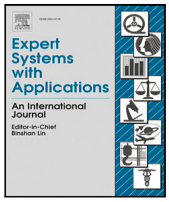
Expert Systems With Applications 236 (2024) 121183

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Look before you leap: Detecting phishing web pages by exploiting raw URL and HTML characteristics

Chidimma Opara [a,*], Yingke Chen [b], Bo Wei [c]

[a] School of Computing, Engineering & Digital Technologies, Teesside University, Middlesbrough, TS1 3BX, UK
[b] Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, NE7 7YT, UK
[c] School of Computing, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK

ARTICLE INFO

Keywords:
Web pages
Phishing detection
Deep neural networks
HTML
URL

ABSTRACT

Phishing websites distribute unsolicited content and are frequently used to commit email and internet fraud. Detecting them before any user information is submitted is critical. Several efforts have been made to detect these phishing websites in recent years. Most existing approaches use hand-crafted lexical and statistical features from a website's textual content to train classification models to detect phishing web pages. However, these phishing detection approaches have limitations, including (1) the tediousness of extracting hand-crafted features, which require specialized domain knowledge to determine which features are useful for a particular platform; and (2) the difficulties encountered by models built on hand-crafted features to capture the semantic patterns in words and characters in URL and HTML content. To address these challenges, this paper proposes WebPhish, an end-to-end deep neural network trained using embedded raw URLs and HTML content to detect website phishing attacks. First, the proposed model automatically employs an embedding technique to extract the corresponding characters into homologous dense vectors. Then, the concatenation layer merges the URL and HTML embedding matrices. Following that, Convolutional layers are used to model its semantic dependencies. Extensive experiments were conducted with real-world phishing data, which yielded an accuracy of 98.1%, showing that WebPhish outperforms baseline detection approaches in identifying phishing pages.

1. Introduction

Phishing has recently become a preferred method of attack for cybercriminals due to its low cost and limited technical skill requirements. The majority of phishing attacks begin with spam emails. Frequently, these emails contain links to phishing web pages. In April 2020 alone, Gmail intercepted over 100 million spam emails daily, including 18 million COVID-19 pandemic phishing attacks.[1] Due to the magnitude of this cyber-attack, industrial and academic experts have put much effort into combating phishing and invented various anti-phishing solutions.

Recent research in phishing detection approaches has resulted in the rise of multiple technical methods, such as augmenting password logins (Chattaraj et al., 2018) and multi-factor authentication (Acar et al., 2013). However, these techniques are usually server-side systems that require the Internet user to correspond with a remote service, which adds further delay in the communication channel. Another popular phishing detection system that relies on a centralized architecture is the phishing blacklist and whitelist methods (Google, 2019). A URL visited by an internet user will be compared with the URL in these lists in real time. Although the list-based methods tend to keep the false positive rate low, however, a significant shortcoming is that the lists are not exhaustive, and they fail to detect zero-day phishing attacks. To mitigate these limitations, researchers have developed several anti-phishing techniques using machine learning models as they are mostly client-side based and can generalize their predictions on unseen data.

Detecting web page phishing using machine learning typically involves extracting an appropriate feature representation from the web page component and training machine learning-based prediction models on that representation. There has been extensive research on lexical features, host-based features, content-based features, and even context and popularity-based features (Buber et al., 2017; Gutierrez et al., 2018; LeCun et al., 2015). While the strategies outlined above have demonstrated success, they do have some limitations: (1) The inconvenient nature of extracting hand-crafted features, which require specialized domain knowledge to determine which features are useful for a given platform; (2) The need to continuously update the hand-crafted features in order to remain relevant when faced with new phishing techniques.

* Corresponding author.
E-mail addresses: c.opara@tees.ac.uk (C. Opara), yingke.chen@northumbria.ac.uk (Y. Chen), bo.wei@newcastle.ac.uk (B. Wei).

[1] J. Tidy, "Google blocking 18 m coronavirus scam emails every day", https://www.bbc.co.uk/news/technology-52319093, accessed: 2020-04-20.

https://doi.org/10.1016/j.eswa.2023.121183
Received 6 October 2020; Received in revised form 9 August 2023; Accepted 10 August 2023
Available online 16 August 2023

Additionally, from our literature review, we found that existing research has commonly studied the performance of models using just raw URLs (Le et al., 2018) and (Bahnsen et al., 2017) or HTML content (Opara et al., 2020) when detecting phishing on web pages using deep learning models. However, based on our research and experiments, we reveal that the information in different parts of a web page can provide different characteristics when detecting phishing. For example, the HTML content of the page gives the semantic and structural characteristics, and the URL can provide insight into leveraging the web page's Internet address. Therefore, leveraging the learned semantic, lexical, and syntactic ambiguities from URL and HTML content to detect phishing is essential.

The objective of this paper is:

1. To accurately detect phishing web pages with minimal false positives and false negatives using only the URL and HTML's raw content on a deep learning model.
2. To provide a phishing detection model that is not reliant on third-party systems (such as search engines and optical readers) and databases (such as blacklists/whitelists).
3. To develop a web page phishing detection model that does not rely on hand-crafted features, which needs extensive domain knowledge.

Consequently, we propose WebPhish, an end-to-end deep neural network model that detects phishing attacks using raw URLs and HTML content. First, we employ an embedding technique to generate homologous dense vectors from the corresponding URL characters and HTML words. The concatenation layer then combines the URL and HTML embedding matrices. Convolutional layers are then used to model the semantic relationships between the words and characters. Specifically, WebPhish uses an embedding technique to generate homologous dense vectors from the corresponding characters in the URL and words in the HTML. The concatenation layer then combines the URL and HTML embedding matrices. Following that, convolutional layers are used to model the semantic relationships between the words. To enable ease of verification and replicability, we have made the dataset available to other researchers interested in this topic.

The following are the main contributions of this work:

- This paper proposes WebPhish, which uses only the raw content of the URL and HTML document of a web page to train a deep neural network model for phishing detection. Manual feature engineering is reduced as WebPhish learns the representation in the features of the HTML document, and we do not depend on any third-party system. Our proposed approach takes advantage of the word and character embedding matrix to present a phishing detection model that automatically accommodates new web content and is, therefore, easily applied to test data.
- A thorough empirical evaluation based on collected real-world data yields results that show that the proposed model significantly outperforms state-of-the-art methods demonstrating the validity of our approach.
- Furthermore, we carried out an ablation study on the efficiency of the proposed model on other textual datasets to demonstrate its adaptability and flexibility beyond the proposed domain.

The remainder of the paper is organized as follows: the next section provides an overview of related works on proposed techniques for detecting phishing on web pages. Section 3 gives an in-depth description of our proposed model, while Section 4 elaborates on the dataset collection and evaluation metrics used to analyze WebPhish. The detailed results of the evaluations of our proposed model are in Section 5. Section 6 discusses the research conducted and the impact of the DNN in the proposed model. Finally, we conclude our paper in Section 7.

2. Related works

This section surveys the state-of-the-art techniques for detecting phishing web pages using manual feature engineering and automatic feature extraction techniques applied to machine-learning algorithms.

2.1. Phishing detection using handcrafted features and machine learning algorithms

The techniques reviewed in this section are based on the assumption that the infrastructure of phishing pages differs from that of legitimate pages. As a result, extracting specific lexical and statistical features from web pages is expected to differentiate phishing web pages from legitimate ones. The referred features include URLs, HTML, and DNS commonly found on web pages (Amrutkar et al., 2017). Recently, NLP-based features such as Bag of Words, ngrams, and TF-IDF have been generated from web page contents to determine their legitimacy (Buber et al., 2017; Gutierrez et al., 2018; LeCun et al., 2015; Rendall et al., 2020). In most published state-of-the-art approaches, these extracted features are used by machine learning algorithms to classify a web page as phishing or legitimate.

Most published state-of-the-art in this section extract bespoke statistical features from the web page components, while others use a combination of existing features while adding a few of theirs. The studies by Kumi et al. (2021), Mohammad et al. (2012a), Rendall et al. (2020), and Smadi et al. (2018) proposed new statistical features to detect phishing web pages.

Specifically, Kumi et al. (2021) extracted eight features from web page content to a Classification Based on Association Rules Algorithm (CBA) to detect phishing web pages. These features include the number of special characters, sensitive words in a URL, and the entropy of the domain name. Evaluations of 700 phishing and 500 legitimate URLs yielded an accuracy of 95.8%. To provide a more in-depth study of the type and number of features that can be extracted from a web page, Mohammad et al. (2012a) developed a neural network model that automatically adapts the network structure to the ever-changing features needed to determine the status of a web page (Mohammad et al., 2012b). The model had 17 features, including the presence of the <a> tag and the ability to request URLs in a domain other than the one entered in the address bar. It was evaluated on 600 legitimate and 800 phishing websites, yielding an accuracy of 92.18% in over 1000 epochs.

Rendall et al. (2020) proposed a multi-layered approach to detect phishing web pages based on 13 DNS-based features and multiple supervised machine learning algorithms. Their method, which included a sensors module (for data collection and preprocessing), a detection engine (with a supervised learning algorithm), and a triage module (for comparison with predefined thresholds), was tested on 17,244 legitimate and 7970 phishing domains, yielding an accuracy of 89% using the SVM algorithm. Also, Yerima and Alzaylaee (2020) proposed a phishing detection approach that took 30 static features from a web page's URL and HTML content and applied them to 1D convolutional neural networks. Experiments on 6,157 legitimate and 4,898 phishing websites achieved an accuracy of 98.2% and an F1-score of 97.6%.

Additionally, some studies focused on detecting malicious domains, such as the study by Maroofi et al. (2020), who proposed COMAR (Classification of COmpromised versus MAliciously Registered Domains), a system capable of distinguishing compromised domains from malicious ones. COMAR uses 38 features to determine the state of a domain. The COMAR model was applied to 41,000 phishing URLs using a random forest classifier, yielding a 97% accuracy.

Some authors focused on evaluating existing features on various machine learning algorithms. Moghimi and Varjani (2016) used nine features derived from a subset of related studies (Aburrous et al., 2010; Lakshmi & Vijaya, 2012; Zhang & Yuan, 2012) and eight new attributes that indicate the relationship between the composition of a web page

and its URL. The proposed features were derived using the Levenshtein distance algorithm (Yujian & Bo, 2007), which approximately identifies similarities in textual string patterns. Utilizing a dataset comprising 1448 phishing and 686 legitimate websites, the algorithm was classified using SVM, giving an accuracy of 99.14%. Chiew et al. (2019) proposed the Hybrid Ensemble Feature Selection (HEFS) framework for machine learning-based phishing detection systems, which were implemented on a Random Forest classifier. HEFS is more computationally efficient and can determine the optimal number of features for a given dataset. For the UCI phishing dataset, their approach achieved an accuracy of 96.17%.

Singh et al. (2015) implemented the Adaline network and backpropagation algorithm with an SVM and neural network on 15 features based on a related study (Mohammad et al., 2012a). The test was performed on a dataset of 179 phishing URLs and 179 legitimate URLs. The authors experimentally demonstrated that the Adaline network with SVM performed better than the backpropagation algorithm, with a prediction accuracy of 99.14%.

Aljofey et al. (2022) extracted features represented by URL character sequences which are combined and fed to train the XGBoost classifier. In particular, they extracted character-level Term Frequency-Inverse Document Frequency (TF-IDF) features from noisy parts of HTML and plaintext of 60,252 web pages to validate the proposed solution. This data contains 32,972 benign web pages and 27,280 phishing web pages. The proposed approach achieved an accuracy of 96.76%.

One of the challenges of phishing detection using machine learning algorithms with handcrafted features is their difficulty in extrapolating to new data, and attackers might be aware of the specific features the model is trained on and can easily bypass it. To mitigate this challenge, Smadi et al. (2018) proposed a neural network model that can adapt to the dynamic nature of phishing emails using reinforcement learning. The proposed model can handle zero-day phishing attacks and mitigate the problem of a limited dataset using an updated offline database. Their experiment yielded a high accuracy of 98.63% for 50 features extracted from a dataset of 12,266 emails.

To explore the application of fuzzy logic in detecting phishing web pages, Barraclough et al. (2013) built a model that combines fuzzy logic and neural networks to expose phishing in online transactions. The novelty combines 288 features from a user-behavior profile, legitimate website rules, PhishTank, and pop-ups from emails. These attributes were trained using supervised learning and yielded an accuracy of 98.5%. Although the authors stated that their model outperformed Netcraft,[2] and Cantina+ (Zhang et al., 2007) it is highly complex and resource-intensive. It is also heavily dependent on manual feature engineering, which is time-consuming.

Some researchers have shifted their focus from desktop to mobile web page security. Their research revealed that web experience on mobile phones differs functionally and structurally from desktop computers. These distinctions are primarily due to the mobile-specific features and capabilities to improve user experience. Amrutkar et al. (2017) proposed the KAYO. This binary classification algorithm distinguishes between legitimate and phishing mobile-specific web pages in real-time. The KAYO model employs machine learning techniques and heuristics derived from the page sources of mobile web pages, URLs, and mobile-specific facilities. Forty-four features were extracted and used in a binomial logistic regression machine-learning technique. KAYO achieved 89% TPR, 8% FPR, and 90% accuracy.

Despite their high accuracy, the proposed algorithms that employ feature engineering techniques have a few limitations: (1) the inconvenient nature of manual feature engineering techniques, which require specialized domain knowledge to determine which features are useful for a given platform; and (2) the challenges faced by models built on predefined features when confronted with new data, as predefined features struggle to extrapolate to new data.

Furthermore, some of the proposed algorithms depend on the content of the associated legitimate web page, as phishers are expected to reproduce legitimate web page content and add handlers to save user data in their preferred repository (Moghimi & Varjani, 2016). This may not be the case, as cybercriminals can easily design a website that looks like a legitimate website without completely replicating its content.

2.2. Phishing detection using automatic feature selection on deep neural networks

Deep Neural Networks (DNN) use layers of stacked nonlinear projections to learn representations of multiple levels of abstraction. The state-of-the-art approaches reviewed in this section employ DNN algorithms to automatically learn the salient features in the web pages to detect phishing.

Bahnsen et al. (2017) proposed a phishing classification scheme that uses only the URLs of a web page as input and implements the model on an LSTM network. The results yielded a 98.7% accuracy on a corpus of two million phishing and legitimate URLs. The authors compared their results with another model that implemented random forest (RF) on 14 lexical and statistical features extracted from the URLs. The LSTM model outperformed the latter model by 5% across all metrics evaluated (accuracy, precision, recall and f1-score). Although RF had a faster runtime than the LSTM network, the authors demonstrated that the LSTM model does not require complete content analysis.

Although the model proposed by Wei et al. (2019) is very similar to the approach proposed by Bahnsen et al. (2017) in that they both use unprocessed URLs as input, the former employs novel word-embedding techniques for automatic feature representation. These features were then applied to convolutional filters to detect phishing URLs. Their experimental results on a dataset of 999,996 legitimate URLs and 523,970 phishing URLs yielded an accuracy of 86.6%.

Le et al. (2018) proposed URLNet, a deep learning model that concatenates character-and character-level word embeddings as input and applies them to convolutional layers. URLNet was evaluated by its application to 5 million URLs, yielding an accuracy of 97.2%.

The model proposed by Ozcan et al. (2021) amalgamated manual and automatic feature extractions. Their hybrid deep learning model uses character embedding representation with 28 NLP features as input to a DNN + LSTM model. Their evaluations on a dataset of 37385 phishing URLs and 36,400 legitimate URLs yielded an accuracy of 98.79% and an F1 score of 98.81%.

Zhang et al. (2021) developed MultiPhish, a phishing detection model that exploits a Variational autoencoder to fuse the text, image, and URL feature information of web pages with neural networks. The proposed model was applied to a dataset of 3887 phishing and 4259 legitimate instances, yielding an accuracy of 97.79%. MultiPhish can detect web pages hosted in compromised domains by adding URL features.

Tang and Mahmoud (2021) extracted character-level features from the URL, which were then applied to a gated recurrent unit (GRU) neural network to determine the maliciousness of the URL. Application of the model on 429, 125 legitimate URLs and 236,362 phishing URLs yielded an accuracy of 99.18%.

A summary of these studies is provided in Table 1.

The majority of the above-mentioned state-of-the-art features are hand-crafted. The only work similar to ours is Le et al. (2018)'s discussion of using a URL's character embedding on convolutions. Our solution leverages the character and word embedding layer to automatically learn vector representations of a web page's URL and HTML content to detect phishing attacks without requiring expert feature engineering. Additionally, as discussed above, our approach concatenates the output of the embedding layer before presenting it for convolutions, preserving the original HTML and URL content.

[2] https://www.netcraft.com

Table 1
Analysis of phishing detection by machine learning based methods using manual and automated features.

Authors	Method	Dataset	Performance
Machine learning based methods using manual feature engineering			
Kumi et al. (2021)	The authors applied 8 features to the CBA algorithm. The measure of the randomness factor in URLs was the most important feature.	700 phishing and 500 legitimate websites.	95.8% accuracy
Moghimi and Varjani (2016)	The authors applied 9 features to the SVM algorithm.	1448 phishing and 686 legitimate websites.	99.14% accuracy
Singh et al. (2015)	15 features from the URL, HTML and networks were implemented on Adaline with SVM.	79 phishing and 179 legitimate URLs.	99.1% accuracy.
Mohammad et al. (2012b)	17 features were implemented on a self-learning neural network.	600 legitimate websites and 800 phishing websites.	92.18% accuracy.
Chiew et al. (2019)	48 features were implemented on a Random Forest Classifier.	2456 legitimate and phishing instances.	96.17% accuracy.
Barraclough et al. (2013)	Used fuzzy logic to derive 288 features implemented on supervised machine learning algorithms.	Not stated.	98.5% accuracy on the Legitimate site rules.
Rendall et al. (2020)	Used a multi-layered approach implemented on supervised machine learning algorithms.	17,244 legitimate and 7970 phishing domains.	89% accuracy on the SVM algorithm.
Yerima and Alzaylaee (2020)	This study applied 30 static features from the URL and HTML content of the web page to a 1D convolutional neural network.	6,157 legitimate and 4,898 phishing websites.	98.2% accuracy and F1-score of 97.6%.
Smadi et al. (2018)	This study applied 50 features from the URL and HTML content of emails to a neural network with reinforcement learning.	12,266 emails.	98.63% accuracy.
Maroofi et al. (2020)	38 features from the URL were applied to a random forest classifier to detect compromised domains.	41,002 URLs.	97% accuracy.
Aljofey et al. (2022)	Extracted character-level Term Frequency-Inverse Document Frequency (TF-IDF) features from noisy parts of HTML and plaintext.	32,972 benign web pages and 27,280 phishing web pages	96.76% accuracy
Amrutkar et al. (2017)	44 mobile-specific features from JavaScript, HTML and URL were applied to a logistics regression algorithm to detect mobile phishing pages.	349,137 benign URLs and 5,231 malicious URLs.	970% accuracy.
Machine learning based methods using automatic feature selection			
Bahnsen et al. (2017)	Applied an embedding of raw URLs on an LSTM network.	2 million phishing and legitimate URL.	98.7% accuracy
Wei et al. (2019)	Applied a word embedding of raw URLs on a Convolutional Neural network.	999,996 legitimate URLs and 523,970 phishing URLs.	86.6% accuracy
Ozcan et al. (2021)	This study proposed a hybrid deep learning model that concatenates NLP features with character embedding before applying them to a DNN+LSTM model.	37385 phishing URLs and 36,400 legitimate URLs.	98.79% accuracy
Le et al. (2018)	Convolutional Neural Networks were applied to both the characters and word embeddings of the URL String to learn the embedding in a jointly optimized framework.	5 million URLs were used.	97.2% accuracy
Zhang et al. (2021)	Developed MultiPhish that uses VAE to fuse the text, image, and URL features.	5887 phishing instances and 4259 legitimate instances.	97.79% accuracy
Tang and Mahmoud (2021)	Extracted character-level features from the URL, which were then applied to a gated recurrent unit (GRU) neural network	429, 125 legitimate URLs and 236,362 phishing URLs	99.18%

3. The proposed model

In this section, we elaborate on the architecture of our proposed deep neural network model, WebPhish.

We define the problem of detecting phishing web pages using their URL and HTML content as a binary classification task of predicting two classes: *legitimate* or *phishing*. Given a dataset with R web pages $\{(u_1, h_1, y_1), \ldots, (u_R, h_R, y_R)\}$, where u_r and h_r for $r = 1, \ldots, R$ represents the URL and HTML content of the rth web page from the dataset, and $y_r \in \{0, 1\}$ is its label. $y_r = 1$ corresponds to a phishing web page and $y_r = 0$ is a legitimate web page. This study aims to automatically obtain the URL representation, $u_r \mapsto U$, and HTML $h_r \mapsto H$, where U and H represent the character and word embedding feature matrices of the raw URL and HTML content, respectively. Subsequently, the embedding matrices are employed to learn the discriminant model $f : X \mapsto Y$ used to classify a given web page, where X is the concatenation of embedding matrices U and H.

Fig. 1 provides an overview of the proposed model, WebPhish. WebPhish accepts a URL and HTML source code as input and then vectorizes the URL's characters and the words in the HTML source code using the Tokeniser utility class.[3] Each integer corresponds to a value in a dictionary that contains the entire corpus's keys, which are the vocabulary terms themselves. In the deep neural network, the tokenized characters and words are fed into the embedding layer as an array, and the weights are trained concurrently with the phishing detection process. The embedding layer, representing words and characters as dense vectors, captures the relationships between words and characters relevant to the task. The embedding layer then semantically maps

[3] https://keras.io/api/preprocessing/text/

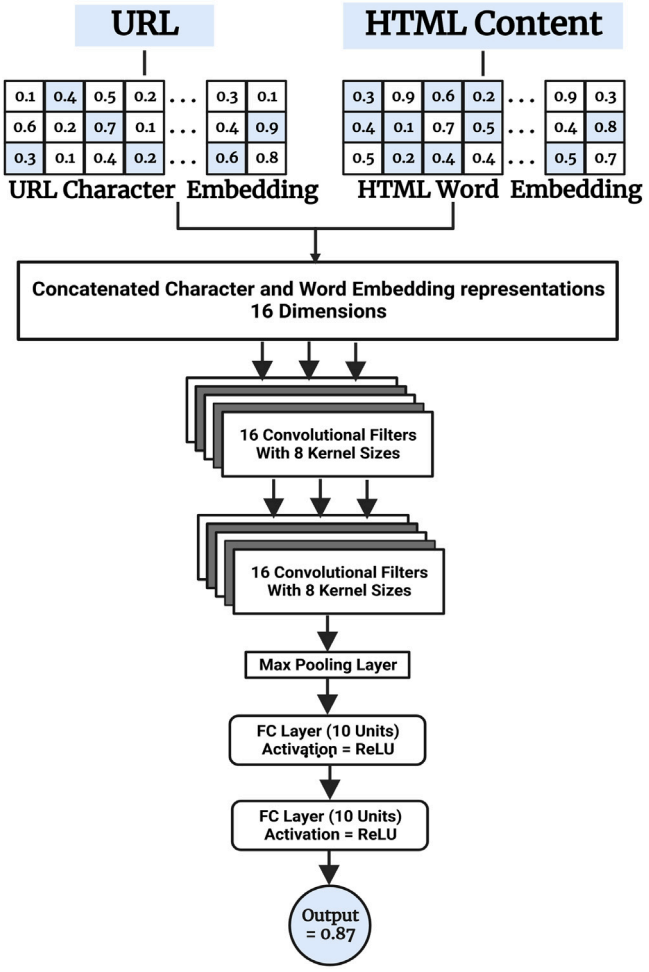


Fig. 1. Overall architecture of WebPhish.

related words such as "login" and "access" within the same embedding space.

Before connecting the embedding layers to the rest of the model, the URL embeddings are concatenated with the HTML embeddings to increase the corpus's vocabulary map. The concatenated features are then passed through convolution layers. CNNs were chosen because they are the best methods to satisfy our requirement to detect more intricate text patterns. We discuss comparisons of the CNN model with other candidates in Section 5.3. Finally, FC layers fuse the results from the convolution layer, and the sigmoid function is used to output the results.

The Deep Neural Network has five layers, namely: (1) input; (2) embedding; (3) concatenation; (4) convolution; (5) FC; and (6) output. The remainder of this section provides a detailed breakdown of each layer.

3.1. Input layer

An advantage of the proposed model is its ability to function directly with raw data. The first step in extracting the automated features in the web page is to create a dictionary of words that occurred in our dataset, with each word having its index.

To create a dictionary of characters for the URLs, we consider each URL a character sequence and then identify all unique alphanumeric and special characters in the URLs in the dataset. 75 unique letters, numbers and special characters with a high frequency in the sequence set are selected to form a character dictionary. Finally, each URL character sequence in the sequence set is tokenised; a corresponding number replaces the original characters individually, such that a one-dimensional digital vector is obtained. This constitutes the character-level corpus of the URLs.

The construction of a word-level corpus of the HTML documents is similar to that of the character-level corpus of the URL. The difference is that HTML documents are segmented into word sequences instead of character sequences. To create a dictionary of words for the HTML documents, we split the content of the HTML document into individual words and treat all punctuation characters as separate tokens. For example, as shown in Fig. 2, <head>, is split into ["<", "head", ">"]. The listed unique words create a dictionary in which every word becomes a feature. We obtained 321,009 unique words from the HTML content dataset. Finally, each HTML document word sequence in the sequence set is tokenized; a corresponding number replaces the original words individually to obtain the one-dimensional digital vector. This constitutes the word-level corpus of the HTML documents.

3.2. Embedding layer

Fig. 2 shows the process in the embedding layer of the proposed model. In the embedding layer, the URL corpus's tokenized characters and the HTML document's tokenized words are automatically converted into feature vectors. Specifically, the raw data is transformed for each input using character embedding matrices into feature representations at the character level. The randomly initialized embedding matrices are gradually modified during training via backpropagation. Although some studies (Li et al., 2019) have used pre-trained Word2vec embeddings as features, other research has shown that learning the embeddings from task data optimized for a particular problem, maybe more efficient (Qi et al., 2018).

Consequently, we obtain the character embedding matrix of each URL $U \in \mathbb{R}^{L_1 \times D}$ such that L_1 is the number of characters in each URL in the dataset, and D is the embedding dimension. We experimentally set $D = 16$ and $L_1 = 180$. URLs over 180 characters were truncated from the 180th character, and URLs shorter than 180 were padded with <PAD> tokens until their lengths reached 180.

For the word embedding matrix of each HTML document $H \in \mathbb{R}^{L_2 \times D}$, L_2 is the number of words in each HTML document in the dataset, and D is the embedding dimension. We experimentally set $D = 16$ and $L_1 = 2000$.

Note: Based on a manual analysis of our dataset, we determined that the longest URL had 156 characters, and the longest HTML document had approximately 1,898 words, excluding spaces. Therefore, we ensured the model did not lose critical web page information.

3.3. Concatenation layer

This layer concatenates the URL character and HTML word embedding matrices into a two-dimensional layer. The concatenation process adds two distinct characteristics to the model. First, it expands the vocabulary size of the embedding space, allowing it to exploit information from rare words and obtain a richer representation capable of capturing sub-word-level information. Second, concatenating the output from the embedding layer (without convolution filtering) preserves the original information of content that can also be used to detect phishing web pages.

3.4. The convolutional neural network

The CNN consists of two major operations: convolution and pooling (Schmidhuber, 2015). The convolution and pooling layers are stacked alternatively in the CNN framework until obtaining the high-level features on which an FC classification is performed (LeCun et al., 2015).

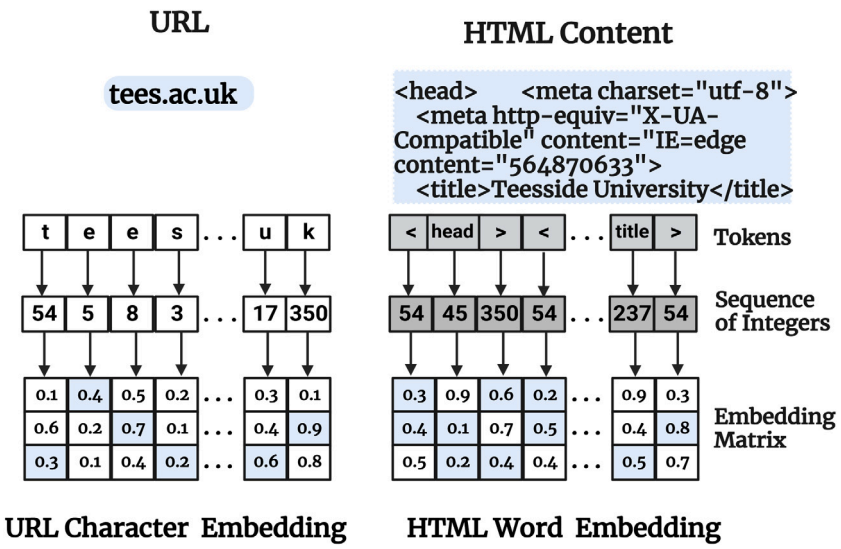


Fig. 2. Configuration of the embedding layer in WebPhish.

Taking the output from the concatenation layer with a shape of [2180, 16], where 2180 is the sum of the length of the URL and HTML and 16 is the dimension, the convolution layer computes the output of neurons connected to the local regions in the input, each computing a dot product between their weights and a small region that they are connected to in the input volume. A ReLU activation function results in a matrix of [2180 × 32 × 8] as the model uses 32 filters and 8 kernels. Max pooling is used for the feature map after the convolution operation to reduce network parameters and extract the most important features. Consequently, the POOL layer performs a downsampling operation along the spatial dimensions, resulting in the output of the convolutional layers being [1090 × 32 × 8], which is then passed to the following FC layers.

3.5. Fully connected layer

The model employs two FC layers that use the output of the convolution layers and merge the resulting features. Specifically, the two FC layers analyze the sequences from the CNN and max-Pooling layers and apply the ReLU activation in each FC layer.

3.6. Output layer

The output layer, the final layer in our model, computes the model's result using the sigmoid function. This output layer, which follows the FC layers, compresses the model's result into a range of 0 to 1, according to the expression: $Q = \frac{1}{1+e^{-q}}$ given the probability of two classes: *legitimate* or *phishing*, where $q = (WR_t + b)$, W and b are the model parameters, and R_t is the input at time step t.

4. Performance evaluation

This section elaborates on the evaluation questions, dataset and evaluation metrics used to access WebPhish.

4.1. Evaluation questions

We look at the following evaluation questions:

- **EQ1:** How accurate is WebPhish at detecting phishing pages compared to other deep learning-based state-of-the-art baselines?
- **EQ2:** How does WebPhish perform compared to hand-crafted features trained using SVM, Logistics Regression, and Random Forest?
- **EQ3:** What are the technical alternatives to the WebPhish model, and how effective are they?
- **EQ4:** How does the WebPhish model perform against other commonly used textual datasets for sentiment classification, such as the United States Airline dataset?

To answer EQ1, in Section 5.1, we conduct experiments comparing WebPhish's performance to that of other baseline approaches on almost 46k phishing and benign webpages. To address EQ2, in Section 5.2, we extract 31 popular features from our dataset's URLs and HTML source code and apply them to three classic machine learning algorithms: SVM, Logistics Regression, and Random Forest evaluating their performance against WebPhish. For EQ3, in Section 5.3, we conduct a controlled experiment to assess performance when using alternative technical options for WebPhish, such as LSTM as the machine learning algorithm and URL or HTML as the only input. To address EQ4, in Section 5.4, we evaluate the model's accuracy using the US Airline dataset in order to demonstrate the model's flexibility on other textual datasets.

4.2. Dataset

To train our models, we used real-world datasets from Alexa.com for legitimate web pages and phishtank.com for phishing web pages to address the evaluation questions above. The details are as follows:

Benign web page dataset. We collated 22,687 benign web pages from the top-ranked Alexa list for this experiment. HTML documents were generated by coding a parser using the Beautiful Soup Library (Richardson, 2017). Beautiful Soup was chosen for two reasons: (1) it is functionally versatile and fast at parsing HTML content, and (2) it does not modify the HTML document object model composition when processing HTML documents. We created a parser that dynamically extracts the HTML source code for each web page from the final landing page.

Phishing webpage dataset. Research has shown that machine-learning classifiers have difficulty coping with imbalanced train sets as they are sensitive to the proportions of the different classes. These algorithms, in particular, tend to favor the majority class, resulting in misleading accuracy. As a result, we collated a balanced phishing set of 22,687 phishing web pages from the study by Korkmaz et al. (2020).

Subsequently, our final corpus contained a balanced dataset of **45,373** phishing and benign instances. A condensed version of our datasets, URLs, and HTML for phishing and legitimate websites have

Table 2
WebPhish hyperparameters.

Hyperparameters	Potential choices	Selected
Number of Conv1D layers	1–3	1
Number of FC layers	1–3	2
Embedding dimension	4–32	16
Optimizer	RMSProp and Adam	Adam
Learning rate	0.0001–0.1	0.0015
Number of epochs	5–30	20
Batch Size	10–30	20

been made available on https://www.kaggle.com/datasets/guchiopara/look-before-you-leap.

Note: In the dataset, all the elements of an HTML document, such as text, hyperlinks, images, tables, lists, etc. and all parts of the URL, including the (subdomain, domain name, path, and query) were used when training the deep learning model. Also, we removed the prefix in URLs such as HTTP:// and HTTPS:// to prevent skewed results on different URL datasets.

4.3. WebPhish experimental setup and metrics

To train WebPhish and its alternatives, a combination of hyperparameters was required. We used a grid search to determine our models' optimal number of CNN layers (1–3) and FC layers (1–3). Additionally, we determined the optimal optimization algorithm for the models (which varied between RMSProp and Adam) across a range of learning rates (from 0.0001 to 0.1).

Table 2 details the selected parameters we found gave the best performance on our dataset bearing in mind the unavoidable hardware limitation. We implemented all WebPhish variants in Python 3.5 on a Tensorflow 1.2.1 backend. We adjusted the batch size for training and testing the model to 20. The Adam optimizer (Kingma & Ba, 2015), with a learning rate of 0.0015, was used to update the network weights. At the same time, we implemented binary cross-entropy to monitor the model's performance. The Early stopping technique (Prechelt, 1998) was adopted to prevent overfitting of the training data. We conducted all WebPhish and baseline experiments on a Google Colaboratory environment with 12 GB GDDR5 VRAM.

4.4. Evaluation metrics

We evaluated the performance of WebPhish using

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

and

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

where TP, FP and FN represent the numbers of True Positives, False Positives and False Negatives, respectively. The recall statistic is calculated as the number of correct results divided by the number of expected results. Finally, the Accuracy of WebPhish was determined using

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

We also used the receiver operating characteristic (ROC) curve and the precision–recall curve in our evaluation. The ROC curve is a probability curve, whereas the Precision–Recall curve depicts the trade-off between precision and recall for various threshold values. The receiver operating characteristic (ROC) curve is plotted against the false positive rate (FPR). In contrast, the precision–recall curve is plotted against the recall values. In addition, we also calculated the f1 score of the test from the precision and recall of the test using the formula

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

Table 3
Result of WebPhish and state-of-the-art baseline models.

Models	Accuracy	Precision	Recall	F1 score	Training time
WebPhish	**0.981**	**0.982**	**0.981**	**0.981**	**491s**
Bahnsen et al. (2017)	0.953	0.963	0.942	0.952	258s
Wei et al. (2019)	0.873	0.804	0.982	0.884	533s
Opara et al. (2020)	0.942	0.909	0.980	0.943	338s
Aljofey et al. (2022)	0.951	0.951	0.953	0.951	135s
Tang and Mahmoud (2021)	0.960	0.948	0.971	0.960	600s

Finally, we evaluated the machine learning model's performance using the K-fold cross-validation technique with k set to 5. Therefore, the data is divided into five folds and iterated five times. Cross-validation reduces problems such as overfitting and underfitting and indicates how the model will generalize to an independent dataset.

5. Results

This section discusses the experiments conducted to evaluate the proposed phishing detection method and their results in terms of each evaluation question.

5.1. Comparing WebPhish with state-of-the-art baselines (RQ 5.1)

We compared WebPhish's performance with existing state-of-the-art approaches for a comprehensive evaluation. These approaches include those by Bahnsen et al. (2017), Wei et al. (2019), Aljofey et al. (2022), Tang and Mahmoud (2021), Opara et al. (2020). The approach proposed by Wei et al. (2019) is a DNN with multiple convolution layers that uses word tokens from a URL as input to determine the maliciousness of the associated web page. Moreover, the model proposed in Bahnsen et al. (2017) accepts the character sequence of a URL as input. It then uses LSTM neural networks to model the URL's sequential dependencies to classify it as phishing or benign.

HTMLPhish (Opara et al., 2020) takes the character and word sequences from the HTML content of a web page as input and uses CNN to learn the semantic dependencies between them. The methods in Bahnsen et al. (2017), and Wei et al. (2019) studies were designed for phishing URL detection, and the study by Opara et al. (2020) was applied to the HTML contents. Aljofey et al. (2022) extracted URL character sequences features with textual content TF-IDF and hyperlink information features to create a feature vector applied to the machine learning models to detect phishing web pages. To ensure an efficient comparison with our dataset, we applied only the TF-IDF features on XGBoost, as discussed in this chapter. Tang and Mahmoud (2021) extracted character-level features from the URL, which were then applied to a gated recurrent unit (GRU) neural network to determine the maliciousness of the URL.

Result: Table 3 summarizes the results of WebPhish and the state-of-the-art baseline techniques. WebPhish exhibited the best performance across all metrics for the dataset. The results demonstrate that the concatenation of the character embedding of the URL and word embedding of the HTML content produces the highest accuracies by exploring the increased vocabulary and sequential patterns in the textual content. Notably, the methods proposed by Bahnsen et al. (2017) performed substantially better than those proposed by Wei et al. (2019) using only word embeddings on URLs and by Aljofey et al. (2022) using TF-IDF on the URLs.

Additionally, we plot the Receiver Operating Characteristic (ROC) and the Precision–Recall curves for each phishing detection technique shown in Figs. 3(a) and 3(b). With decreasing FPR, we observe a growing gap between WebPhish and the baseline approaches, except for Opara et al. (2020). Apart from WebPhish, the only other approach that achieves meaningful recall (TPR) at lower FPRs is Opara et al. (2020) (albeit this comes with a high computational cost). Even with

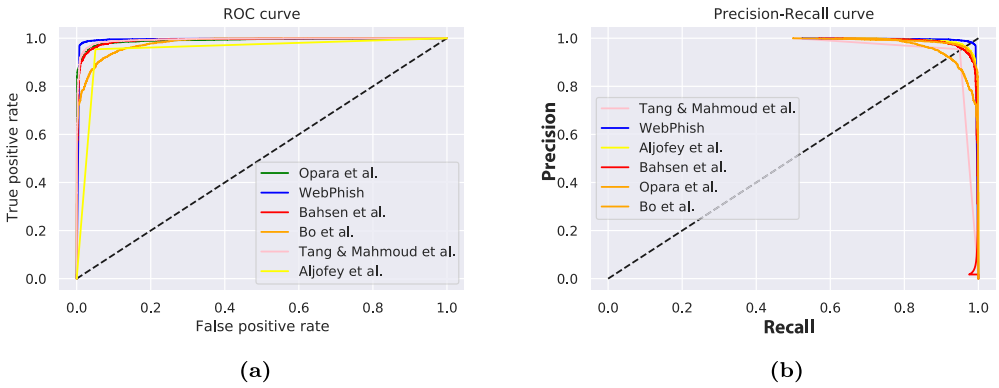


Fig. 3. ROC and precision–recall curves of WebPhish and its state-of-the-art baselines.

low FPR values of 0.2, 0.3 and 0.4, which are required for operational deployment, WebPhish achieves a better recall than Bahnsen et al. (2017).

The WebPhish model performed the best in terms of the F1 score, indicating that the model's overall performance is robust regarding precision and recall. In other words, the proposed methodology accurately detects phishing incidents and avoids misclassifying too many legitimate pages as phishing. This shows that the proposed phishing technique balances well between accuracy and recall.

5.1.1. Performance of the WebPhish model in terms of computational time

The main limitation of the proposed phishing detection model is that it is substantially more computationally intensive than other comparative models in terms of training time (see Table 3), with an average elapsed training time that is approximately two times higher than that of Bahnsen et al. (2017). Overall, the proposed model is more computationally complex than other benchmarked methods, except for the model by Wei et al. (2019). In terms of runtime efficiency, the model requires 0.094s to evaluate whether a website is harmful based on its URL and HTML content. According to Fui-Hoon Nah et al. the acceptable wait time for web users to retrieve information is 2s. Therefore, when applied in real-time, the performance of our model does not negatively impact the website's usability.

5.2. Comparing WebPhish with hand-crafted features trained on simpler baseline models (RQ2)

This section compares WebPhish's deep neural network effect on phishing web page detection using raw URL and HTML content to simpler baseline models trained on hand-crafted features. We used three machine learning models: logistics regression, kernel SVM, and a random forest classifier. We chose these models because these traditional classifiers were commonly used in sequence detection systems (Mirończuk & Protasiewicz, 2018) and are relevant baselines to compare with WebPhish. We used **31** features detailed in Table 4 culled from Adebowale et al. (2019), Amrutkar et al. (2017), Mohammad et al. (2012a), and Zhang et al. (2007).

5.2.1. URL features

For the hand-crafted features extracted from the URL, research has shown that phishing web pages developers frequently exploit an Internet user's familiarity with a website (Dhamija et al., 2006) by adding terms to the URL that may trick a user into thinking that somehow the malicious website is the real website. Widely used terms to access genuine websites, like admin and account, make them particularly vulnerable to imitation. Therefore, the creator of a phishing website would intuitively use ambiguous terms at the URL's start. Therefore, including those terms in the URL is regarded as a feature. Many malicious domain names are hosting systems IP addresses (Fette et al., 2007; McGrath & Gupta, 2008). We counted the combination of numbers in a URL and the percentage of numbers in the hostname as a feature. Also, phishers create several subdomains to include tricky terms, for example, PayPal as a subdomain. This could make phishing URLs longer (McGrath & Gupta, 2008). Therefore, we included the URL length, if the URL consists of a subdomain, the number of subdomains, and the number of dots as features. Furthermore, the number of punctuation marks, such as semicolons, hyphens, and underscores, are included in our URL feature set.

5.2.2. HTML features

For the HTML feature set, variables such as the number of white spaces, the presence of internal and external links, and the number and presence of images were extracted because of their relevance when differentiating between a phishing and legitimate web page.

A binary classifier is taught using the extracted features, which are provided when the features are collected. We empirically set the number of trees as 70 for the random forest classifier, the penalty for the logistics regression as *L1*, and the kernel bias function (RBF) of the non-linear SVM as 50.0.

Result: Table 5 shows the precision, recall, and F1 score of WebPhish compared with the traditional machine learning classifiers. Figs. 4(a) and 4(b) show the traditional machine learning classifier ROCs. WebPhish outperforms all state-of-the-art techniques (with an improvement in accuracy of at least 2%) in all categories and metrics.

The WebPhish classifier correctly predicted approximately 98% of the 2,252 instances of phishing in the test set. In contrast, the SVM, logistic regression, and RF classifiers correctly predicted 88%, 87%, and 94% of the phishing instances, respectively. The outputs of the traditional machine learning classifiers show the limitations of hand-crafted features. This highlights the importance of the temporal robustness of our proposed method.

In addition, the ROC curve in Fig. 4(a) shows that the RF classifier yields better results than the logistic regression and SVM classifiers, with its curve being further from the hyperplane.

Furthermore, Table 6 above is the confusion matrix for the proposed model (WebPhish) and the traditional machine learning classifiers on the dataset. With 4538 instances in the test set, the classifier correctly predicted "phishing" 2202 times and "legitimate" 2237 times for the WebPhish model yielding a precision of 98.1%.

In the scenario where the HTML content is cloned, and the URL has been spoofed, the model using these features will leverage the URL and HTML features to determine the maliciousness of the web page. For example, if the HTML content has been cloned word by word, and the URL is spoofed by using only the IP address of the website, then the HTML features, including the length of the page, number of images and external links, will not suffice as they will be same in both the cloned and legitimate web page. However, the model will rely on URL characteristics such as the number and percentage of digits in the URL to identify the maliciousness of the web page.

Table 4
Extracted hand-crafted features from both URL and HTML contents on Web pages.

Features	Description
URL features	
Number of misleading words in the URL, such as "login" and "bank."	Authors of phishing webpages often exploit the familiarity of users to a webpage (Amrutkar et al., 2017) by including words in the URL that can mislead a user into believing that the phishing webpage is a legitimate webpage. Words such as login and bank are commonly used in the URL of the login webpage for benign websites that are highly prone to imitation.
Number and % of digits	Many phishing domain names are simply IP addresses of the machines that host them. To detect phishing presence, we calculated the number of digits in a URL and the percentage of digits in the hostname.
Number of forward slashes, question marks, dots, hyphens, underscores, equal signs, semi-colons and ampersands	Phishers use some meaningless characters to confuse the victim. Therefore, they can also use some punctuation characters, especially ".", ";", "!", "&", "%", etc. Increased value has more tendency to be a phishing webpage
Number and Presence of sub-domains and two letter subdo-mains	Legitimate URLs typically have fewer subdomains; Also, phishers can use the subdomain names as if they were domain names. Furthermore, they can use several subdomains with names that are similar to the original ones. As a result, fewer subdomains increase the likelihood of a legitimate web page.
Length of the URL	Long URLs can be used by phishers to hide the suspicious part in the address bar. We averaged the lengths of the URLs in the dataset. Intuitively, the longer a URL is, the more likely it is to be a phishing URL.
HTML Features	
Number of NoScript	Intuitively, a benign webpage developer will include more NoScript in the code to ensure a good experience for even the most security-savvy user.
Presence and number of em-bedded JavaScript	A webpage that includes embedded JavaScript loads faster than a page that must refer to external code. According to previous research (Fette et al., 2006), legitimate websites will have more embedded JavaScript than phishing websites.
Presence and number of exter-nal and internal JavaScript	Legitimate web pages use more internal and external JavaScript for advertising and analytics than malicious web pages
Number and Presence of inter-nal and external links	Legitimate web pages have more internal links pointing to subdirectories on the website and fewer external links pointing to other legitimate websites. This feature also improves the user experience of the website, which is considered by legitimate website creators. Phishing webpages usually use external resources like their phishing targets to enrich their content in order to deceive users that the page is legal, resulting in very few internal links and many external links in a phishing webpage.
Presence of iframes	IFrame is an HTML tag that displays an additional webpage within the one that is currently displayed. Phishing website developers use the "iframe" tag and make it invisible, i.e., without frame borders. As a result, the presence of iframes indicates a phishing website. This feature counts the number of strings that contain "iframe" in a script.
Presence of and number of images	Legitimate web pages contain more images for aesthetic and improved user experience, whereas malicious web pages contain fewer images.
Percentage of white spaces in the HTML content	Legitimate websites will have more white space between the code to ensure a good experience even for a security-savvy user. As a result, the more whitespaces there are in the HTML content, the more likely the website is legitimate.

Table 5
Result of WebPhish and hand-crafted features on traditional models.

Models	Accuracy	Precision	Recall	F1 score	Training time
WebPhish	**0.981**	**0.982**	**0.981**	**0.981**	**491s**
Kernel SVM + hand-crafted Features	0.873	0.836	0.924	0.878	125s
Logistics Regression + hand-crafted Features	0.884	0.865	0.908	0.885	65s
Random Forest Classifier + hand-crafted Features	0.945	0.942	0.948	0.945	90s

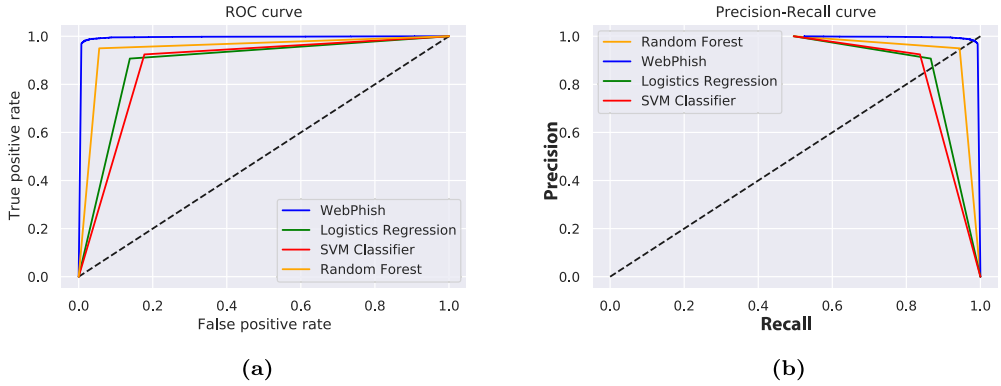


Fig. 4. ROC and precision–recall curves of webphish and hand-crafted features on traditional models.

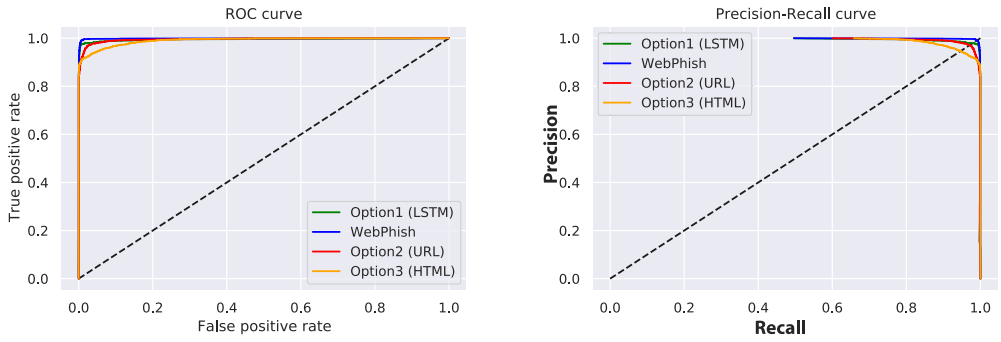


Fig. 5. ROC and precision–recall curves of WebPhish and its variants.

Table 6
Confusion matrix of WebPhish and hand-crafted features on traditional models.

	WebPhish		Random forest classifier		Kernel SVM		Logistics regression	
Legitimate	2237	50	2156	131	2090	197	1968	319
Phishing	49	2202	117	2134	162	2089	207	2044
	Legitimate	Phishing	Legitimate	Phishing	Legitimate	Phishing	Legitimate	Phishing

5.3. Alternative technical options for WebPhish (RQ3)

In addition to the proposed end-to-end framework, which combines URL embedding at the character level with HTML content, we evaluated alternative technical options for implementing WebPhish. Here we examine the following technical alternatives:

- Option1 (LSTM): This deep learning model learns URL features and HTML content representations using RNNs, particularly **LSTM**. The character embedding matrix is applied to the LSTM layer, whose output is concatenated and sent to the dense layers. Then, the sigmoid layer outputs the classification results.
- Option2 (URL): This CNN model is trained exclusively on the **URL**. The embedding layer's character embedding matrix is applied to the CNN and max-pooling layers, which are then passed to the dense layers. The results are output via the sigmoid layer.
- Option3 (HTML): This is a CNN model trained exclusively on the **HTML content**. The embedding layer's word embedding matrix is applied to the convolution and max-pooling layers, and the results are passed to the dense layers. The sigmoid layer outputs the results.

Result: In Fig. 5(a) and 5(b), we show the ROC curve of WebPhish and its alternative options. Amongst the alternative options, Option3 (HTML) was the least performing across all evaluated metrics. This outcome is because phishing web pages, especially those hosted on compromised websites, are known to systematically copy the legitimate web page source code in other to blend in effortlessly. In contrast, Option1 (LSTM) performed more closely to the proposed model, albeit having a higher training time. Furthermore, a random selection of false positive URLs showed the presence of known phishing words such as "login" and "account", which typically occur in phishing URLs. However, the model can accurately classify a high percentage of the legitimate URLs because, upon closer examination of our dataset, we discovered that some words more frequently occur in phishing web pages than legitimate ones and vice versa. For example, in our dataset, the word "login" appeared 99.8% more often in phishing than in legitimate URLs. Consequently, the model generates a prediction based on the semantic meaning of the highlighted text and its co-occurrence on the page.

In general, Table 7 indicates that WebPhish significantly outperforms the three options: Option1 (LSTM), Option2 (URL) and Option3 (HTML). WebPhish achieved an average of 98.1% across its precision, F1 score, and recall metrics on the dataset. WebPhish leverages the strength of the alternatives and consistently produces better results while capturing local and temporal patterns in the data. Furthermore, the precision, recall, and F1 score for WebPhish are well-balanced as their values are similar. This indicates that WebPhish can detect phishing web pages accurately when implemented in the real world. Furthermore, WebPhish achieved the lowest FPR of 2% as it leverages the strengths of both the URL and HTML input to consistently produce better results while capturing local and temporal patterns in the data. This indicates that the proposed model can learn the characters and words in the dataset and their associated semantic meanings to identify phishing web pages.

Table 7
Result of WebPhish and its alternative options.

Models	Accuracy	Precision	Recall	F1 score	Training time
WebPhish	**0.981**	**0.982**	**0.981**	**0.981**	**491s**
Option2 (URL)	0.959	0.956	0.964	0.960	193s
Option3 (HTML)	0.965	0.970	0.961	0.965	533s
Option1 (LSTM)	0.954	0.965	0.943	0.954	614s

5.4. Application on sentiment classification (RQ4)

We evaluated our model's performance on the publicly available US airline dataset to demonstrate its adaptability to other textual datasets. The US airline dataset was sourced from CrowdFlower's Kaggle databases.[4] There are 14,640 tweets in this data set. American Airlines, United Airlines, US Airways, Southwest Airlines, Delta Airlines, and Virgin Airlines are among the airlines that have used these tweets. Sentiment classification techniques can aid researchers and decision-makers at airlines in better comprehending their customers' feelings, opinions, and satisfaction.

We fed WebPhish the actual text tweeted by customers and the airline sentiment confidence associated with it. The airline sentiment confidence metric is a numeric value that indicates the degree of certainty associated with categorizing a tweet as neutral, positive, or negative.

[4] https://www.kaggle.com/crowdflower/twitter-airline-sentiment

Table 8
Result of WebPhish and state-of-the-art models on the US airline dataset classifying 3 classes: Positive, negative and neutral.

	Corpus size	No. of classes	Algorithm	Accuracy in %
Rustam et al. (2019)	14,640	3 Classes	Decision Tree	63
			Random Forest	75.8
			SVM	78.5
			Gaussian Naive Bayes	43.8
			AdaBoost	74.6
			Logistic Regression	78.7
			Gradient Boosting	73.4
			Decision Tree Classifier	68.6
			Voting Classifier	79.2
Tang and Liu (2016)	14,640	3 Classes	Gated GRU	74
WebPhish	**14,640**	**3 Classes**	**DCNN**	**80**

Table 9
Result of WebPhish and state-of-the-art models on the US airline dataset classifying 2 classes: Positive and negative.

	Corpus size	No. classes	Algorithm	Accuracy in %
Naseem and Musial (2019)	11,542	2 Classes	BiLSTM	93.6
Khan and Malviya (2020)	11,542	2 Classes	Deep RNN	93
WebPhish	**11,542**	**2 Classes**	**DNN**	**94.1**

5.4.1. Implementation

Using the evaluation metrics detailed in Section 4.4, we applied the WebPhish model to the US Airline dataset to evaluate the model's ability to differentiate between positive and negative emotions in text.

Note: as the US Airline dataset has been publicly available since 2015, previous studies have been conducted on classifying its sentiments. We compared the performance of WebPhish on the US Airline dataset with the following studies: In Rustam et al. (2019), the authors applied a VC to classify tweets according to their emotions. The VC is based on logistic regression and SGDC and uses a soft voting mechanism to obtain the final prediction.

In addition, using TF-IDF as a feature extraction mechanism, the authors implemented a phrase-level analysis on seven classification algorithms: decision tree, RF, SVM, Gaussian Naive Bayes, AdaBoost, logistic regression, gradient boosting, and the VC. We also compared WebPhish with the system proposed by Tang and Liu (2016). Using Doc2Vec embeddings on the US Airline dataset, Tang and Liu (2016) explored a bi-directional GRU network for sentiment analysis of Twitter data directed at US airlines. They fed a trained word vector to the bi-directional GRU network using a skip-gram model, which was initialized by the existing GloVe model (Socher et al., 2013).

We compared our model with the DICET method proposed in Naseem and Musial (2019). DICET is an automated text pre-processor fed into a Bidirectional LSTM with attention to detecting Twitter sentiment analysis. Also, Khan and Malviya (2020) proposed a sentiment analysis model that extracts relevant features from the US airline dataset using a Hadoop cluster. The obtained features are fed into a deep RNN network to perform the classification, providing two classes: positive and negative.

Naseem and Musial (2019) and Khan and Malviya (2020) removed the neutral class from the US Airline dataset during experimentation, thereby reducing the dataset to 11,541 tweets. Conversely, Rustam et al. (2019) and Tang and Liu (2016) experimented with the entire corpus of the US Airlines dataset of 14,640 instances. Consequently, WebPhish was trained and evaluated on full and abridged datasets to ensure fairness.

Tables 8 and 9 detail the result of WebPhish and other state-of-the-art techniques on the US airline dataset. WebPhish indicates a significant improvement in efficiency compared with current state-of-the-art techniques trained on the US airline Twitter datasets.

WebPhish, with an accuracy of 94%, works better than other models for the US airlines dataset. We can effectively conclude that our proposed model is a robust solution and applies to other text classification fields beyond social engineering.

6. Discussion

6.1. Why does WebPhish outperform the baselines?

In this study, the proposed model WebPhish, effectively tackled the detection of phishing web pages using only the URL and HTML raw content on a deep learning model.

The advantages of WebPhish over the baseline models include:

1. The approach presented in this study leverages the dual input approach using both the raw URL and HTML content to detect phishing attacks. For example, in the scenario where the URL is spoofed, the HTML content is relied upon to provide a convincing result on the maliciousness of the web page. Also, unlike existing techniques, our solution does not rely on manual feature engineering. It exploits the URL and HTML content's character and word embedding matrices to detect a phishing attack without expert feature engineering.
2. Choice of semantic level to use in the embedding layer: Existing research influenced the semantics level chosen for the embedding layer (Opara et al., 2019). Word-level embeddings are limited to the training data's unique dictionary of words. Consequently, it cannot generate useful feature representations for newly introduced words in the test data. According to our analysis, character-level embeddings outperform word-level embeddings. With a limited number of existing characters in languages, the character embedding layer is less likely to encounter characters not present in the test dataset. As a result, it is simple to apply to test data. However, in the phishing detection task, the approaches proposed by Wei et al. (2019) and HTMLPhish struggled to distinguish information for scenarios in which phishing URLs and HTML documents attempt to impersonate benign web pages using obfuscation techniques. This is because convolution filters produce similar output from a sequence of characters with the same spelling or only minor differences. Consequently, CNNs based on URLs or HTML alone are insufficient for obtaining detailed structural information from a web page. Hence, we considered the concatenation of both web page component embeddings.

   In summary, WebPhish uses the URL's character embedding matrices and the HTML's word embedding matrices to accommodate unseen words in the test data, yielding better results than the other variants.

Table 10
The impact of the FC layers.

Models	Accuracy	Training time
1 FC Layers	0.979	475s
Proposed Model (2 FC Layers)	0.981	491s
3 FC Layers	0.980	492s

Table 11
The impact of the convolutional layers.

Models	Accuracy	Training time
1 Convolutional Layer	0.980	480s
Proposed Model (2 Convolutional Layers)	0.981	491s
3 Convolutional layers	0.980	493s

3. Introducing the concatenation layer before feeding the input features to the convolution layers: Unlike previous work (Le et al., 2018), which concatenates the outputs of convolution layers, WebPhish combines the outputs of embedding layers. Concatenating the output of the embedding layer (without convolutional filtering) preserves the original information regarding the content useable for detecting malicious web pages, as demonstrated in our experiments.
   In WebPhish, before connecting the embedding layers to the rest of the model, URL embeddings are concatenated with HTML embeddings, increasing the corpus's vocabulary map. The concatenated features are then passed through convolution layers. Consequently, WebPhish can learn patterns based on the URL characters and HTML word embeddings.

6.2. How does WebPhish perform on cloned websites?

WebPhish's main objective is to accurately determine the maliciousness of a given website using its HTML and URL. When its HTML content and URL are spoofed precisely in the same format, WebPhish uses the character and word level characteristics of the HTML and URL to distinguish these spoofed websites from the benign ones.

For example, suppose the URL is spoofed by misspelling the characters in the links or by using non-Latin characters to make homographic URLs. Because this proposed approach is trained on a dataset of multilanguage websites and uses character embeddings to learn the text's semantic, lexical, and syntactic ambiguities, the model will detect these URL spoofing methods. When an Internet user clicks on a shortened URL, the full URL is disclosed before the web page is opened. As a result, the model can still evaluate the full URL to determine its maliciousness.

If the HTML content is cloned word for word, the proposed method will depend on the URL content to determine the maliciousness of the website. To summarize, the proposed approach uses both the contents of URLs and HTML to remain effective in the face of a phishing attack.

6.3. The importance of the number of layers in the DNN on the phishing detection accuracy

To demonstrate the importance of the DNN layers in WebPhish in detecting phishing, we examine FC layers and CNN layers' effect on the proposed model efficiency.

Table 10 shows the results of our evaluations on the effect of the FC layers. Intuitively, we expect that more FC layers will increase the model's accuracy. However, our analysis found that the proposed model's configuration of 2 FC layers gave our task's best performance on the dataset, while an extra layer did not improve the accuracy. The proposed model achieved an accuracy of 97.9%, 98.1%, and 98% with 1, 2, and 3 FC layers.

Table 11 shows the effect of the number of CNN layers in the model. We found that 2 CNN layers gave the best balance of training time of 491 s and accuracy of 98.1%. When using 1 and 3 CNN layers, WebPhish can achieve 98% and 98.1% accuracy and a training time of 475 s and 492 s, respectively.

Table 12
The impact of the embedding layer.

Models	Accuracy	Training time
Proposed Model	0.981	491s
Proposed Model without Embedding layer	0.941	268s

Furthermore, we demonstrated the importance of the embedding layer in the DNN model in phishing detection. We analyzed this by checking the performance of WebPhish on the dataset when the embedding layer was replaced with hand-crafted features from the URL and HTML characteristics on the CNN and FC layers. Table 4 lists the hand-crafted features. Although the hand-crafted features' training time is shorter than the embedding features, a 4% drop is observed across all metrics in Table 12. This highlights the character embedding matrix's importance when analyzing textual content in the URL and HTML. It also demonstrates that the tedious hand-crafted feature engineering process can overlook salient characteristics that differentiate a phishing web page from a legitimate one.

6.4. Limitations of WebPhish

While we argue that WebPhish can detect zero-day phishing attacks that contain known phishing HTML and URL content, if the zero-day phishing attack involves a rare manipulation of the web page content, WebPhish may miss it. Additionally, due to the rare manipulation of web page content on web pages, the model will need to be retrained to remain relevant. Additionally, as the proposed model is not visual based, the model cannot classify web pages whose DOM is predominantly embedded images or flashes.

7. Conclusion

In this paper, we propose a technique for detecting web page phishing. We used a DNN, more precisely Convolutional Neural Networks, to capture the semantic relationships inherent in raw URL and HTML content while initiating automatic feature extraction via character and word embedding techniques. The proposed method overcomes some limitations associated with existing approaches that rely on hand-crafted features or one raw web page component and enables straightforward extrapolation to new data. Additionally, as it takes the proposed model 0.094s to evaluate whether a website is harmful, therefore, when applied in real-time, the performance of our model will not negatively impact the website's usability when deployed in the real world. The next step will be to examine our model's performance against other types of phishing attacks, such as spear phishing.

CRediT authorship contribution statement

Chidimma Opara: Conceptualization, Data curation, Methodology, Software, Writing – original draft. **Yingke Chen:** Investigation, Writing – review & editing, Supervision, Validation. **Bo Wei:** Writing – review & editing, Validation, Supervision, Visualization, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by a scholarship of the Petroleum Technology Development Fund Nigeria.

References

Aburrous, M., Hossain, M. A., Dahal, K., & Thabtah, F. (2010). Intelligent phishing detection system for e-banking using fuzzy data mining. *Expert Systems with Applications*, *37*(12), 7913–7921.

Acar, T., Belenkiy, M., & Küpçü, A. (2013). Single password authentication. *Computer Networks*, *57*(13), 2597–2614.

Adebowale, M. A., Lwin, K. T., Sanchez, E., & Hossain, M. A. (2019). Intelligent web-phishing detection and protection scheme using integrated features of images, frames and text. *Expert Systems with Applications*, *115*, 300–313.

Aljofey, A., Jiang, Q., Rasool, A., Chen, H., Liu, W., Qu, Q., & Wang, Y. (2022). An effective detection approach for phishing websites using URL and HTML features. *Scientific Reports*, *12*(1), 1–19.

Amrutkar, C., Kim, Y. S., & Traynor, P. (2017). Detecting mobile malicious webpages in real time. *IEEE Transactions on Mobile Computing*, *16*(8), 2184–2197.

Bahnsen, A. C., Bohorquez, E. C., Villegas, S., Vargas, J., & González, F. A. (2017). Classifying phishing URLs using recurrent neural networks. In *Electronic crime research (eCrime), 2017 APWG symposium on* (pp. 1–8). IEEE.

Barraclough, P. A., Hossain, M. A., Tahir, M., Sexton, G., & Aslam, N. (2013). Intelligent phishing detection and protection scheme for online transactions. *Expert Systems with Applications*, *40*(11), 4697–4706.

Buber, E., Dırı, B., & Sahingoz, O. K. (2017). Detecting phishing attacks from URL by using NLP techniques. In *International conference on computer science and engineering* (pp. 337–342). IEEE.

Chattaraj, D., Sarma, M., & Das, A. K. (2018). A new two-server authentication and key agreement protocol for accessing secure cloud services. *Computer Networks*, *131*, 144–164.

Chiew, K. L., Tan, C. L., Wong, K., Yong, K. S., & Tiong, W. K. (2019). A new hybrid ensemble feature selection framework for machine learning-based phishing detection system. *Information Sciences*, *484*, 153–166.

Dhamija, R., Tygar, J. D., & Hearst, M. (2006). Why phishing works-Proceedings of the SIGCHI conference on human factors in computing systems. In *CHI, vol. 6* (p. 581).

Fette, I., Sadeh, N., & Tomasic, A. (2006). *Learning to detect phishing emails*: *Technical report*, Carnegie-Mellon University Pittsburgh PA.

Fette, I., Sadeh, N., & Tomasic, A. (2007). Learning to detect phishing emails. In *Proceedings of the 16th international conference on world wide web* (pp. 649–656).

Google (2019). Google safe browsing. http://code.google.com/apis/safebrowsing/. (Accessed 30 September 2019).

Gutierrez, C. N., Kim, T., Della Corte, R., Avery, J., Goldwasser, D., Cinque, M., & Bagchi, S. (2018). Learning from the ones that got away: Detecting new forms of phishing attacks. *IEEE Transactions on Dependable and Secure Computing*, *15*(6), 988–1001.

Khan, M., & Malviya, A. (2020). Big data approach for sentiment analysis of twitter data using hadoop framework and deep learning. In *2020 International conference on emerging trends in information technology and engineering* (pp. 1–5). IEEE.

Kingma, D., & Ba, J. (2015). Adam: a method for stochastic optimization (2014), 15. arXiv preprint arXiv:1412.6980.

Korkmaz, M., Kocyigit, E., Sahingoz, O. K., & Diri, B. (2020). Deep neural network based phishing classification on a high-risk url dataset. In *International conference on soft computing and pattern recognition* (pp. 648–657). Springer.

Kumi, S., Lim, C., & Lee, S.-G. (2021). Malicious url detection based on associative classification. *Entropy*, *23*(2), 182.

Lakshmi, V. S., & Vijaya, M. (2012). Efficient prediction of phishing websites using supervised learning algorithms. *Procedia Engineering*, *30*, 798–805.

Le, H., Pham, Q., Sahoo, D., & Hoi, S. C. (2018). URLnet: Learning a URL representation with deep learning for malicious URL detection. arXiv preprint arXiv:1802.03162.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436.

Li, Y., Yang, Z., Chen, X., Yuan, H., & Liu, W. (2019). A stacking model using URL and HTML features for phishing webpage detection. *Future Generation Computer Systems*, *94*, 27–39.

Maroofi, S., Korczyński, M., Hesselman, C., Ampeau, B., & Duda, A. (2020). Comar: Classification of compromised versus maliciously registered domains. In *2020 IEEE European symposium on security and privacy* (pp. 607–623). IEEE.

McGrath, D. K., & Gupta, M. (2008). Behind phishing: An examination of phisher modi operandi. *LEET*, *8*, 4.

Mirończuk, M. M., & Protasiewicz, J. (2018). A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications*, *106*, 36–54.

Moghimi, M., & Varjani, A. Y. (2016). New rule-based phishing detection method. *Expert Systems with Applications*, *53*, 231–242.

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2012a). An assessment of features related to phishing websites using an automated technique. In *2012 International conference for internet technology and secured transactions* (pp. 492–497). IEEE.

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2012b). Predicting phishing websites based on self-structuring neural network. In *International conference for internet technology and secured transactions* (pp. 492–497). IEEE.

Naseem, U., & Musial, K. (2019). Dice: Deep intelligent contextual embedding for twitter sentiment analysis. In *2019 International conference on document analysis and recognition* (pp. 953–958). IEEE.

Opara, C., Wei, B., & Chen, Y. (2019). HTMLPhish: Enabling accurate phishing web page detection by applying deep learning techniques on HTML analysis. arXiv preprint arXiv:1909.01135.

Opara, C., Wei, B., & Chen, Y. (2020). HTMLPhish: Enabling phishing web page detection by applying deep learning techniques on HTML analysis. In *2020 International joint conference on neural networks* (pp. 1–8).

Ozcan, A., Catal, C., Donmez, E., & Senturk, B. (2021). A hybrid DNN–LSTM model for detecting phishing URLs. *Neural Computing and Applications*, 1–17.

Prechelt, L. (1998). Early stopping-but when? In *Neural networks: Tricks of the trade* (pp. 55–69). Springer.

Qi, Y., Sachan, D. S., Felix, M., Padmanabhan, S. J., & Neubig, G. (2018). When and why are pre-trained word embeddings useful for neural machine translation? arXiv preprint arXiv:1804.06323.

Rendall, K., Nisioti, A., & Mylonas, A. (2020). Towards a multi-layered phishing detection. *Sensors*, *20*(16), 4540.

Richardson, L. (2017). *Beautiful soup* (4.6.0 ed.). URL: https://www.crummy.com/software/BeautifulSoup.

Rustam, F., Ashraf, I., Mehmood, A., Ullah, S., & Choi, G. S. (2019). Tweets classification on the base of sentiments for US airline companies. *Entropy*, *21*(11), 1078.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Singh, P., Maravi, Y. P., & Sharma, S. (2015). Phishing websites detection through supervised learning networks. In *International conference on computing and communications technologies* (pp. 61–65). IEEE.

Smadi, S., Aslam, N., & Zhang, L. (2018). Detection of online phishing email using dynamic evolving neural network based on reinforcement learning. *Decision Support Systems*, *107*, 88–102.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1631–1642).

Tang, Y., & Liu, J. (2016). Gated recurrent units for airline sentiment analysis of Twitter data. arxiv.

Tang, L., & Mahmoud, Q. H. (2021). A deep learning-based framework for phishing website detection. *IEEE Access*, *10*, 1509–1521.

Wei, B., Hamad, R. A., Yang, L., He, X., Wang, H., Gao, B., & Woo, W. L. (2019). A deep-learning-driven light-weight phishing detection sensor. *Sensors*, *19*(19), 4258.

Yerima, S. Y., & Alzaylaee, M. K. (2020). High accuracy phishing detection based on convolutional neural networks. In *2020 3rd international conference on computer applications & information security* (pp. 1–6). IEEE.

Yujian, L., & Bo, L. (2007). A normalized Levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(6), 1091–1095.

Zhang, Y., Hong, J. I., & Cranor, L. F. (2007). Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on world wide web* (pp. 639–648).

Zhang, N., & Yuan, Y. (2012). Phishing detection using neural network. *CS229 Lecture Notes*.

Zhang, L., Zhang, P., Liu, L., & Tan, J. (2021). Multiphish: Multi-modal features fusion networks for phishing detection. In *ICASSP 2021-2021 IEEE international conference on acoustics, speech and signal processing* (pp. 3520–3524). IEEE.