Article

PhishTransformer: A Novel Approach to Detect Phishing Attacks Using URL Collection and Transformer

Sultan Asiri [1,2], Yang Xiao [1,*] and Tieshan Li [3]

1 Department of Computer Science, The University of Alabama, Tuscaloosa, AL 3487-02903, USA; saasiri@crimson.ua.edu
2 Department of Computer Science, King Khalid University, Abha 62529, Saudi Arabia
3 School of Automation Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China; litieshan073@uestc.edu.cn
* Correspondence: yangxiao@ieee.org

Abstract: Phishing attacks are a major threat to online security, resulting in millions of dollars in losses. These attacks constantly evolve, forcing the cyber security community to improve detection systems. One major problem with current detection systems is that they cannot detect new phishing attacks, such as Browser in the Browser (BiTB) and malvertising attacks. These attacks hide behind legitimate Uniform Resource Locators (URLs) and can evade detection systems that only analyze a web page URL without exploring the page content. To address this problem, we propose PhishTransformer, a deep-learning model that can detect phishing attacks by analyzing URLs and page content. We propose only using URLs embedded within a webpage, such as hyperlinks and JFrames, to train PhishTransformer. This helps reduce the number of features that need to be extracted from the page content, which makes training the model more efficient. PhishTransformer combines convolutional neural networks and transformer encoders to extract features from website URLs and page content. These features are then used to train a classifier that can distinguish between phishing attacks and legitimate websites. We tested PhishTransformer on a dataset of 10,000 URLs. Our results show that PhishTransformer can achieve an F1-score of 99%, precision of 99%, and recall of 99%. This result suggests that PhishTransformer is a promising new approach to phishing detection.

Keywords: phishing attacks; real-time; detection systems; deep learning; tiny uniform resource locators; Browser in the Browser (BiTB)

check for updates

Citation: Asiri, S.; Xiao, Y.; Li, T. PhishTransformer: A Novel Approach to Detect Phishing Attacks Using URL Collection and Transformer. *Electronics* **2024**, *13*, 30. https://doi.org/10.3390/electronics13010030

Academic Editor: Aryya Gangopadhyay

Received: 7 November 2023
Revised: 28 November 2023
Accepted: 7 December 2023
Published: 20 December 2023

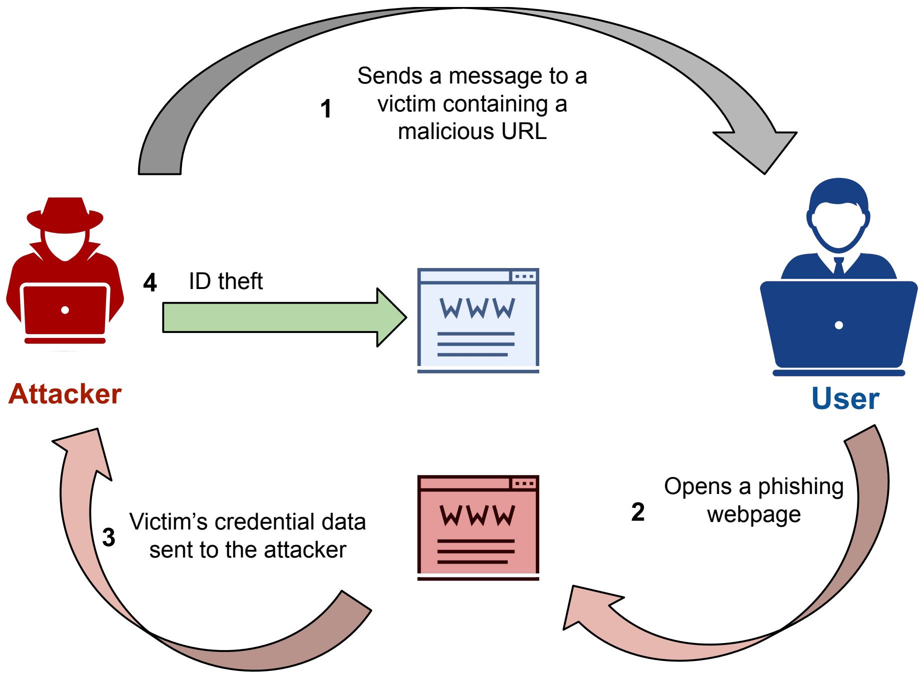
1. Introduction

Phishing attacks are a type of cybercrime and social engineering attack where malicious actors deceive users into entering sensitive information or downloading malicious software [1]. It started in 1996, when a group of hackers attacked America Online (AOL) accounts [2,3]. These attacks aim to steal victims' identities and gain access to their sensitive information, such as login credentials and financial details. Phishing attacks often use human nature, focusing on emotions like fear, urgency, or curiosity to manipulate victims into sharing their confidential information [4,5]. Attackers typically use social media platforms, emails, and other forms of online communication to reach their victims [1,6,7]. To get victims' trust, they may masquerade as trustworthy sources, such as banks, government agencies, or well-known companies. Phishing posts often contain threatening messages encouraging victims to act immediately, such as opening a malicious link [8]. These links can lead to fake websites that look legitimate to deceive victims into entering their credential information, which can cause consequences ranging from financial losses to identity theft [1,8].

Phishing attacks have increasingly affected individuals and companies in the last two years [9]. According to IBM's 2020 Cost of a Data Breach Report [10], 20% of companies experiencing a malicious data breach fall victim to it due to lost or stolen credentials, and

17% face breaches through direct phishing attacks. The increase in phishing attacks shows a significant threat to cybersecurity, particularly in email and social media platforms. As a result, companies are working to build efficient detection systems to detect phishing attempts. Most companies rely on human knowledge and awareness to detect phishing attacks. However, phishing attacks have become increasingly sophisticated and difficult to detect, and human error remains unavoidable. Therefore, it is crucial to implement additional security layers to protect companies from the consequences of human mistakes.

Figure 1 illustrates a basic example of phishing attacks. In phishing attacks, attackers gain access to users by sending messages with malicious URLs via social media platforms or email applications. These URLs direct users to a phishing webpage. To deceive users, attackers design the phishing webpage to look like a legitimate site. Users then enter their login credentials, which are returned to the attackers. This allows the attackers to access the legitimate site using the stolen information. Phishing attacks are now more robust and hard to detect because the attacker uses various techniques, such as Browsers in the Browser attacks (BiTB) attacks, to bypass the current detection systems.



1
Sends a message to a victim containing a malicious URL
4
ID theft
WWW
Attacker
User
2
Opens a phishing webpage
3
Victim's credential data sent to the attacker
WWW

Figure 1. An example of a phishing attack.

Phishing attacks are becoming increasingly intelligent and challenging. Existing approaches, such as heuristic rules, blacklists, and URL-based models, are often ineffective against modern phishing techniques. Attackers constantly evolve their methods, employing techniques such as BiTB [11], Watering Hole Attacks [12], and Clickjacking attacks [13] that bypass current phishing detection tools.

In the case of BiTB, a malicious website runs inside a legitimate website by exploiting vulnerabilities in the browser [11]. In this attack, an attacker uses JavaScript code to create a hidden IFrame or window that runs a malicious website inside the legitimate website. Unfortunately, users are typically unaware of this attack and may accidentally interact with the malicious website by clicking on a link or entering their personal information. Watering hole attacks compromise a legitimate webpage frequently visited by its intended targets and include hyperlinks to a fake webpage designed to steal personal information [12]. Once the target visits a legitimate webpage, they may accidentally click on the hyperlink and be directed to the fake webpage. Clickjacking uses hidden hyperlinks or transparent overlays to trick users into clicking on a button or hyperlink they did not intend to click on [13]. This technique is often used with website spoofing to make the fake webpage appear more convincing. In these attacks, attackers use web development tools such as

HTML and JavaScript to hide attacks inside a legitimate webpage. Therefore, analyzing webpage content is essential to protect users from being victims of phishing webpages.

Current phishing detection systems rely on analyzing either the main URLs of webpages [14–16] or the entire content of webpages, including HTML and JavaScript [17]. URL-based detection systems are effective at identifying known phishing websites. However, they are vulnerable to bypass by attackers who create new malicious URLs or use other techniques such as redirects, BiTB attacks, or Clickjacking attacks. On the other hand, whole-page analysis systems are more computationally demanding and challenging to develop and maintain, as they need to handle a wider range of features in website design and code.

This research assumes that even though a webpage's main URL looks benign, it does not necessarily indicate that it is safe from phishing attacks. Users who access a legitimate webpage tend to trust all URLs and frames within it, making them vulnerable to BiTB, Watering Hole Attacks, and Clickjacking attacks. To overcome this problem, a deep learning (DL) detection model that can analyze all URLs, including hyperlinks within HTML and JavaScript code, is more effective in detecting phishing attacks than existing detection methods. As explained, the Watering Hole attacks, BiTB attacks, and Clickjacking attacks have one thing in common: embedding malicious code in legitimate content. Therefore, by analyzing all URLs in the webpage content, we analyze the malicious.

This research investigates how to develop a DL model that accepts all URLs in HTML and JavaScript, including hyperlinks and IFrame URLs as input, to detect phishing attacks accurately. This problem requires collecting a dataset, designing a feature representation, and designing a DL model that uses these feature representations to classify a suspected webpage as phishing or benign.

To address these limitations, we propose a novel approach that analyzes all URLs within a webpage's content, including hyperlinks and IFrame URLs, to detect phishing attacks effectively. This comprehensive approach enables our deep learning (DL) model to identify malicious URLs that may be embedded within legitimate content, effectively mitigating Watering Hole, Clickjacking, and BiTB attacks. Focusing on these informative features can reduce the model complexity while maintaining or improving the detection accuracy. Based on that, our contribution is as follows:

- We propose a URL collection transformer, which represents the first phishing detection system that utilizes both the main URL of a webpage and all URLs attached to it, to the best of our knowledge.
- We propose a data pipeline that collects all URL embeddings within the suspected webpage to pass to a DL model for classification.
- We comprehensively analyze the model with the real-time dataset.

Our novel approach considers the webpage URL and all URLs embedded within the webpage, including iframes, hyperlink URLs, and other embedded links. This approach offers several advantages:

- Focusing on embedded URLs rather than the entire HTML and JavaScript code significantly reduces the input dimension, leading to a more efficient and less complex model.
- Embedded URLs serve as rich sources of information that can reveal potential phishing attempts, including BiTB attacks, Clickjacking schemes, and other malicious techniques concealed within seemingly legitimate URLs.
- By incorporating embedded URLs into the analysis, the model can capture a more comprehensive picture of the webpage's potential malicious intent, improving detection accuracy.

The rest of this paper is organized as follows. Section 2 reviews related works, Section 3 describes our proposed data pipeline and model, Section 4 provides a comprehensive analysis of our data pipeline and model performance on new data, and Section 5 concludes the paper with a discussion and future work.

2. Related Work

The cybersecurity community recently proposed various methods for detecting phishing attacks. These methods are categorized into two main groups: URL-based and hybrid approaches.

URL-based uses machine learning (ML) model to extract and analyze semantic and syntactic features of webpages' main URLs [14–16]. The authors in [14] propose the URLNet model, a deep learning approach for classifying URLs as benign or phishing. It utilizes both character-level and word-level features to analyze URLs. The model first tokenizes the URL at the character level, extracting unique characters and symbols while replacing infrequent ones with an unknown value, and then converts each character into a k-dimensional vector. The model combines two embedding matrices for word-level tokenization: M1 for word embedding and M2 for character embedding. Unique words are extracted from the URL and assigned unique IDs, forming the M1 matrix. Each word is then tokenized into characters, generating the M2 matrix. M2 is added to M1 to create a new word embedding matrix, M3. M3 is then passed through Convolutional Neural Network (CNN) layers to extract character and word-level features.

The authors in [16] propose a parallel neural joint model for classifying URLs as benign or phishing. The model takes a URL as input and processes it through image and semantic data preprocessing pipelines. The image pipeline converts the URL into a 2D matrix, while the semantic pipeline splits the URL into word-level and character-level representations. The model then uses two neural networks, IndRNN [18] and CapsNet [19], to extract features from the URL. The output of these networks is then passed to an attention mechanism, which helps the model focus on the most important parts of the input. The final output of the model is a classification of the URL as benign or phishing.

The authors in [15] propose a Transformer-based model for phishing detection. The model takes a URL as input and encodes it using a character-level embedding. The embedding is then passed to a Transformer encoder, which extracts features from the URL. The output of the encoder is then passed to a classification layer, which outputs a prediction of whether the URL is benign or phishing.

Unfortunately, attackers bypass URL-based detection methods by injecting legitimate websites with phishing URLs, such as Clickjacking and BiTB attacks.

Hybrid approaches combine HTML and Javascript content to detect phishing attacks. The authors in [17] propose Web2vec, a phishing webpage detection method based on multidimensional features driven by deep learning. It uses URLs, HTML content, and document object model (DOM) structures as input for a CNN that can classify web pages as phishing or benign. Web2vec takes URL, page content, and DOM structure information for feature extraction. The model applies one-hot encoding to represent webpages as vectors and uses CNN for local feature extraction and bidirectional long-term memory (BiLSTM) for sequence modeling. An attention mechanism is incorporated to highlight important features. As a result, their method achieved a high performance. However, web2vec [17] has three limitations: it is time-consuming, requires considerable resources to perform in a real-time application, and an attacker can bypass this detection system by using Client-Side Cloaking.

The authors in [20] propose a phishing detection method based on client-side features such as extracting features from HTML and URLs. They divide HTML features into four categories: login form, hyperlink, CSS, and web identity. First, they check whether the suspected webpage contains a login form. If so, they check whether the domain name in its action link is similar to the domain name in the address bar. Second, their method extracts six features from webpage hyperlinks, such as the number of hyperlinks, no hyperlinks, if the webpage has hyperlinks that contain a domain name different from the primary domain name, etc. Third, their methods check whether the webpage connects to an external CSS file. Fourth, their methods check whether the suspected website copyright keywords match the main domain name. This method is based on heuristic rules that attackers can bypass by changing the features of the phishing website. Another limitation is that extracting many features requires time-consuming and human efforts.

Combining the webpage URL and its content to detect whether it is phishing or benign is an important approach for enhancing phishing detection accuracy. However, the effectiveness of this approach depends on the specific features extracted from the webpage content. Utilizing the entire HTML and JavaScript code can significantly increase the input dimension, leading to a more complex, computationally expensive model exposed to overfitting.

Based on the literature, we summarise their limitation as follows:

- Deciding whether the webpage is secure based only on its URL will likely result in an incomplete assessment of its security. It may lead to false positives or false negatives in identifying phishing attacks. Therefore, other factors, such as the webpage's content, must be considered.
- Learning a model using webpage content such as JavaScript and HTML can be time-consuming due to the complexity of the model required and the size of the input feature.

3. Methodology

Most of the recent phishing detection models are either simple models by analyzing only webpages' main URLs to detect phishing attacks like in [14,16,21] or more complicated like using more features such as HTML, DOM, JavaScript content like in [17,22]. Using only the URL feature requires a simple model; however, it might result in a higher rate of false positives and negatives due to the limited scope of information considered for phishing detection. On the other hand, using a more complicated model that incorporates features like HTML, DOM, and JavaScript content can enhance phishing detection accuracy; however, it might be more time-consuming during the training phase and when making real-time predictions. Therefore, this paper combines the advantages of URL-based and hybrid models. Our methodology involves training the model using URL features extracted from various sources within the webpage, such as URLs embedded within HTML hyperlinks, JavaScript (JFrame), and IFrame elements. As a result, we combine the simplicity of the URL feature and extract critical insights from JavaScript and HTML content. The architecture of Phishtransformer, as shown in Figure 2, contains three main parts: data collection, data preprocessing, and DL model.

3.1. Data Collection

First, we download phishing blacklist datasets from Phisharmy [23], PhishTank [24], and a benign dataset from Alexa's top 100,000 websites [25]. We then use a web parsing tool to extract URLs from the HTML and JavaScript content of the webpages. We scrape all URLs in hyperlinks, IFrames, JFrames, and URL regular expressions. To collect the URLs within each website, we must ensure that each URL is correct, online, and error-free. To do this, we open each URL in the dataset and perform the following steps:

- We check whether the URL is online. If the URL is not online, we skip it and move on to the next URL.
- We obtain the current website URL. This is the URL of the webpage that we are currently processing.
- We scrape every URL inside the webpage, whether in JavaScript or HTML codes. This includes URLs in hyperlinks, IFrames, and URL regular expressions.

As a result, we create a list X for each webpage that contains all URLs inside the webpage.

$$X_1 = [u_1\, u_2\, u_3\, \dots \dots\, u_m], \tag{1}$$

where m is a fixed size representing the number of URLs within the website, and u is a URL embedded in the website X_1, either in HTML or JavaScript content. Since we have a list of benign and phishing websites, we generate a list D of all websites, $D = [X_1, X_2, X_3, \dots, Xn]$, where n is the size of the dataset.

As a result, our dataset is as follows:

$$D = \begin{bmatrix} u_{11} & u_{12} & u_{13}, \dots & u_{1m} \\ u_{21} & u_{22} & u_{23}, \dots & u_{2m} \\ u_{31} & u_{32} & u_{33}, \dots & u_{3m} \\ . & & & \\ . & & & \\ u_{n1} & u_{n2} & u_{n3}, \dots & u_{nm} \end{bmatrix}, \quad (2)$$

where u_{11} is the first URL in the first webpage in the dataset, and u_{12} corresponds to the second URL in the same webpage, and so forth.

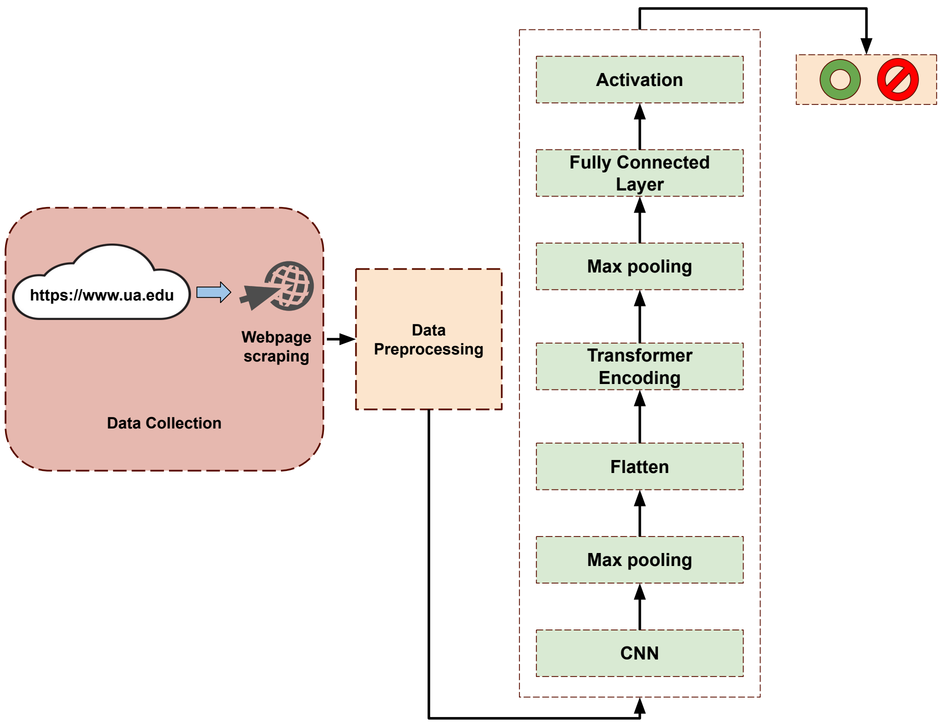


Figure 2. Proposed Method.

3.2. Data Preprocessing

As shown in Figure 3, our data processing contains four steps: data scraping, cleaning, tokenization, padding, and concatenation. Given our focus on the URL dataset in this paper, each feature is important, making us retain all features without cleaning.

In the context of data tokenization, we use a character-level approach for tokenizing URLs. Character-level tokenization is a text preprocessing technique used in NLP. Its main advantage is that it allows the model to handle unknown words more effectively since each character is already an available token. First, we break down each URL into individual characters. Then, we convert each character into a numeric value. Since we work with URLs, there are 92 possible characters, such as letters, digits, and special characters. Two special indexes are also introduced: $< PAD >$ for padding and $< UNK >$ for unknown characters. To enable the model to distinguish individual URLs within a webpage, we introduce two additional indexes: $< STA >$ to mark the beginning of a URL and $< SEP >$ to indicate its end. This approach provides the model with a better comprehension of each URL's features instead of simply concatenating all URL characters embedded within a webpage. Therefore, for each URL inside the webpage, u_{nm} is tokenized into characters c, and each character is converted into its corresponding integer. This operation results in the structured representation of each URL denoted as $u_n = [c_{1n}, c_{1n}, c_{1n}, \dots, c_{dn}]$, where d is a fixed size representing the length of the URL.

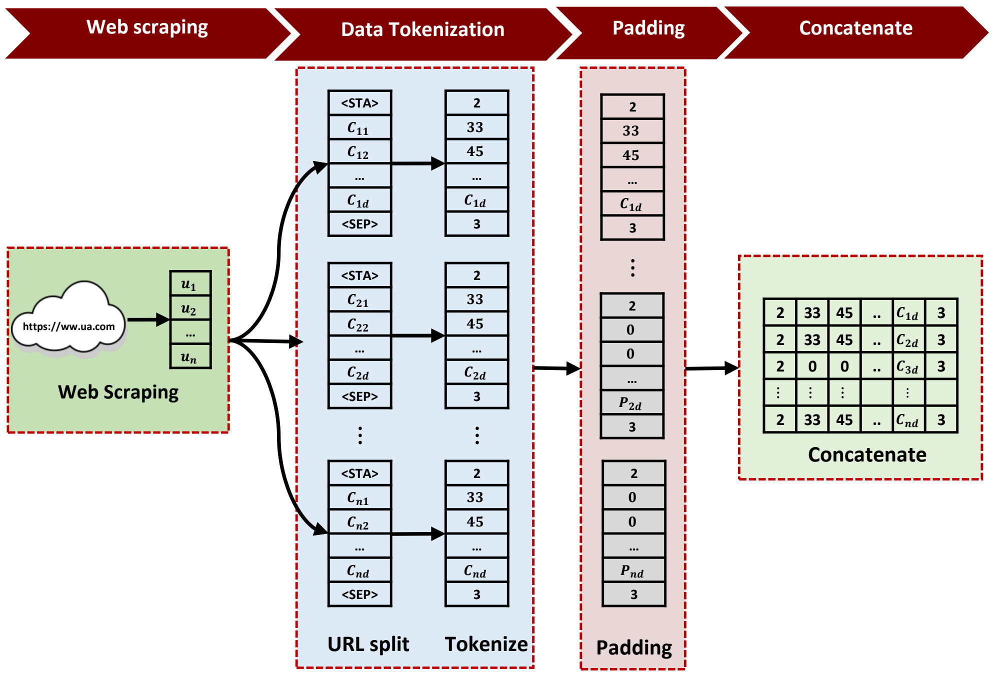


Figure 3. Data Preprocessing Steps.

In the padding steps, we start by determining the length of URLs and the quantity of URLs within a webpage. To establish the optimal fixed length for URLs, we examine all URLs associated with all webpages in the dataset. This identified length becomes the fixed length for URLs in our dataset. If a URL surpasses this fixed length, we trim it down; otherwise, we extend it by appending $< PAD >$ until it reaches the fixed length. The same approach is applied to the number of URLs per webpage. Webpages exceeding the fixed number of URLs are truncated, while shorter lists are padded with $[< STA >, < PAD >, \ldots, P_{d-1}, < SEP >]$ tokens, where d represents the fixed length of URLs.

Finally, we concatenate this structured URL representation and pass it to a single-layer neural network embedding layer. The embedding layer maps the structured representation of the URL into a vector representation. This vector representation is passed to our DL model.

3.3. Deep Learning Model

As shown in Section 3, our input data is a nested sequence, where each row represents a webpage, and each webpage contains a sequence of URLs. As a result of that, we have high-dimension input data. Therefore, we combine two models, CNN [26] and transformer encoder. The CNN layer can learn the local features of the URLs, such as the characters and symbols they contain. The transformer encoder can then learn the long-range dependencies between the URLs, such as the order in which they appear on the webpage and the relationships between them. This combination of CNN and transformer encoder allows us to learn the input data's local and global features.

As shown in Figure 2, the vector representation is transferred to a DL model for training. First, it passes to a convolutional layer to extract the local feature using Equation (3)

$$h_i = f(D_i \cdot W + b), \tag{3}$$

where h_i is the output of the convolutional layer for input i, D_i is the input data for input i, W is the weight matrix of the convolutional layer, b is the bias, and f is the nonlinear activation function of the convolutional layer. Equation (3) essentially performs a dot product between the input data and the weight matrix. The dot product calculates the similarity between the input data and the filter at each location. The output of the dot product is then passed through the nonlinear activation function, which helps to introduce non-linearity into the model.

The local features extracted by the convolutional layer are then passed to the max pooling layer to down-sample the input feature.

$$mp = max(h_i), \tag{4}$$

where mp is the max pooling layer output, and h_i is the output value of the convolution operation. Max-pooling works by taking a small window of input data and selecting the maximum value in the window. The output mp is then passed to the flattening layer to convert a multi-dimensional array into a one-dimensional array before passing it to the transformer encoder layer.

In this paper, we use a transformer encoder [27] due to its ability to capture dependencies between distant symbols in URLs by using multi-head attention mechanisms that allow the model to attend to different parts of the input sequence. For example, URLs can contain information in their position, such as the presence of certain subdomains or directory paths. Furthermore, the self-attention mechanism captures positional information of terms in the input sequence without recurrent connections or convolutional filters. The transformer encoder contains a positional encoder, self-attention layer, normalization, residual connection, and feedforward neural network.

First, the embedding output for each character is passed to the positional encoding layer to add information about the position of each character. In this step, we add the character's location as a signed number starting from zero to the $(n-1)$th dimension, where n is the fixed number representing the URL size. Then, each character-embedding vector is passed to the self-attention layer. Each character-embedding vector is multiplied by three randomly initialized weights, Query (Q), Key (K), and Value (V), before passing it to the self-attention layer. Then, using Equation (5), the self-attention layer calculates a score for each character in the input sequence to the current character:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{5}$$

where d_k is a dimension vector of K, then the output of self-attention (Z) passes along with the input embedding (X) to the normalization layer $LayerNorm(X+Z)$. The normalization layer helps stabilize the training of the transformer model.

Second, the output of the transformer block passes to a global pooling layer. The global pooling layer takes the maximum value from each dimension of the input vector. This helps to reduce the dimensionality of the input vector.

Finally, the output is sent to a fully connected layer for classification. The fully connected layer is a traditional neural network layer with a sigmoid activation function. The sigmoid activation function helps to produce a probability distribution over the output classes. The output of the fully connected layer is then used to predict the class, either phishing or benign.

4. Experiments

To evaluate the performance of our model, we followed a five-step process: data collection, data analysis, model training, model evaluation, and comparison with state-of-the-art models.

4.1. Data Collection

We started by gathering the necessary datasets for our research. To obtain the phishing URLs, we downloaded a phishing blacklist dataset from two reputable sources, namely PhishArmy [23] and PhishTank [24]. These sources provide a comprehensive list of known phishing websites. For the benign dataset, we needed a collection of legitimate URLs. We manually selected 50,000 URLs from Alexa's top 100,000 websites dataset [25]. By manually selecting URLs, we ensured they were benign and represented many legitimate websites. Initially, our dataset included 100,000 URLs, with an equal distribution of phishing and benign URLs. We followed the process outlined in Section 3.3 to extract the necessary features. We carefully scraped all URLs by analyzing each webpage's HTML and JavaScript content in our dataset. However, during the scraping process, we encountered many offline URLs. As a result, we only managed to obtain 50,000 URLs, 50% of which are phishing and the remaining 50% benign. We divided the dataset into three subsets to evaluate our deep learning model. The training set was allocated 70% of the data for the model to learn from. The testing set is 20% of the data, allowing us to assess the model's performance on unseen URLs. Lastly, the remaining 10% was designated as the validation set, which was used to fine-tune the model and ensure its robustness.

4.2. Data Analysis

We comprehensively analyzed our dataset to determine the best approach for applying our DL model. By answering two key questions, we gained valuable insights that helped us to develop a robust DL model.

- What is the maximum, minimum, and average number of characters in the URL in the dataset?
- What is the maximum, minimum, and average number of URLs within a webpage in the dataset?

The first question focused on understanding the characteristics of URLs within our dataset. Specifically, we sought to determine the maximum, minimum, and average number of characters in a URL. The second question investigated the maximum, minimum, and average number of URLs within a webpage. The results of our analysis, as presented in Figure 4, display the differences between the benign and phishing datasets. Notably, the benign dataset displayed more characters in the URL and a higher number of URLs within each webpage than the phishing dataset. For instance, the maximum number of characters in a URL within the benign dataset was 5623, while the maximum number of characters in a URL within the phishing dataset was 2852.

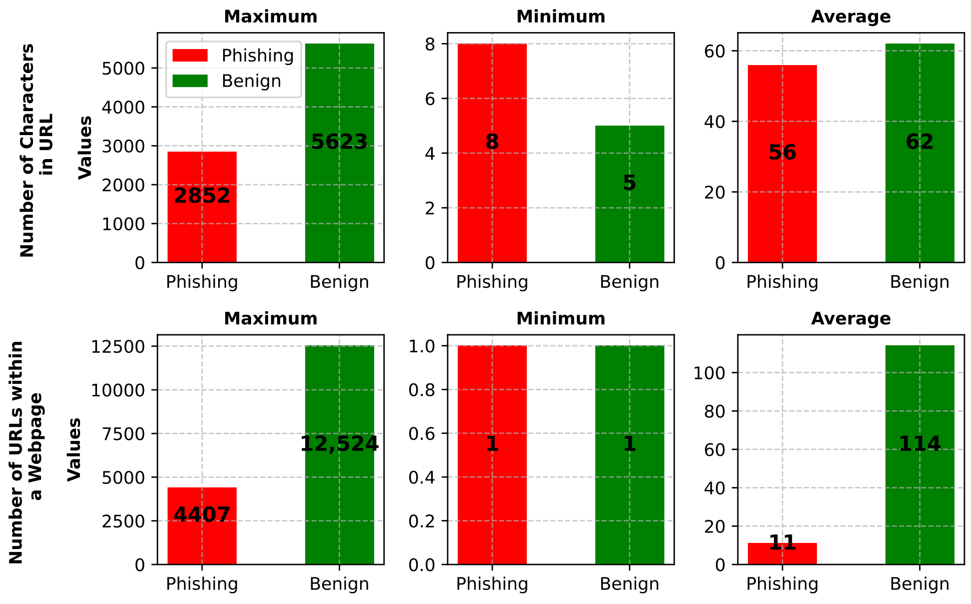


Figure 4. Data Analysis (top graphs are URLs' length; down graphs are number of URLs within a webpage).

Similarly, the average number of URLs within a webpage was 114 for the benign dataset and 11 for the phishing dataset. These findings suggest that the benign dataset is more complex than the phishing dataset, with more characters in URLs and more URLs within webpages. Given these differences, it was essential to be cautious when selecting the fixed lengths of both URLs and the number of URLs within webpages to train our model effectively. For the URL length, we chose a fixed number of 100 characters (n). This decision was driven by the observation that the average length of URLs in benign and phishing datasets was around 62 characters. However, it is worth noting that numerous URLs presented lengths significantly exceeding 62 characters. By choosing a fixed length of 100 characters, we ensure that all URLs in the dataset have the same length.

Regarding the number of URLs within webpages (m), we select 100 as a fixed number. This choice is derived from the average number of URLs for benign and phishing datasets within a webpage. Fixing the number at 100 helped prevent the bias towards either the benign or phishing dataset and accommodated the reality of webpages with larger numbers of URLs beyond our chosen limit. To maintain consistency in the dataset, URLs shorter than 100 characters are padded, while URLs longer than 100 characters are cropped. This approach guarantees that all URLs within the dataset share the same fixed length, enabling the effective training of the DL model. By establishing fixed lengths for URL characters and the number of URLs within webpages, we ensured a standardized dataset that could be utilized effectively in training our model.

4.3. Model Evaluation

We evaluated our model using four metrics: Accuracy, true positive rate (TPR), precision, and F1 score. Accuracy measures the overall ratio of correctly classified benign and phishing URLs. TPR (also known as recall) is the ratio of correctly predicted benign URLs to the total number of benign URLs. Precision is the ratio of correctly predicted benign URLs to the total number of URLs predicted as benign. The F1 score is a harmonic mean of TPR and precision, considering false positives and false negatives.

The equations for TPR, $precision$, and $F1score$ are as follows:

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

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

$$Precision = \frac{Y_{TP}}{Y_{TP} + Y_{FP}}, \tag{8}$$

$$F1 = 2 \times \frac{Precision \times recall}{Precision + recall}, \tag{9}$$

where TP True positive is a correctly predicted benign URL, TN True positive is a correctly predicted phishing URL, FP False positive is an incorrectly predicted benign URL, and FN False negative is an incorrectly predicted phishing URL. We chose these metrics because they are all commonly used to evaluate the performance of phishing detection models. TPR is a good measure of the model's ability to identify all benign URLs. At the same time, precision is a good measure of the model's ability to avoid incorrectly classifying phishing URLs as benign. The F1 score is a more comprehensive metric considering TPR and precision.

4.4. Hyper-Parameters Settings

We divided the hyperparameter settings into model input and model parameters. The parameters for the model input are the URL length, number of features, and number of URLs within the webpage. As shown in Table 1, we chose 100 for the URL length and the number of URLs within the webpage. We chose 95 for the number of features. We explain why these fixed lengths are chosen in Section 4.2.

The model parameters are the number of hidden attention layers, kernel size, filter size, pool size, learning rate (LR), batch size, epoch, and dropout. For other values, we used the default parameters provided by the Keras library.

Table 1. Parameters Tuning.

Type	Parameter	Value
Model Input	URL length	100
	Number of features (Special symbol)	95
	Number of URLs within the webpage	100
Model	Number of hidden attention layers	10
	Kernel size	128
	Filter size	64
	Pool size	3
	Learning Rate (LR)	0.0001
	Batch	16
	Epoch	100
	Dropout	0.05
	Loss function	Binary Crossentropy
	Optimizer	Adam

We used 10 for the number of hidden attention layers, 128 for the CNN kernel size, 64 for the CNN filter size, and 3 for the CNN pool size. We used a decaying learning rate to reduce the LR if the model validation loss stopped improving for ten epochs. As shown in Figure 5, a learning rate between 0.001 and 0.0001 yields the best model performance based on the evaluated metrics. We used the early stopping technique to stop the training once the model stopped improving or become worse for ten epochs. Therefore, the optimal hyperparameters were LR = 0.0001, epoch = 100, batch size = 16, and dropout = 0.05.

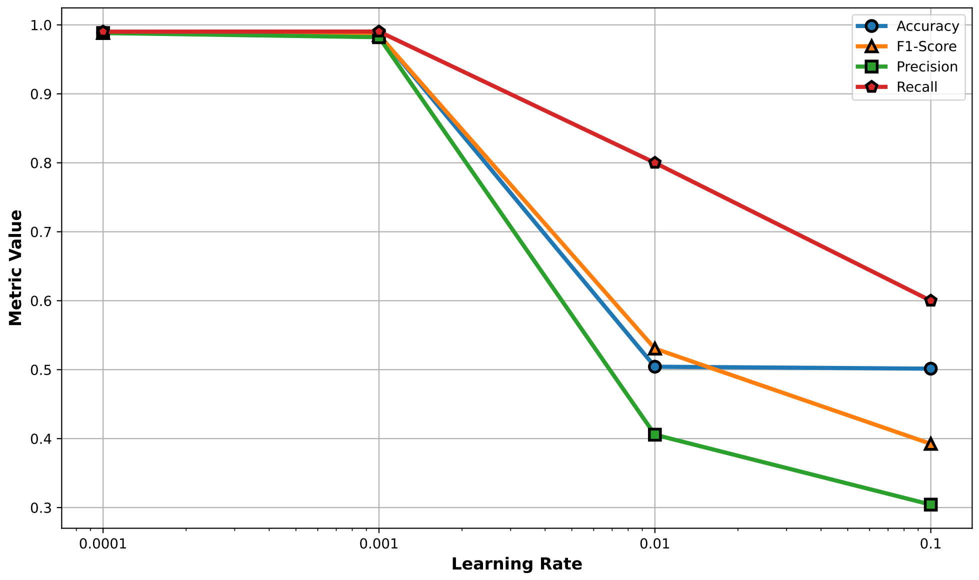


Figure 5. Comparison of Different Metrics at Various Learning Rates.

4.5. Model Training

After data collection and preprocessing, we initiated the model training phase. Since our dataset is limited, we used k-fold cross-validation for model training. This technique involves dividing the dataset into multiple folds and iteratively training the model on each fold while using the remaining folds for validation. Therefore, it ensures that every data point contributes to training and validation, maximizing the utilization of our dataset. Hence, this approach helps to prevent overfitting, as the model encounters a diverse range of data during training, reducing its reliance on specific data points. As shown in Figure 6, the training set is randomly split into five folds, with four folds reserved for training and one fold reserved for validation. This process is repeated, with each fold taking turns as the validation fold. Consequently, each training fold contains 31,462 samples, while each validation fold contains 7866 samples. After training the model, we used the test set with 10,000 for evaluations. As shown in the Figure 7, there is a strong performance across all metrics. The model achieves an average accuracy of 0.989, a precision of 0.988, a recall of 0.990, and an F1-score of 0.988, suggesting that the model is reliable and accurate in classifying the data. In addition to that, the consistency across different folds indicates that the model generalizes well to new, unseen data. This validates the effectiveness of using k-fold cross-validation in our training phase, as it helps to mitigate the risks of overfitting while making the most out of our limited dataset. Therefore, we are confident that our model is robust and can be effectively deployed for further analysis or production use.

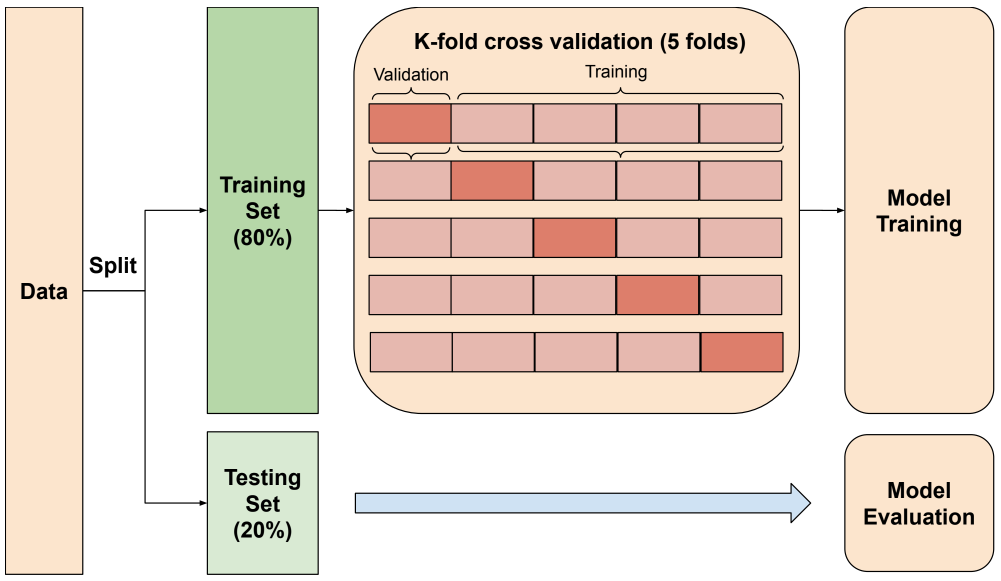


Figure 6. Training the model using K-fold cross-validation.

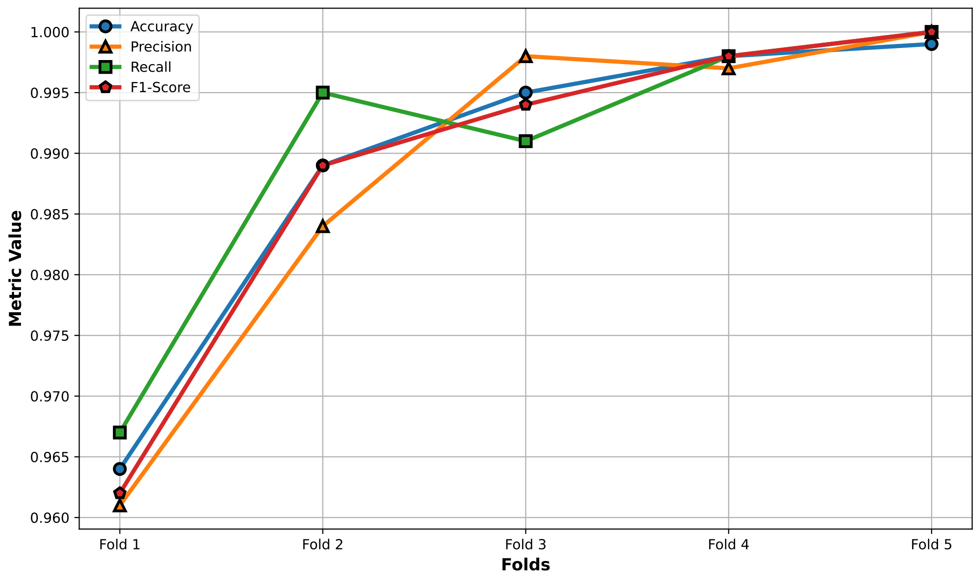


Figure 7. Five-folds cross-validation.

4.6. Comparison of Phishing Detection Models

We conducted two experiments to test the effectiveness of our model. In the first experiment, we evaluated the performance of our data pipeline with different feature extraction models, including Support Vector Machine (SVM) [28], CNN, BiLSTM, and our CNN+Transformer encoder. As shown in Table 2, we evaluate each model's performance using the Precision, Recall, and F1 metrics. The SVM shows the worst performance with a precision of 0.84, a recall of 0.85, and an F1 score of 0.84. The CNN model has a precision of 0.98, a recall of 0.97, and an F1 score of 0.98. The BiLSTM model has a precision of 0.95, recall of 0.96, and F1 score of 0.80. Our model shows the highest performance with a precision of 0.99, recall of 0.99, and F1 score of 0.99.

Table 2. Compression on the performance of our data pipeline with different feature extraction models.

Model	Precision	Recall	F1 Score
SVM	0.84	0.85	0.84
CNN	0.98	0.97	0.98
BiLSTM	0.95	0.96	0.95
Our model	0.99	0.99	0.99

As shown in Table 2, our model performs well compared to the other three models. Our CNN+Transformer encoder achieved the best performance, with precision, recall, and F1 scores of 0.99, 0.99, and 0.99, respectively. This high performance is because our model can extract local and global features. The extraction of local features is important as our input data consists of a nested array containing 100 URLs, with each URL further divided into 100 tokens. Additionally, the long-range dependency feature plays a vital role, since each URL within a webpage contains meaningful information that can significantly impact the classification outcome. The transformer encoder's ability to model long-range dependencies provides a comprehensive understanding of URL patterns, enabling accurate analysis of all input components. While the CNN model also exhibited good performance, with precision, recall, and F1 scores of 0.98, 0.97, and 0.98, respectively, it fell short of our CNN+Transformer encoder due to its inability to capture the dependency relationships among characters within URLs.

In the second experiment, we compared the performance of our model to two state-of-the-art phishing detection models, URLNET [14] and MPURNN [29]. Both models are trained using URL-based features but differ in feature extraction and classification methods. URLNET [14] extracts a URL representation from three levels of features: character-level, word-level, and character-word level. Each feature is then passed to a set of CNN layers, followed by a fully connected layer for classification. MPURNN [29], on the other hand, only extracts character-level features. These features are then converted into unique binary vectors using one-hot encoding. The vector representation is then passed to a BiLSTM for training and classification.

This experiment compares our approach with the standard approach of URL-based state-of-the-art models. To achieve this, we train these models using our training and validation sets and test them with our test set. As shown in Table 3, our model performs significantly better than the other two. Our model achieved a precision, recall, and F1 score of 0.99, 0.99, and 0.99, respectively, which are 16%, 17%, and 15% higher than URLNET and 29%, 31%, and 27% higher than MPURNN. This is because our model extracts more informative features from webpages by using all URLs inside the webpage.

Table 3. Compression on state-of-the-art of phishing detection model.

Model	Precision	Recall	F1 Score
URLNET [14]	0.93	0.85	0.87
LSTM + one hot encoding [29]	0.80	0.82	0.80
Our model	0.99	0.99	0.99

5. Conclusions

In this paper, we propose a new phishing detection model, PhishTransformer. PhishTransformer consists of two main components: a new input pipeline that extracts all URLs within a webpage and a combination of the CNN and Transformer encoder for feature extraction.

The new input pipeline allows our model to consider each webpage's context, essential for phishing detection. It extracts all URLs within a suspected webpage, such as hyperlinks, Jframes, and other embedded objects. These URLs are then passed to our deep learning model, a combination of CNN and Transformer encoder.

The CNN extracts local features from each URL, such as the URL length, the presence of certain characters, and the presence of certain keywords. The Transformer encoder extracts long-range dependency features, such as the order of the words in the URL and the relationships between the words. Combining these two features allows our model to achieve state-of-the-art results on the phishing detection task.

We evaluated our model on a dataset of 10,000 new URLs. Our results show that PhishTransformer outperforms state-of-the-art models. This result suggests that PhishTransformer is a promising new approach to phishing detection. Additionally, our results show that using all URLs embedded within a webpage can help increase phishing detection performance.

One major limitation of our model is the size of the input pipeline. It can reduce the speed of our model, as some webpages may contain more than 1000 URLs, each of which is 100 tokens long. It can take some time for our model to produce a result.

In our future work, we plan to address this limitation by designing a model that reduces the size of the input pipeline by using data compression or other techniques. We also plan to evaluate our model on a larger dataset of URLs to assess its performance further.

Author Contributions: Methodology, S.A. and Y.X.; Writing—original draft, S.A.; Writing—review & editing, Y.X. and T.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Our data was collected from three different sources. For the benign dataset, we used the Alexa Top 1 Million Sites dataset available on Kaggle: https://www.kaggle.com/datasets/cheedcheed/top1m. Phishing data was collected from PhishTank https://phishtank.org/phish_search.php?page=2&active=y&valid=y&Search=Search and PhishArmy https://phishing.army/. The details of the datasets are explained in Sections 3.1 and 4.2.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Asiri, S.; Xiao, Y.; Alzahrani, S.; Li, S.; Li, T. A Survey of Intelligent Detection Designs of HTML URL Phishing Attacks. *IEEE Access* **2023**, *11*, 6421–6443. [CrossRef]
2. Rekouche, K. Early phishing. *arXiv* **2011**, arXiv:1106.4692.
3. Ollmann, G. The phishing guide understanding & preventing phishing attacks. *Ngs Softw. Insight Secur. Res.* **2004**, 1–72.
4. Gupta, B.B.; Tewari, A.; Jain, A.K.; Agrawal, D.P. Fighting against phishing attacks: State of the art and future challenges. *Neural Comput. Appl.* **2017**, *28*, 3629–3654. [CrossRef]
5. Xiao, Y.; Li, C.C.; Lei, M.; Vrbsky, S.V. Differentiated Virtual Passwords, Secret Little Functions, and Codebooks for Protecting Users from Password Theft. *IEEE Syst. J.* **2014**, *8*, 406–416. [CrossRef]
6. Sun, B.; Wu, K.; Xiao, Y.; Wang, R. Integration of mobility and intrusion detection for wireless Ad Hoc networks. *Int. J. Commun. Syst.* **2007**, *20*, 695–721. [CrossRef]
7. Lei, M.; Xiao, Y.; Vrbsky, S.V.; Li, C.C. Virtual Password Using Random Linear Functions for On-line Services, ATMs, and Pervasive Computing. *Comput. Commun. J.* **2008**, *31*, 4367–4375. [CrossRef]
8. Alabdan, R. Phishing attacks survey: Types, vectors, and technical approaches. *Future Int.* **2020**, *12*, 168. [CrossRef]
9. Shoaib, M.; Umar, M. Phishing detection model using feline finch optimisation-based LSTM classifier. *Int. J. Sens. Netw.* **2023**, *42*, 205–220. [CrossRef]
10. Security, I.; Institute, P. Cost of a Data Breach Report. 2020. Available online: https://www.ibm.com/security/data-breach-report/ (accessed on 13 August 2020).

11. mrd0x developers. Browser In The Browser (BITB) Attack. Available online: https://mrd0x.com/browser-in-the-browser-phishing-attack/ (accessed on 2 July 2022).
12. Wright, G.; Bacon, M. What is a Watering Hole Attack? Available online: https://www.techtarget.com/searchsecurity/definition/watering-hole-attack (accessed on 6 October 2022).
13. Balduzzi, M.; Egele, M.; Kirda, E.; Balzarotti, D.; Kruegel, C. A solution for the automated detection of clickjacking attacks. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, Beijing, China, 13–16 April 2010; pp. 135–144.
14. Le, H.; Pham, Q.; Sahoo, D.; Hoi, S.C. URLNet: Learning a URL representation with deep learning for malicious URL detection. *arXiv* **2018**, arXiv:1802.03162.
15. Xu, P. A transformer-based model to detect phishing URLs. *arXiv* **2021**, arXiv:2109.02138.
16. Yuan, J.; Chen, G.; Tian, S.; Pei, X. Malicious URL detection based on a parallel neural joint model. *IEEE Access* **2021**, *9*, 9464–9472. [CrossRef]
17. Feng, J.; Zou, L.; Ye, O.; Han, J. Web2Vec: Phishing Webpage Detection Method Based on Multidimensional Features Driven by Deep Learning. *IEEE Access* **2020**, *8*, 221214–221224. [CrossRef]
18. Li, S.; Li, W.; Cook, C.; Zhu, C.; Gao, Y. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 5457–5466.
19. Sabour, S.; Frosst, N.; Hinton, G.E. Dynamic routing between capsules. *Adv. Neural Inf. Process. Syst.* **2017**, 3859–3869.
20. Jain, A.K.; Gupta, B.B. Towards detection of phishing websites on client-side using machine learning based approach. *Telecommun. Syst.* **2018**, *68*, 687–700. [CrossRef]
21. Wu, C.Y.; Kuo, C.C.; Yang, C.S. A phishing detection system based on machine learning. In Proceedings of the 2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA), Tainan, Taiwan, 30 August–1 September 2019; pp. 28–32.
22. Zhu, E.; Chen, Y.; Ye, C.; Li, X.; Liu, F. OFS-NN: An effective phishing websites detection model based on optimal feature selection and neural network. *IEEE Access* **2019**, *7*, 73271–73284. [CrossRef]
23. Mao, J.; Tian, W.; Li, P.; Wei, T.; Liang, Z. Phishing-alarm: Robust and efficient phishing detection via page component similarity. *IEEE Access* **2017**, *5*, 17020–17030. [CrossRef]
24. Developer, P. Phishtank Dataset. Available online: https://phishtank.org/developerinfo.phpl (accessed on 5 June 2021).
25. Ghodke, S. Alexa Top 1 Million Sites. Available online: https://www.kaggle.com/datasets/cheedcheed/top1m (accessed on 29 April 2022).
26. Yang, J.; Deng, F.; Lv, S.; Wang, R.; Guo, Q.; Kou, Z.; Chen, S. Multi-applicable text classification based on deep neural network. *Int. J. Sens. Netw.* **2022**, *40*, 277–286. [CrossRef]
27. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, 6000–6010.
28. Kumar, S.; Chaube, M.; Nenavath, S.; Gupta, S.; Tetarave, S. Privacy preservation and security challenges: A new frontier multimodal machine learning research. *Int. J. Sens. Netw.* **2022**, *39*, 227–245. [CrossRef]
29. Bahnsen, A.C.; Bohorquez, E.C.; Villegas, S.; Vargas, J.; González, F.A. Classifying phishing URLs using recurrent neural networks. In Proceedings of the 2017 APWG Symposium on Electronic Crime Research (eCrime), Phoenix, AZ, USA, 25–27 April 2017; pp. 1–8. [CrossRef]