Machine Learning System to predict phishing sites using HTML, URL & metadata

Marking Scheme: Experimentation based with Significant Software Development

by

Niklas Lange

RegNo: 202320011

In partial fulfilment of the
requirements for the degree of
Bachelor of Science
in
CS408: Individual Project

Department of
Computer and Information Sciences

Except where explicitly stated all the work in this report, including appendices, is my own and was carried out during my final/fourth year. It has not been submitted for assessment in any other context.

March, 2026

Abstract

Since the start of the internet, phishing has been a persistent problem, with attempts becoming more sophisticated and common in recent years. Prevention models are therefore needed to classify sites correctly as either phishing or legitimate at scale. This paper builds and compares three ML models — Random Forest, Logistic Regression and K-Nearest Neighbours — and a CNN model, using URL, metadata and HTML features collected from phishing and legitimate websites. The models were evaluated on both historical and modern datasets, with a key finding being that all models experienced significant accuracy degradation on modern data, with ML models dropping by $\sim$30% and the CNN by $\sim$48%, despite performing best on historical data.

Acknowledgements

This project has been both a personal and academic challenge and would not have been possible without the support I received. I, of course, would like to thank my project supervisor, Andreas Neofytou, whose advice and guidance have made this project as much of a success as I feel it is. I would also like to extend my gratitude to the Computing Science staff at Strathclyde who I could always ask for help and guidance.

Contents

1 Introduction

As cyber attacks become more frequent and sophisticated, a serious challenge for individuals, organisations and national security arises. According to a survey by the UK home office titled 'Cyber security breaches survey 2024', the most common type of breaches were phishing attacks making over 80% of all attacks. Furthermore, 243,000 scams have been removed as of February 2026 according to the UK National Cyber Security center, that have used one or more URLs [1], demonstrating how widespread phishing scams are that use some sort of website. These phishing attacks deceive individuals into sharing sensitive information such as, but not limited to passwords, financial information or personal details. With yearly estimated cost of cyber crimes in the UK being billions of pounds per year [2]. As digitalization increases, the need to prevent such attacks is a major challenge with detection systems becoming increasingly more important.

To address this problem, several approaches have been proposed to mitigate the phishing threats from educating potential targets and creating blacklists that block known URLs that where identified as malicious and then blocked. Although effective against previously identified threats, they struggle to detect newly created ("day-zero sites") phishing websites. This has required more advanced techniques and the use of Machine Learning to better predict if a site is malicious or legitimate. Attackers continuously evolve their techniques and find new loopholes resulting in a continuous arms race between phishing attempts and ways of prevention. Currently some of the best prevention system use a mix of heuristic, list-based and phishing detection model often trained on engineered features [3]. With some more recent models using the raw data to train, deep learning models like using the raw HTML data of a site, that do not rely on heavy feature engineering. However, these approaches are trained on historical datasets and may not be generalized to modern phishing attacks. Furthermore there is a lack of comparisons of traditional feature based models and deep learning approaches. This raises the questions regarding robustness and long-term effectiveness of current phishing detection models.

This paper aims to investigate detection models to predict if a site is legitimate or a phishing site using a mix of raw HTML, URL and metadata from each site. Then using 3 features based models - Random Forest, Logistic Regression and K-Nearest Neighbours (KNN) -

and a deep learning model - Convolutional Neural Network (CNN). The models will both utilize historical dataset sets and newly collected data to evaluate model performance.

The following research questions (RQs) are addressed in this study:

RQ1 : How effectively can machine learning models detect phishing websites using a combination of URL, HTML, and metadata features?

RQ2 : Which feature categories contribute most to detection performance?

RQ3 : How well do models trained on historical datasets generalise to modern phishing data?

RQ4 : To what extent does detection performance degrade when evaluated on newly collected real-world phishing websites?

The remainder of the paper is organized as follows. Section 2 presents the background literature for predicting phishing sites. Section 3 details the specification and the design, with section 4 going into greater depth of the implementations of the software build. Section 5 contains the evaluation results and analysis of the proposed methods. Section 6 will discuss the findings and their implications. Finally, Section 7 summarizes the conclusion and indicates future studies.

2 Background Literature

2.1 The Phishing Threat landscape

Phishing has been often synonymous with emails, where malicious actors are gaining access to sensitive information through deceitful means. Although phishing emails are still prevalent there is a rise in more personalized and sophisticated phishing attacks like using websites to gain users' trust and access to sensitive information. With National Cyber Security Centre (UK) having already removed 433,000 URLs that were associated with scams [1] . Although phishing attacks can be mitigated through having strategies in place to reduce the damage of malicious actors getting these sensitive information and training staff there is no guarantee organisation or individuals will be 100% secure against these threats [4]. That is why a detection system is needed to add another protection layer against malicious actors to prevent more phishing attacks. This can be simple heuristic based detection system or more advanced Machine Learning(ML) and deep learning system which have slowly been used more to combat these attacks.

2.2 Traditional detection approaches and their limitations

Traditional detection approaches can be put into three categories: list-based, heuristic-based and visual similarities.

List-based phishing prevention method can be put down in either whitelist, a list of sites the device can only access, and blacklist, a list of sites the device cannot access. Although these are effective, the issue is for whitelists are often too restrictive and are likely to stop a user from accessing legitimate sites, which makes it for many organisations unfeasible to implement and for personal uses too difficult to work with. Meanwhile blacklists have the opposite problem in that it gives the user freedom to access all the sites except known phishing sites, with the major limitation being if a URL is not on the blacklist the user will be in danger of getting scammed. To address this limitation, heuristic-based approaches were developed.

A heuristic approach may help solve this issue as it follows some sort of rules from the data it gains from the website like URL, text content, DNS, digital certification and website

traffic. The benefit of a heuristic method is that it may better detect Zero-hour phishing more effectively than a black-list based method, but as phishing attacks evolve and become more sophisticated these models may soon become insufficient [3]. Another suggestion was instead of a reactive approach to use a more proactive approach of looking at legitimate site and their visual clues.

Another option is to look at visual similar website instead of rules. That checks known website that are more likely to be phishing like retail and personal finance site and using visual similar data like: CSS, text layout, source code, the website logo, screenshot and any other visual element. Training models on this data. Although this can be very effective against known website duplicates it requires more resources than the other methods.But as it requires a lot of resources it has only limited uses making this option one of the most challenging to correctly implement and maintain on large scale.

While each of these approaches contributes to phishing detection, they share a common limitation — they rely on predefined rules or known examples, making them ineffective against novel, zero-day attacks. This has driven the adoption of Machine Learning techniques, which can learn patterns from data without requiring manually defined rules [3].

2.3 Machine Learning (ML) Approaches

Unlike heuristic models that rely on manually defined rules, ML models can be used to detect phishing website by using the large scale data to train models and without the need to define rules.

The central idea of the ML approach is to use the massive amount of data that can be collected from legitimate websites that can be accessed through API like Common Crawl and then merge the data with open source phishing data based like PhishTank that display phishing sites, to then apply feature engineering and then train models. These features can be URL based with feature engineering models may include but not limited: checking if certain symbols are present. length of the URL, if the IP is present, counting the subdomain. Furthermore combined with the website codes features like: number of external hyperlinks, ratio of hyperlinks and if internal/external CSS is used. Turning all these features into a

metric, often a binary one, to then train and test models [5].

ML models chosen to perform predicting if a site is phishing or legitimate, these are often classification models. Some of the most commonly used classification models are Random Forest, K-Nearest Neighbours and Logistic Regression, with Random Forest being the most widely adopted, appearing in 31 out of 57 ML-based studies reviewed by Safi and Singh [3], achieving accuracies of up to 97.41% [6]. While Logistic Regression and KNN accuracy achieved up to 95.56% and 99.04% respectively [7].

Although ML models are effective, they still need some sort of feature extraction which is often requires manual input and human decision making. This has the potential of making older ML models obsolete even if you train them on newer data as these models may not be optimised and trends may not be discovered because of the feature engineering. That is why there has been more research into using deep learning models to solve this where models do not require feature engineering.

2.4 Deep Learning (DL) Approaches

Instead of relying on any feature engineering, deep learning uses raw data to create a model and looks by itself for patterns, with the central idea being that these models will find hidden patterns better and do not rely on human input to avoid any human errors or assumption that may result in false classification.

These models can use raw data like HTML, URL and metadata that allows to train and test DL models to find hidden pattern collected from similar sources like the ones previously mentioned for ML. Compared to ML models DL models do require more data points and can easily become a blackbox.

Convolutional Neural Networks (CNN) is one of the most common DL models, using convolutional layers to automatically extract spatial patterns without manual feature engineering. This in theory makes CNN well suited to deal with phishing attacks by finding patterns in the raw data that a feature engineer may have not done, with this method being the closes algorithm to avoid any human error. According to recent research where a CNN was only trained on raw URLs and HTML code, resulting in the model achieving n accuracy of 98.1% [8]. While other models improved CNN by adding a self-

attention mechanism, allowing the model to focus on the most relevant parts of the input simultaneously, resulting in an accuracy of 97.82 % [9]. With CNN models have been reported as achieving the highest accuracy in phishing detection [3]. While these models achieve the highest accuracy, the trade-off between performance and interpretability remain an open challenge.

As DL models are strong with the lack of human input, there arises the issue of dealing with a blackbox. As these models are currently some of the best phishing detection models interpreting the results may be more difficult compared to the previous models as there was an explanation to all of them.

2.5 Summary and Research Gaps

With phishing attacks becoming more sophisticated, from simple emails and messages to more advanced attacks by using websites, different approaches have been suggest to combat phishing attempts that use websites. From list-based models, like a whitelist, one being too restrictive while the other offers no protection against zero-day sites. To more advanced models like heuristic models that follow some sort of rules that may become redundant after a time period, to now where in the age of big data ML and DL models are starting to get implemented with ML models still needing feature engineering while DL models can be difficult to interpret.

As the advanced models need to be constantly retrained to stay relevant there has been a gap to actually see the effect when training a model on these kind of sites has. While there has also been little comparison to DL and ML models while using HTML, URL and Metadata of the site to train these models, with this combination not being used before [8].

This project addresses this gap by using an older dataset and a new collected one with all the previous mentioned features, while also collecting and then comparing it to RF, LR, KNN and CNN models. Furthermore also evaluating the importance of each feature for each ML model.

3 Specification & Design

3.1 Methodology

This project will follow an iterative methodology, building on the product from the initial prototype through continuous refinement [10]. As this project is experimental in nature — as established in Section 2, limited research has examined URL, HTML and metadata simultaneously — the optimal configurations could not be determined in advance. Each stage will inform the next stage , as the data collection will inform what kind of additional preprocessing will be done, the feature engineering will influence the initial ML models and initial model results guided further refinement. This approach was particularly suited as issues will arise that this methodology will handle well such as missing metadata values, feature selection and model optimisation.

3.2 Analysis

The following models were selected for this project. For ML, RF, LR and KNN have been selected as these are some of the most common models and have shown strong results for this task [3]. While for the DL model CNN was chosen as it has demonstrated the highest reported accuracy in phishing detection literature, achieving up to 99.7% accuracy[9].

To train these models three categories were selected URL features, HTML features and Metadata. As URL has been proven effective in literature [7], HTML has shown to capture structural patterns [8], and the metadata adds domain registration context [5]. As all three of these feature categories can be collected on a large scale all this data can be used to train ML and DL models and as identified in Section 2, there is a gap to see how effective all of this data is combined to classify sites. This combination directly addresses RQ1 and RQ2 as outlined in introduction.

This project will have the following **functional** requirements:

- The system must classify a website as phishing or legitimate given a URL, HTML content and WHOIS metadata
- The system must train and evaluate three ML models — Random Forest, Logistic Regression and K-Nearest Neighbours — with the data having been featured

engineered.

- The system must train and evaluate a CNN model on raw URL, HTML and metadata text.
- The system must extract URL-based features from raw URL strings including length, character counts and structural patterns.
- The system must extract HTML-based features from raw HTML content including link ratios, script counts and form analysis
- The system must extract binary metadata features from WHOIS data including but not limited to registration status and email presence.
- The system must evaluate all models on both a historical dataset and a modern dataset separately.
- The system must produce evaluation metrics for each model including accuracy, precision, recall, F1 score, false positive rate and false negative rate.
- The system must generate feature importance scores for the Random Forest model.
- The system must evaluate each feature category individually and the combination for the ML models.

3.2.1 Non-Functional Requirements

This project will have the following **non-functional** requirements:

- The system shall process the full dataset of around 60,000 records - training and testing combined - within a reasonable timeframe on consumer hardware.
- The system shall fit all preprocessing transformations and scalers on training data only to prevent data leakage from test sets.
- The system shall be reproducible — all preprocessing, feature engineering and model training steps shall be version controlled and documented.
- The system shall handle missing metadata values gracefully without crashing.

- The system shall produce all evaluation plots and results tables automatically without manual intervention.
- The system shall maintain separate ML and DL datasets to prevent data leakage between feature engineering approaches.

3.3 System Design

The system design can be put into the following six sections: Data Source, Data Merging, Preprocessing, ML/DL Dataset split, ML Models & CNN Model, and Evaluation.

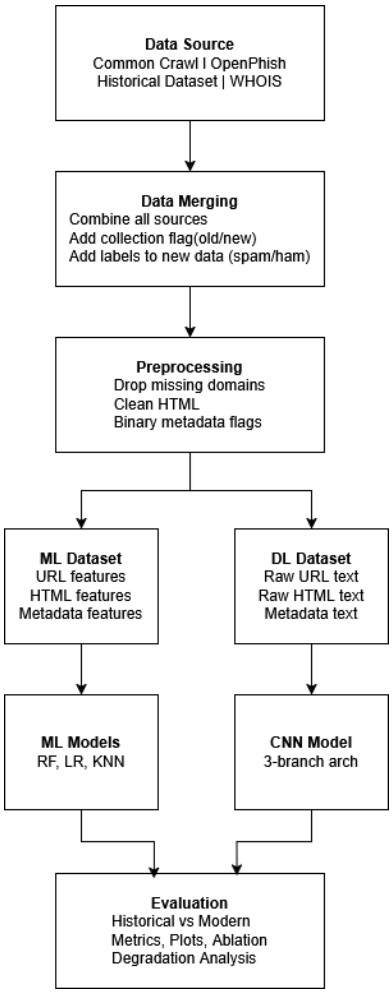

Figure 1: System data pipeline

3.3.1 Data Source

Four sources were used to collect the data, that can be split into the modern and historical dataset. The historical dataset used was originally collected for a phishing detection study using raw URL and HTML features [8]. The modern data was collected from OpenPhish for the phishing sites and Common Crawl for the legitimate. WHOIS was used to then collect the metadata for all of the data points. Data collection was facilitated using the warcio, python-whois and requests Python libraries.

3.3.2 Data Merging

Before the data was merged the modern data was labeled 'spam' or 'ham' - same as the historical data - respectively for phishing and legitimate entries. Then the modern and historical data was labeled either 'old' or 'new' and than all merged together.

3.3.3 Preprocessing

Rows with missing domains were dropped as this indicated WHOIS metadata could not be retrieved. The HTML was cleaned for easier feature engineering and binary metadata flags were created, before the data was then split up between ML and DL dataset.

3.3.4 ML/DL Dataset split

The dataset was split up with the ML dataset going through further feature engineering for the URL and HTML, with all raw Metadata features getting dropped. For DL URL was kept as a raw string and the same with HTML, with the Metadata field containing field containing registrar, nameservers, org, country, status as concatenated text, and a binary flag that shows if it has been updated.

3.3.5 ML Models & CNN Model

At this stage ML models are getting trained using the feature engineered data — Random Forest, Logistic Regression and K-Nearest Neighbours. With the DL model using a three-branch architecture processing URL, HTML and metadata separately, with character tokenization being used for the URL and word tokenization for the metadata and HTML.

3.3.6 Evaluation

All models evaluated on historical test set and modern dataset separately, producing metrics, plots, ablation study and degradation analysis.

4 Product

4.1 Implementation

The implementation can be broken down into six sections: Data Collection, WHOIS Pipeline, Preprocessing and Feature Engineering, ML Models, DL Model, and Software and Library Information. All code was written in Python.

4.1.1 Data Collection

The data collected can be split into two sub-categories: Historical and Modern. The historical dataset is of size 45,373 (see Tables 1, 2 & 3 for more details) with three columns: Category (ham/spam), URL, and HTML. The data can be accessed on Kaggle [11]. It is worth noting that the historical dataset had URL prefixes removed by its creator, meaning URLs did not contain http:// or https:// prefixes. This was handled during feature extraction by prepending a default http:// prefix where absent.

Table 1: Historical dataset shape

Column	Size	Total Percentage (%)
Total size	45,373	100
Category spam	22,686	50
Category ham	22,687	50

Table 2: Sample URLs from the historical dataset - DO NOT CLICK ON THE LINKS

Index	URL
25715	www.isoc.org/internet/history/
18810	www.aeon-jp.cc
40421	miolkoijhjhjhb.gq/CC_POSTALE
18744	www.theatlantic.com/unbound/digitalreader/dr20...
28869	dimacs.rutgers.edu/Workshops/Faster/

Table 3: Sample HTML content from the historical dataset

Index	HTML Content (truncated)
34050	`<!DOCTYPE html><!--[if IE 8]><html class="ie8...`
28401	`<!DOCTYPE html><html class="no-js" lang="ja"...`
30395	`<!DOCTYPE html><!--[if IE 8]><html class="no-j...`
16411	`<!DOCTYPE html>', '<html itemscope="" itemtype...`
17942	`<!DOCTYPE html>', '', '<html dir="rtl"...`

For the modern dataset, data was collected using Common Crawl and OpenPhish. The

`warcio` library was used to stream 8,000 pages of URL and HTML data from the CC-MAIN-2025-47 Common Crawl crawl. WARC paths were randomly shuffled to ensure a representative sample, with pages shorter than 500 characters filtered out and checkpoints saved every 1,000 pages to prevent data loss. For phishing data, a data collection pipeline was built using the Python `requests` library to fetch live URLs from the OpenPhish feed, with `BeautifulSoup4` used to parse the returned HTML. This ran in a continuous loop until a sufficient number of phishing pages were collected. The modern dataset resulted in a near-balanced class distribution of 51.4% legitimate and 48.6% phishing sites.

Table 4: Modern dataset shape

Column	Size	Total Percentage (%)
Total size	15,571	100
Category spam	7,571	48.6
Category ham	7,999	51.4

Table 5: Sample URLs from the modern dataset - DO NOT CLICK ON THE LINK

Index	URL
58126	https://docs.fieldsquared.com/knowledge-base/u...
58411	https://ediciones.ucm.cl/producto/estrabismo-y...
52329	https://www.guneydogusac.com.tr/pf
50388	https://roblox.com.py/users/1333299643/profile
57282	https://cme.uchicago.edu/em-senior-lecture-cas...

Table 6: Sample HTML content from the modern dataset

Index	HTML Content (truncated)
58126	`<!DOCTYPE html>\r\n <html lang="en-US">...`
58411	`<!doctype html>\n<html lang="es">...`
52329	`<!DOCTYPE html>\n<html><head>...`
50388	`<!DOCTYPE html>\n<!--[if IE 8]> <html class...`
57282	`<!DOCTYPE html>\n<!--[if IEMobile 7]> <html...`

As shown in Table 5, the modern dataset contains varied URLs including potentially suspicious domains such as `roblox.com.py`, which mimics the legitimate `roblox.com` domain — a common phishing technique. Following data collection, WHOIS metadata was retrieved before all sources were merged into a single dataset.

4.1.2 WHOIS Pipeline

WHOIS is a domain lookup protocol that allows users to retrieve publicly available ownership and registration information for a domain name [12]. The `python-whois` library was used to build a data collection pipeline for all URLs. Before querying WHOIS, the `tldextract` library was used to extract the root domain from each URL. A 0.5 second delay was applied between requests to avoid rate limiting. The fields collected included creation date, expiration date, updated date, registrar, name servers, organisation, country, status and emails. As this pipeline queries each domain individually it is computationally expensive to run.

4.1.3 Preprocessing and Feature Engineering

All data sources were merged into a single dataset with a `collected` column added to distinguish historical (`old`) from modern (`new`) entries. Rows with missing domain values were dropped as this indicated WHOIS metadata could not be retrieved. The HTML column was cleaned to remove formatting artefacts introduced during data collection, where some entries had been stored as stringified Python lists rather than raw HTML strings. The following columns were dropped as they were not suitable for modelling: `creation_date` and `expiration_date` — removed due to the datasets spanning different time periods making date comparisons unreliable — and `timestamp` and `source` — removed as these are collection artefacts with no predictive value. Furthermore columns like `status`, `country` and `name_servers` have been changed to binary values with it if it has that value a 1 is assigned and if not 0, with the only field being added is `email_count`, which counts the number of emails.

Table 7: Preprocessed dataset shape and class distribution

Column	Size	Total Percentage (%)
Total size	60,574	100
Category spam	29,888	49.3
Category ham	30,686	50.7
Historical (old)	45,004	74.3
Modern (new)	15,570	25.7

After that the ML dataset feature engineering, extracting 38 numerical features across three categories. With the URL getting engineered to capture lexical and structural characteristics

Table 8: Preprocessed dataset missing values per column

Column	Missing Values	Missing (%)
Category	0	0.0%
HTML	0	0.0%
URL	0	0.0%
Domain	0	0.0%
Registrar	0	0.0%
Name Servers	0	0.0%
Org	0	0.0%
Country	0	0.0%
Status	16,583	27.4%
Has Updated Date	0	0.0%
Has Emails	0	0.0%
Has Status	0	0.0%
Has Org	0	0.0%
Has Country	0	0.0%
Has Name Servers	0	0.0%
Has Registrar	0	0.0%
Email Count	0	0.0%

of the URL string, with previous URL feature engineering explored in other papers getting used [7]. In addition HTML features engineered so they capture the content, including link ratios and script counts which have been shown to differ between phishing and legitimate sites [8]. Lastly the metadata features were converted to binary presence flags given the high rate of missing WHOIS data, with email count retained as a numerical feature to capture additional signal [5].

4.1.4 ML Models

Three ML models were implemented — Random Forest (RF), Logistic Regression (LR) and K-Nearest Neighbours (KNN) — using the `scikit-learn` Python library. RF is an ensemble model that constructs 100 decision trees and combines their predictions through majority voting. LR calculates a weighted sum of all input features and passes the result through a sigmoid function to produce a classification probability. KNN classifies each instance by finding the five most similar training examples and taking a majority vote of their labels. The following hyperparameters were used: RF — 100 estimators; LR — maximum iterations set to 1,000 to ensure convergence; KNN — k=5. `StandardScaler` was applied to the feature matrix prior to training LR and KNN, as both models are

Table 9: Engineered features by category

Category	Count	Features
URL	16	url_length, domain_length, path_length, num_dots, num_hyphens, num_underscores, num_slashes, num_at, num_question_marks, num_equals, num_ampersands, num_digits, num_subdomains, has_ip, has_https, has_port
HTML	14	num_links, num_external_links, num_internal_links, ratio_external_links, num_forms, num_inputs, has_password_input, num_scripts, num_external_scripts, num_iframes, num_images, num_meta_tags, has_title, html_length
Metadata	8	has_updated_date, has_status, has_emails, has_org, has_country, has_name_server, has_registrar, email_count
Total	**38**	

sensitive to feature magnitude — LR due to its weighted sum calculation and KNN due to its Euclidean distance metric. Random Forest requires no scaling as decision tree splits are invariant to feature scale. The dataset was split 80/20 using historical data only for training and testing, with the modern dataset reserved entirely as an unseen test set to evaluate generalisation and answer RQ3 and RQ4.

4.1.5 DL model

The DL model chosen was CNN, using the PyTorch framework to create a three-branch architecture - one per feature: HTML, URL, and Metadata. Each branch will be processed independently as seen in some of the top CNN models [9] and then combined, with the final output being a binary classification.

With each branch having its own unique attribute. The URL branch will use character-level tokenization with a vocab size of 128, and a limit on the maximum character length of 200, this kind of tokenization was used as URLs have no natural word boundaries, hence looking at the individual character makes more sense. While for the HTML branch and Metadata branch word-level tokenization was chosen as there are meaningful words and natural language content in the HTML and the metadata has meaningful registration

terms. Furthermore word-level tokenization has been used in previous research for similar tasks [7]. Table 10 details the configuration of each branch.

Table 10: CNN branch architecture details

Branch	Input	Tokenization	Vocab Size	Max Length	Embedding	Conv Filters	Output
URL	Raw URL string	Character-level	128	200 chars	64	$64 \rightarrow 128 \rightarrow 64$	64
HTML	Raw HTML text	Word-level	10,000	500 words	128	$128 \rightarrow 256 \rightarrow 128$	128
Metadata	Metadata text	Word-level	5,000	100 words	64	$64 \rightarrow 128 \rightarrow 64$	64
Combined output after concatenation							**256**

The individual branches are then combined and trained on the historical dataset split up 80/20 for testing and training before then testing on the modern dataset. With standard hyperparameter values being selected following established practice in text-based CNN literature — Adam optimiser with a learning rate of 0.001 [9], batch size of 64, dropout of 0.5 to prevent overfitting [13], and sigmoid activation with BCELoss for binary classification. Ten training epochs were used due to CPU hardware constraints, with the best model selected by validation loss.

Table 11: CNN training configuration

Parameter	Value
Framework	PyTorch
Loss function	Binary Cross Entropy (BCELoss)
Optimiser	Adam
Learning rate	0.001
Batch size	64
Epochs	10
Dropout	0.5
Activation (output)	Sigmoid
Model selection	Best validation loss
Hardware	CPU

4.1.6 Library and Software Information

All the code was written in Python and the following Software and Libraries were used.

Table 12: Software and library versions used in this project

Software/Library	Version	Purpose
Python	3.9.13	Core programming language
pandas	2.1.1	Data manipulation and analysis
numpy	1.26.0	Numerical computing
scikit-learn	1.5.0	ML model implementation
PyTorch	2.8.0	CNN implementation
beautifulsoup4	4.12.2	HTML parsing
requests	2.32.5	HTTP data collection
warcio	1.7.5	Common Crawl WARC processing
python-whois	0.9.6	WHOIS metadata collection
tldextract	5.3.0	Domain extraction
statsmodels	0.14.6	Statistical significance testing
matplotlib	3.9.4	Results visualisation
seaborn	0.13.2	Statistical visualisation
joblib	1.5.3	Model serialisation
openpyxl	3.1.5	Excel file handling
pyarrow	21.0.0	Parquet file handling

4.2 Verification & Validation

4.2.1 Verification

The following verifications where done for during the software development of this project:

- Initial checks after each pipeline stage - checking row counts, missing values and checking if label distribution matches expectation
- Verified that during preprocessing 370 rows with an missing domain would be removed and comparing the actual results with the expected
- Verified ML dataset is the expected output after the feature engineering.
- Verified the encoding was correct the category (spam $= 1$, ham $= 0$) and binary encoding.
- Verified train/test split was random to insure a fair distribution.
- Verified best model was saved by validation loss not training loss.

4.3 Validation

Functional requirements

The following functional requirements where validated with the evidence shown in the Product section 4 and the Results section 5.

Table 13: Functional requirements validation

Requirement	Status	Evidence
Classify websites as phishing or legitimate	Met	Confusion matrices, accuracy results
Train and evaluate RF, LR and KNN	Met	Results tables (historical and modern)
Train and evaluate CNN	Met	CNN results and training history
Extract URL-based features	Met	Table 9, 16 features extracted
Extract HTML-based features	Met	Table 9, 14 features extracted
Extract binary metadata features	Met	Table 9, 8 features extracted
Evaluate on historical and modern datasets	Met	Separate results tables for each dataset
Produce accuracy, precision, recall, F1, FPR, FNR	Met	Results summary tables
Generate RF feature importance scores	Met	Feature importance plot and CSV
Evaluate each feature category individually	Met	Ablation study results

Non-functional requirements

Table 14: Non-functional requirements validation

Requirement	Status	Evidence
Process 60,000 records within reasonable timeframe on consumer hardware	Met	RF trained in 1.1s, LR in 1.9s, KNN in 0.0s on CPU
Fit scalers on training data only to prevent leakage	Met	StandardScaler fitted on X_train only, applied to test sets separately
Reproducible — version controlled and documented	Met	random_state=42 throughout, all scripts documented
Handle missing metadata values gracefully without crashing	Met	Table 8, zero missing values after preprocessing
Produce all plots and results tables automatically	Met	All plots and CSVs generated automatically by evaluation script
Maintain separate ML and DL datasets to prevent leakage	Met	df_ml.parquet and df_dl.parquet saved separately

5 Results & Evaluation

5.1 Evaluation Process

The following process was evaluated on a dataset containing HTML, URL and metadata features, in the dataset itself there was a split between historical (around 45,004 data entries) and a modern dataset (15,570 data entries). For the evaluation the historical dataset was used for training the models (80% or 36,000 data entries) and then tested on the historical test dataset (20% or 9,000 data entries) and also on the modern dataset (100% or around 16,000 data entries).

The false negative (FN) rate is discussed in greater detail than the false positive (FP), true positive (TP) or true negative (TN) rate, as a missed phishing site poses a greater threat to the target than a false alarm. The evaluation metrics used are defined in Table 15.

Table 15: Evaluation metrics used in this project

Metric	**Description**	**Formula**	**Applied To**
Accuracy	Proportion of correctly classified instances	$\frac{TP+TN}{TP+TN+FP+FN}$	All models
Precision	Proportion of predicted phishing that are truly phishing	$\frac{TP}{TP+FP}$	All models
Recall	Proportion of actual phishing sites correctly detected	$\frac{TP}{TP+FN}$	All models
F1 Score	Harmonic mean of precision and recall	$\frac{2\times Precision\times Recall}{Precision+Recall}$	All models
False Positive Rate (FPR)	Proportion of legitimate sites wrongly blocked	$\frac{FP}{FP+TN}$	All models
False Negative Rate (FNR)	Proportion of phishing sites missed	$\frac{FN}{FN+TP}$	All models
AUC-ROC	Area under ROC curve — overall discriminative ability	—	All models
Average Precision	Area under precision-recall curve	—	All models

Furthermore an ablation study was evaluated on each feature category independently and a degradation analysis was conducted to quantify the performance drop between the historical and modern datasets. In addition, the McNemar test [14] and five-fold cross-validation [15]

were used to assess statistical significance and model robustness respectively.

5.2 Results of Evaluation

The results are split into the following sections: ML Model Performance on Historical Data, CNN Model Performance on Historical Data, Model Generalisation to Modern Data, and Feature Category Analysis.

5.2.1 ML Model Performance on Historical Data

Table 16 shows the performance of each ML model on the historical test set. RF achieved the highest accuracy of 96.84%, outperforming KNN (93.89%) and Logistic Regression (91.58%). RF also recorded the lowest FNR of 2.87%, meaning it missed the fewest phishing sites of the three ML models. Logistic Regression performed the worst across all metrics.

Table 16: ML model performance on historical test data

Model	Accuracy	Precision	Recall	F1	FPR	FNR
Random Forest	0.9684	0.9653	0.9713	0.9683	0.0344	0.0287
KNN	0.9389	0.9452	0.9308	0.9379	0.0531	0.0692
Logistic Regression	0.9158	0.9140	0.9164	0.9152	0.0848	0.0836

These results are further supported by the ROC curve in Figure 2, where RF achieved the highest AUC score of 0.994, followed by KNN and Logistic Regression both at 0.975.

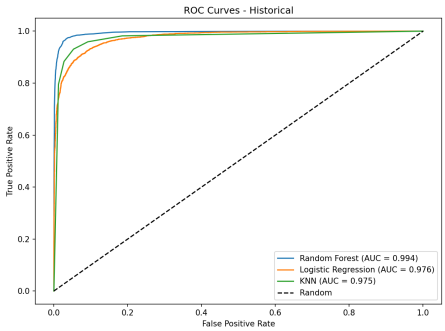

Figure 2: ROC curve of ML models on historical data

The five-fold cross-validation results in Table 17 confirm the stability of these results, with all models recording a standard deviation below 0.004. The McNemar test results in Table 18

further confirm that all pairwise differences between models are statistically significant ($p < 0.0001$).

Table 17: Five-fold cross-validation results

Model	**Mean F1**	**Std F1**
Random Forest	0.9673	0.0011
KNN	0.9028	0.0014
Logistic Regression	0.8961	0.0037

Table 18: McNemar test results for pairwise model comparisons

Model A	**Model B**	**p-value**
Random Forest	Logistic Regression	<0.0001
Random Forest	KNN	<0.0001
Logistic Regression	KNN	<0.0001
Significance threshold: $\alpha = 0.05$		

5.2.2 CNN Performance on Historical Data

Table 19 shows the CNN achieved the highest accuracy of all models on the historical dataset at 98.96%, with an AUC score of 0.999 and an FPR of just 0.29%. The training and validation curves in Figure 3 show a consistent downward trend in loss and stable accuracy across all 10 epochs, with training and validation accuracy remaining closely aligned, indicating no signs of overfitting.

Table 19: CNN performance on historical data

Dataset	**Accuracy**	**Precision**	**Recall**	**F1**	**FPR**	**FNR**	**AUC**
Historical	0.9896	0.9970	0.9819	0.9894	0.0029	0.0181	0.9990

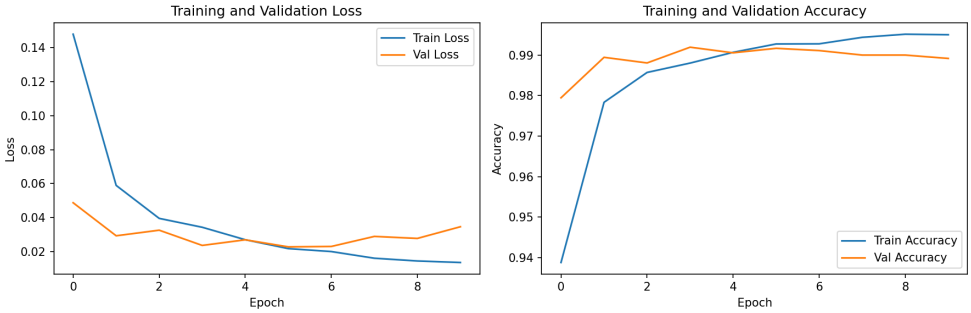

Figure 3: CNN training and validation loss and accuracy over 10 epochs

5.2.3 Model Generalisation to Modern Data

All models experienced a significant drop in performance when evaluated on the modern dataset, as shown in Table 20. The ML models recorded accuracies between 65.26% and 67.68%, while the CNN dropped to 51.01% — close to random chance. Notably, the CNN recorded an FPR of 93.37%, meaning it classified the vast majority of legitimate sites as phishing, while maintaining a low FNR of 2.09%, indicating it predicted almost everything as phishing regardless of the actual label.

Table 20: Model performance on modern data

Model	Accuracy	Precision	Recall	F1	FPR	FNR
Random Forest	0.6768	0.6763	0.6430	0.6592	0.2913	0.3570
Logistic Regression	0.6696	0.6952	0.5706	0.6268	0.2368	0.4294
KNN	0.6526	0.6909	0.5167	0.5912	0.2188	0.4833
CNN	0.5101	0.4981	0.9791	0.6603	0.9337	0.0209

The drop in AUC scores further confirms this degradation. RF recorded the highest AUC on modern data at 0.721, followed by Logistic Regression at 0.711 and KNN at 0.653, compared to the CNN which dropped to 0.536 — barely above random. The ROC curves for the modern dataset are shown in Figure 5 and Figure 6.

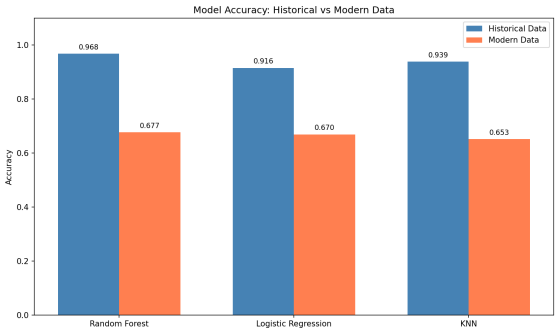

Figure 4: Model accuracy comparison: historical vs modern data

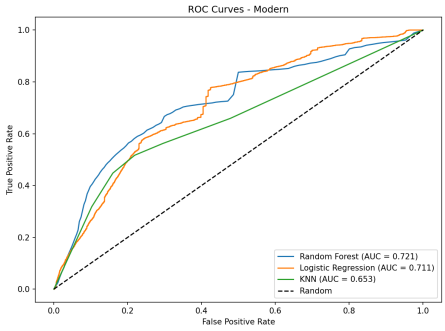

Figure 5: ROC curve of ML models on modern data

As shown in Table 21, the CNN experienced the largest degradation of 48.45%, while Logistic Regression recorded the smallest drop of 26.89%. RF and KNN dropped by 30.12% and 30.49% respectively.

Table 21: Performance degradation from historical to modern data

Model	**Historical**	**Modern**	**Absolute Drop**	**Drop (%)**
Random Forest	0.9684	0.6768	0.2917	30.12%
Logistic Regression	0.9158	0.6696	0.2462	26.89%
KNN	0.9389	0.6526	0.2863	30.49%
CNN	0.9896	0.5101	0.4795	48.45%

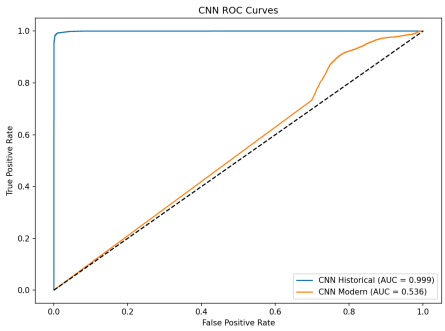

Figure 6: ROC curve of CNN model on modern data

5.2.4 Feature Category Analysis

Figure 7 shows the accuracy of each ML model across individual feature categories. HTML features scored the highest mean accuracy of 0.973 across all three ML models, followed by URL features at 0.870 and metadata features at 0.685. This pattern was consistent across RF, LR and KNN.

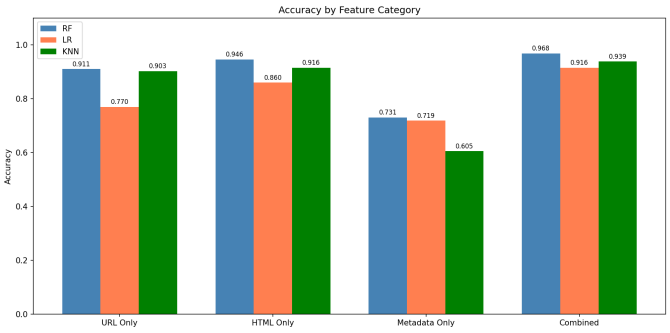

Figure 7: Accuracy by feature category across ML models

The ablation study in Table 22 and Figure 8 shows the RF model performance across all feature combinations. HTML alone achieved 94.61% accuracy, outperforming URL alone at 91.06% and metadata alone at 73.11%. Adding metadata to either HTML or URL provided no meaningful improvement — HTML + Metadata (94.55%) performed marginally below HTML alone (94.61%). The full feature combination achieved the highest accuracy of 96.84%, confirming that all three categories contribute complementary signal.

Figure 9 shows the top 15 feature importances from the RF model. `html_length` was the most important feature with an importance score of 0.175, followed by `num_links`. The highest ranked metadata feature was `has_updated_date`, while `path_length` was the most

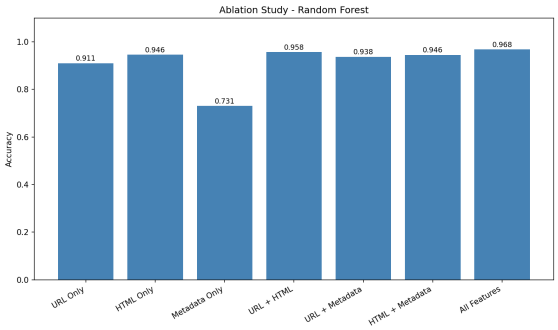

Figure 8: Ablation study results for Random Forest

Table 22: Ablation study results — Random Forest

Feature Combination	Features	Accuracy	F1
Metadata Only	7	0.7311	0.7248
URL Only	16	0.9106	0.9090
HTML Only	14	0.9461	0.9452
URL + Metadata	23	0.9376	0.9367
HTML + Metadata	21	0.9455	0.9448
URL + HTML	30	0.9578	0.9573
All Features	38	0.9684	0.9683

important URL feature.

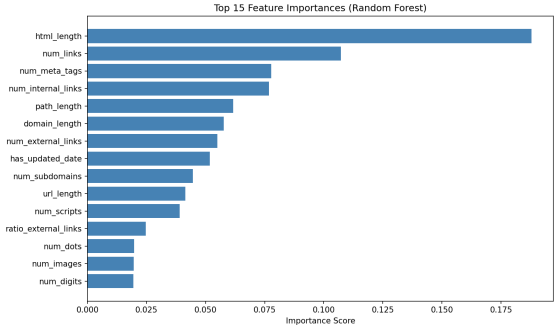

Figure 9: Top 15 feature importances from Random Forest

5.3 Returning to the Research Questions

5.3.1 RQ1

How effectively can machine learning models detect phishing websites using a combination of URL, HTML, and metadata features? All the models (RF, LR, KNN, CNN) are effective when training and testing on the historical dataset, with all models achieving an accuracy higher than 91%. The CNN model performed the best with an accuracy of 98%. Given the correct training data, there is enough evidence to conclude that combining HTML, URL and metadata features can effectively train classification models to detect phishing from legitimate websites.

5.3.2 RQ2

Which feature categories contribute most to detection performance?

The best single feature category was HTML, scoring as high as 0.946 in accuracy for the RF model, as seen in figure 7. With the RF model trained on the URL and metadata feature datasets getting only 0.911 and 0.731 respectively, this shows how HTML is the strongest single feature. URL also performed well, while metadata performed the most poorly on its own, with no model achieving a higher accuracy than 74%. When combining two features there were improvements, with all combinations achieving the same accuracy or higher, with the URL and HTML combination achieving an accuracy score of 0.958, while using all features achieves an accuracy of up to 0.968.

Furthermore, when looking at the individual features of the dataset as seen in figure 9, `html_length` scored the highest feature importance in the RF model, with 8 out of the top 15 features coming from the HTML dataset. This demonstrates that HTML is the most important feature category, followed by URL, with metadata performing the poorest.

5.3.3 RQ3

How well do models trained on historical datasets generalise to modern phishing data?

Table 20 provides evidence that models trained on historical data perform poorly when

tested on a modern phishing and legitimate dataset. The highest achieving accuracy model (RF) only achieved a score of 0.6768. Furthermore, the CNN model performed significantly worse with an FPR of 0.9337, rendering it ineffective as it classifies the vast majority of legitimate sites as phishing. Additionally, this degradation is consistent across all models, with every metric — Precision, Recall, F1, FPR, FNR and AUC — scoring lower on the modern dataset, providing clear evidence that models trained on historical data generalise poorly to modern phishing data.

5.3.4 RQ4

To what extent does detection performance degrade when evaluated on newly collected real-world phishing websites? Table 21 provides evidence of the extent to which performance degrades. LR achieved the lowest relative percentage drop of 26.89% across all models, while the CNN, the top performing model on the historical dataset, had the highest drop of 48.45%. Notably, whilst all three ML models fell within a degradation range of 26.89% to 30.49%, the CNN dropped by 48.45%, nearly 20 percentage points more than the worst performing ML model. This suggests that whilst the CNN learns better from raw text, it is more susceptible to shifts in distribution from historical to modern data, compared to the feature engineered ML models. This is clear evidence that detection performance degrades significantly on modern data.

6 Discussion & Reflection

6.1 Interpreting the Results

The results demonstrate that HTML, URL and metadata can be effectively used to train DL and ML models. With the RF model achieving a 96.84% accuracy, which as seen in Safi & Singh's systematic literature review puts it close to some of the top ML models [3]. While the KNN and LR models reached a similar expected performance seen in other research mentioned in this report, with all models achieving at least 91.6% accuracy when trained and evaluated on the historical dataset.

Further inspection of the ML models shows how HTML engineered features produced the strongest performance on its own compared to the other feature categories, scoring higher accuracy with URL close behind and the metadata performing the worst. Results improved when combining two or more feature categories together. This was expected as other papers such as Opara et al. [8] provide similar evidence that HTML and URL categories are effective at classifying sites.

The CNN model achieved the highest accuracy of 98.96% when trained and evaluated on the historical data. This was expected as existing literature has concluded that CNN achieves the highest accuracy when comparing DL and ML models for phishing site detection [3]. With this paper's CNN performing at a similarly high accuracy to state of the art models [8, 9] when being trained on HTML, URL and metadata.

When comparing the historically trained models on modern data, some degradation was expected, however all models experienced significant performance losses. The ML models dropped by approximately 30%, with RF achieving the highest accuracy amongst them. In contrast, the CNN model, which performed best on the historical dataset and has been shown in other research to perform very well, experienced an accuracy drop of approximately 48%. This may be attributed to the CNN's character and word-level tokenisation learning patterns specific to historical phishing data, whereas the ML models rely on deliberately engineered features such as URL structure and HTML link ratios that remain more stable indicators of phishing behaviour regardless of when the data was collected. This suggests that ML models are more robust against data degradation, while the CNN is more

susceptible to it, with the model producing an FPR of approximately 93%, severely limiting its practical application.

For real-world deployment, this highlights the importance of frequently retraining state of the art models on the most recent data, as failure to do so risks the model becoming ineffective against modern phishing attacks.

6.2 Reflection

The greatest insight from this dissertation was the high degradation of each model when looking at the modern data, with the CNN performing the worst out of all of them. This should have been explored in greater depth, comparing more DL models with the ML models to see if there was a similar impact as for the CNN. Another worthwhile exploration could have been comparing the differences between the older and newer HTML. Additionally, more time should have been allocated to the collection of modern phishing sites, as more data was planned to be collected but was slowed down due to limited tools available for live phishing data collection.

6.3 Challenges

The initial challenge of this project was the data collection and creation of the pipelines. Although a good amount of data was already collected, software tools needed to be researched, built and tested to ensure high quality and relevant data to answer the RQs. Furthermore, a significant amount of data cleaning needed to be done and tested to ensure data integrity was maintained. Lastly, all the metadata needed to be collected, with research required on how to most effectively achieve this, with the WHOIS library being chosen. Although it could collect most of the data for each URL, rate limiting resulted in a slow pipeline. Furthermore, there were issues with missing metadata values, with the WHOIS status value missing 27% of entries, which then needed to be handled gracefully.

6.4 Limitations

For this project there were three notable limitations that affected performance. The first is that more data could have been collected to achieve better results. More data could have

improved model performance, with more historical and modern data being used to train each model. Furthermore, the legitimate data used is assumed to be legitimate, as it was randomly collected from the internet. Although there is evidence that a notable proportion of sites on the internet are suspicious or malicious, this assumption needed to be made as it was not feasible to verify all of them. Another limitation was that the system was constrained by the hardware available. With one of the non-functional requirements being that training and testing shall be completed within a reasonable time-frame on consumer hardware, this especially limited the CNN model, which was only trained for 10 epochs.

6.5 Future Work

Future work should look at degradation and ways to effectively mitigate it for both ML and DL models. As there is evidence that the CNN degraded the worst whilst also being the top performing model on historical data, degradation is a serious threat to real-world deployment. One concrete direction would be periodic retraining on newly collected modern phishing data to reduce distribution shift over time. Additionally, HTML, URL and metadata features should be explored in greater depth, as this combination resulted in strong models with each feature category improving overall results. Furthermore, mixing historical and modern data during training should be explored to evaluate the impact on degradation. Finally, transformer-based models should be investigated, as they may capture more complex patterns in raw HTML and URL text compared to the CNN architecture used in this project.

7 Conclusion

This paper investigated the use of HTML, URL and metadata features to classify websites as either phishing or legitimate, using three ML models — Random Forest, Logistic Regression and K-Nearest Neighbours — and a CNN model. Additional data pipelines were built to collect modern phishing data and WHOIS metadata for all entries.

The results demonstrated that combining HTML, URL and metadata features is an effective approach for phishing detection, with all models achieving at least 91.6% accuracy on the historical dataset. HTML was identified as the strongest individual feature category, followed by URL, with metadata performing the poorest on its own. The CNN achieved the highest accuracy of 98.96% on historical data, consistent with existing literature.

The most significant finding was the substantial degradation all models experienced when evaluated on modern data. ML models dropped by approximately 27–30%, while the CNN dropped by approximately 48%, despite being the top performing model on historical data. This demonstrates that ML models are more robust to distribution shift, while CNN models are more susceptible to it. For real-world deployment, this highlights the critical importance of retraining models frequently on the most recent data, particularly for state of the art DL models, to remain effective against modern phishing attacks. This highlights the risk of relying on older trained models in real-world deployment without the continuous retraining of the model with the most recent data.

References

[1] UK National Cyber Security Centre. Phishing scams — guidance and overview. https://www.ncsc.gov.uk/collection/phishing-scams, 2025. Accessed: 2026-03-20.

[2] UK Cabinet Office. The cost of cyber crime, 2011.

[3] Asadullah Safi and Satwinder Singh. A systematic literature review on phishing website detection techniques. *Journal of King Saud University - Computer and Information Sciences*, 35(2):590–611, 2023.

[4] Himanshu Gautam, Vishal Kumar, and Vinamra Sharma. Phishing prevention techniques: Past, present and future. In Krishan Kant Singh Mer, Vijay Bhaskar Semwal, Vishwanath Bijalwan, and Rubén González Crespo, editors, *Proceedings of Integrated Intelligence Enable Networks and Computing*, pages 83–98, Singapore, 2021. Springer Singapore.

[5] S. Das Guptta, K. T. Shahriar, H. Alqahtani, D. Alsalman, and I. H. Sarker. Modeling hybrid feature-based phishing websites detection using machine learning techniques. *Annals of Data Science*, pages 1–26, 2022. Epub ahead of print. PMID: 40479161; PMCID: PMC8935623.

[6] Sakib Shahriar Shafin. An explainable feature selection framework for web phishing detection with machine learning. *Data Science and Management*, 8(2):127–136, 2025.

[7] Brij B. Gupta, Krishna Yadav, Imran Razzak, Konstantinos Psannis, Arcangelo Castiglione, and Xiaojun Chang. A novel approach for phishing URLs detection using lexical based machine learning in a real-time environment. *Computer Communications*, 175:47–57, 2021.

[8] Chidimma Opara, Yingke Chen, and Bo Wei. Look before you leap: Detecting phishing web pages by exploiting raw url and html characteristics. *Expert Systems with Applications*, 236:121183, 2024.

[9] Yahia Said, Ahmed A. Alsheikhy, Husam Lahza, and Tawfeeq Shawly. Detecting phishing websites through improving convolutional neural networks with self-attention mechanism. *Ain Shams Engineering Journal*, 15(4):102643, 2024.

[10] Roger S. Pressman and Bruce R. Maxim. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 8th edition, 2014.

[11] R. Dhirr. Look for mallory: Web page phishing detection dataset. https://www.kaggle.com/datasets/rmdhirr/look-for-mallory, 2023. Accessed: 2026-03-30.

[12] WHOIS. Whois lookup. https://www.whois.com/whois/. Accessed: 2026-03-30.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[14] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2009.

A Appendix

A.1 User Guide

The full source code and data for this project can be found at: https://github.com/HerrNiklasLange/Machine-Learning-System-to-predict-phishing-sites-using-HTML-URL-metadata

A.1.1 Prerequisites

The following software and libraries are required to run this project. All library versions are listed in Table 12.

- Python 3.9.13
- All dependencies can be installed by running:

```
pip install -r requirements.txt
```

A.1.2 Project Structure

The project is organised as follows:

A.1.3 Project Structure

```
thesis/
 data_collection_raw/   # Raw data collection scripts
 data_merged/           # Merged historical and modern dataset
 data_new/              # Modern collected dataset
 data_old/              # Historical dataset
 DL_model/              # CNN model scripts
 ML_model/              # ML model scripts (RF, LR, KNN)
 models/                # Saved trained models
 plots/                 # Generated evaluation plots
 pre_processing/        # Preprocessing and feature engineering
 requirements.txt       # Required dependencies
```

Note: Each folder contains a `README.txt` file with further instructions specific to that stage of the pipeline. Furthermore all file paths in the scripts are absolute and will need to be updated to match your local directory structure before running.

A.1.4 Running the Pipeline

The pipeline should be run in the following order:

1. Run data collection scripts in `data_collection_raw/`
2. Run preprocessing scripts in `pre_processing/`
3. Train ML models by running scripts in `ML_model/`
4. Train CNN model by running scripts in `DL_model/`

A.1.5 Expected Outputs

- Trained models saved to `models/`
- Evaluation plots saved to `plots/`
- Results tables generated automatically as CSV files

A.1.6 Known Issues

- WHOIS rate limiting may significantly slow metadata collection
- CNN training is CPU-bound and may take several hours
- A checkpoint file is saved every 1,000 pages during data collection to prevent data loss

A.2 List of abbreviations used in this dissertation

Table 23: List of abbreviations used in this dissertation

Abbreviation	Full Term
AUC	Area Under the Curve
BCELoss	Binary Cross Entropy Loss
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma-Separated Values
DL	Deep Learning
DNS	Domain Name System
F1	F1 Score (Harmonic Mean of Precision and Recall)
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
KNN	K-Nearest Neighbours
LR	Logistic Regression
ML	Machine Learning
NCSC	National Cyber Security Centre
NLP	Natural Language Processing
RF	Random Forest
ROC	Receiver Operating Characteristic
RQ	Research Question
TN	True Negative
TP	True Positive
TLD	Top-Level Domain
URL	Uniform Resource Locator
WARC	Web ARChive
WHOIS	Who Is (domain registration protocol)