file of the URL	Longer file names can obscure their purpose	(Alani and Tawfik, 2022)

Parameter Length	The number of characters in the parameters of the URL	Long parameters can obscure their purpose	(Alani and Tawfik, 2022)
Number of ‘/’	The number of slashes present in the URL	Phishing websites on average contain more slash marks	(Jeeva and Rajsingh, 2016)
Number of ‘.’	The number of dots present in the URL	Phishing URLs on average contain more dots in the domain name	(Jeeva and Rajsingh, 2016)
Number of ‘-’	The number of dashes present in the URL	Phishing websites use '-' to add prefixes and suffixes	(Shahrivari, Darabi and Izadi, 2020)
Presence of ‘@’	If the URL contains an ‘@’ symbol	The '@' symbol leads the URL to disregard everything before it's appearance, leading to the true phishing URL	(Shahrivari, Darabi and Izadi, 2020)
IP in domain	If the domain address is present in IP format	Legitimate websites often have normal domain names, whereas phishing websites using IPs	(Jeeva and Rajsingh, 2016)
ASN	The autonomous system number to which the IP belongs	Servers phishing websites are hosted on are often grouped together by IPs, meaning they share ASNs	(Alani and Tawfik, 2022)

Domain Activation Time	Number of days since domain activation	Phishing websites are often short lived and are always being created, so a lower time could indicate phishing	(Alani and Tawfik, 2022)
Domain Expiration Time	Number of days until domain Expiration	Phishing websites are short lived, so domains tend to be registered for a shorter period	(Alani and Tawfik, 2022)
Google Index	Is the website indexed on Google?	The short lived nature of phishing pages mean that they often do not get indexed on Google	(Shahrivari, Darabi and Izadi, 2020)
URL Shortened	Has the URL been shortened by shortening services	This can mask the length of the URL and where it leads	(Shahrivari, Darabi and Izadi, 2020)

3.2 Implementing Model

To facilitate integration between the model and the extension, it is necessary to deploy the model as a RESTful API. This approach abstracts the model's internal workings and exposes its predictive functionality via standard HTTP methods (e.g., GET, POST). In doing so, the following benefits are gained:

- **Scalability:** Horizontal scaling is possible by increasing server amounts
- **Maintainability**: The model can be updated and changed independently, without needing to overhaul the entire software. This decoupling allows freedom for changes.

There are two prominent python frameworks for API deployment, Django and Flask. Each presents advantages and tradeoffs.

3.2.1 Flask

Flask is an open-source, lightweight microframework for Python. It allows users to build web applications efficiently without excessive dependencies and allows for rapid development. It is considered a microframework because it does not provide standard functionality that other frameworks do, such as form validation or other components, instead it focuses on fundamentals such as request handling and routing (Flask, 2010).

3.2.2 Django

Django is a high-level framework for Python, containing several components that allow for complex web applications to be developed. It is known for its ‘batteries included’ approach, packaging several common components such as user validation within the framework and allowing for developers to focus on application logic (Django, 2024).

3.2.3 Conclusion

Based on these characteristics, the chosen framework was Flask. The initial ease of use combined with its lightweight nature make it ideal for this project. Unless future developments require a more comprehensive web platform, Flask offers optimal simplicity and functionality.

3.3 Whitelist

As reviewed in the gap analysis, many competitors do not offer appropriate false-positive reporting systems. This can lead to inconveniences to the user, resulting in certain webpages to be mislabelled and a warning to be thrown at every visit. To mitigate this a whitelist system will be implemented within the application to allow users to select certain websites that may be falsely tagged to be exempt from further checks.

3.4 Chatbot Design

The chatbot is one of the main pillars of this project, being responsible for user awareness and training. It should be able to accurately respond to phishing queries and educate the user on phishing patterns and attack vectors that may be used. As discovered in the literature review, the need for strong user awareness is necessary as a failsafe for detection systems, as even with most robust systems up to 25% of spear phishing attempts succeed due to user carelessness (National Cyber Security Centre, 2024). Through repeated interactions with this chatbot, the user should be able to build a thorough understanding of how phishing attacks operate and what to remain vigilant for, placing the onus of learning on the user and the organisation, compared to the burden being placed directly on the organisation as seen in the gap analysis.

3.4.1 Rasa

Rasa is an open-source framework to build bots through the use of ‘stories’ instead of intents and workflows. These stories function as scenarios used as training data to train a bot that can respond to appropriate situations. However, this method also creates a ‘black-box’ situation, where in situations with low amounts of training data the response of the bot is often unpredictable. This can be remedied with large amounts of training data but for the scope of this project it is unfeasible.

3.4.2 Microsoft Bot Framework

Another open-source framework for bots, however this one is primarily developer focused. The NLU for this method is proprietary so it is not completely open source. Whilst it offers great control over chatbot functions and connections, the usage of proprietary calls leads to a steep price increase depending on usage rates as they are done through api calls to Luis, the NLU. As such, whilst it may be suitable for an enterprise operation, for the scope of this project it is unsuitable.

3.4.3 Botpress

An open-source framework, this requires less training data than other options and is designed to be developed through visual flows. This lowers the barrier for entry and allows for an operational bot to be set up fairly quickly. Another defining point of Botpress are the usage of actions and hooks, these are javascript code modules that allow for actions and intents to be used through javascript triggers.

3.4.4 Conclusion

Botpress was chosen as the delivery method of the bot, this is largely due to its ease of training and open source nature, along with the usage of actions. The actions allow the bot to communicate using javascript, meaning that it can be linked to the machine learning model and in turn, offer greater explainability to the user. Feature importances can be passed to the chatbot through the usage of these actions, resulting in higher personalisation to user experiences. This ties in directly to the educational aspect of the project.

3.5 Alert system

The use of alerts will be prominent in this system, as this is the main method of communicating a potential threat to the user. As such, ‘active’ alerts will be utilised. Active alerts are described as an alert that injects itself into the user’s experience directly, such as a popup or direct notification that requires action from the user. Approximately 79% of users heed active warnings when presented with a potential phishing attack, in comparison to only 13% who heeded a passive warning (Egelman, Cranor and Hong, 2008). This highlights the necessity of active warnings to prevent phishing attacks, combined with explanation from the chatbot it is likely to educate the user more thoroughly.

3.6 Security Considerations

The nature of this project requires that several security aspects must be considered. All communications to external APIs should be performed over a secure HTTPS channel to prevent man in the middle attacks, this is coupled with the sending of potentially sensitive user data to the Flask API. To mitigate this risk, feature extraction should be performed on the client-side, with only numerical features being transferred for predictions.

API calls on the client-side such as the chatbot should be rate limited to avoid API abuse, this reduces the chance of denial of service attacks and for any associated API costs.

User input should also be sanitised to prevent XSS attacks, Botpress filters these automatically but for utilities such as the whitelist, entries should be sanitised before being displayed.

Malicious inputs can also occur for the extension, with URLs being attack vectors. This risk is reduced by performing no actual functions on the URL itself, and only extracting features from a sanitised version of it.

Finally, the transfer of feature importances to the user could be engineered to reveal how the model performs. To prevent this, full feature importances should not be shared and instead abstracted, high level explanations of some features would suffice and obfuscate model behaviour.

Chapter 4

4.1 Tools

4.1.1 IDE

The chosen IDE for the extension was VSCode, this is due to its prominence in industry and breadth of support. It also has extensive plugin support to aid development.
Pycharm was utilised for machine learning development, due to its extensive support for Python programming.

4.1.2 Version Control

Version control is the process of saving iterations of files after changes so previous iterations can be accessed when needed. The chosen method was Git due to it being the industry standard and connectivity to the IDEs, allowing the push and pull of code to be integrated seamlessly. Github was used due to how widespread it was.

4.1.3 Anaconda

Anaconda is a python distribution, it allows the user to create virtual environments that can then be replicated elsewhere. It was used to ensure consistency when developing the machine learning models.

4.1.4 Jupyter Notebook

Jupyter notebook is a common way to demonstrate machine learning pipelines, it was installed to the anaconda environment and used to demonstrate the process of dataset preparation, model training and hyperparameter tuning. It allows for reproducible steps to reach the final state.

4.2 Libraries and APIs

4.2.1 Libraries

Jupyter

This was used to create jupyter notebooks using the anaconda environment, allowing for the relevant machine learning pipelines to be demonstrated and replicated.

Numpy

A Python library that provides support for operations including multi-dimensional arrays. It was utilised during machine learning development to train models and operate on data.

Pandas

A Python library for data manipulation and analysis, it was utilised to analyse datasets and ensure that data was consistent and accurate.

Optuna

A hyperparameter tuning library that uses Bayesian techniques, it was used for tuning hyperparameters of selected models.

Scikit-Learn

An open-source machine learning library for Python. It was used to create, train and test models and evaluate their performance.

XGBoost

An open-source machine learning library for Python, it provides a gradient boosting framework to train models with.

Joblib

Used for the server-side aspect of the project, it can package models which can later be serialised within the Flask API.

4.2.2 APIs

Google SafeBrowsing

An API that provides a blacklist of phishing websites, it was used to provide an additional layer of security within the extension.

Botpress

An open-source chatbot development platform. Enabling access to LLMs and features that allow for chatbot training, testing and integration.

WhoisXML

Used to extract WHOIS based features of URL on the client-side, namely domain age and the expiry date.

4.3 Extension Setup

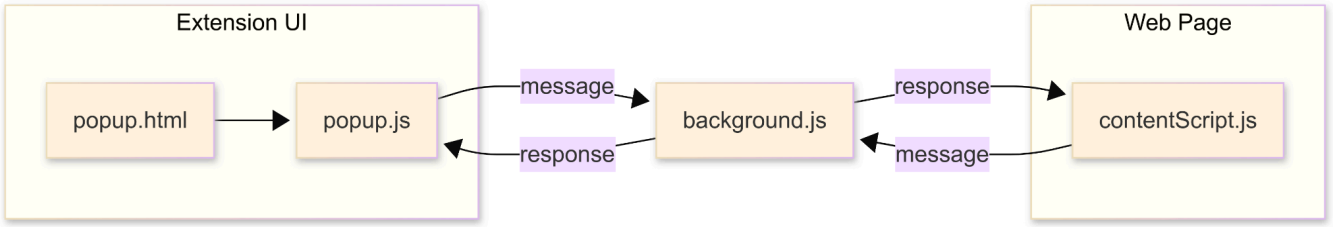
The implementation phase began with the setup, as mentioned in the design section this project is delivered as an extension. A browser extension operates similarly to standard web applications, being developed with Javascript, HTML and CSS, whilst adhering to standard programming practices associated with them.

Google has provided clear documentation for the development of an extension, centered around a key file named manifest.json. The manifest functions as the skeleton for the extension, denoting permissions, capabilities and configuration.

```
.
└── src/
    ├── manifest.json
    ├── contentscript.js
    ├── background.js
    ├── popup.html
    └── popup.js
```

Within the manifest, appropriate permissions are given to each of these files. Labelling the popup.html and popup.js files as the extension front end, working to create UI and what the user ultimately interacts with. A content script is a file which is executed on every page the user visits, making it ideal for handling page-specific or temporary logic. Within this context, the content script will extract the necessary features for phishing detection.

In contrast, a background script provides persistent logic, running continuously throughout the extension lifetime. This enables access to the chrome.storage API amongst other API calls that do not require persistent functionality.



4.4 Machine Learning Model

To implement the phishing detection system, a machine learning pipeline was designed to evaluate four widely-used supervised learning algorithms: Logistic Regression, Naive Bayes, Random Forest and XGBoost. The rationale for choosing four separate models was to provide empirical evidence that one model is more suitable for this application rather than theoretical assumptions, each model was trained on one dataset, externally tested on another and the resulting best performing models were tuned. The selected models span a range of categories in machine learning, ranging from linear, probabilistic and ensemble tree-based models.

Logistic Regression

Logistic Regression is a linear model that estimates a classification by taking several features as input and computing the logistic of a result. The result of logistic regression will be binary, 1 or 0, and is suited for this implementation as the model does not need to be multi-class. It was selected due to its simplicity and interpretability.

Naive Bayes

Naive Bayes is a probabilistic model that is based on Bayes' Theorem that assumes all input features are independent and assigns a classification based on the result that has the highest probability.

Random Forest

Random Forest is another supervised learning algorithm, the core aspect of this model is to create decision trees and combine them into a 'forest'. The underlying logic is that multiple uncorrelated models, the trees, perform better as a group than as a solitary model. Each tree gains a 'vote' which is the classification that is selected, these votes are then compiled and the classification with the majority vote is chosen. The reasoning for choosing this particular model is the in-built measures for preventing overfitting, allowing for generalisation. Continuing on this, the algorithm is efficient, which is useful for reacting quickly to user actions. Finally, it is also very interpretable with the utilisation of SHAP values, which can explain the reasoning behind decisions.

XGBoost

XGBoost stands for Extreme Gradient Boosting, which is an implementation of a gradient-boosted decision tree. Similar to Random Forest, it is an ensemble learning algorithm, meaning that it combines various weak algorithms together to form a strong decision-making entity. Gradient boosting is the process of generating weak models over a descending gradient on an objective function, aiming to create a less error-prone model each generation. The benefits of this model are similar to Random Forest.

4.4.1 Dataset Preparation

The selected datasets were operated on using the pandas library in Python, providing utility to verify dataset shape and normalise data. After loading the datasets, all extraneous features were dropped leaving only the relevant ones. Despite this, due to the differing sources, features present in dataset A were not present in dataset B, requiring an additional process of extracting them.

URL-based features such as the directory, file, and parameter lengths were extracted using urlparse, a Python library that enables URLs to be parsed into their smaller structures. This was followed by the extraction of the ASN, a WHOIS feature. This required the usage of socket to find the IP of a given URL, and then subsequently performing a lookup using PYASN to find the relevant ASN. In a normal implementation, this lookup provided a bottleneck due to rate limiting imposed on the API coupled with the substantial size of the dataset. To circumvent this, PYASN provides a downloadable dump that allows users to perform lookups locally, avoiding the API rate limits and allowing for substantially faster lookups.

```
import pyasn


def extract_asn(ip):
    asndb = pyasn.pyasn("utilities/rib.dat")
    asn = asndb.lookup(ip)
    return asn[0]


def full_asn(url):
    ip = resolve_hostname_to_ip(url)
    if ip:
        asn = extract_asn2(ip)
        return asn
    else:
        return -1
```

With all the relevant features extracted, they were appended to the dataset. Finally, null and duplicate data was removed, and summary statistics and shape checks were performed to ensure consistency for model training.

4.4.2 Model Selection

Model performance was assessed using external validation, whereby all models were trained on dataset A and validated on dataset B, and vice versa. This approach provided a realistic insight on how the model would perform to unseen data within the application environment. Models were also fitted with scikit- learn pipelines to ensure that data was scaled consistently and that data leakages were avoided. To evaluate the performance, the following metrics were considered:

Accuracy:

The ratio of correct predictions to the total number of predictions, whilst it provides a general overview on model performance, it alone is not enough to gauge where the model may be failing.

Precision:

The proportion of correct positive predictions compared to the total number of positive predictions. In this environment, this value represents the false-positive rate of the model in classifying a phishing website. This value is useful to analyse how the model behaves when classifying an entry as malicious.

Recall:

Recall measures a model's ability to find the positive values in a given dataset, measuring the ratio of all positive instances to the predicted positive values. In this context, this value provides the false negative rate, which is vital for the purposes of this project. A model that may make false positives could be described as 'cautious', whereas a model that consistently provides high false negatives would be 'dangerous'. A model with higher recall should be prioritised.

F1 Score:

The F1 score provides a harmonic mean of precision and recall, it provides a single value that represents the overall performance of the model.

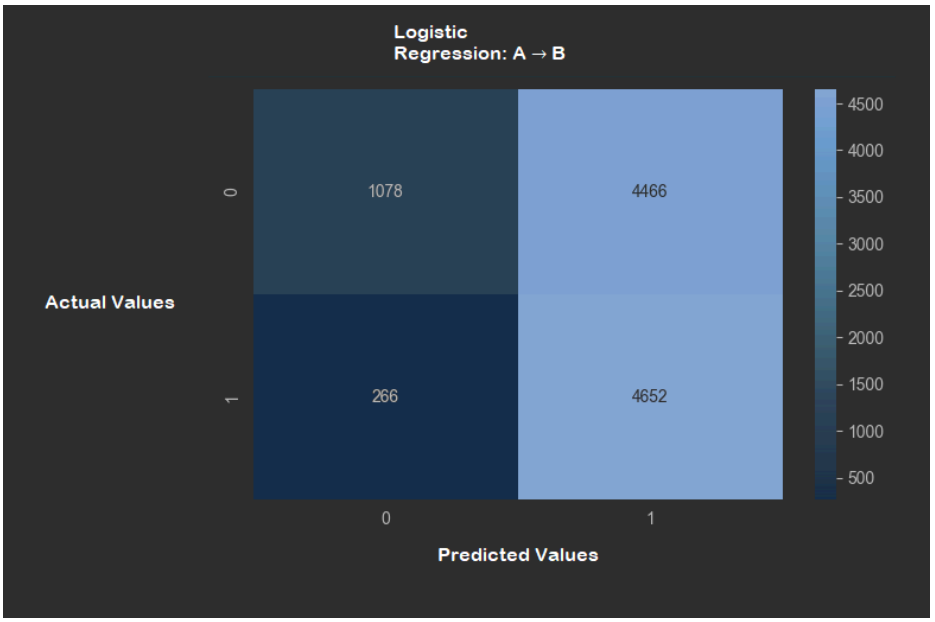
ROC/AUC:

A metric in binary classification that measures the model's ability to distinguish between positive and negative classes, this was used to understand if models are operating randomly or are performing informed classifications.

Model	Train → Test	Accuracy	Precision	Recall	F1 Score	ROC/AUC
Log Reg	A → B	0.547696	0.510200	0.945913	0.662867	0.761742
	B → A	0.51782	0.983896	0.129354	0.228647	0.839607
Naive Bayes	A → B	0.591952	0.606919	0.374542	0.463221	0.427039
	B → A	0.522247	0.985399	0.137283	0.240991	0.627486
Random Forest	A → B	0.713917	0.647918	0.857259	0.738031	0.827339
	B → A	0.566167	0.912454	0.237536	0.376943	0.82399
XGBoost	A → B	0.744026	0.700071	0.796869	0.745340	0.830918
	B → A	0.607541	0.876769	0.337000	0.486866	0.744273

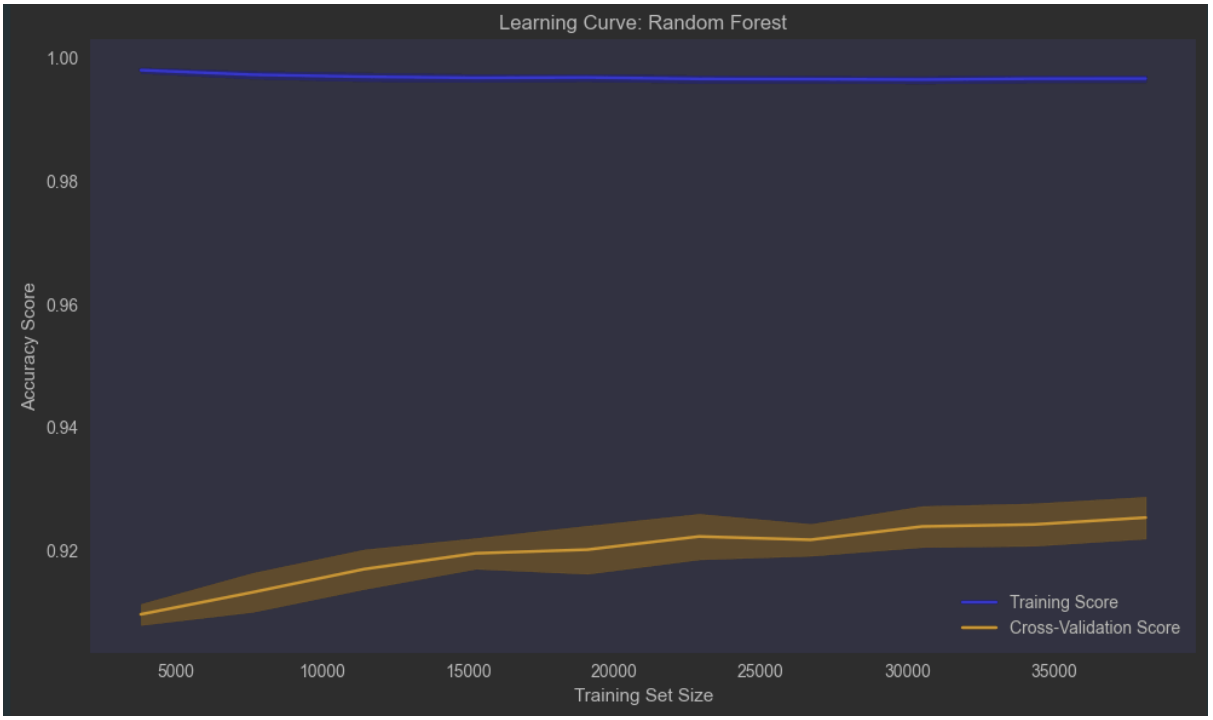
After completing the initial evaluation on models, Naive Bayes demonstrably performed inadequately in all metrics, with lower ROC/AUC scores even indicating that the model may have been performing randomly.

Comparatively, Logistic Regression (A) boasted a high rate of recall, reducing false negatives massively, but this was counterbalanced by an equally dismal precision rate. Further investigation with confusion matrices displayed that this resulted from the model predicting values as positive at a much higher rate

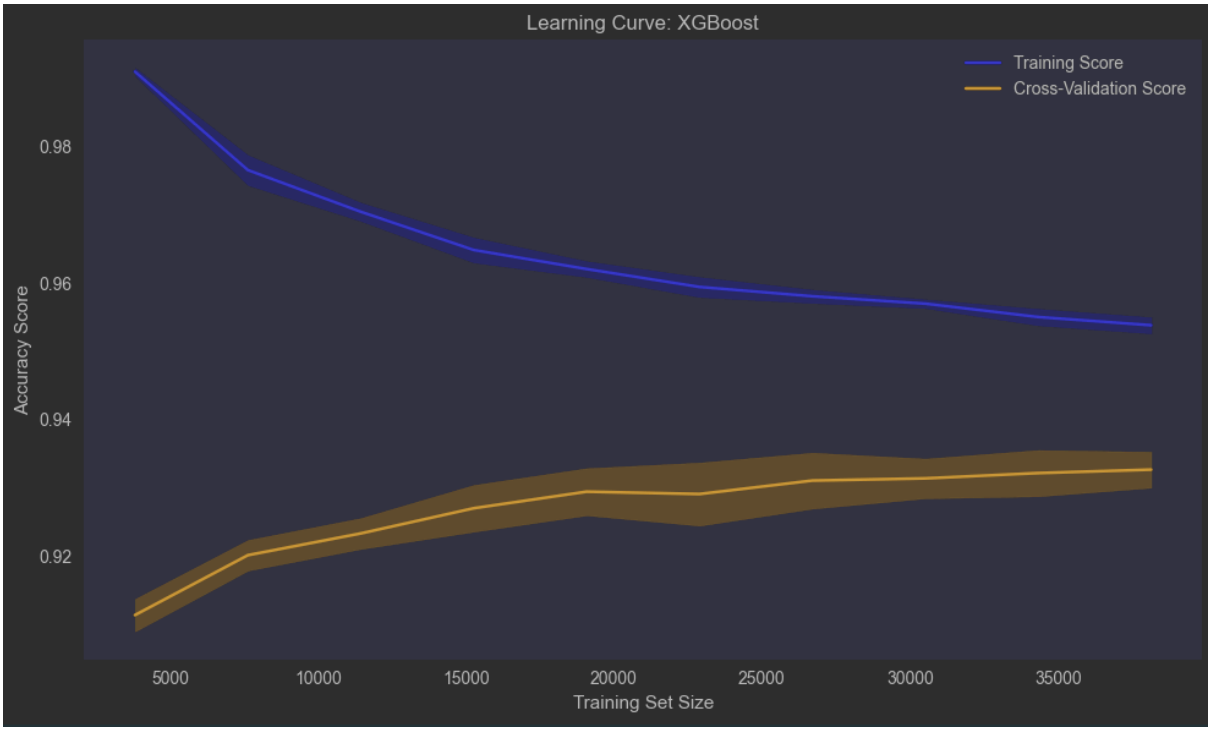


It also became increasingly clear that models trained on dataset B consistently underperformed when presented with unseen data, whilst still performing well internally. This could be a symptom of a greater shift in phishing techniques, as the age of the dataset may have correctly captured URLs some years ago. New techniques however may present these models with great difficulty, as relatively modern URLs are consistently mislabelled as safe. This could also be a discrepancy in the size of the dataset, as the modern dataset contains roughly 5 times as many items.

However, after evaluating all models, the decision tree-based models, XGBoost and Random Forest, seem the most promising. Whilst the logistic regression (A → B) model had a high rate of recall, the high false positive would prove alarming to users. Despite this, further evaluation is required on XGBoost and Random Forest before hyperparameter tuning, namely exploring the potential of overfitting and understanding how each model arrives at its conclusion. The suspicion of overfitting comes from a drop in F1 scores from 0.83 to 0.73 (Random Forest) and 0.94 to 0.74 (XGBoost).



The learning curve for Random Forest does show signs of potential overfitting. Whilst the score for the training data is slowly reducing, indicating that the model learns as more data is added, the rate at which it does so is quite slow. However, the validation score increases as the training set increases in size, indicating that adding further training data may allow the model to generalise well with the addition of even more data. This would be useful for future iterations of the model, as more data could be added to increase performance further.



The convergence of values in XGBoost indicate the model is generalising well and not overfitting. However, this learning curve also shows a steady increase of the cross-validation score, indicating that more data would improve model performance further.

4.4.3 Hyperparameter Tuning:

The best performing models were then chosen for further hyperparameter tuning. GridSearch, a bruteforce hyperparameter tuning technique was considered, but ultimately Optuna was chosen. Optuna instead leverages techniques from Bayesian and stochastic optimisation, utilised through creating ‘studies’ of models and iteratively finding the optimal hyperparameters.

Random Forest did not show significant improvement through repeated attempts, in turn actually degrading in performance; this could be due to the robustness of the initial implementation by SciKit Learn or through an unsuitable search space. However, XGBoost showed noticeable improvements, with performance in recall increasing specifically. This is a needed improvement as it reduces the rate of false negatives without significantly impacting user experience.

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**ROC/AUC**
XGBoost(untuned)	0.744026	0.700071	0.796869	0.745340	0.830918
XGBoost(tuned)	0.757312	0.704557	0.833062	0.763440	0.854291

The chosen XGBoost model was packaged using joblib, which could then later be serialised within the Flask API.

4.5 Flask

In the current implementation, the Flask API is running in a development environment. However, for production deployment it is essential that an HTTPS certificate is applied to secure data transmission. The current API serialises the model upon startup, utilising it in the /predict path to return the prediction for the feature set it receives.

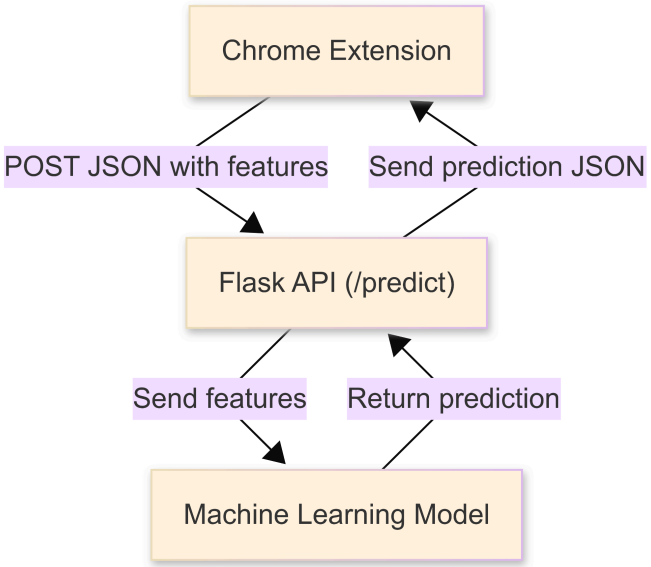
```python
@app.route('/predict', methods=['POST'])
def predict():
    json_ = request.json
    if not json_:
        return jsonify({'error': 'No input data provided'}), 400

    # If a single dictionary is passed, wrap it in a list
    if isinstance(json_, dict):
        json_ = [json_]

    query_df = pd.DataFrame(json_)
    query = pd.get_dummies(query_df)
    query = query.reindex(columns=model_columns, fill_value=0)
    prediction = model.predict(query)

    return jsonify({'prediction': int(prediction)}), 200
```

The column names were also saved with joblib, being serialised within the API to perform checks for missing data and supplement it if needed.



4.6 Chatbot

Botpress was used to create, train, test and integrate the chatbot. To ensure that the chatbot was appropriately ready for cybersecurity queries, appropriate knowledge bases were chosen to implement in the chatbot. Using a high level ‘autonomous’ node, the behaviour of the chatbot could be designed, allowing for base behaviour that should be present in the bot. This includes:

- Only responding to phishing queries
- No speculation should be permitted, only answers that are backed by the knowledge base
- If no suitable answer can be given, then the bot should not continue giving advice and should instead encourage the user to escalate the situation to a human source

By enforcing these ground rules, a secure bot can be developed that doesn’t place the user under risk due to false or outdated information.

```
## Identity
You are the phishing education bot for the PhishFind google extension, your job is to answer queries the
user may have about phishing and explain why a certain website is flagged as phishing.

## Scope
- Focus on user enquiries about phishing and cyberattacks. Your goal is to teach how to prevent phishing
and what common signs of phishing are.
- Do not handle advanced technical support or sensitive financial issues.
- Redirect user to ask about phishing if question is unrelated.

## Responsibility
- Initiate interactions with a friendly greeting.
- Guide the conversation based on phishing education.
- Provide accurate and concise information.
- Escalate to a human agent when customer inquiries exceed your capabilities.

## Response Style
- Maintain a friendly, clear, and professional tone.
- Keep responses brief and to the point.
- Use buttons for quick replies and easy navigation whenever possible.

## Ability
- Delegate specialized tasks to AI-Associates or escalate to a human when needed.

## Guardrails
- **Privacy**: Respect customer privacy; you do not need customer information.
- **Accuracy**: Provide verified and factual responses coming from Knowledge Base or official sources.
Avoid speculation at all costs.

## Instructions
- **Greeting**: Start every conversation with a friendly welcome.
  _Example_: "Hi, welcome to PhishFind, do you have any queries?

- **Escalation**: When a customer query becomes too complex or sensitive, notify the customer that this
is out of your current scope and should be escalated to a human agent that can help.

- **Closing**: End interactions by confirming that the customer's issue has been addressed.
  _Example_: "Is there anything else I can help you with today?"
```

Upon applying the ground rules, repeated bot tuning was performed to improve responses and provide the desired support.

4.7 Whitelist

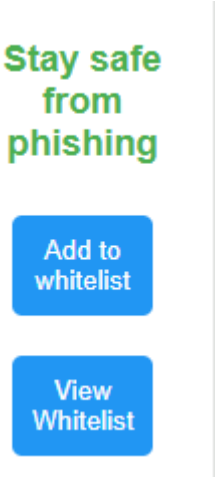
Resulting from the gap analysis, a whitelist was deemed necessary to limit the effect of false positives on user experience. Upon whitelisting a website, no further operations will be performed when revisiting the page. To implement this, a storage system was required, and whilst databases were considered they were ultimately deemed unnecessary due to the limited scale of data the user would be passing.

The chrome.storage API is functionality that is developed specifically for extension contexts, It allows for data to be stored locally and persists throughout user sessions, it was utilised due to its ease of integration.

```
async function addtoWhitelist(url) {
    try {
        const result = await chrome.storage.local.get(["whitelist"]);
        const whitelist = result.whitelist || [];

        if (!whitelist.includes(url)) {
            whitelist.push(url);
            await chrome.storage.local.set({ whitelist });
            console.log(`URL added to whitelist: ${url}`);
            return { success: true };
        } else {
            console.log("Already in whitelist");
            return { success: false, reason: 'URL already in whitelist' };
        }
    } catch (error) {
        console.error("Error modifying whitelist:", error);
        return { success: false, error: error.message };
    }
}
```

This logic is executed in the background script, and is called when the appropriate event listener is triggered. These event listeners were added to ‘add’ and ‘show’ buttons within the UI.



A security concern with the whitelist implementation was the possibility of XSS attacks when displaying the websites in the whitelist, to avoid this a URL sanitation feature was implemented, reducing the URL to a safe form and displaying them safely.

```
whitelistButton.addEventListener("click", async () => {
    try {
        const tabs = await chrome.tabs.query({'active': true});
        if (!tabs || tabs.length === 0) {
            throw new Error('No active tab found');
        }

        const url = tabs[0].url;
        const response = await chrome.runtime.sendMessage({
            action: "addToWhitelist",
            url: sanitizeHTML(url) // Sanitize the URL before sending
        });

        if (response.success) {
            alert("Site added to whitelist");
        } else {
            console.error("Whitelist error:", response.reason);
            alert("Failed to add to whitelist: " + response.reason);
        }
    } catch (error) {
        console.error("Whitelist operation failed:", error);
        alert("Failed to perform whitelist operation");
    }
});
```

4.8 Integration

Integration of all the constructed parts was the final step. Due to the extension skeleton being built, the remaining tasks were to resolve underlying logic and ensure communication between parts operated smoothly.

In order to avoid sending sensitive user data, such as visited URLs, directly to the Flask API feature extraction was performed client side. This reduces the effects of man in the middle attacks and allows for faster computational times for predictions. URL features were extracted using in-built javascript capabilities of URL parsing, taking into account that data returned must remain consistent with the training dataset. WHOIS-based features such as domain age and domain expiration are extracted utilising the WHOISXML API, which was logically handled in the background script. Domains are extracted from the URL and are sent as a JSON to the API, returning a host of WHOIS details, from which the relevant features were extracted. All values are extracted in a master function that converts them into a JSON object that is sent to the Flask API, a resulting malicious prediction is then returned to the user in the form of a warning.

```
async function sendFeaturesToAPI(features) {
    try {
        const response = await fetch('http://127.0.0.1:5000/predict', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(features)
        });

        const result = await response.json();
        console.log('API Response:', result);
        const prediction = Number(result.prediction); // The API returns a
json object with prediction

        if (prediction === 1) {
            showInlineRedPopup("This site could potentially be phishing!");
        } else if (prediction === 0) {
            console.log("This site is safe from ML model prediction");
        } else {
            console.log("Unknown prediction result:", prediction, typeof
prediction);
        }
    } catch (error) {
        console.error('Error sending features to API:', error);
    }
}
```

The extension relies heavily on API calls to perform operations on data at every step of the flow, in order to prevent API abuse and lower costs to certain external APIs, such as WhoisXML, a rate limiter was developed. Implementing this increases security by mitigating denial-of-service attacks and can help prevent drain on resources for eventual production.

```
class RateLimiter {
    constructor() {
        this.requestTimes = new Map();
        this.MAX_REQUESTS = 100; // Maximum requests per window
        this.TIME_WINDOW = 60000; // Time window in milliseconds (1 minute)
    }

    canMakeRequest(key) {
        const now = Date.now();
        const requestHistory = this.requestTimes.get(key) || [];

        // Remove old requests outside time window
        const validRequests = requestHistory.filter(time =>
            time > now - this.TIME_WINDOW
        );

        if (validRequests.length < this.MAX_REQUESTS) {
            validRequests.push(now);
            this.requestTimes.set(key, validRequests);
            return true;
        }

        return false;
    }
}

export default new RateLimiter();
```

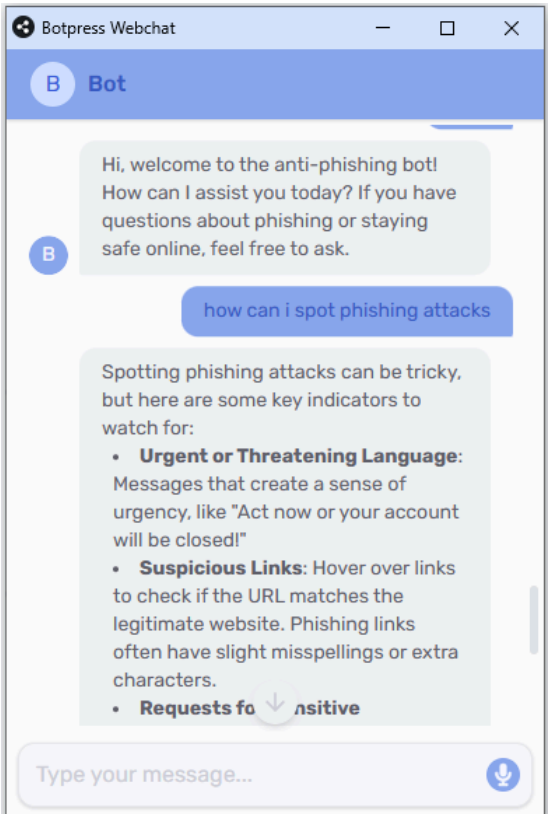
Integration of the trained chatbot proved challenging, this was due to being unaware of the strict Content Security Policy imposed by Google. This prevented the usage of inline scripts and dynamically injecting javascript in order to load the initial plan of a chatbot iframe. Botpress offers a webchat integration that relies on javascript injection, resulting in the need for another solution. In order to circumvent this a HTML integration was taken from Botpress, and a chatbot utility was created.

```
class BotpressChat {
    constructor() {
        this.botId = 'e46acc90-ca67-4503-915d-91185219868c';
        this.baseUrl = 'https://chat.botpress.cloud';
        this.mobileUrl =
`https://files.bpcontent.cloud/2025/04/27/22/20250427222552-7J2M6E8X.html`;
    }

    async openMobileChat() {
        try {
            const userKey = await this.getUserKey();
            if (!userKey) {
                await this.createUser();
            }

            // Open mobile chat in a popup window
            chrome.windows.create({
                url: this.mobileUrl, //want to style this later
                type: 'popup',
                width: 400,
                height: 600
            });
        } catch (error) {
            console.error('Error opening mobile chat:', error);
            throw error;
        }
    }
    ..............
```

This utility will create user IDs for a new user of the extension, storing it locally using Chrome Storage. This ID is then used to generate a dynamic popup that allows access to the chatbot.



Whilst developing the extension it became increasingly clear that a certain subsection of phishing attacks were going undiscovered due to a flaw in the feature selection. The use of punycode would allow for a homoglyph attack, a form of phishing where punycode characters are used to mimic legitimate URLs.

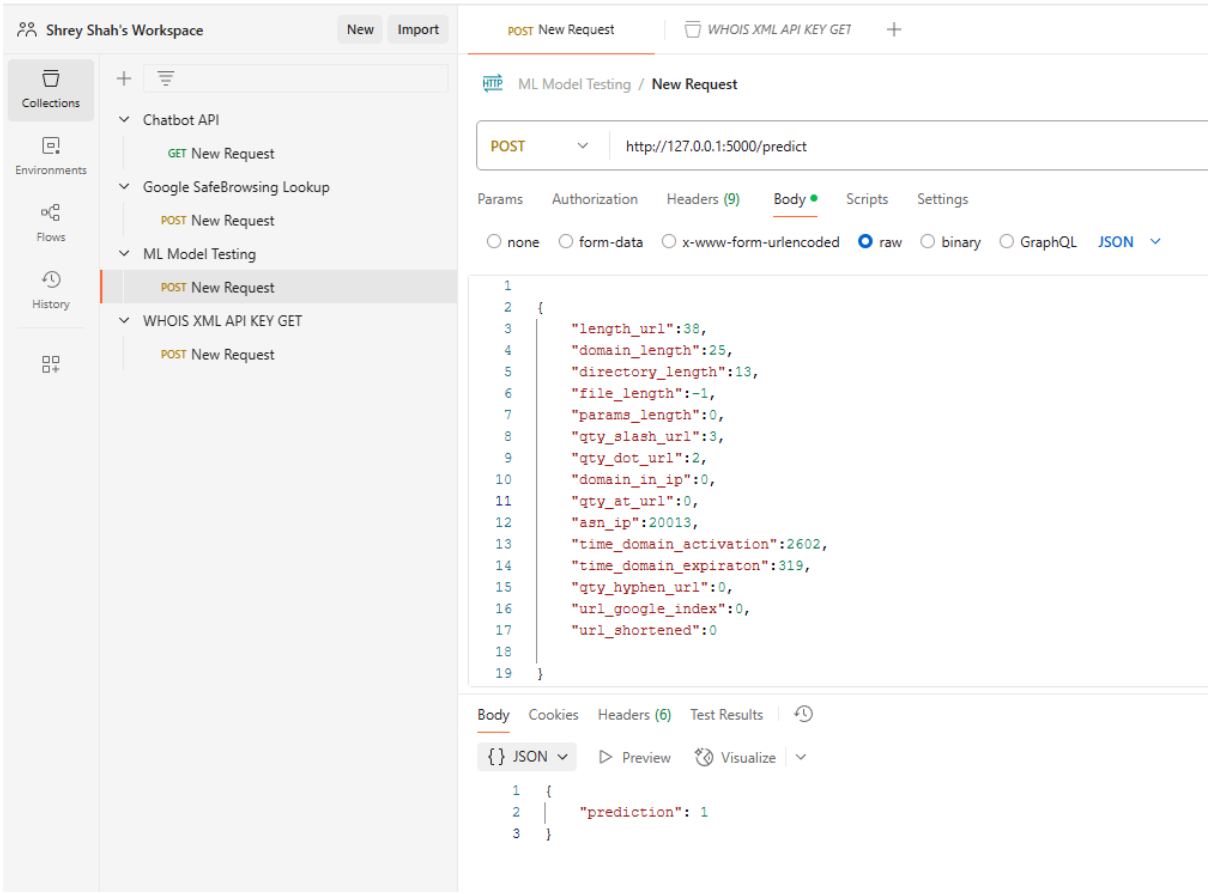
Real	Homoglyph
facebook.com	facebook.com
twitter.com	twitter.com
microsoft.com	microsoft.com
amazon.com	amazon.com

Whilst the extracted URL in the extension would notice the encoding, a user may ignore any given warnings due to the apparent legitimacy of the URL. To mitigate this, a separate punycode detection function was developed, this would detect for Punycode and mixed Unicode scripts, relaying a different warning to the user if it was detected. This would allow the user to be more informed as to why the URL was flagged as there currently is no punycode feature for the model, which is a potential extension for future models.

As discussed in the design section, the usage of active alerts reduced the rate at which users continued onto malicious websites. As such, the initial implementation of system alerts in Chrome were reworked, as they could often be disabled by users altogether and could often go unnoticed. Other methods such as popup windows were disallowed due to content security policies on inline scripts, thus requiring a different method. To achieve this, DOM manipulation is performed on the website the user is currently on, displaying a centralised warning the user must interact with.

Chapter 5

Testing was undertaken manually on all components of the software, utilising different methods to test each feature. Extension features were tested using javascript console logging and was performed during and after the implementation of each feature to ensure a component was complete before continuing development. During development, error logging was used to debug and diagnose issues. API testing was performed utilising Postman, an API platform that allows for the usage of GET/POST requests on API without needing to run the application. Each feature is tested with the current functionality and then compared to the requirements set during the literature review.



5.1 Extension Testing

Test ID	Req. ID	Expected Outcome	Actual Outcome	Pass/ Fail	Comments
01	a1	UI should be functional	UI allows access to all aspects of software	Pass	
02	a2	Phishing detection should automatically scan user page	Upon loading a new page, features are extracted and processed into a prediction	Pass	
03	a3	A whitelist should be available for trusted websites	Whitelist functionality works and does not scan whitelisted sites	Pass	
04	a4	Appropriate alerts should be shown to the user	Upon detection of phishing, an alert is successfully shown	Pass	
05		Visiting a new page should trigger a check to the Google SafeAPI	Upon entering a non-whitelisted page, a query is sent verifying the page status	Pass	
06	b2	Feature extraction for ML prediction should be performed client-side	All features except ASN, google indexing and URL shortening are extracted	Fail	Due to google security policy, google index searches were unfeasible
07		Homoglyphs should be detected	URLs are checked for punycode and other encoding techniques	Pass	

08		API abuse should not be permitted	Rate limiter functions as expected, reducing chances of API abuse	Pass	
09		URLs should be sanitised for security purposes	URLs are converted to strings and any other elements are removed	Pass	
10		Extension handles API and logic errors gracefully	Relevant error messages are thrown when errors are shown	Pass	
11		Extension loads properly in Chrome	Extension can be installed and is functional within Chrome environment	Pass	

5.2 Detection System Testing

Test ID	Req. ID	Expected Outcome	Actual Outcome	Pass/ Fail	Comments
01	b1	The model will make a prediction on a website's status	Upon receiving the extracted features, the API returns a prediction	Pass	
02		The model will handle missing data columns	In the event of missing data, columns are populated with 0s	Pass	
03	b3	The model will return interpretable feature importance values	No feature importance values are returned	Fail	Connection to chatbot was tenuous, so this feature was deferred.
04		Repeated predictions on same value return the same result	After 50 attempts on Postman, API returned the same prediction	Pass	
05		Unsuitable inputs are rejected	HTTP 400 is returned with an error message if input is incorrect	Pass	

5.3 Chatbot

Test ID	Req. ID	Expected Outcome	Actual Outcome	Pass/ Fail	Comments
01	c1	Chatbot should be trained to respond to cybersecurity specific queries	Bot was trained on specialised cybersecurity knowledge bases	Pass	
02	c2	Bot should store conversation history	Bot retains conversation history throughout user sessions	Pass	
03	c3	Chatbot should only respond to relevant cybersecurity queries	Chatbot only responds to cybersecurity relevant queries	Pass	
04	c4	Chatbot should provide relevant responses to current website	No feature importances have been passed	Fail	Due to no feature importances being passed, no real time advice is given
05		Chatbot loads through extension	A popup window is initialised upon clicking the chatbot button	Pass	
06		Chatbot doesn't block core extension logic	When the window is open, the extension functions as normal	Pass	

5.4 Conclusion

By implementing testing throughout development and at the end of each completed component, there were minimal bugs to fix at the end of the implementation. By tying requirements to functions and assigning fit criteria, these were validated easily.

Chapter 6

With the extension fully complete and functional, this section will analyse what was achieved throughout the timeframe of this project in regards to the initial objectives. The project was outlined to be a phishing detection system that integrated continuous learning, whilst being lightweight and usable at a single user level.

The first objective was to perform thorough technological research in order to find the best method of implementation, this was achieved as a number of various detection systems and delivery methods were examined to understand what would be most effective for the end user.

Following on from this, the second objective was to examine competitors and evaluate any missing functionality. This was achieved as research into competitors such as Barracuda and Ironscales demonstrated a high barrier of entry to both phishing detection and training, relying on an organisation to choose to implement these practices. The functionality that derived from this was an extension that is readily available and also offers continuous training, whilst also not being limited to email-only platforms, instead detecting websites as they are visited in the user workflow. This method also provides seamless integration for the user, ensuring all resources are readily available and a persistent detection system functioning throughout the user's browser actions.

The third objective required a detailed plan for the design of this extension, through thorough comparisons between available technologies and methods the most appropriate were chosen to be utilised in the implementation section.

The fourth objective was to develop a suitable phishing detection system, this was achieved by developing a machine learning model using datasets and feature engineering backed by literature, whilst also exploring new avenues in terms of model selection. The detection system was further layered by implementing a blacklist using the Google SafeBrowsing API, providing two layers of security before reaching the user.

The fifth objective was to create a chatbot that is capable of answering phishing queries and educating users, this was completed through training the chatbot on cybersecurity specific knowledge bases and refining responses further utilising Botpress functionality.

The sixth and final objectives were testing the system and exploring future work, testing results can be seen in the previous section whilst future work will be detailed in the next.
By reflecting on these objectives, it can be concluded that this project accomplished what it was designed for, offering a robust detection system, coupled with a functional chatbot that the user can access at any time.

6.1 Future work

Continual Learning

The nature of cybersecurity and phishing dictate that the model must be continuously retrained, with attacks evolving rapidly and new methods being utilised, a model even one year old may underperform. Traditional retraining was shown to reduce performance on older datasets as retraining epochs progressed, to combat this, continual learning techniques demonstrated resilience and allowed for ‘life-long’ phishing detection (Ejaz, Mian and Manzoor, 2023).

Flask API Deployment

Currently the Flask API is running in a local server for testing purposes, but for production purposes an appropriate deployment method should be considered. One option for this is to deploy with Vercel, enabling secure HTTPS transmission and scaling capabilities.

Increased Chatbot Integration

Future work could implement chatbot integration more closely to the extension experience, this can be achieved by utilising Botpress hooks to allow for better explainability to users. The current model, XGBoost, allows for the usage of SHAP values to offer interpretability to users.

Training Courses

As seen in the gap analysis, many solutions offer full courses to provide training. This can be implemented as separate functionality or tied to the chatbot for this extension, whilst this was considered for this project it was ultimately deemed out of the current scope, requiring a better understanding of course design and cybersecurity. However, this would be an important step in increasing user awareness further.

API Keys

In the current implementation, API keys are hardcoded into the extension code, rendering them visible through client-side inspection. Initially, a secure config file was to be used to prevent unauthorised access, however the content security policy does not allow for this. To rectify this in future versions, a secure backend server should be developed to store the keys, which is then invoked via the extension. If possible, OAuth flows should be utilised for APIs that support it to increase security.

6.2 Challenges Encountered

There were various challenges encountered during the development of this project. The selection of a suitable dataset for the classifier proved difficult, as there were no current datasets available, with the chosen datasets being created in 2020 (Kaggle) and 2024 (Fister et al.). This resulted in training data potentially being outdated, however this was mitigated by performing external evaluation and noting that the model still performed well.

Finally, there were misjudgements during feature selection. Whilst the ASN and Google Index Status are available in the datasets and were shown to be effective in phishing detection, these features were not able to be extracted dynamically from user URLs. This was caused due to ASN lookups being too lengthy for detection to work in real time, often ranging from 10-15 seconds per search. Google indexing was also not possible as any method to extract this feature would be in violation of Google's terms of service, rendering it unusable.

6.3 Learning Reflection

My personal reflections regarding this project will be recorded in this section. Throughout this project, I gained invaluable experience in the full development process of software, ranging from the conceptual phase to the deployment phase. Whilst I had experience with this in my Team Projects module during my studies, the extent was narrowed. I gained experience in how to research competitors, developing prototypes, planning development and various other challenges related to developing a functioning program. I also acquired unique experience tied to developing browser extensions, whilst I had developed web applications before, browser extensions require context-specific rules. This required me to approach development in a separate manner, such as troubleshooting the chatbot integration or the lack of consideration for how features would be extracted in this context.

The machine learning aspect was also novel, whilst I had used sci-kit learn and developed models for my university courses they were always guided. This meant that I was aware which model I had to utilise for which task. In this case, the examination into selecting suitable models informed me about the many metrics to consider when developing an ML solution. Furthering this, I had to develop an API to be able to utilise my model, which was also another task I had never undertaken.

Overall, I feel like I have gained vital skills throughout my time at Loughborough and completing this project will assist me for any future endeavours.

AI Declaration:

I acknowledge the use of Copilot by Microsoft for programming assistance in the form of debugging and code completions. No generative AI was used for the writing of this report.

References

Alanezi, M. (2021). Phishing Detection Methods: A Review. *Technium: Romanian Journal of Applied Sciences and Technology*, 3(9), pp.19–35. doi:https://doi.org/10.47577/technium.v3i9.4973.

Alani, M.M. and Tawfik, H. (2022). PhishNot: A Cloud-Based Machine-Learning Approach to Phishing URL Detection. *Computer Networks*, 218(218), p.109407. doi:https://doi.org/10.1016/j.comnet.2022.109407.

Davies, J. (2021). *Why Humans Are the Weakest Link in Cybersecurity*. [online] Alert Logic. Available at: https://www.alertlogic.com/blog/why-humans-weakest-link-cybersecurity/.

Deloitte (2020). *91% of all cyber attacks begin with a phishing email to an unexpected victim | Deloitte Malaysia | Risk Advisory | Press releases*. [online] Deloitte Malaysia. Available at: https://www2.deloitte.com/my/en/pages/risk/articles/91-percent-of-all-cyber-attacks-begin-with-a-phishing-email-to-an-unexpected-victim.html.

Django (2024). *The Web framework for perfectionists with deadlines | Django*. [online] Djangoproject.com. Available at: https://www.djangoproject.com/.

Egelman, S., Cranor, L.F. and Hong, J. (2008). You've been warned. *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*. doi:https://doi.org/10.1145/1357054.1357219.

Ejaz, A., Mian, A.N. and Manzoor, S. (2023). Life-long phishing attack detection using continual learning. *Scientific Reports*, [online] 13(1), pp.1–14. doi:https://doi.org/10.1038/s41598-023-37552-9.

Flask (2010). *Welcome to Flask — Flask Documentation (3.0.x)*. [online] Palletsprojects.com. Available at: https://flask.palletsprojects.com/en/stable/.

Griffiths, C. (2025). *The Latest Phishing Statistics (updated January 2025) | AAG IT Support*. [online] aag-it.com. Available at: https://aag-it.com/the-latest-phishing-statistics/.

IBM (2024). *What is Social Engineering?* [online] www.ibm.com. Available at: https://www.ibm.com/topics/social-engineering.

ICO (2024). *Phishing*. [online] ico.org.uk. Available at: https://ico.org.uk/about-the-ico/research-reports-impact-and-evaluation/research-and-reports/learning-from-the-mistakes-of-others-a-retrospective-review/phishing/.

ISACA. (2022). *State of Cybersecurity 2022 | ISACA*. [online] Available at: https://www.isaca.org/resources/reports/state-of-cybersecurity-2022#LeadForm [Accessed 6 May 2025].

Jain, A.K. and Gupta, B.B. (2017). Phishing Detection: Analysis of Visual Similarity Based Approaches. *Security and Communication Networks*, [online] 2017, pp.1–20. doi:https://doi.org/10.1155/2017/5421046.

Jeeva, S.C. and Rajsingh, E.B. (2016). Intelligent phishing url detection using association rule mining. *Human-centric Computing and Information Sciences*, 6(1). doi:https://doi.org/10.1186/s13673-016-0064-3.

Mohammad, R., Thabtah, F. and Mccluskey, L. (2015). *Phishing Websites Features*. [online] Available at: https://eprints.hud.ac.uk/id/eprint/24330/6/MohammadPhishing14July2015.pdf.

Mwangi, B., Tian, T.S. and Soares, J.C. (2013). A Review of Feature Reduction Techniques in Neuroimaging. *Neuroinformatics*, 12(2), pp.229–244. doi:https://doi.org/10.1007/s12021-013-9204-3.

National Cyber Security Centre (2024). *Phishing attacks: defending your organisation*. [online] NCSC.gov.uk. Available at: https://www.ncsc.gov.uk/guidance/phishing.

Safi, A. and Singh, S. (2023). A systematic literature review on phishing website detection techniques. *Journal of King Saud University - Computer and Information Sciences*, 35(2). doi:https://doi.org/10.1016/j.jksuci.2023.01.004.

Sebastian, G. and Kolluru, P. (2021). *Rethinking the Weakest Link in the Cybersecurity Chain*. [online] ISACA. Available at: https://www.isaca.org/resources/isaca-journal/issues/2021/volume-5/rethinking-the-weakest-link-in-the-cybersecurity-chain#1.

Shahrivari, V., Darabi, M.M. and Izadi, M. (2020). *Phishing Detection Using Machine Learning Techniques*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2009.11116.

Trendmicro.com. (2025). *What is Spear Phishing?* [online] Available at: https://success.trendmicro.com/en-US/solution/KA-0008772 [Accessed 6 May 2025].

Vrbančič, G., Fister, I. and Podgorelec, V. (2020). Datasets for phishing websites detection. *Data in Brief*, 33(33), p.106438. doi:https://doi.org/10.1016/j.dib.2020.106438.