IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 November 2022, accepted 9 January 2023, date of publication 18 January 2023, date of current version 23 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3237798

SURVEY

A Survey of Intelligent Detection Designs of HTML URL Phishing Attacks

SULTAN ASIRI[1], YANG XIAO[1], (Fellow, IEEE), SALEH ALZAHRANI[1], SHUHUI LI[2], (Senior Member, IEEE), AND TIESHAN LI[3,4], (Senior Member, IEEE)

[1]Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35487, USA
[2]Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL 35401, USA
[3]School of Automation Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
[4]Navigation College, Dalian Maritime University, Gaoxinyuan, Dalian 116026, China

Corresponding author: Yang Xiao (yangxiao@ieee.org)

The work of Tieshan Li was supported in part by the National Natural Science Foundation of China under Grant 51939001 and Grant 61976033.

ABSTRACT Phishing attacks are a type of cybercrime that has grown in recent years. It is part of social engineering attacks where an attacker deceives users by sending fake messages using social media platforms or emails. Phishing attacks steal users' information or download and install malicious software. They are hard to detect because attackers can design a phishing message that looks legitimate to a user. This message may contain a phishing URL so that even an expert can be a victim. This URL leads the victim to a fake website that steals information, such as login information, payment information, etc. Researchers and engineers work to develop methods to detect phishing attacks without the need for the eyes of experts. Even though many papers discuss HTML and URL-based phishing detection methods, there is no comprehensive survey to discuss these methods. Therefore, this paper comprehensively surveys HTML and URL phishing attacks and detection methods. We review the current state-of-art deep learning models to detect URL-based and hybrid-based phishing attacks in detail. We compare each model based on its data preprocessing, feature extraction, model design, and performance.

INDEX TERMS Intelligent detection, HTML, URL, phishing attacks, social engineering, machine learning, natural language processing.

I. INTRODUCTION

Phishing attacks are cybercrime using social engineering to deceive users into stealing their information, such as personal identity, financial information, etc. Masquerading as legitimate sources, attackers can reach victims by sending fraudulent messages using emails (such as Gmail, Outlook, etc.) or social media platforms (like Twitter, Facebook, etc.). Users become vulnerable if they input their information or download attachment files [1].

In recent years, there has been an increase in social media platform attacks since it is easy for attackers to reach many users from anywhere in the world by posting a single message [2]. According to [2], the Anti-Phishing Working Group (APWG) reports the number of phishing attacks increased by 250000 in one month in Jan 2021. In addition, the number of business compromises increased 56% from the last quarter of 2020 to the first quarter of 2021. Fig. 1 shows that the most targeted industries in 2021 are financial institutions, social media, and web emails [2]. Based on Fig. 1, attackers' primary focus is to steal victims' financial information or identities by targeting financial industries and social media platforms, respectively. Attackers also might send malicious software that leads to other cyberattacks, such as malware attacks, ransomware attacks, etc.

Increasing phishing attacks in recent years and their cybersecurity threats have become an important issue to be solved [3]. Most current organizations rely on using human knowledge to detect these attacks [4]. Nevertheless, phishing attacks are complex for the human eyes to identify, even for an expert, due to the similarity between legitimate and fake messages. Therefore, cybersecurity experts pay more attention to the message attachments, such as Uniform Resource Locators (URLs) or email IDs, etc., to recognize phishing

The associate editor coordinating the review of this manuscript and approving it for publication was Tony Thomas.

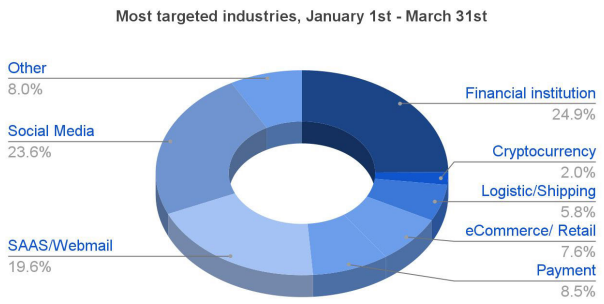


FIGURE 1. APWG report 2021 [2].

messages. Nevertheless, attackers improve their attack techniques by using new methods to design phishing attacks that are hard to detect. For example, they design a phishing URL and webpage that look similar to a benign URL, such as https://www.facebook.com/, https://www.faceb00k.com/, https://www.facebook.edu/, etc. Therefore, it is essential to determine methods to distinguish a phishing URL from a benign URL. As a result, researchers have proposed several solutions in recent years with high accuracy to detect phishing attacks, such as blacklist [5], traditional machine learning [6], and Deep Learning (DL) [7], [8], [9], [10]. We provide a brief analysis of each solution as follows.

- Blacklists are lists of websites' URLs that are most likely phishing websites. The systems block all URLs or IPs in this list. However, this method has a significant drawback. A system must have a phishing attack URL to block it; it does not detect it if the URL is not on the list.
- Traditional machine learning models are utilized to detect phishing attacks. However, traditional machine-learning models need to extract features manually [11]. Hence, it requires human effort and is time-consuming to extract a set of features [12]. These features are based on the available URLs. Thus, when attackers design new phishing URLs, it increases the feature analysis and extraction, leading to a high feature dimension [10]. Even with this effort of analyzing large sets of features and high dimensions, it cannot avoid being attacked with the new phishing URLs [10].
- The advantage of using a DL approach to detect phishing URLs is that a model extracts the features without human efforts for both text and image. However, there are some problems due to the intelligent design of the phishing attacks and the phishing webpage created by new DL methodologies. For example, a model is trained to detect long URLs. But, it does not detect tiny URLs [13]. Also, DL has some drawbacks, such as it requires a huge dataset to train, test and validate the models [12], [14], [15]. It is also costly due to the complexity of DL models [12].

Different types of data can be used to detect phishing attacks, such as URL-based [5], [6], [8], [9], content-based [16], [17], and hybrid-based [10], [18]. URL-based methods extract URL information without exploring anything else, such as webpage content, title, etc. URL-based has the advantage of detecting a phishing message without clicking on the URL and encountering the risk of downloading and installing malicious software. However, extracting only URL-based features causes a lack of vital features of the phishing webpage, such as page title and page code [10]. It is also hard to analyze tiny URLs using only URL-based. Content-based approaches extract information from webpage content such as images, JavaScript, text, HyperText Markup Language (HTML) code, etc. Nevertheless, the content-based approach makes users or systems open the webpage and extract content, leading to a possibility of an attack by downloading malicious software and installing it. The hybrid-based content feature integrates the URL-based and content-based features.

There are several survey papers conducted reviewing phishing detection methods. Each of these papers examines the state-of-art from a different perspective. We summarize them as follows.

First, the authors in [19], [20], [21], and [22] survey state-of-art detection methods based on various feature extraction, selection, and their performance. The authors in [19] survey phishing emails and their detection methods based on Natural Language Processing (NLP) and machine learning. Their survey analyzes spoofing emails and different features extracted from content-based, such as email text, URL-based, or hybrid-based. The authors in [20] survey state-of-art features extract methods from webpages such as URL, HTML, Cascading Style Sheets (CSS), etc. They analyze different types of features, such as URL-based and content-based, and their performance with various machine learning models. The authors in [22] analyze state-of-art detection methods of phishing URL-based using machine learning. Their survey reviews the history of phishing attacks and the current detection methods such as list-based, heuristic strategy, and machine learning-based. They also review a set of frameworks to detect phishing attacks, such as web browser extensions, phone applications, etc. The authors in [21] report a survey on the usage of URL-based, source codes, and image features with machine learning to detect phishing webpages. They study the feature selections for each type, such as tracking the login screen from source code, etc. These surveys [19], [20], [21], and [22] did not investigate applying DL models for feature extraction. They analyze different feature selections and traditional machine learning performance. However, traditional machine learning models have the drawback of manually extracting features from the input data. This problem leads to another problem, such as the extracted feature based on the provided input dataset. Therefore, new datasets need a new feature selection hence increasing dataset dimension. The authors in [22], and [19] briefly mention some DL models such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), etc. Nevertheless, they did not review the state-of-art DL models,

TABLE 1. Limitation of existing survey papers.

			DL Models		Data Preprocessing		
Ref	Dataset	Phishing Attacks	Supervised	Unsupervised	Cleaning	Tokenization	Embedding
[19]	×	×	✓	×	×	×	×
[20]	×	×	✓	×	×	✓	×
[21]	×	×	✓	×	×	✓	×
[22]	✓	✓	✓	×	×	✓	×
[23]	×	✓	✓	×	×	×	×
[24]	✓	×	✓	×	×	×	×
[25]	✓	×	✓	✓	×	×	×
[26]	×	✓	✓	×	×	×	✓
[27]	✓	✓	✓	✓	×	×	✓
Ours	✓	✓	✓	✓	✓	✓	✓

nor did they review each model's data preprocessing and feature extraction.

Second, the authors in [23] and [24] survey the state-of-art detection method based on the learning type. The authors in [23] investigate Artificial Intelligence (AI) techniques of phishing detection such as DL, machine learning, hybrid-based learning, and scenario-based learning for building phishing detection. They study each method and compare its performance, advantage, and limitation. However, their study [23] lacks information about state-of-art models built using DL models. Even though their paper includes a section on DL models, they review some traditional machine learning models instead, such as Support Vector Machines (SVMs), Decision Trees (DT), etc. Therefore, they did not provide more in-depth details of actual models built using DL. Furthermore, it does not provide information on the state-of-art methods of handling data preprocessing and feature extraction. The authors in [24] investigate website phishing detection methods through URL-based, content-based, and hybrid-based. They study these types based on list-based, heuristic rule, and learning-based detection methods. Furthermore, they compare the methods based on their performance and limitation, usage of the third party, and language independence. Nevertheless, their survey reviews a few DL models without providing in-depth details of how these models handle the input data. For example, it did not provide the proposed feature extraction methods for HTML data.

In addition, Table 1 shows the limitation of current studies of phishing detection methods. It shows that most current studies did not explain in-depth data preprocessing. They only study the DL models used in each method without going into more detail about how the model handles input data, such as data tokenization, feature selection, and feature extraction. Therefore, a crucial step is not explained. Studying and comparing the current state-of-art data preprocessing methods is very important due to the disparity in the model's performance with each method. Table 1 also shows a few papers that investigate the unsupervised learning models [25], [27]. In [27], the authors only study one type of unsupervised learning, such as k-means clustering. The authors in [25] study multiple unsupervised learning models. However, their study lacks an explanation of the data prepossessing.

Based on the previous literature, Table 1, and to the best of our knowledge, we summarize the existing survey papers' limitations as follows:

- There is a lack of a survey paper that in-depth reviews DL models to detect phishing attacks. Most of the current surveys only mention the models without their details.
- There is a lack of a survey paper investigating data preprocessing for DL models, such as input tokenization, feature selection, and extraction.
- Although many researchers use these methods, there is a lack of up-to-date comprehensive studies on applying unsupervised URL-based DL models to detect phishing attacks.

Therefore, this paper provides a comprehensive survey investigating URL-based and HTML data preprocessing and DL models to detect phishing attacks. The main contributions of this paper are listed as follows:

- We survey current state-of-art DL models to detect phishing attacks. We investigate their strengths and weaknesses in data preprocessing approaches, such as data cleaning, tokenization, and embedding.
- We categorize the detection methods based on the type of data (hybrid-based, URL-based) and the learning style (supervised, unsupervised) and provide a survey for each category. Then, we study each category starting from input data, data cleaning, tokenization, embedding, feature extraction, and DL model until the model's output.
- We analyze and compare the detection methods' strengths and weaknesses in data preprocessing and feature extraction. In addition, we conduct a general comparison of the models' design and performance.

The rest of the paper is organized as follows. Section II presents a short survey of URL phishing attacks, evaluation models, dataset resources used for training, and the DL process. Section III reviews the state-of-art phishing detection model based on URL and hybrid features. In Section IV,

FIGURE 2. URL components.

we compare these methods. In Section V, we discuss some limitations and challenges of state-of-art DL phishing detection models. Finally, we conclude the paper and present future direction in Section VI.

II. PHISHING ATTACKS, DATASETS, AND DEEP LEARNING PROCESS

A. PHISHING ATTACKS

As discussed in the above section, phishing attacks have increased rapidly. The increase is partially due to the simplicity of the design of these attacks. They can be done anywhere without being in the same place as the victims. These advantages made phishing attacks one of the most dangerous attacks that persons and organizations suffer. It works by sending victims fake URLs using email and social media. The URLs transfer the victims to fake websites to deceive them into sharing personal information such as credit card numbers, login information, etc. Most of the time, phishing URLs are hard to identify due to the similarity between phishing and benign URLs. URL contains four parts, as shown in Fig. 2. First, the URL protocol shows us how the data is transmitted. The second part is the host, including the Top-Level Domain (TLD) and a Second-Level Domain (SLD) [28]. TLD helps distinguish the purpose of the domain; for example, edu represents the domain for education purposes, as shown in Fig. 2. SLD usually contains the website name. The third part is a path with the exact location of the required webpage. The final part can be a query and may have many parameters. Also, it can be an anchor which transfers the user to a specific location on the webpage, as shown in Fig. 2.

There are different techniques that attackers use to design fake URLs. First, an attacker can design a phishing URL by adding Structured Query Language (SQL) injection or Cross-Site Scripting (XSS) attack [29]. Second, the attacker can use the organization name (i.e., SLD) with a different TLD, for example, www.ua.com, instead of www.ua.edu [30]. In this way, the inexpert user may fall into this trap and be a victim of the attack. Therefore, organizations focus on designing systems that identify phishing attacks by analyzing hostname and a combination of SLD and TLD. Third, URLs use a different protocol, such as 'shttp.' Fourth, an attacker can design a URL like a well-known organization URL by simply changing its spelling. Fifth, an attacker can design a URL with random words, resulting in a long URL [4]. Sixth, attackers design URLs and wrap them behind tiny URLs. Tiny URL is a service that shortens the URLs by producing a new URL with a different pattern [31], such as "https://tinyurl.com/brbm97cx" is a tiny URL of the URL in Fig. 2. A traditional detection model fails to detect tiny URLs because they have different patterns than the original URLs. Finally, attackers create a website utilizing well-known blog hosting platforms such as Google Sites [32] and embed the phishing URL as a hyperlink in a fake blog. Hence, the attackers hide the phishing URL inside a legitimate website. Thus, phishing detection techniques may not detect these attacks because they may analyze the URL generated by the Google Sites, a legitimate URL instead of the phishing URL.

B. DATASETS

Designing an excellent model needs to have substantial datasets to train it. Due to the growing number of phishing attacks in recent years, the need for high-quality and new datasets for better model performance has increased. Therefore, we must first review their datasets to understand the state-of-art models better. Thus, Table 2 lists phishing and benign widely used datasets. This section reviews the most common phishing detection datasets used in state-of-art papers as follows.

- PhishTank [33] is the first dataset in the table due to its high usage in phishing detection models. It is a blacklist operated by Cisco Talos Intelligence Group (Talos) containing around 7 million phishing URL records [33]. According to their website [33], two million records are verified as phishing. In addition, 11000 URLs of the two million are online, and the remaining are offline because the phishing URLs are removed whenever the attacks are made. If a URL is online, its webpage could be viewed using PhishTank.
- Alexa [34] is a website organized by Amazon that provides analytic tools to analyze the websites' performance. Such as a website ranking, a tool that lists millions of benign website URLs ranked based on probability.
- JoeWein [35] is an open-source API containing a blacklist of 15000 phishing URLs.
- Common Crawl [36] is a website that crawls through the web and provides many datasets of benign websites and their metadata such as page content, URL, etc.
- OpenPhish [37] is a platform that contains 18 million phishing URLs. It also provides metadata such as page content, IP address, etc.

Table 3 shows examples of phishing URLs. It shows that the most common attack is misspelled well-known websites, such as the URLs in rows 6 and 8.

C. DEEP LEARNING PROCESS FOR PHISHING ATTACK DETECTION

This section reviews the essential DL steps and their applications in state-of-art phishing detection papers. As shown in Fig. 3, DL generally has four steps: data collection, data preprocessing, model training, and model evaluation.

1) DATA PREPROCESSING

Data prepossessing is the first step in machine learning, and DL tasks [46], [47], [48]. It prepares raw data, such as text,

TABLE 2. List of the datasets.

	Ref	Name	Last update	Type	Link
1	[33]	PhishTank	2021	Phishing	https://phishtank.org/developer_info.php
2	[34]	Alexa	2021	Benign	https://www.alexa.com/topsites
3	[35]	JoeWein	2021	Phishing	https://joewein.net/
4	[36]	Common Crawl	2021	Benign	https://commoncrawl.org/2021/10/september-2021-crawl-archive-now-available/
5	[37]	OpenPhish	2021	Phishing	https://openphish.com
6	[38]	Phishing Army	2021	Phishing	https://www.phishing.army/
7	[39]	Kaggle	2021	Benign	https://www.kaggle.com/datasets/cheedcheed/top1m
8	[40]	UCI ML repository	2021	Phishing	https://archive.ics.uci.edu/ml/datasets/phishing+websites
9	[41]	Phishing-Dataset	2022	Phishing	https://github.com/GregaVrbancic/Phishing-Dataset
10	[42]	Parsed DMOZ data	2022	Benign	https://doi.org/10.7910/DVN/OMV93V
11	[43]	Yahoo	2022	Benign	https://webscope.sandbox.yahoo.com/
12	[44]	Yandex	2022	Benign	https://yandex.com/dev/xml/
13	[45]	Phishload	2022	Phishing	https://www.medien.ifi.lmu.de/team/max.maurer/files/phishload/index.html

TABLE 3. Example of URL in the datasets.

	Ref	Datasets	Phishing URL example	Type
1	[33]	PhishTank	https://zimbria.creatorlink.net/	Phishing
2	[33]	PhishTank	https://bancraporinternet.interhants.com/login	Phishing
3	[33]	PhishTank	https://intrbnk.co	Phishing
4	[33]	PhishTank	https://dbs-finance.com/	Phishing
5	[37]	OpenPhish	https://datenighthub.com/rfp/index.php	Phishing
6	[37]	OpenPhish	https://www.facbook.info/mobile.html	Phishing
7	[37]	OpenPhish	https://www.msj.go.cr/	Phishing
8	[37]	OpenPhish	https://dlscord.net/giveaway/steam	Phishing

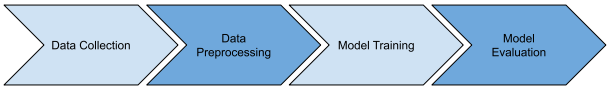


FIGURE 3. DL process.

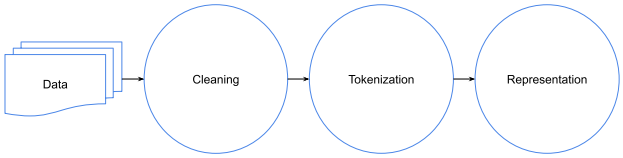


FIGURE 4. Data preprocessing.

images, etc., for model classification. Generally, data preprocessing has different steps based on the type of data and its goals [49]. Thus, this section analyzes the data preprocessing applied for URL-based and hybrid-based phishing detection. As shown in Fig. 4, raw data go through three steps, data cleaning, tokenization, and representation.

First, data cleaning removes unwanted information from the input, such as stopwords, data noise, special characters, etc. However, a URL contains special characters representing essential detail in the URL. For example, the hash sign '#' represents a fragment in the URL. Therefore, removing a special character must be done carefully, so the URL does not lose important details. Thus, most of the URL-based models do not require data cleaning, such as [7], [10], [29], [50], [51], [52], [53], and [18]. However, the authors in [54] aim to extract meaningful words from the URLs. Accordingly, they remove all special characters from the URLs.

Second, data tokenization is to split the text into tokens. Each token can be character-level, word-level, or sentence-level. Character-level is to split the input data into characters. Character-level is helpful when the small detail of the input is important such as single characters or unknown words. Word-level is to split the input data into words. Word-level is helpful when the goal is to learn word meaning. It is beneficial, especially with text classification, such as sentiment analysis. Sentence-level is to split the input data into sentences. The authors of phishing detection papers have different methods for handling input data. For the URL-based, the input data can be split into character-level, such as [7], [29], and [55]. It can also be split into word-level, such as [53]. In addition, some papers use a combination of word-level and character-level, such as [10], [50], [51], [54], and [52]. Splitting the URLs into word-level can be done by using the URL symbols such as '.', ':', '//', etc. The authors in [10] go beyond word-level and character-level. They propose a hybrid-based model using three different data: URL, Document Object Model (DOM), and page content. Therefore, they propose splitting page content (HTML) data into word-level and sentence-level.

Third, data representation converts the input data into numeric values to train a DL model. These values should capture the text's lexical, semantic, and syntactical meaning, so the model trains on these features [56], [57]. In recent years, data representation in NLP is improved rabidly by proposing different techniques such as one-hot encoding, FastText [58], Bidirectional Encoder Representations from Transformers (BERT) [59], Skip-Gram [60], Bag of words [61], etc. Each one of these methods looks at the

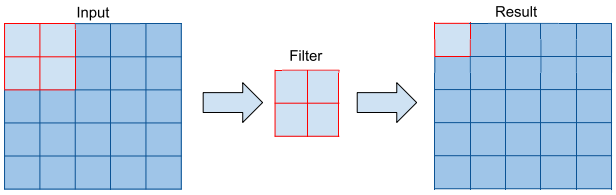


FIGURE 5. Convolutional layer.

problem from a different perspective. A bag of words is a traditional data representation technique used in different data types, such as images, text, etc. In the text, it collects words in documents with a number of its appearance [61]. Therefore, it discards grammar and word order. FastText [58] is an open-source library published by Facebook AI for text classification and word embedding. FastText handles each word as an n-gram; hence each word is split into smaller words. It is helpful, especially with the combination of words in URLs, e.g., checkout is broken into che, hec, eck, and out. Then, each part of the n-gram is hashed to a unique number. Then, it calculates the sum of the n-grams for each word. Then, it calculates the dot product for each word and its neighbor. Finally, it passes the result to the sigmoid function to calculate the match score. One-hot encoding converts each token into a vector with a value of 1 for each sample corresponding to its original category and zeroes to others. Skip-gram is a machine learning model to generate a vector from a text introduced in [60]. Skip-gram handles the text as a word pair, i.e., (context, target). First, the model encodes the text using the one-hot encoding introduced above. Then, the model starts by tokenizing the first word in the sentence and adding the next word in the same pair. For example, the skip-gram generates two pairs $< hello, new >$ and $< hello, world >$ for 'hello new world.' Then, the model moves to the second word in the sentence, 'new.' In this case, there is a word on its left. Therefore, the model adds the word on the left and the right as a target in the pair. For example, the target is the word 'new' in the previous sentence; the model generates two pairs $< new, world >$ and $< new, hello >$. Then, these pairs are passed to a neural network to learn the context of the sentence.

2) DEEP LEARNING MODELS

After data preprocessing, the data is passed to a DL model for training. In recent years, there has been an increase in the number of DL models to detect phishing attacks, such as phishingnet [54], Convolutional Gated Recurrent Unit (CGRU) [29], URLNET [51], etc. These models have one thing in common: applying multiple DL layers such as convolutional, pooling, and fully connected layers. The convolutional layer is essential to the CNN [62]. It is a filter applied to input data to map its features [63]. As shown in Fig. 5, the filter is smaller than the actual data, so it starts from the beginning and slides the window until the end of the input data. Each window calculates the dot products of the input data with the filter. As a result, the dot product generates a new matrix highlighting the main feature of the input data. Thus, the model is trained using these features. The pooling layer applies a pooling operation to reduce the feature dimension generated by the convolutional layers. Many pooling operations can be used, such as max pooling, min pooling, average pooling, etc. Finally, a fully connected layer (FCN) is a traditional neural network. Equation 1 summarizes the FCN operation.

$$h_i = f\left(w_i^T x + b_i\right), \tag{1}$$

where f is the activation function, w_i^T is the weight, b_i is the bias belonging to the previous layer i, and x is the input matrix.

Another standard DL model is the sequence model such as RNN [64], Gated Recurrent Unite (GRU) [65], etc. These models work on sequential data such as text classification, video recognition, etc. Their goal is to remember the previous input to classify the current input. For example, phishing attacks are a type of social engineering attack it reaches a user using social media, email, etc. Thus, phishing attacks contain text such as message content, URL, etc. Text is a sequence of data where the beginning, middle, and end matter to understand the text. Thus, the model should be trained based on the previous feature and generate a new one passed to the next layer. The most common sequence model is Long Short Term Memory (LSTM) [66]. As shown in Fig. 6, LSTM uses three gates for this case: an input gate, a forget gate and an output gate. It also uses two states: the hidden state and the cell state. The hidden state stores information from the previous layer. The cell state is the main difference between LSTM from RNN. It carries information to the entire chain with a minor change in each layer. Thus, each layer receives features from the previous layers. The model accepts inputs from the feature extraction layer, previously hidden state, and cell state and then passes them to the forget and input gates, as shown in Fig. 6. The forget gate decides whether the input from the previous state is relevant or not using Equation (2), and the output is either 0 or 1. Therefore, this output decides which value from the cell state is forgotten. The input gate calculates the importance of the input using Equations (2) and (3) and passes its value to the cell state. Finally, the output gate calculates the value of the hidden state for the next layer. As shown in Fig. 6, the output gate contains information on the previous input using Equation (2). It also contains information on the new value in the cell state using Equation (4). Accordingly, it calculates the value of the hidden state for the following layer using Equation (5).

$$\left(\begin{Bmatrix} i_t \\ f_t \\ o_t \end{Bmatrix}\right) = sigmoid\left(\begin{Bmatrix} W_i \\ W_f \\ W_o \end{Bmatrix} h_{t-1} + \begin{Bmatrix} U_i \\ U_f \\ U_o \end{Bmatrix} X_t + \begin{Bmatrix} b_i \\ b_f \\ b_o \end{Bmatrix}\right), \tag{2}$$

$$\widetilde{C} = \tanh(U_c \bullet X_t + W_c h_{t-1} + b_c), \tag{3}$$

$$C_t = f_t \bullet C_{t-1} + i_t \bullet \widetilde{C}_t, \tag{4}$$

$$h_t = O_t \bullet \tanh(C_t), \tag{5}$$

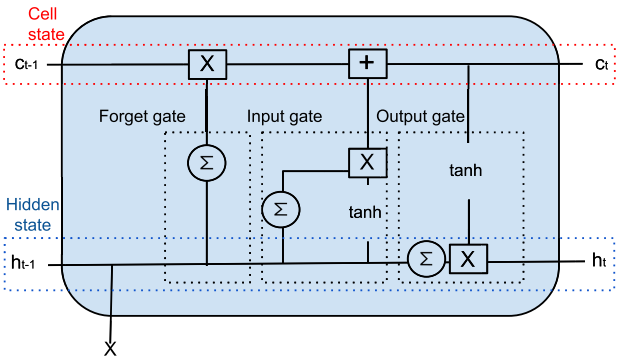


FIGURE 6. LSTM layer.

where i is the input gate, f is the forget gate, and O is the output gate; $\widetilde{C}$ is the current state of the cell memory, C_t is the memory cell state, and h_t is the hidden layer; W and U are the weighted matrixes of each gate, and b is the bias value of each gate; X_t is the current input.

These two models are common in many text classification problems such as sentiment analysis [67], [68], spam detection [69], [70], phishing detection [10], [50], [51], [54].

3) EVALUATION

Before we review the state-of-art methods in the next section, we need to explain the metrics used to evaluate the performance of the proposed models. Most of the state-of-art phishing detection models are binary classification. Therefore, a model contains two classes, positive and negative classes. The positive classes are benign webpages, and the negative classes are phishing webpages. Five standard metrics have been used to assess the performance of the DL model: accuracy, True Positive Rate (TPR), False Positive Rate (FPR), precision, and F1 score. Some papers use all these metrics to evaluate their models, and others use some of these metrics. First, the TPR, also called recall, represents the probability of the correct prediction of the positive class, i.e., a model correctly predicts the input as a benign webpage. Second, the FPR represents the probability of the incorrect prediction of the positive class, i.e., a model predicts a benign webpage, but it is phishing webpage. Third, precision is the probability of correctly predicting a URL as phishing among all correctly predicted values. We have their formulas as follows.

$$Accuracy = \frac{Y_{TP} + Y_{FN}}{Y_{TP} + Y_{TN} + Y_{FP} + Y_{FN}}, \tag{6}$$

$$TPR(recall) = \frac{Y_{TP}}{Y_{FP} + Y_{TN}}, \tag{7}$$

$$FPR = \frac{Y_{FP}}{Y_{FP} + Y_{TN}}, \tag{8}$$

$$Precision = \frac{Y_{TP}}{Y_{TP} + Y_{FP}}, \tag{9}$$

$$F1 = 2 \times \frac{Precision \times recall}{Precision + recall}, \tag{10}$$

where Y_{TP}, true positive, is the number of URLs where the model correctly predicts it as phishing; Y_{FP}, false positive, is the number of URLs where the model incorrectly predicts it as phishing; Y_{TN}, true negative, represents the correct prediction of the negative class, where the model correctly predicts the input as a phishing webpage; Y_{FN}, false negatives, represents the probability of the incorrect prediction of the negative class, where the model predicts a phishing webpage where it is benign.

III. DETECTION METHODS

In this section, we survey the state-of-art phishing detection methods. We divide this section into two subsections based on training types, e.g., supervised and unsupervised learning. We review the data preprocessing and classification methods for each subsection.

A. SUPERVISED LEARNING ALGORITHMS

Supervised learning is a DL task designed by training the model using a dataset that contains input and its labels as output. This dataset includes at least two kinds of labels, e.g., benign or phishing. Using these labels, the model can measure its performance accuracy. This subsection comprehensively studies supervised learning models with URL-based and hybrid-based methods.

1) URL-BASED

URL-based relies on extracted features from only URLs using NLP.

The authors in [7] propose using the LSTM model on the URL character level to extract features from the URL. The model accepts URL as input. Then, they split the URLs into character-level. Then, they convert each URL into one-hot encoding. After that, they convert one-hot encoding output into 128-dimension embedding, which is then passed to the LSTM layers. A sequence model like LSTM helps extract sequence features by learning the characters' patterns for a long period. LSTM learns the sequence patterns for each input. The authors in [7] use LSTM to classify the URL as benign or phishing. As we discuss in Subsection II-C, LSTM uses three gates: input gates, forget gates, and output gates. First, the input gates control the input value to the model from the outside. In this model, the input value comes from the character embedding layer. Second, the forget gates manage which values are passed and ignored. Third, the output gates control which information is present as a model output. Then, the model output is given to the sigmoid function for classification.

The authors in [29] propose a CGRU. CGRU uses a set of malicious keywords included in the URLs to train the model. CGRU is a DL model combining the convolutional layer and GRU. The CGRU accepts an URL as input. The URL usually has some keywords representing different types of attacks. Table 4 shows some examples of malicious keywords. Therefore, the authors in [29] propose using character-level tokenization except for these malicious keywords. For character embedding, they create a dictionary containing 95 URL characters. Then, they convert

each character and the keyword to its corresponding low-dimensional vector. These keywords are essential words that usually represent different attacks, such as XSS attacks that typically have a phrase such as "script" in their malicious URL. The authors in [29] use a CNN to extract URL features before passing them to the classification model. After getting the keywords character embedding, the authors in [29] use a hash to convert these data into a list of integers where each character is equivalent to an integer. The input to the model is a two-dimensional vector spliced into a floating-point matrix and transformed from the original characters. Then, they use Keras embedding to transfer the vector into a 200×64 matrix. The embedding is trainable since it updates the vector in every epoch. After getting the input tensor, it is passed to the multi-core convolutional layer. The convolutional layer receives the input and then performs convolutional operations. The authors in [29] apply 128 convolution layers of different sizes. The convolution layer generates significant training parameters overfitted quickly. Then, instead of using the pooling layer, they use multiple GRU to reduce the number of parameters. GRU is a sequence model. The main advantage of using GRU instead of RNN is that RNN has a short memory. Therefore, GRU helps solve this issue by using reset gates and an update gate. The update gate decides that the information is kept, and the reset gate determines that the information is forgotten. The following equations explain GRU gates and the activation function.

$$u_t = \sigma(W_u \bullet [h_t - 1, C_t]), \tag{11}$$

$$r_t = \sigma\left(W_r \times [h_{t-1}, C_t]\right), \tag{12}$$

$$\widetilde{h_t} = \tanh(W[r_t \times h_{t-1}, C_t]), \tag{13}$$

$$h_t = u_t \times \widetilde{h_t} + (1 - u_t) \times h_{t-1}, \tag{14}$$

where W is the list of the feature extracted from the convolutional layer W=[c_1,c_2,c_3,..,c_n], $C_t \in$ W, and u_t is the update gate. u_t gets the input c_t from the update gate and the previous hidden state and then multiplies its weight W to get its sigmoid value. The reset gate has the same formula as the update gate, with different weights and gate usage. Then, the last feature from GRU is used in the classification layer. Using GRU helps the model retain a critical feature that might get lost during the training. Therefore, GRU keeps these features stored using reset and update gates until the classification layer. Finally, the result of GRU is passed to the fully connected neural network for classification.

The authors in [50] propose a Texception as a classification model that accepts URLs as input. Then, they extract two types of features for classification word-level and character-level. First, they split URLs into words using URL symbols such as ".?/ $-_=$ %&@+;" then they use the FastText [58] pre-trained model to convert words into a vector representation. Second, they split URLs into characters for character-level embedding. They propose a method that converts each character into a corresponding integer by designing a table containing all characters with their index and converting it into a vector representation. This vector is trainable;

TABLE 4. Malicious keywords [10].

URL stack type	Malicious keyword
SQL Injection	and, or, xP_, substr, utl, benchmark, shutdown, hex, sqlmap, md5, hex, select, union, drop, delect, concat, orderby, exec
XSS Attack	script, frame, eval, prompt, alert, JavaScript, cookie, onclick, Onerror, prompt
Sensitive File Attack	access_log, text/plain, phpinfo, proc/self/cmdline,/fckeditor/
Directory Travel	..., 1, ..|,	
Normal	base64, wget, curl, redict, upload, ping, shal, java.lang

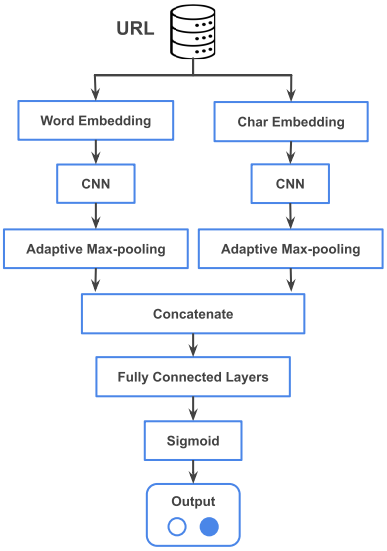


FIGURE 7. Texception architecture [50].

therefore, it updates its value during the training. Texception has two paths for word-level tokenization and character level, as shown in Fig. 7. For each route, the input is passed to multiple blocks. Each block contains one dimension CNN layer with different filter sizes, batch normalization, max-pooling layer, and Rectified Linear Unit (ReLU) activation layer. The output of the block is passed into the adaptive max-pooling layer [71]. Then, the two paths are concatenated and passed into two fully connected layers. Finally, the output is passed to the sigmoid function for classification.

The authors in [51] propose a URLNet model to classify a URL as benign or phishing. The authors in [51] propose to tokenize the URL based on world-level and character-level features, as shown in Fig. 8. First, they tokenize the URL as the character level. Then, they extract all unique characters and symbols in the URLs with their frequencies appearing in the dataset. Then, they replace the characters with less than 100 frequencies with an unknown value (UNK). Then, the model converts each character into a k-dimensional vector by randomly initializing the character with a random number. It updates the value during the training. Second, they tokenize the URL as the word-level feature. They combine two embedding matrices in this step, one for word embedding

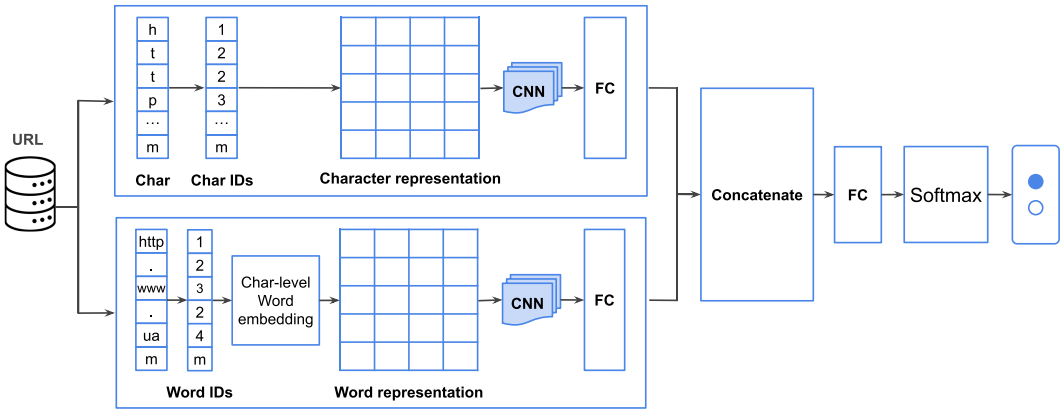


FIGURE 8. URLNet module architecture [51].

$M1$ and another for character embedding $M2$. First, the model extracts all unique words in the dataset by splitting the URL using special symbols such as ". / ?", etc. Then, they assign each word with a unique ID and convert each word into an embedding matrix $M1 = R^{k*L1}$, where $k = 32$, and $L1$ is the maximum URL length in words. Second, the model tokenizes each word into character-levels and obtains its embedding using character-level embedding to generate a new matrix ($M2 = R^{k*L2}$), where $L2$ is the maximum length of URL in character. Then, $M2$ sums with $M1$ to develop a new word embedding $M3 = M1 + M2$. Then, $M3$ is passed to the CNN layers. Next, the output of the CNN layers is passed to the fully connected CNN layers to extract features from the character-level and the word-level in the URL. The CNN outputs are concatenated, passing the result to the fully connected layer and SoftMax for classification.

The authors in [52] propose a parallel neural joint model with a URL as the input to classify whether it's benign or phishing. After the model accepts a URL as an input, it passes the URL through two different data preprocessing processes (image and semantic), as shown in Fig 9. First, the model splits the URL into word-level and character levels, illustrated in Fig 9. Second, it converts each character into its corresponding integer and uses one-hot encoding to generate a vector. Then, it converts each URL into a 2D Matrix by converting each character into a decimal using ASCII. Finally, it reshapes each URL into 2D matrices. The authors in [52] propose using two models, Independent Recurrent Neural Network (IndRNN) and Capsule Net (CapsNet). IndRNN is trained using the semantic feature. Therefore, IndRNN accepts word-level and character-level representations and then applies multiple IndRNNs. IndRNN is a member of sequence neural network models. Accordingly, each neuron processes the output of all neurons independently. Each neuron updates its hidden status using Equation (15).

$$h_t = \sigma\left(W_{st} + U \odot h_{t-1} + b\right), \tag{15}$$

where h_t is the hidden status of the t_{th} layer, W_{st} is the input weight of the vector representation of the t_{th} character in the URL, U is the recurrent weight, and b is the bias value. As shown in Fig. 9, CapsNet is trained on URL 2D matrices. The main advantage of CapsNet is using a pattern's position as a feature while training the model. Therefore, the authors in [52] use vector capsules, dynamic routing, and squash functions to replace neurons, pooling, and ReLU, respectively, in CNN. The output of CapsNet and IndRNN is passed to the attention mechanism. The attention mechanism helps the model utilize the vital part of the input sequence. Therefore, the sequence model must have a fixed size with different URL lengths. Any input with less length is filled with pad value. Hence, it is essential to use the critical part of the input and ignore the rest by the attention mechanism. The authors in [52] propose a method to calculate the attention mechanism using equation (16) for both semantic and visual features:

$$u_i = \tanh(W \times X_j + b), \tag{16}$$

$$u_c = [u_{sem}, u_{vis}], \tag{17}$$

where W is the input weight and u_i is the hidden representation i-th layer. Then, the method concatenates the attention output of the semantic u_{sem} and visual representation u_{vis} and calculates the weight as follows:

$$a_i = \exp(u_i^t u_c) / \sum_i \exp(u_i^t u_c), \tag{18}$$

where u_c is the value of concatenating the attention output of the semantic u_{sem} and visual representation u_{vis}. Finally, the method calculates the weight of the multi-model vector based on the a_i as follows:

$$V_c = \sum_i a_i[Sem_i, Vis_i], \tag{19}$$

where Sem_i is the value of the semantic model, and Vis_i is the value of the visual feature model.

The final result of equation (19) is used as the input to the Sigmoid function.

The authors in [54] propose a PhishingNet model to detect a phishing attack using URL only. The model input is a set of URL U = $u_1, u_2, u_3, \ldots, u_i$, and the output is to be either phishing or benign. This model includes four parts, as shown

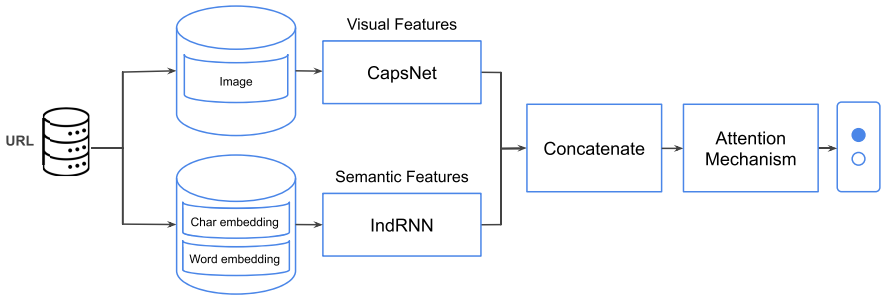


FIGURE 9. Parallel neural joint model architectures [52].

in Fig. 10: an attention mechanism, a character CNN model, a feature fusion method, and a classification method. The data preprocessing includes the following three stages.

- At the first stage, a word in a URL doesn't have a meaning most of the time. Therefore, it is hard to extract semantic meaning from it. Another difficulty is that some words in a URL are a joint of two words, for example, aboutus, loginmember, and emailacceess. Based on these issues, extracting syntax meaning from URLs is challenging. Therefore, the authors in [54] propose a method using the Viterbi algorithm. First, they collected URLs using Common Crawl and extracted their domain names to build their vocabulary. Then, they combine this list of vocabulary with common English names. Next, they use the regular expression pattern [a-zA-Z]+|[0-9]+ to get a token sequence. Finally, they use Viterbi to extract a sequence of the segmented words.
- At the second stage, as shown in Fig. 10, the input is passed through two models: an attention-based hierarchical RNN module and a char CNN module. Therefore, the input needs two types of data preprocessing. First, the attention RNN input, as shown in Fig. 11, is split based on word level, and all special characters such as '.', '/',':', etc., are removed. After that, the authors in [54] split each word into character levels. Then, they transfer each word into a character sequence L_wc and add padding. The module obtains a matrix representing URL characters where each row is a character. Then, using k-dimensional, the input is converted into $M_w = R^{L_w \times L_{wc} \times k}$. The authors in [54] propose character embedding k-dimensional trained using the Skip-gram model. Thus, the authors in [54] propose a k-dimensional skip-gram model. It is trained on 3,295,473,093 unclassified URLs. The goal of k-dimensional is to transfer each character to a vector.
- At the third stage, for the char CNN input, as shown in Fig. 12, the input is converted into a character sequence and then transferred into a representation matrix $M_c = R^{L_c \times k}$ using k-dimensional character level embedding.

For the attention mechanism, after converting the input into a representation matrix, as shown in Fig. 10, Bidirectional Long Short memory (BiLSTM) with the attention mechanism extracts character-level representation from the input matrix. Then, use another BiLSTM to extract word-level features from character-level representation. Next, the input is passed through five convolutional blocks after converting the input into representation matrix M_c as shown in Fig. 11. Each block has four convolutional layers with batch normalization and ReLU activation. Finally, the convolutional block is concatenated as a character-level spatial feature representation.

The authors in [54] propose a cost-sensitive cross-entropy objective function to handle the imbalanced data as follows

$$L_1(p, y) = -(\alpha y log(p) + \beta(1 - y) log(1 - P)), \tag{20}$$

where p is the probability of the predicted value of the legitimate class and y is the corresponding label in one-hot encoding representation. It uses α and β to adjust the penalty miss-classification using the formula below:

$$[\alpha, \beta] = \begin{cases} \left[\frac{n_{all}}{2n_0}, \frac{n_{all}}{2n_1}\right] : n_0 \neq 0 \wedge n_1 \neq 0 \\ \left[\frac{N_{all}}{2n_0}, \frac{n_{all}}{2n_1}\right] : Otherwise \end{cases} \tag{21}$$

where n_0 is the number of benign URLs, n_1 is the number of phishing URLs, and n_{all} is the number of all samples. Therefore, if either n_0 or $n_1 = 0$, it replaces it with N_{all}, the number of all samples in all training sets, n_0, the number of legitimate samples in all training sets, and n_1, the number of phishing samples of all training sets. They use the following objective function [72] to train the model by minimizing the objective.

$$L_2 = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} R\left(x_i, y_j\right), \tag{22}$$

$$R(x_i, y_j) = \begin{cases} (-(x_i - y_i - \gamma))^p : x_i - y_j < \gamma \\ 0 : Otherwise, \end{cases} \tag{23}$$

where x_i is the classifier output of the phishing URL, y_i is the output of the legitimate example, m is the total number of classification outputs of phishing URLs, and n is the total number of classifications of benign URLs. We have $0 < \gamma \leq 1$ and $p > 1$. Therefore, the proposed training objective is defined as follows:

$$L = \theta L_1 + (1 - \theta) L_2, \tag{24}$$

where $\theta \in [0, 1]$

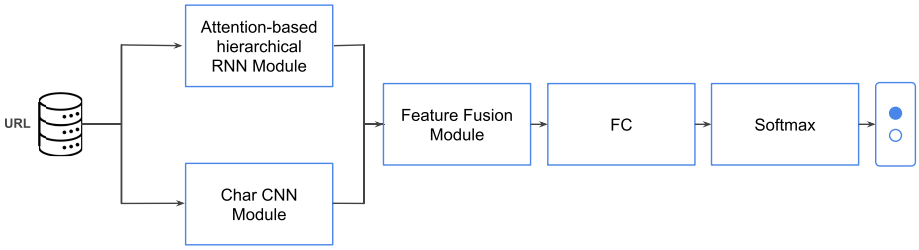


FIGURE 10. PhishingNet overview [54].

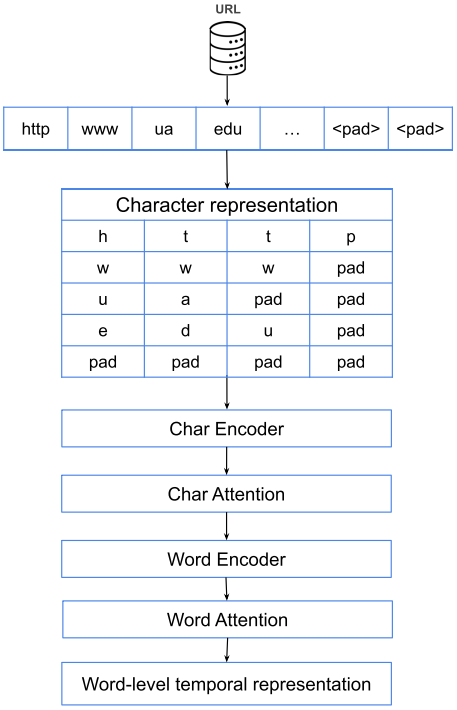


FIGURE 11. The process in Attention-based RNN mechanism [54].

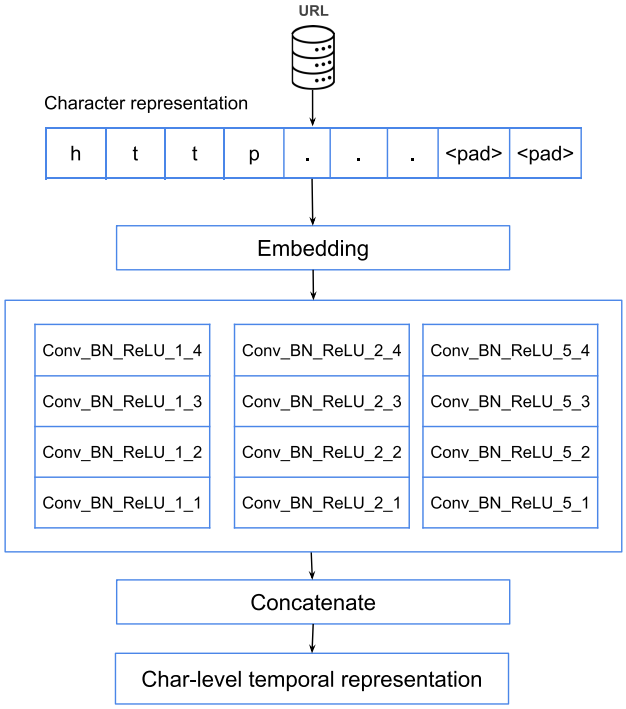


FIGURE 12. The process in character CNN step [54].

2) HYBRID-BASED

This section reviews some state-of-art DL models that use the hybrid-based feature to detect phishing attacks. Hybrid-based features combine URLs and other features such as HTML content, JavaScript, Hyperlink, email headers, etc.

The authors in [10] propose a hybrid-based model in which the input combines URL and other features such as page content, email header, email signature, HTML, etc. The authors in [10] propose a web2vec model to classify webpages as phishing or benign. The model's inputs are a set of webpages $N = \{P_1, P_2, P_3, \ldots, P_n\}$ and outputs are +1 (benign) or -1 (phishing). In addition, each webpage is a set of features $Pi = \{u_i, h_i, d_i\}$ where u, h, and d represent a URL, a page content, and a DOM structure, respectively. Therefore, the model has three parts: data preprocessing, feature extraction, and classification. For the URL data preprocessing, the authors in [10] tokenize the input into word and character levels.

- For the character level tokenization, the URLs are split into character levels to get a sequence of characters.
- Each character sequence is normalized into fixed-length by zero-padding, and the method assigns each character to a unique number.
- Each character sequence with its corresponding integer is encoded to generate a one-dimensional vector using one-hot encoding.
- For the word-level tokenization, web2vc splits the URLs into word-level by dividing each URL into the protocol, SLD, TLD, path, and query by separators '.', ':', '//', etc. After splitting the URLs into word sequences, each word is assigned a unique integer.
- Each integer is encoded using one-hot encoding.

Second, web2vec extracts word-level and sentence-level corpora from text information from the page content. Word-level corpora of page content are similar to word-level corpora of URLs. Sentence-level corpora of page content are obtained using '.' as the separator since only text information is used instead of multimedia, CSS, and HTML tags. Third,

web2vec extracts corpora from the DOM structure. HTML pages containing tags with nested relationships. Therefore, web2vec extracts main tags from DOM, such as head, body, title, etc., and avoids using their attributes, texts, and comment nodes. There are three steps to extract corpora from the DOM structure as follows.

- First, construct the DOM tag sequence for each webpage, obtain the root node of the DOM tree, and use it as the current layer and the first element of the tag sequence.
- Second, use the breadth-first strategy to traverse layer by layer starting from the current layer. This specific method is to traverse the child nodes of the nodes in the current layer from left to right and save them in a sequence.
- Third, repeat until all layers are scanned and return to the tag sequence.

After completing the above steps, the webpage is converted into a sequence of DOM tags. Then, the DOM tag sequences formed by all webpages are assembled, and the HTML tags are regarded as the words constituting the sequences, thereby constructing word-level corpora.

In the embedding step, the authors in [10] propose a one-hot encoding method of transferring words to vectors for webpage representation. For example, at the URL character-level, each URL is converted into vector representation using one-hot encoding $g_i = \{g_{i1}, g_{i2}, g_{i3}, \ldots, g_{im}\}^T$ where i-th g is the i-th character in g and m is the size of the character dictionary extracted from URLs in the dataset. Each URL is converted into one-hot matrix where $G_{m*n} = \{g_1, g_2, g_3, \ldots, g_n\}$ where n is the size of the URL character-level. Then, apply a single neural network to the matrix G to extract the character embedding.

$$S^c = WG = \begin{bmatrix} w_{11} & w_{12} & \ldots & w_{1m} \\ w_{21} & w_{22} & \ldots & w_{2m} \\ \vdots & \vdots & \ldots & \vdots \\ w_{p1} & w_{p2} & \cdots & w_{pm} \end{bmatrix} \times \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ g_{p1} & g_{p2} & \cdots & g_{mn} \end{bmatrix}, \tag{25}$$

where S^c is the embedding representation matrix for the character level, W is the weight matrix of the embedding layer $W \in R^{p*m}$, and p is the embedding dimension. Repeat the same method for DOM and page content. After extracting the representation matrix for the input, web2vec concatenates these features into one representation; therefore, webpage P_i representation is $P_i = \left(S_i^c, S_i^w, C_c^s, C_i^w, D_i\right)$ where S_i^c and S_i^w are representation vectors of URLs; c and w represent the character-level and the word-level, respectively; C_c^s and C_i^w are representation vectors of webpage content, and s represents the sentence-level; D is the representation vector of DOM. All these vectors have fixed lengths with the same dimension.

Web2vec applies two well-known DL algorithms for feature extraction: CNN for extracting local features and BiLSTM for processing sequence information. As shown in Fig. 13, web2vec applies multiple convolutional kernels, and each kernel applies the convolutional operation. It also uses pooling to reduce the dimension of the feature map with the maximum pooling. The convolutional operation is $h_i = f(w_i x + b_i)$, where w is the weight, x is the input matrix, and b is the bias value.

The convolutional operation output has a large number of parameters. Therefore, web2vec uses the maximum pooling to reduce these parameters as follows.

$$X_j = max(h_i). \tag{26}$$

After CNN extracts the local feature, a sequence model BiLSTM is used to learn the relationships among extracted features. BiLSTM is an improved version of LSTM with a two-direction LSTM forward and backward. LSTM as GRU helps increase the algorithm's memory to learn the sequence of the model. LSTM uses three gates for this case, an input gate, a forget gate, and an output gate.

The model accepts the input from the extracted features and uses the following equations:

$$\left(\begin{Bmatrix} i_t \\ f_t \\ o_t \end{Bmatrix}\right) = \left(\begin{Bmatrix} W_i \\ W_f \\ W_o \end{Bmatrix} h_t - 1 + \begin{Bmatrix} U_i \\ U_f \\ U_o \end{Bmatrix} X_t + \begin{Bmatrix} b_i \\ b_f \\ b_o \end{Bmatrix}\right), \tag{27}$$

$$\widetilde{C} = \tanh(W_c h_t - 1 + U_c X_t + b_c), \tag{28}$$

$$h_t = u_t \times \widetilde{h}_t + (1 - u_t) \times h_{t-1}, \tag{29}$$

$$C_t = f_t \bullet C_{t-1} i_t \bullet \widetilde{C}_t, \tag{30}$$

$$h_t = o_t \bullet \tanh(C_t), \tag{31}$$

where i is the input gate, f is the forget gate, and o is the output gate; $\widetilde{C}$ is the current state of the cell memory, C_t is the memory cell state, and h_t is the hidden layer; W and U are the weighted matrixes of each gate, and b is the bias value of each gate.

Finally, the output of the BiLSTM is passed to an attention mechanism for a better decision to highlight the important value of the input feature by giving each feature a different mechanism as follows:

$$a = \tanh(h), \tag{32}$$

$$ar = softmax(w^T a), \tag{33}$$

$$x = \sum_t \alpha h, \tag{34}$$

where h is the previous layer output value, ar is the attention of h, and x is the feature vector. The feature vector outputs from the attention mechanism for each input, such as URL, HTML, and DOM, are concatenated and then passed to a fully connected layer and a sigmoid function for the prediction.

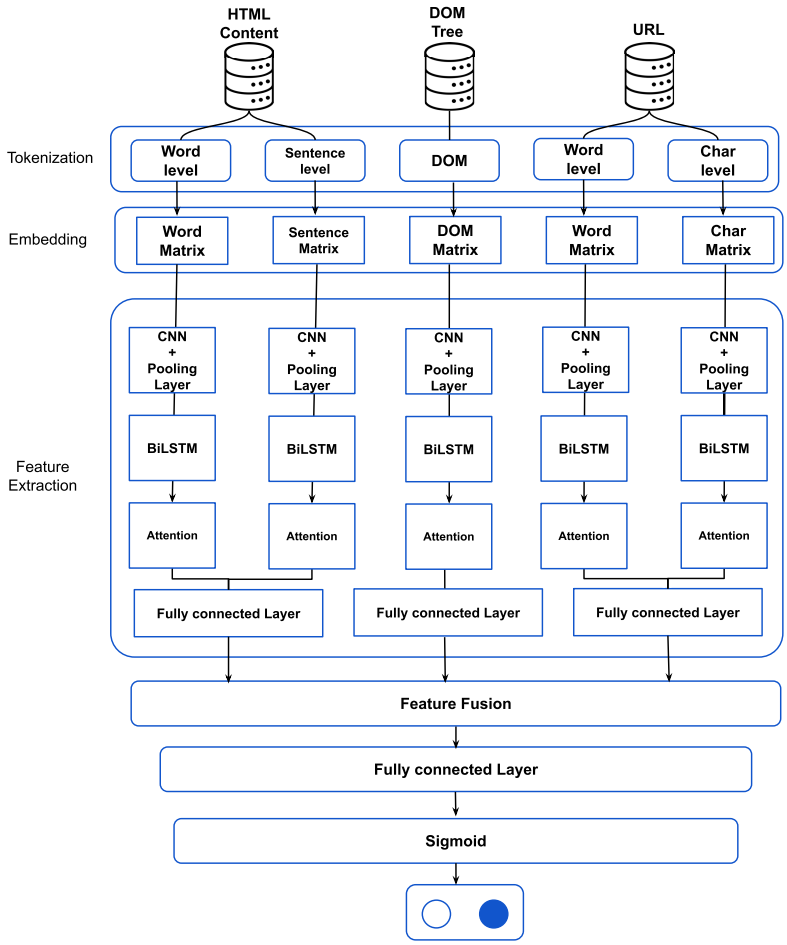


FIGURE 13. Web2vec framework [10].

The feature vector outputs from the attention mechanism are concatenated to a fusion vector for a fully connected layer and a sigmoid function for the prediction.

The authors in [18] propose the OFS-NN model to detect a phishing website. Fig. 14 shows the model architecture. First, the model accepts the URL as input. Then, the model passes the URL to the blacklist and whitelist and classifies it as phishing or benign, respectfully. Suppose the URL is not on these lists. In that case, it passes to OSS-NN in three steps: feature extraction, feature selection, and classification. The authors in [18] propose to extract three features: URL set features, page content set features, and domain features. First, they extract the address bar feature from the URL, such as URL length, whether the URL contains an IP address, using a tiny URL, or using symbols such as @,//,., etc. Second, they extract abnormal features from URLs such as a request URL, URL of anchor, links in tags, etc. Third, they extract HTML and JavaScript features such as whether the website disables the right-click, whether it allows popup windows, etc. Finally, it extracts domain features such as age, website traffic, the number of pages linking to this domain, etc. As shown in Table 5, each feature has a value between -1 to 1. The URL which contains the corresponding feature is labeled as 1; otherwise, it is labeled as -1. Example feature items containing three values 1,0,-1 in Table 5 include 2, 7, and 14. These values have a condition; for example, for item 2, if the URL is more extended than 75 characters, it is assigned as 1, and if the URL is less than 53, it is set as -1; otherwise, it is set as 0 (meaning no effect in the model). These features are extracted manually for each URL and used as input to the neural network for classification. The authors in [18] propose the Feature Validity Value (FVV) index's sensitive model. This model helps select the optimal features from Table 5.

$$FVV = P\left(A_X = positive\ and\ Y = positive\right) + P\left(B_X = negative\ and\ Y = negative\right), \quad (35)$$

$$p = \sum_{i=1}^{m} \frac{FVV_i}{(2 \times m)}, \quad (36)$$

where m is the number of features. This threshold helps reduce the number of features based on the URL inputs. The authors in [18] design a neural network model with three layers (input layer, hidden layer, and output layer). The input layer is the optimal feature vector. Then, each input is connected to a hidden layer connected to multiple fully

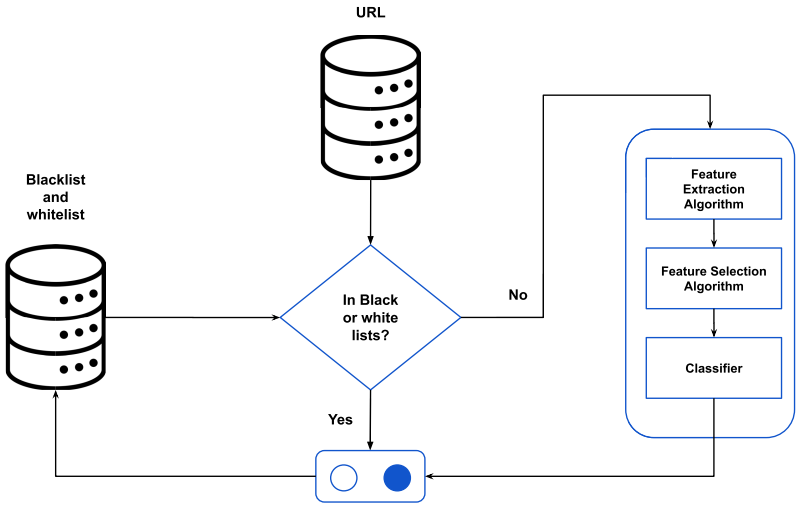


FIGURE 14. OFS-NN Model architecture [18].

connected hidden layers. Finally, the output of the hidden layer is passed into ReLU as an activation function.

B. UNSUPERVISED LEARNING ALGORITHMS

Unsupervised learning is a DL task where the model learns and clusters the data based on a hidden pattern. The unsupervised model extracts these patterns without data labels and human interaction. The model learns the differences and similarities between the data and then groups the output.

The authors in [55] propose to handle this issue as an anomaly detection problem. Anomaly detection often adopts a data mining method that detects unusual data [73]. Therefore, the authors in [55] propose training a convolutional autoencoder on only benign URLs. As shown in Fig. 15, the model accepts URLs as inputs and unsupervised learns from them to extract abnormal scores. Then, the scoring is passed to CNN as a URL feature to classify them as phishing or benign. Thus, the model has two steps: data preprocessing and phishing URL detection based on convolutional Auto-encoder(AE). The main idea of data preprocessing is to transfer input characters into vectors and produce character-level representation. Therefore, the authors in [55] use a built-in python function ord() to split a URL into a sequence of characters. It converts each character to its corresponding integer by extracting ASCII code, passing it to one-hot encoding, and creating a vector of size (the size of a dictionary). The vector is fed to the AE to extract character-level representation. Then, it is used as the input to the model. The AE is unsupervised learning which encodes an input into different dimensions and decodes it to the same dimension. The AE can be used in various data science applications such as data compression, feature extraction, dimension reduction, etc. For example, the authors in [55] use the AE to maximize the reconstruction error of unknown URLs. They also use a convolutional operation to extract the feature. The outputs of the convolutional process are passed to the max-pooling layer

TABLE 5. OFS-NN propose features [18].

No	Phishing Item	Value
1	URL contains an IP Address	1,-1
2	Length of the input URL	1,0,-1
3	Uses URL shortening services	1,-1
4	URL contains @ symbol	1,-1
5	URL uses redirect symbol /	1,-1
6	URL contains. Symbol	1,-1
7	Number of '.' in the URL	1,0,-1
8	Protocol and SSL certificate status	1,0,-1
9	Domain registration length	1,-1
10	Favicon	1,-1
11	Uses non-standard port	1,-1
12	HTTPs token in the domain part	1,-1
13	Request URL	1,-1
14	URL of anchor	1,0,-1
15	Links in Meta, Script, Link	1,0,-1
16	Server Form Handler	1,0,-1
17	Submits Info.To mail	1,-1
18	Abnormal URL	1,-1
19	Website forwarding	1,0,-1
20	Status bar customization	1,-1
21	Disables right-click	1,-1
22	Uses pop-up window	1,-1
23	Iframe redirection	1,-1
24	Age of domain	1,-1
25	DNS record	1,-1
26	Website traffic	1,0,-1
27	PageRank	1,-1
28	Google index	1,-1
29	Number of links pointing to the page	1,0,-1
30	Statistical-reports based features	1,-1

to reduce the dimension. The convolutional AE has two parts, compressing the data (encoding) and decompressing the data (decoding). After extracting the character-level feature, the model uses Euclidian distance to calculate the loss. The loss function is the distance between the input benign URL x_i and the reconstructed URL $\hat{x}_i$ as follows:

$$L_{MSE} = \sum_{i} \left(x_i - \hat{x}_i\right), \tag{37}$$

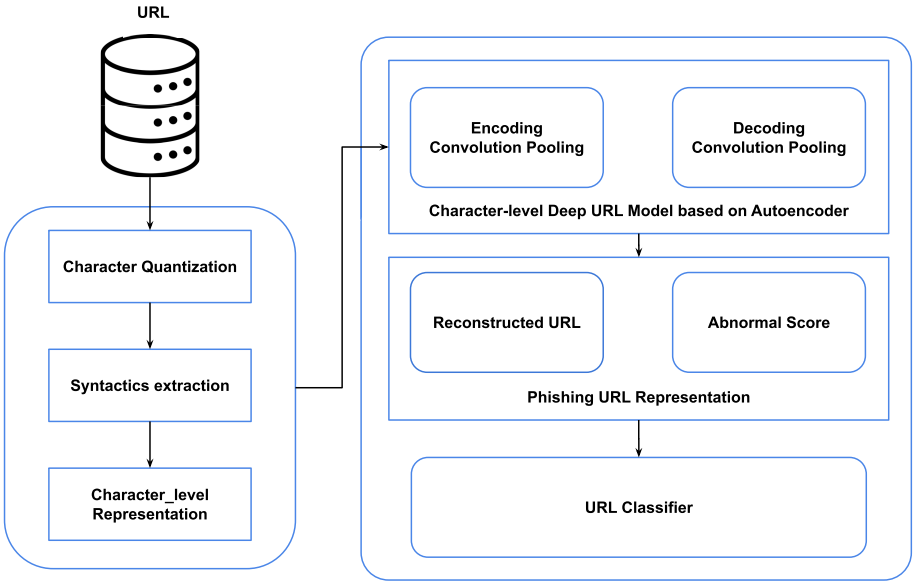


FIGURE 15. Proposed method [55].

Then, the authors in [55] use Stochastic Gradient Descent (SGD) to train the model with the loss function L_{MSE} to minimize the loss value as follows.

$$NN = argmin_{\theta} \sum_{x_i} \sum_{i} \left(x_i - \hat{x}_i\right), \tag{38}$$

where NN is the basic neural network method applied to the loss function L_{MSE}, and θ is the encoding and decoding parameter to minimize the loss function L_{MSE}.

Furthermore, the authors in [55] propose to use two classification methods: an abnormal score and a CNN classifier. First, the abnormal score is the distance between the input benign and the output of the convolutional AE. They define the abnormal score S as follows:

$$S_t\left(x_i\right) = d\left(x_i, \widehat{x_i}\right) = \left\|x_i - \widehat{x_i}\right\|_2^2 \tag{39}$$

where S_t is the abnormal score with the threshold t. They apply the threshold technique to the abnormal score. Finally, they use the CNN models to classify the reconstructed URL image $\widehat{x_i}$ with the SoftMax optimization and cross-entropy as a loss function.

1) URLDEEPDETEC [53]

The authors in [53] propose supervised (LSTM) and unsupervised learning (clustering) models. They propose URLdeepDetetc, which accepts a URL as input and detects whether it is phishing or benign. The model splits the URL at the word-level using Natural Language Toolkit (NLTK). Since the model has two techniques, each has a different data preprocessing. First, for supervised learning, the model converts each word into vectors using Wrod2vec embedding [74]. Word2vec is a word embedding pre-trained neural network model. It creates a vector for each word. Therefore, each word, its synonym, and the word with a close relationship have a close vector value; for example, king and man have an immediate value. The Word2vec outputs are passed to the LSTM model. Second, for unsupervised learning, the authors propose to extract lexical features from the input, such as the URL length, the number of non-numeric in the URLs, the number of digits in the URL, etc. Then, the model gets the sum value of these features. These values are passed into the k-means clustering.

IV. COMPARISONS OF THE STATE-OF-ART DETECTION METHODS

This section compares state-of-art papers based on: data preprocessing, feature extraction, methods, and results.

A. DATA PREPROCESSING

Data preprocessing is an essential step in DL. It contains three parts: data cleaning, tokenization, and embedding. Data cleaning removes unwanted information from the datasets, such as special symbols (like @, //, #, etc.), step words, data noise, etc. Data tokenization splits the text into pieces based on the embedding level; for example, it splits the text based on the character level or the word level. Data embedding transforms each token into a corresponding integer and maps it into a vector representation. Many examples of data embedding in NLP have been done; some of them are trained in a massive amount of data, such as BERT, Word2vec, etc. Therefore, in the previous state-of-art papers, the authors use a different technique to handle the data preprocessing, as shown in Table 6.

The authors in [10] propose combining the character and word levels. Therefore, they split the URL into character sequences and word sequences. For the word sequences, they

TABLE 6. Data preprocessing comparison of state-of-art papers.

Ref	Data cleaning	Data tokenization	Data embedding	Disadvantage
[7]	No cleaning needed	Character level	Keras embedding	Keras embedding is training based on the input data; therefore, if the word is outside of the input, it is considered as [UNK].
[10]	URL: No cleaning needed	Character level and word level	Web2vec	Time-consuming due to three different types of data
[18]	No cleaning needed	No tokenization	Using the extracted feature of each URL as input to the model.	Human efforts to extract features for each input.
[29]	No cleaning needed	Character level	Keras embeddings	Keras embedding is training based on the input data; therefore, if the word is outside of the input, it is considered as [UNK].
[50]	No cleaning needed	Character level and word level	Word-level: Pre-trained fastText model Character-level: Bag-of words.	Using a pre-trained function with a URL that contains random words produces a large number of [UNK] values.
[51]	No cleaning needed	Character level and word level Word-level: Combine Bags-of-words and character embedding	Bag-of-words	Time-consuming due to embedding character-level and word-level twice.
[52]	No cleaning needed	Character level and word level	Semantic features: One-hot encoding Visual feature: Convert each character into decimal using ASCII and reshape it into a 2D matrix	Time-consuming due to converting each URL into decimal value and reshaping it into a 2D matrix.
[54]	Cleaning the URL from special characters such as (':','/',' +',etc)	Character level and word level	Skip-gram embedding	Cleaning a URL from a symbol character results in removing information since the special characters in the URL have meaning.
[55]	No cleaning needed	Character level	One-hot encoding + Autoencoder	Time-consuming
[53]	No cleaning needed	Word-level	Word2vec Lexical features	Using a pre-trained function with a URL that contains random words produces a large number of [UNK] values. Time-consuming + less accurate with new datasets

split it based on URL symbols such as'.', ':', '//', etc. The URL is divided into the protocol, TLD, SLD, paths, and query using these symbols. Then, they use one-hot encoding to convert the character and word sequences into vectors.

The authors in [29] define a set of keywords extracted from the URL. Then, they tokenize each URL character level except for the collection of keywords. Next, they use a hash to map each character into a vector. The main drawback of this paper is that using the keywords makes the model assume that the phishing URL is only the one with these keywords.

The author in [7] proposes using one-hot encoding to convert the sequence of characters into vectors. The main drawback of this method is that converting a vector into embedding based on the train data results in an unknown value when the word is out of input data. For example, let us assume that faceb00k is not in the dataset; the model is not trained on it, so it generates [unk] instead, even though this word might be a phishing word.

The authors in [54] propose to extract a meaningful value from the URL. Then, they use different preprocessing for the CNN model and the attention RNN. Therefore, they divide the URL into sequences of words for the attention RNN. Next, they divide it into two sequences of characters. Then, they use the skip-gram model to convert the sequence into a vector. The main drawbacks are time-consuming and removing the special characters containing essential information regarding the URL.

The authors in [50] propose extracting features from the word level and the character level. It uses the FastText pre-trained model to convert words into a vector representation. For character-level embedding, they propose a method that converts each character into a corresponding integer using a table containing all characters with their index and converting it into a vector representation.

The authors in [52] propose to split the URL into two features: a semantic feature and a visual feature. First, the semantic feature is split the URL into the character level and the word level. Then, use one-hot encoding to convert each level into vectors. The visual feature converts the character level into a 2D matrix by reshaping the ASCII of each input.

The authors in [18] propose using 30 features extracted from the URL. Each feature is assigned with 1 and -1 based on their existence in the corresponding URL. Then, use the FVV index to evaluate the feature-based usefulness in detecting phishing URLs.

The authors in [55] propose using the ord() function to convert each character to its corresponding integer. Then, they use one-hot encoding to convert each sequence into vectors. Finally, they use the AE to extract the character-level

TABLE 7. Comparison of feature extraction of the state-of-art papers.

Ref	Feature extraction	Advantage	Disadvantage
[7]	LSTM	Extract the relationship between each sequence.	Overfitting and lost information for long dependency text
[10]	CNN + BiLSTM	Extract local and global feature	Time-consuming
[18]	FCN	Extract local feature	Time-consuming
[29]	CNN + GRU	Extract local and global feature	Time-consuming
[50]	CNN	Extract local feature	It does not extract the relationship in the text.
[51]	CNN	Extract local feature	It does not extract the relationship in the text
[52]	CapsNet + IndRNN	Extract local and global feature	Time-consuming
[54]	CNN + BiLSTM	Extract local and global feature	Time-consuming
[55]	CNN	Extract local feature	It does not extract the relationship in the text
[53]	LSTM K-means	Extract the relationship between each sequence. Unsupervised clustering the input into two groups based on a distance relationship.	Overfitting and lost information for long dependency text It is easily affected by the outlier, resulting in a large number of false positive and false negative predictions

representation. The main drawback is that using the AE focuses on extracting as much information rather than much-related information, resulting in time-consuming and noise.

The authors in [53] propose two data preprocessing techniques for supervised and unsupervised learning. First, for supervised learning, the URL is split into word-level tokens using NLTK. Then, each token is passed into Word2vec and converted into word embedding. NLTK is usually used with natural language. Therefore, it splits words based on symbols such as dots, spaces, etc. However, many symbols in the URL that NLTK doesn't use for splitting produce many combination words. Hence, using Word2vec with combination words has a large [UNK] value. Second, for unsupervised learning, the sum of URL lexical features such as URL length, the number of the alphabet in the URL, and the number of the digit in the URL, converts each alphabet in the URL domain into a number, etc. Using the lexical feature results in time efforts and is less accurate with the new datasets.

B. FEATURE EXTRACTION

Feature extraction is where the model reduces the number of features from the input and uses the most related training features. This step differentiates between DL and machine learning. Machine learning needs a human to extract features from the input data. However, DL focuses on learning from the input and its label to extract the most related features. As shown in Table 7, the most common algorithm to extract features in recent years is CNN. The authors in [10], [29], [54], [50], [51], [75], and [55] propose using CNN for feature extraction.

The authors in [10], [29], and [54] propose a combination of CNN and sequence models such as BiLSTM and GRU to extract features. The advantage of using a combination of two models is that CNN extracts spatial features, whereas the sequence extracts the relationship feature in the URL. However, as a result, using a combination of two models is time-consuming.

The authors in [7] propose using LSTM to extract the relationship feature. The advantage of using only LSTM is to extract the relationship feature. However, LSTM struggles with raw data since LSTM can't capture long dependency data.

The authors in [18] propose a neural network that contains three layers: input, hidden, and output. The hidden layer is a combination of seven fully connected layers. Using this model with 30 features makes the model suffer from overfitting. Also, the model needs to extract features manually from the input, which is time-consuming.

The authors in [52] propose a DL model for each semantic or visual feature. For semantic features, the authors propose using the IndRNN model. For visuals, the author proposes using the CapsNet model. Then, they concatenate the output of these features and pass it into the attention mechanism. The disadvantage is replicating the features using the

TABLE 8. General comparisons of the state-of-art papers.

Ref	Feature extraction	Methods	Advantage	Disadvantage	TPR	FPR	F1
[7]	LSTM	A simple binary classification problem	Fast	The model suffers from handling a new URL.	93%	Not used	98%
[10]	CNN + BiLSTM	Combine URL, page content, and DOM tree	The ability to extract features from different resources, which makes the model able to detect tiny URL	Time-consuming.	98%	25%	99%
[18]	FCN	Using multiple features such as page content, URL, domain	The ability to extract features from different resources which makes the model able to detect tiny URL	Time-consuming	95%	Not used	96%
[29]	CNN + GRU	Multi-classification	The ability to detect different types of attacks using URLs.	The model assumes that the input has the defined keywords The model assumes that the URL is always a long URL	99%	Not used	99%
[50]	CNN	Using pre-trained model such as FastText.	Increase the embedding accuracy with known words.	It does not extract the relationship in the text. It results in a large number of [UNK] values due to using a pre-trained model	48%	Not used	Not used
[51]	CNN	Build a word embedding with combination of word-level and character-level	It extracts more information of the URL	It does not extract the relationship in the text	98%	Not used	Not used
[52]	CapsNet + IndRNN	Extract local and global feature	It detects the exact match of phishing URL	Time-consuming	99%	99%	99%
[54]	CNN + BiLSTM	Imbalanced datasets	Handel the imbalanced data Time-consuming	Overfitting	Not used	0.002%	Not used
[55]	CNN	Anomaly detection	The ability to detect a new URL	Time-consuming	89%	Not used	Not used
[53]	LSTM	Learning the relationship of the vector produce by word2vec model.	The model speed	The model suffers from handling a new URL.	97.7%	Not used	98.3%
	K-means	Clustering the extracted feature into two groups	Simplicity	Outlier affect the model performance.	95.7%	Not used	96.8%

character-level two times in visual and semantic features. Therefore, it is time-consuming.

The authors in [53] propose two models: LSTM and k-means. LSTM is used with word2vec to extract the relationship of each word in the URL. K-means clusters the input URL features into benign or phishing based on its lexical feature.

The authors in [51], [55], and [50] propose to use CNN only. CNN helps extract local features from the text. However, it does not learn the long relationship feature from the text. Based on the previous observation, we can see that the combination between CNN and BiLSTM is the most used feature extraction.

C. GENERAL COMPRESSION

Each paper handles the problem in a different matter. In [54], the authors see the problem as a simple binary classification. Their proposed model is to use simple embedding with LSTM. Due to the simplicity of this model, it is fast, but applying it to real-world data has difficulty. Because their embedding model is simple and trained on the input data, resulting in [UNK] value as soon as the model gets new data. Their model also assumes that the URL input is a long URL.

In [29], the authors view the problem as a multi-classification problem. Their model classifies phishing based on the type of attacks. Therefore, their solution is to add a keyword for each attack. Their model also assumes that the input is always a long URL.

The authors in [10] solve the issue of tiny URLs by looking at the problem as a combination of multiple features such as webpage content, URL, and DOM. Thus, they combine these features to detect a phishing website. This model might solve the problem of tiny URLs because the model extracts feature from page content, URL, etc., instead of only using the URL features, which are hard to detect with tiny URLs. However, the disadvantages of this model are the time-consuming, significant effort, and it was trained on a small amount of data, making it less accurate with a new dataset. Also, it is risky in phishing attacks to click on the URL, resulting in downloading malicious software.

In [55], the authors look at the problem as anomaly detecting. The model is trained to detect benign URLs, and consequently, it measures the distance between the new input and the benign URL. As a result, the model can detect a zero-day URL.

In [7], the authors look at the problem as an imbalance problem. They propose a loss function that can measure the loss among all records in the dataset rather than each batch. Their solution makes the model overfitting and is also time-consuming.

In [51], the authors use CNN to extract features from the character level and word level in the URL. First, they convert each word and character to its corresponding integer. Then, they propose two embeddings for word and character levels. The word-level embedding is combined with word-level and character-level embedding. As a result, their model can extract local features, ignoring the relationship feature between the URL words.

The authors in [51] propose to combine the word level and the character level. It uses the FastText pre-trained model to convert words into a vector representation. For character-level embedding, they propose a method that converts each character into a corresponding integer using a table containing all characters with their index and converting it into a vector representation. They use CNN to extract features for both the character level and the word level. This model has two drawbacks; first, since the proposed model uses special characters to split the word, the pre-trained model results in more [UNK] value. For example, assuming the URL is ''www.example.coom/aboutus,'' using their method to split the word results in ''www'', ''example'', and ''aboutus''; therefore, the word ''aboutus'' is an unknown word for the FastText pre-trained model, and it results in a [UNK] value. Second, the model is not able to detect tiny URLs.

The authors in [52] propose to combine local features using CapsNet and spatial features using IndRNN. Therefore, their model detects the input's relationship feature and detects each input's local feature. They use CapsNet to use the advantage of the capsule of detecting the specific location of each feature in the input. However, using two features replicates the input data, resulting in time-consuming and reducing the model productivity of detecting new input data.

The authors in [18] propose manually extracting 30 features from the input layer. Usually, extracting many features fails when the attacker designs new phishing URLs. It adds more features to the list of analysis and extraction, leading to a high feature dimension. They use the proposed feature selection to select the feature. Then, they use a simple neural network for training the model.

The authors in [53] propose two models: supervised and unsupervised learning. For supervised learning, they split the URL using NLTK. Then, they use LSTM with word2vec for training the model. For unsupervised learning, they manually extract five features and get the sum of these features. The sum is passed to the k-means clustering model. The k-means model groups the input into two groups. The k-means model is a simple machine-learning clustering model. Therefore, the drawback is that the input outlier affects the cluster performance.

Table 8 shows the general comparison of these papers.

V. DISCUSSION

The introduction section explains the phishing attacks and their impacts from the user perspective and within the organization. The problem of phishing attacks is an underrated problem. Recent phishing URLs are more complex than before. An attacker can use DL or NLP to mimic the URL. We explored in previous sections the state-of-art of phishing detection methods. We have seen different methods to solve phishing detection. Even though these methods show promising results, they usually have limitations affecting the model performance. We discuss the potential problem and existing solutions' limitations as follows:

A. URL CLEANING

URL is a combination of symbols, numbers, and characters. It is also a cluster of multiple parts such as TLD, SLD, subdomain, and path. Therefore, designing a real-time phishing detection application requires cleaning the data. It is essential to consider which part of the URL will be used to train and test the model because using the whole URL or some part of the URL might improve or kill the model performance. For example, two websites can have the same hostname with a different subdomain: https://www.example.com/ is different from https://example.com/. Therefore, assuming that these two URLs are the same might result in vulnerability to phishing attacks. Also, a URL path might cause a model to perform poorly due to the randomness in the path, even on benign webpages. Therefore, cleaning the URL is an open research question worth studying.

B. ZERO-DAY ATTACKS

There is a need to design a model to detect new URL types. The attackers do not use the same pattern to design a fake URL every time. They usually design a URL for a one-time attack and then remove it. They work to deceive users into thinking that the URL is genuine so that they can design a new URL pattern in each attack. Therefore, designing a model that works well only on the patterns trained shows a bad result with real-time phishing attacks. Designing a model to detect new data has been proposed, such as [54] and [51]. However, these models are trained on an imbalanced dataset toward a benign URL. As a result, training the model in an imbalanced dataset makes the model unaware of the patterns of phishing URLs. Therefore, it must be trained on a large dataset to make the model learn the pattern from these URLs.

C. TIME-CONSUMING

The time-consuming problem in the DL model affects the user time and the type of machine that runs the experiment. Designing a massive model with multiple layers needs high-performance computing to train it. Most proposed models are trained using two models [10] or only one sequence model, such as LSTM or GRU [7]. These methods are time-consuming for training large data. As a result, some papers propose using CNN because it can train on a large dataset with less time. However, using CNN alone makes the model unable to detect the URL's relationship pattern. We explore in Section II some phishing attacks URL, e.g., attackers use the same SLD as a well-known URL but with a different TLD. Therefore, extracting the relationship pattern among URLs is important to classify the URL as a whole.

The time-consuming is a NLP problem. It has been shown to have a promising result with the standard English language. However, designing a lightweight model to detect a URL with combined features is a research direction worth studying.

D. TINY URL

The problem of tiny URL phishing attacks is relatively new. There is a lack of research that solves this problem. To solve this problem, some researchers add new features to some models, such as page content [10], email content, etc., to allow the models to learn in different aspects. It can solve the problem but it is time-consuming to train the data. This type of problem needs the dataset to be unbiased. Since the model uses features such as page content, the feature extraction steps need to learn the pattern that differentiates between phishing and benign websites. If the dataset is biased, the model will have a high value of FPR. The previous section explores a tool that directs the model to the original URL and conducts the classification part as soon as the user clicks on the tiny URL [53]. This method can solve the problem of getting the original URL from the tiny URL. However, it might result in the attacker stealing information from the user as soon as the tools open the URL by downloading and installing malicious software. Therefore, this problem still exists, and exploring it is required to solve the potential vulnerability of opening insecure URLs.

E. DETECT DIFFERENT TYPES OF ATTACK

An attacker can design multiple attacks; each attack impacts the system differently. Therefore, the model's ability to detect attacks can make the system use protection based on the attack type. For example, in [29], the authors proposed using keywords to detect different attacks. Their methods solve the issue of classifying the URL based on the attack type using specific keywords. However, it does not detect a phishing URL out of these words. Therefore, designing a model to detect phishing attacks without depending on sets of keywords is worth exploring.

F. WORD TOKENIZATION

Word tokenization is an essential method in NLP. It is crucial to extract the meaningful word from the sentence. This problem has been studied for many years in NLP. It aims to split the sentence into words and use these words in the word embedding layer. However, URLs contain a joint word such as (aboutus), with using a simple method in URL results in many unknown words. Therefore, using a powerful pre-trained model such as BERT results in low performance due to the high number of [UNK] values. Therefore, designing a model that can extract a meaning word from joint words is essential to improve the model performance in real-time data.

VI. CONCLUSION AND FUTURE WORK

In recent years, DL has become essential for solving cybersecurity problems such as phishing attacks. It gained attention due to its ability to extract features from the input data automated instead of manually. This survey studies state-of-art DL models to detect phishing attacks. Its importance lies in analyzing each DL model on every level, from input data to the model output. The importance of data preprocessing is at the same level as the DL model. The data preprocessing affect the model performance in any task, especially once the model implements an application to detect real-time data. For example, the model must be able to classify any input data even if it was not part of the model's dataset. Therefore, in this paper, we pay more attention to data preprocessing and highlight its weaknesses and strengths. Then, we analyze each DL model's design and highlight its strengths and weaknesses.

The data preprocessing has three steps: cleaning, tokenization, and embedding. In the cleaning step, each part of input data, such as URL or HTML, has meaning, including special symbols. Thus, cleaning the input data results in losing critical features. In the tokenization step, the input data is mostly joint words such as abebooks, ieeexplore, etc. Hence, using character-level tokenization is suited more to handle such input data. Finally, in the embedding step, we believe that URL and HTML embedding need more improvement to be as good as prebuilt text embedding, such as the transformers model. Since the existing embedding models are only trained in a small dataset, they perform well with similar data, and the performance drops rapidly with real-time detection. Therefore, building a DL embedding model and training on a large URL and HTML dataset are vital to enhance its performance.

Due to the size of DL models, they are time-consuming and resource-intensive. However, building a real-time phishing detection application requires efficient performance to be usable by end users. Therefore, building a DL model using a combination of CNN to reduce the model size and LSTM to learn the input long-time dependency is more suited for such an application.

As we discussed in Section V, there is potentially future work that needs to be explored. First, we plan to perform many experiments to explore more model limitations with real-time data to detect zero-day attacks. Then, we plan to design a model to solve the experiment's limitation. We also plan to explore designing a lightweight model and running it on small computers, such as the Internet of Things (IoT) devices, to evaluate and enhance the model's performance. We also plan to design a model trained on URLs, page content, and JavaScript. Furthermore, we also plan to solve the vulnerability that causes by opening an insecure URL, which might result in downloading and installing malware on the user's device. Finally, we plan to leverage a powerful model such as transformers to design a model that learns and extracts depth information from URL, HTML, and JavaScript input.

REFERENCES

[1] Y. Zhang, Y. Xiao, K. Ghaboosi, J. Zhang, and H. Deng, "A survey of cyber crimes," *Secur. Commun. Netw.*, vol. 5, no. 4, pp. 422–437, 2012.
[2] APWG Developers. (2021). *Phishing Activity Trends Report*. [Online]. Available: https://apwg.org/trendsreports/

[3] M. Lei, Y. Xiao, S. V. Vrbsky, and C.-C. Li, "Virtual password using random linear functions for on-line services, ATM machines, and pervasive computing," *Comput. Commun.*, vol. 31, no. 18, pp. 4367–4375, Dec. 2008.

[4] P. Burda, L. Allodi, and N. Zannone, "Don't forget the human: A crowd-sourced approach to automate response and containment against spear phishing attacks," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS PW)*, Sep. 2020, pp. 471–476.

[5] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, "PhishNet: Predictive blacklisting to detect phishing attacks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–5.

[6] W. Zhang, Y.-X. Ding, Y. Tang, and B. Zhao, "Malicious web page detection based on on-line learning algorithm," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 4, Jul. 2011, pp. 1914–1919.

[7] A. C. Bahnsen, E. C. Bohorquez, S. Villegas, J. Vargas, and F. A. González, "Classifying phishing URLs using recurrent neural networks," in *Proc. APWG Symp. Electron. Crime Res. (eCrime)*, 2017, pp. 1–8.

[8] B. Cui, S. He, X. Yao, and P. Shi, "Malicious URL detection with feature extraction based on machine learning," *Int. J. High Perform. Comput. Netw.*, vol. 12, no. 2, pp. 166–178, 2018.

[9] Y. Fang, C. Zhang, C. Huang, L. Liu, and Y. Yang, "Phishing email detection using improved RCNN model with multilevel vectors and attention mechanism," *IEEE Access*, vol. 7, pp. 56329–56340, 2019.

[10] J. Feng, L. Zou, O. Ye, and J. Han, "Web2Vec: Phishing webpage detection method based on multidimensional features driven by deep learning," *IEEE Access*, vol. 8, pp. 221214–221224, 2020.

[11] H. Cheng, J. Liu, T. Xu, B. Ren, J. Mao, and W. Zhang, "Machine learning based low-rate DDoS attack detection for SDN enabled IoT networks," *Int. J. Sens. Netw.*, vol. 34, no. 1, pp. 56–69, 2020.

[12] S. Christin, É. Hervet, and N. Lecomte, "Applications for deep learning in ecology," *Methods Ecol. Evol.*, vol. 10, no. 10, pp. 1632–1644, Oct. 2019.

[13] A. Aggarwal, A. Rajadesingan, and P. Kumaraguru, "PhishAri: Automatic realtime phishing detection on Twitter," in *Proc. eCrime Res. Summit*, Oct. 2012, pp. 1–12.

[14] H. Ma, Y. Zuo, and T. Li, "Vessel navigation behavior analysis and multiple-trajectory prediction model based on AIS data," *J. Adv. Transp.*, vol. 2022, pp. 1–10, Jan. 2022.

[15] J. Fang, B. Li, and M. Gao, "Collaborative filtering recommendation algorithm based on deep neural network fusion," *Int. J. Sens. Netw.*, vol. 34, no. 2, pp. 71–80, 2020.

[16] E. S. Gualberto, R. T. De Sousa, T. P. De Brito Vieira, J. P. C. L. Da Costa, and C. G. Duque, "The answer is in the text: Multi-stage methods for phishing detection based on feature engineering," *IEEE Access*, vol. 8, pp. 223529–223547, 2020.

[17] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 841–851, Jan. 2019.

[18] E. Zhu, Y. Chen, C. Ye, X. Li, and F. Liu, "OFS-NN: An effective phishing websites detection model based on optimal feature selection and neural network," *IEEE Access*, vol. 7, pp. 73271–73284, 2019.

[19] S. Salloum, T. Gaber, S. Vadera, and K. Shaalan, "Phishing email detection using natural language processing techniques: A literature survey," *Proc. Comput. Sci.*, vol. 189, pp. 19–28, Jan. 2021.

[20] M. Korkmaz, O. K. Sahingoz, and B. Diri, "Feature selections for the classification of webpages to detect phishing attacks: A survey," in *Proc. Int. Congr. Hum.-Comput. Interact., Optim. Robotic Appl. (HORA)*, Jun. 2020, pp. 1–9.

[21] C. Singh and Meenu, "Phishing website detection based on machine learning: A survey," in *Proc. 6th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Coimbatore, India, 2020, pp. 398–404. [Online]. Available: https://ieeexplore.ieee.org/document/9074400/authors#authors, doi: 10.1109/ICACCS48705.2020.9074400.

[22] L. Tang and Q. H. Mahmoud, "A survey of machine learning-based solutions for phishing website detection," *Mach. Learn. Knowl. Extraction*, vol. 3, no. 3, pp. 672–694, Aug. 2021.

[23] A. Basit, M. Zafar, X. Liu, A. R. Javed, Z. Jalil, and K. Kifayat, "A comprehensive survey of AI-enabled phishing attacks detection techniques," *Telecommun. Syst.*, vol. 76, pp. 139–154, Oct. 2020.

[24] M. Vijayalakshmi, S. M. Shalinie, M. H. Yang, and M. R. Meenakshi, "Web phishing detection techniques: A survey on the state-of-the-art, taxonomy and future directions," *IET Netw.*, vol. 9, no. 5, pp. 235–246, Sep. 2020.

[25] N. Q. Do, A. Selamat, O. Krejcar, E. Herrera-Viedma, and H. Fujita, "Deep learning for phishing detection: Taxonomy, current challenges and future directions," *IEEE Access*, vol. 10, pp. 36429–36463, 2022.

[26] A. Odeh, I. Keshta, and E. Abdelfattah, "Machine LearningTechniquesfor detection of website phishing: A review for promises and challenges," in *Proc. IEEE 11th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2021, pp. 0813–0818.

[27] S. Salloum, T. Gaber, S. Vadera, and K. Shaalan, "A systematic literature review on phishing email detection using natural language processing techniques," *IEEE Access*, vol. 10, pp. 65703–65727, 2022.

[28] T. Mahjabin, Y. Xiao, T. Li, and C. L. P. Chen, "Load distributed and benign-bot mitigation methods for IoT DNS flood attacks," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 986–1000, Feb. 2020.

[29] W. Yang, W. Zuo, and B. Cui, "Detecting malicious URLs via a keyword-based convolutional gated-recurrent-unit neural network," *IEEE Access*, vol. 7, pp. 29891–29900, 2019.

[30] M. Somesha, A. R. Pais, R. S. Rao, and V. S. Rathour, "Efficient deep learning techniques for the detection of phishing websites," *Sādhanā*, vol. 45, no. 1, pp. 1–18, Dec. 2020.

[31] A. Aljofey, Q. Jiang, Q. Qu, M. Huang, and J.-P. Niyigena, "An effective phishing detection model based on character level convolutional neural network from URL," *Electronics*, vol. 9, no. 9, p. 1514, Sep. 2020.

[32] Google Developers. (2020). *Google Site*. [Online]. Available: https://sites.google.com/

[33] Phishtank Developer. (2020). *Phishtank Dataset*. [Online]. Available: https://phishtank.org/developerinfo.phpl

[34] Alexa Developer. (2020). *Alexa Dataset*. [Online]. Available: https://www.alexa.com/topsites

[35] JoeWein Developer. (2020). *JoeWein Dataset*. [Online]. Available: https://joewein.net/bl-log/bl-log.html

[36] Common Crawl Developer. (2020). *Common Crawl Dataset*. [Online]. Available: https://commoncrawl.org/2021/10/september-2021-crawl-archive-now-available/

[37] OpenPhish Developer. (2020). *OpenPhish Dataset*. [Online]. Available: https://openphish.com

[38] A. Draghetti. (2020). *Phishing Army: The blocklist to Filter Phishing!* [Online]. Available: https://www.phishing.army/

[39] S. Ghodke. (Jan. 2018). *Alexa Top 1 Million Sites*. [Online]. Available: https://www.kaggle.com/datasets/cheedcheed/top1m

[40] Accessed: Jul. 10, 2021. [Online]. Available: https://archive.ics.uci.edu/ml//index.php

[41] G. Vrbančič, I. Fister, and V. Podgorelec, "Datasets for phishing websites detection," *Data Brief*, vol. 33, Dec. 2020, Art. no. 106438. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340920313202

[42] G. Sood. (Feb. 2021). *Parsed DMOZ Data*. [Online]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi%3A10.7910%2FDVN%2FOMV93V

[43] *Webscope | Yahoo Labs*. Accessed: Aug. 10, 2021. [Online]. Available: https://webscope.sandbox.yahoo.com/

[44] *Yandex.xml*. [Online]. Available: https://yandex.com/dev/xml/

[45] Accessed: Aug. 10, 2021. [Online]. Available: https://www.medien.ifi.lmu.de/team/max.maurer/files/phishload/

[46] D. Nayak and H. Perros, "Automated real-time anomaly detection of temperature sensors through machine-learning," *Int. J. Sens. Netw.*, vol. 34, no. 3, pp. 137–152, 2020.

[47] M. Guo, C. Guo, C. Zhang, D. Zhang, and Z. Gao, "Fusion of ship perceptual information for electronic navigational chart and radar images based on deep learning," *J. Navigat.*, vol. 73, no. 1, pp. 192–211, Jan. 2020.

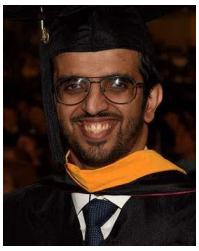
[48] N. Wu, X. Wang, B. Lin, and K. Zhang, "A CNN-based end-to-end learning framework toward intelligent communication systems," *IEEE Access*, vol. 7, pp. 110197–110204, 2019.

[49] J. Atiga, M. Hamdi, R. Ejbali, and M. Zaied, "Recurrent neural network NARX for distributed fault detection in wireless sensor networks," *Int. J. Sens. Netw.*, vol. 37, no. 2, pp. 100–111, 2021.

[50] F. Tajaddodianfar, J. W. Stokes, and A. Gururajan, "Texception: A character/word-level deep learning model for phishing URL detection," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 2857–2861.

[51] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, "URLNet: Learning a URL representation with deep learning for malicious URL detection," 2018, *arXiv:1802.03162*.

[52] J. Yuan, G. Chen, S. Tian, and X. Pei, "Malicious URL detection based on a parallel neural joint model," *IEEE Access*, vol. 9, pp. 9464–9472, 2021.
[53] M. Sameen, K. Han, and S. O. Hwang, "PhishHaven—An efficient real-time AI phishing URLs detection system," *IEEE Access*, vol. 8, pp. 83425–83443, 2020.
[54] Y. Huang, Q. Yang, J. Qin, and W. Wen, "Phishing URL detection via CNN and attention-based hierarchical RNN," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 112–119.
[55] S.-J. Bu and S.-B. Cho, "Deep character-level anomaly detection based on a convolutional autoencoder for zero-day phishing URL detection," *Electronics*, vol. 10, no. 12, p. 1492, Jun. 2021.
[56] U. Kamath, J. Liu, and J. Whitaker, *Deep Learning for NLP and Speech Recognition*, vol. 84. Cham, Switzerland: Springer, 2019.
[57] N. Cao, W. Huo, T. Lin, and G. Wu, "Application of convolutional neural networks and image processing algorithms based on traffic video in vehicle taillight detection," *Int. J. Sens. Netw.*, vol. 35, no. 3, pp. 181–192, 2021.
[58] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "FastText.Zip: Compressing text classification models," 2016, *arXiv:1612.03651*.
[59] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[60] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
[61] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 3. Washington, DC, USA: IEEE Computer Society, Oct. 2003, pp. 1470–1477.
[62] Y. Zhu, Y. Zuo, and T. Li, "Modeling of ship fuel consumption based on multisource and heterogeneous data: Case study of passenger ship," *J. Mar. Sci. Eng.*, vol. 9, no. 3, p. 273, Mar. 2021.
[63] A. Géron, *Hands-On Machine Learning With Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, Mar. 2017, pp. 355–357.
[64] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, 1982. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554
[65] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.
[66] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[67] J. Wang, L.-C. Yu, K. R. Lai, and X. Zhang, "Tree-structured regional CNN-LSTM model for dimensional sentiment analysis," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 28, pp. 581–591, 2020.
[68] A. Yenter and A. Verma, "Deep CNN-LSTM with combined kernels from multiple branches for IMDb review sentiment analysis," in *Proc. IEEE 8th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2017, pp. 540–546.
[69] F. Wei and T. Nguyen, "A lightweight deep neural model for SMS spam detection," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, Oct. 2020, pp. 1–6.
[70] H. Yang, Q. Liu, S. Zhou, and Y. Luo, "A spam filtering method based on multi-modal fusion," *Appl. Sci.*, vol. 9, no. 6, p. 1152, Mar. 2019.
[71] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.
[72] L. Yan, R. H. Dodier, M. Mozer, and R. H. Wolniewicz, "Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, 2003, pp. 848–855.
[73] H. H. Addeen, Y. Xiao, J. Li, and M. Guizani, "A survey of cyber-physical attacks and detection methods in smart water distribution systems," *IEEE Access*, vol. 9, pp. 99905–99921, 2021.
[74] J. Ya, T. Liu, P. Zhang, J. Shi, L. Guo, and Z. Gu, "NeuralAS: Deep word-based spoofed URLs detection against strong similar samples," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–7.
[75] J. Song, R. Paul, J. Yun, H. Kim, and Y. Choi, "CNN-based anomaly detection for packet payloads of industrial control system," *Int. J. Sens. Netw.*, vol. 36, no. 1, pp. 36–49, 2021.



SULTAN ASIRI received the B.S. degree in computer information systems from King Khalid University, Saudi Arabia, and the master's degree in computer information science from Gannon University, Erie, PA, USA. He is currently pursuing the Ph.D. degree with the Department of Computer Science, The University of Alabama, Tuscaloosa. His research interests include social engineering attacks, applied NLP, and deep learning.

YANG XIAO (Fellow, IEEE) received the B.S. and M.S. degrees in computational mathematics from Jilin University, Changchun, China, in 1989 and 1991, respectively, and the M.S. and Ph.D. degrees in computer science and engineering from Wright State University, Dayton, OH, USA, in 2000 and 2001, respectively.

He is currently a Full Professor with the Department of Computer Science, The University of Alabama, Tuscaloosa, AL, USA. He had directed 20 doctoral dissertations and supervised 19 M.S. theses/projects. He has published over 300 Science Citation Index (SCI)-indexed journal articles (including over 60 IEEE/ACM TRANSACTIONS) and 300 Engineering Index (EI)-indexed refereed conference papers and book chapters related to these research areas. His current research interests include cyber–physical systems, the Internet of Things, security, wireless networks, smart grid, and telemedicine. He was a Voting Member of the IEEE 802.11 Working Group, from 2001 to 2004, involving the IEEE 802.11 (Wi-Fi) standardization work. He is a fellow IET and AAIA. He received the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING Excellent Editor Award 2022. He has served as a Guest Editor over 30 times for different international journals, including the *IEEE Network*, in 2007, *Mobile Networks and Applications* (MONET) (ACM/Springer), in 2008, IEEE WIRELESS COMMUNICATIONS, in 2006 and 2021, textitIEEE Communications Standards Magazine, in 2021, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, in 2021, IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, in 2021, and IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC), in 2022. He also serves as the Editor-in-Chief of *Cyber-Physical Systems* journal, *International Journal of Sensor Networks* (IJSNet), and *International Journal of Security and Networks* (IJSN). He has been serving as an Editorial Board Member or an Associate Editor for 20 international journals, including the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, from 2007 to 2014, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, from 2007 to 2009, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, from 2014 to 2015, IEEE TRANSACTIONS ON CYBERNETICS, since 2020, and IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING (TNSE), since 2022.

SALEH ALZAHRANI received the bachelor's degree in information systems from King Khalid University, Abha, Saudi Arabia, and the master's degree from St. Mary's University, San Antonio, TX, USA, in 2018. He is currently pursuing the Ph.D. degree with The University of Alabama. His research interests include computer security, cyber security, and malware analysis and detection.

SHUHUI LI (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Southwest Jiaotong University, Chengdu, China, and the Ph.D. degree in electrical engineering from Texas Tech University, in 1999. From 1988 to 1995, he was at the School of Electrical Engineering, Southwest Jiaotong University, where his fields of research interest include modeling and simulation of large dynamic systems, dynamic process simulation of electrified railways, power electronics, power systems, and power system harmonics. From 1995 to 1999, he was engaged in research on wind power, artificial neural networks, and applications of massive parallel processing. He joined Texas A&M University-Kingsville as an Assistant Professor, in 1999, and then as an Associate Professor, in 2003. He worked with the Oak Ridge National Laboratory for simulation system development on supercomputers, in 2004 and 2006. He joined The University of Alabama as an Associate Professor, in 2006. His current research interests include renewable energy systems, power electronics, power systems, electric machines and drives, FACTS, intelligent control, microgrids, and distributed generation.

TIESHAN LI (Senior Member, IEEE) received the B.S. degree in ocean fisheries engineering from the Ocean University of China, Qingdao, China, in 1992, and the Ph.D. degree in vehicle operation engineering from Dalian Maritime University (DMU), Dalian, China, in 2005. From 2007 to 2015, he has held various Postdoctoral/Senior Research Associate (SRA)/ Visiting Scholar positions at the Shanghai Jiao Tong University, City University of Hong Kong, and University of Macau. He has been a Full Professor with DMU since 2011, and he has been served as a Chair Professor since 2021. He is currently a Tenured Professor with the University of Electronic Science and Technology of China. His research interests include intelligent learning and control for nonlinear systems, multi-agent systems, and their applications to unmanned vehicles.