applied sciences

MDPI

Article

A Phishing-Attack-Detection Model Using Natural Language Processing and Deep Learning

Eduardo Benavides-Astudillo [1,2,*], Walter Fuertes [2], Sandra Sanchez-Gordon [1], Daniel Nuñez-Agurto [2] and Germán Rodríguez-Galán [2]

[1] Department of Informatics and Computer Science, Escuela Politécnica Nacional, Quito 170525, Ecuador
[2] Department of Computer Sciences, Universidad de las Fuerzas Armadas ESPE, Sangolquí 171103, Ecuador
* Correspondence: diego.benavides@epn.edu.ec

Abstract: Phishing is a type of cyber-attack that aims to deceive users, usually using fraudulent web pages that appear legitimate. Currently, one of the most-common ways to detect these phishing pages according to their content is by entering words non-sequentially into Deep Learning (DL) algorithms, i.e., regardless of the order in which they have entered the algorithms. However, this approach causes the intrinsic richness of the relationship between words to be lost. In the field of cyber-security, the innovation of this study is to propose a model that detects phishing attacks based on the text of suspicious web pages and not on URL addresses, using Natural Language Processing (NLP) and DL algorithms. We used the Keras Embedding Layer with Global Vectors for Word Representation (GloVe) to exploit the web page content's semantic and syntactic features. We first performed an analysis using NLP and Word Embedding, and then, these data were introduced into a DL algorithm. In addition, to assess which DL algorithm works best, we evaluated four alternative algorithms: Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), Gated Recurrent Unit (GRU), and Bidirectional GRU (BiGRU). As a result, it can be concluded that the proposed model is promising because the mean accuracy achieved by each of the four DL algorithms was at least 96.7%, while the best performer was BiGRU with 97.39%.

Keywords: phishing; deep learning; natural language processing; NLP; Keras embedding; GloVe; LSTM; BiLSTM; GRU; BiGRU

Citation: Benavides-Astudillo, E.; Fuertes, W.; Sanchez-Gordon, S.; Nuñez-Agurto, D.; Rodríguez-Galán, G. A Phishing-Attack-Detection Model Using Natural Language Processing and Deep Learning. *Appl. Sci.* **2023**, *13*, 5275. https://doi.org/10.3390/app13095275

Academic Editor: Luis Javier García Villalba

Received: 5 March 2023
Revised: 10 April 2023
Accepted: 10 April 2023
Published: 23 April 2023

1. Introduction

1.1. Theoretical Background

Social engineering is a form of cyber-attack where the attacker uses psychological manipulation, unsuspecting behavior [1], or naive personality traits [2] to deceive and defraud the victim. One of the ways to implement this type of attack is phishing through the use of a fraudulent web page, which is often a copy of a legitimate page [3]. In other words, phishing is a social engineering attack that aims to deceive users into committing fraud against individuals or organizations. According to [4], phishing is a crime employing social engineering and technical subterfuge to steal consumers' identity data and financial account credentials. Nevertheless, since [5], a single concept has been agreed upon: "Phishing is a scalable act of deception whereby impersonation is used to obtain information from a target".

The most-common method to detect that a web page is phishing is identifying if such a page is on a blacklist. The problem is that these blacklists only store the URLs of pages previously found to be phishing, i.e., this method does not help to detect new phishing pages [6]. Various solutions use Machine Learning (ML) to overcome this problem of phishing detection. In recent years, a branch of ML named Deep Learning (DL) has achieved better accuracy than traditional ML algorithms [7]. The DL models perform better than the ML models in terms of accuracy, but the ML models perform better than the DL models in terms of computation time [8].

On the other hand, NLP is a technique to represent the common language of humans [9]. In the DL works reviewed, which analyze the text in phishing pages, there are two possibilities: the sequential and non-sequential approaches. Generally, the text input to the DL algorithms is non-sequential, i.e., the order in which the words are entered does not matter, producing a deficiency in the semantic meaning of the input text [10,11]. The approach taken in this research is sequential, whereby the data sequence is encapsulated, semantic and syntactic meaning is preserved, and relationships between words are described using spatial distance [12]. There are several sequential methods, such as Word2Vec and FastText, but we decided to use Keras Embedding with GloVe. The advantage of Keras Embedding with GloVe over other embedding techniques is that it is a sequential representation approach that facilitates semantic and syntactic relationships between words. In addition, by using Keras Embedding with GloVe, we have the following advantages [13]:

- Better performance: Pre-trained word embeddings such as GloVe have been trained on massive amounts of text data, meaning they will likely have learned high-quality representations of words.
- Reduced computational cost: Using pre-trained embeddings such as GloVe can save the time and resources required to train its embeddings.
- Improved generalization: Pre-trained embeddings such as GloVe have been trained on diverse and large corpora, meaning they can capture semantic and syntactic properties of words common across different domains and tasks.
- Flexibility: Pre-trained embeddings such as GloVe are available in different dimensions (we used 100 dimensions between 50, 100, 200, and 300), meaning you can choose the dimensionality that best suits your task and data.

1.2. Motivation

This research proposes a model to help users detect whether a web page is phishing. The primary approach to detect new phishing pages is using traditional ML algorithms [14]. However, in ML, a branch of algorithms called DL is known to provide more accurate results as long as they are fed more data [15]. Therefore, this study evaluated four algorithms derived from the Recurrent Neural Network (RNN) [16], which are well suited for text analysis.

Currently, several solutions use DL algorithms to detect phishing pages. Nevertheless, most need to emphasize the words' order or meaning. One of the advantages of our proposal is using the Keras Embedding Layer [17], by which contextual information can be discovered within the sequence of words, thus taking advantage of the semantic and syntactic characteristics, i.e., the meaning of the text entered [18].

According to [19], among the main countermeasures against phishing is the use of ML. However, it also highlights that technicians should consider semantics-based techniques. Furthermore, Reference [20] reported in a systematic literature review that RNN-type algorithms are the most-widely used to solve phishing attack detection problems, followed by Convolutional Neural Networks (CNNs).

1.3. Contributions

In this study, we propose a model to detect phishing attacks using words extracted from the HTML code of web pages. These are then analyzed and passed through an embedding layer, taking into account their semantic and syntactic features. Finally, these data are pre-processed and entered into DL algorithms. Therefore, the main contributions of this study are the following:

- Extract only linguistic features from the HTML code of the text obtained from Web pages and perform word embedding processing with the analyzed text, taking into account its semantic and syntactic features.
- Determine the appropriate length of words or tokens to input to the DL algorithms to balance mean accuracy and training time.

- Obtain better performance detecting phishing attacks by implementing a model combining NLP and DL, using Keras Embedding Layer with GloVe.
- Conduct a comparative study with four DL algorithms to evaluate which offers better results. We used two unidirectional algorithms: LSTM and GRU, and their two bidirectional versions: BiLSTM and BiGRU.

The rest of this article is structured as follows: Section 2 examines the related work. Section 3 describes in detail the methodology used. Section 4 explains the experimental setup. Section 5 presents the results and their discussion, and finally, Section 6 presents the conclusions and future work.

2. Related Work

Before conducting a study on the primary articles, we first conducted a meta-review of systematic literature reviews (SLRs), literature reviews, and surveys from the last three years in the general field of phishing detection and deep learning to identify whether or not there are any identified studies focusing on the use of NLP for the analysis of the textual content of suspicious web pages.

The authors of the SLR [21] conducted an in-depth study on 100 articles to mitigate phishing email attacks using NLP mixed with traditional ML or DL algorithms. The study describes that a high percentage of 38% of the articles are oriented toward using the email body to detect attacks. It is striking that there is not a relevant percentage about detecting phishing attacks based on the content of web pages.

Survey [22] comprehensively reviews all aspects surrounding phishing attacks and their countermeasures, including DL techniques. It even references three solutions [23–25] that use text analysis of web pages to detect attacks. However, these solutions do not use NLP to detect phishing attacks.

Next, once the search for information was conducted in secondary studies where we could not find articles proposing a solution similar to ours, we decided to do a more in-depth search, this time only on primary articles. To find articles related to this topic, we used the methodology proposed by Barbara Kitchenham [26] in a summarized way to perform an SLR. To follow this methodology, we first designed the research question; next, we explored the scientific databases in which we entered our search string based on the research question, applied the inclusion and exclusion criteria, applied the quality criteria, and finally, extracted the results.

2.1. Research Question

Our main objective was to develop a model that allows us to detect phishing attacks with high accuracy based on web pages' text content without wasting the text's intrinsic richness [27]. To achieve this, we used NLP; next, this dataset feeds a DL algorithm that classifies whether the data entered are from a phishing or ham page (when a web page is benign or not phishing). Based on this, the research question is defined as follows:

Which primary studies give a solution to detect phishing attacks using NLP and DL algorithms?

2.2. Search the Relevant Documents

The search was performed in the following scientific databases:

- Elsevier
- Scopus
- Web of Science
- IEEE Xplore
- ACM Digital Library
- Springer

2.3. Search String

This review aimed to find all primary studies on phishing attack detection methods combining NLP and DL. Thus, the following string was entered into each scientific database:

Phishing and ("Natural Language Processing" or NLP) and ("Deep Learning" or LSTM or BiLSTM or GRU or BiGRU)

2.4. Inclusion and Exclusion Criteria

The years for which we performed our search were 2017 to 2023 because this is when there has been the most movement in the DL algorithms and NLP exploitation. We only included articles in English. Furthermore, we only included primary articles that offer a concrete solution for detecting phishing attacks.

2.5. Quality Evaluation of Research

For the quality criteria of the search, it was determined that the review would only be performed in primary works that apply the analysis of the text contained in web pages, discarding other methods, such as URL analysis, because they were not our object of study. In addition, as described above, the review was performed on reputable scientific databases and considered only articles in Computer Science or similar.

2.6. Results of the Review

After applying the search string, we identified 28 articles (ten from Web of Science, nine from Scopus, five from Elsevier, and four from IEEE Explore), eliminating the identical results between the scientific databases, resulting in twenty articles. From the analysis of the 20 articles, the two main vectors of phishing attacks that stood out were those carried out through phishing emails or web pages. However, this research concentrated on the text contained in the web pages; for this reason, 13 articles whose solutions were oriented toward phishing emails were not considered. Finally, once the quality, inclusion, and exclusion criteria were applied, the seven articles presented in Table 1 were selected as the inputs for the present study. As we can see, published research in this common area still needs to be available.

Table 1. Related research.

Article	DL	NLP	Web Page Text
[28]	Yes	Yes	No
[29]	Yes	Yes	No
[30]	Yes	Yes	No
[31]	Yes	Yes	No
[32]	Yes	Yes	No
[33]	Yes	No	Yes
[15]	Yes	No	No
Our proposed model	Yes	Yes	Yes

Although the articles [28–32] of Table 1 implement DL and NLP techniques, all of them focus their research on analyzing only the content of the URL address of the web pages instead of the content of the web pages.

The article of [28] is not oriented toward detect phishing on web pages and is not based on detecting phishing in the body of emails, but the approach to giving scores to the words is interesting. This score depends on two rankings obtained from these words, according to the ranking in which they appear in a phishing email, Phishing Rank(w), or in legitimate emails, Legitimate Rank(w). For example, in this study, the following 20 words obtained the highest score: account 21.45%, it is 15.00%, click 14.11%, mailbox 9.59%, Cornell 9.58%, link 9.37%, verify 8.83%, customer 8.63%, access 8.50%, reserved 8.03%, dear 7.85%, log 7.70%, accounts 7.61%, paypal 7.52%, complete 7.37%, service 7.15%, protecting 6.95%,

secure 6.94%, mail 6.70%, and clicking 6.63%. The hierarchical LSTM algorithm was used in this model.

The authors of [29] performed an algorithm using CNN self-attention. The proposed algorithm obtained an accuracy of 95.6%, even better than the accuracy obtained by the hybrid CNN-LSTM approach. In contrast to our proposal, this work did not analyze words, but only URLs.

In [30], the authors used three steps to detect whether a page is phishing. First, they extracted the lexical and host-based properties of a website. Second, they combined URL features, NLP, and host-based properties to train machine learning and DL models. NLP was only applied to the URL to detect if there was a similar word in that URL but not identical to a known domain. Furthermore, a single DL algorithm, the deep neural network, was used, resulting in an accuracy of 96.6%. The article did not perform an analysis of the content of the web pages. This study obtained an accuracy of 96.60%, although it did not indicate which DL algorithm was used.

The work of [31] is interesting because it included NLP and two DL algorithms (LSTM and CNN) to build a hybrid model to detect phishing attacks. Still, in contrast to our work, this one is not oriented toward analyzing the web pages' contents, but the URLs and images of the web pages.

An empirical study was performed in the model proposed by [15] only to compare and determine which algorithm is better at detecting phishing attacks, LSTM, CNN, or LSTM combined with CNN. It was determined that CNN was superior to the others in this experiment. On the other hand, the web pages' body was not analyzed, but the URL of each website was. In this study, NLP was not used for pre-processing.

In [32], the application of NLP in detecting web pages was studied. For this, the application of traditional ML and DL models was performed. The deep learning algorithms used were LSMT, GRU, and BiRNN. In this study, the analysis was performed only on URLs and not on the text of the web pages.

In article [33], the authors proposed a model combining long-term recurrent convolutional and graph convolutional networks algorithms to detect phishing attacks on URL and HTML content. However, they did not use NLP.

An experiment to detect phishing attacks using four DL algorithms like ours is [34]. The authors performed an intensive empirical study to determine which of the four algorithms performed best at detecting a web page by its URL. They set multiple parameter values before running each DL algorithm. This article did not use NLP nor analyze the web pages' textual content. It was concluded that no algorithm produced the best measures on all performance metrics.

3. Proposed Model Based on NLP and DL

This article proposes a model based on NLP and DL to perform a pre-analysis of the text entered into the DL algorithm. For this purpose, our model comprises four consecutive phases from the HTML code: word parsing, data pre-processing, feature representation, and feature extraction, as illustrated in Figure 1.

Figure 1. Phishing attack detection—overview of the proposed mode.

Thus, with our method, first, by word parsing, the input text is divided into words, and the relationships between these words are obtained; then, by pre-processing with NLP, two phases are used, data pre-processing and feature representation, by which the most-important words to be analyzed are obtained, in addition to the importance of the meaning of the order in which the words are entered. Finally, by DL, the essential features

of the input text are obtained automatically, and the algorithm is trained. The final goal of the model is to obtain greater precision in detecting phishing attacks.

An essential contribution of our model is that before these data are entered into the DL algorithms, it goes through a word embedding process, specifically Keras Embedding, and GloVe. As an additional contribution, four DL algorithms were evaluated to determine which best fit our model.

The phishing detection problem is a binary classification task because the algorithm can only result in two options: phishing or ham. Thus, a dataset with 10,373 rows and two columns was obtained after processing. Therefore, the first column contains clean text, i.e., text without characters that negatively affect the accuracy of the experiment. The second column indicates whether that row is phishing or not. For practical purposes, phishing = 1 and ham = 0. Each of the stages and sub-stages of the model is described in the following subsections.

3.1. Word Parsing

From the Phishload database [35], we obtained 10,373 records of HTML code from phishing and ham pages. However, if we enter all that text to be analyzed by our selected DL algorithms, the results will be less accurate. For example, let us examine the following set of words found in the vast majority of HTML code of phishing pages: "DOCTYPE HTML PUBLIC". The algorithms will detect that these features are an essential indicator to decide that a page is phishing when it is not. For this reason, in our model, we first cleaned the text using the Regular Expressions and four Natural Language Toolkit (NLTK) tools. In Figure 2, you can see all the steps followed in word parsing.

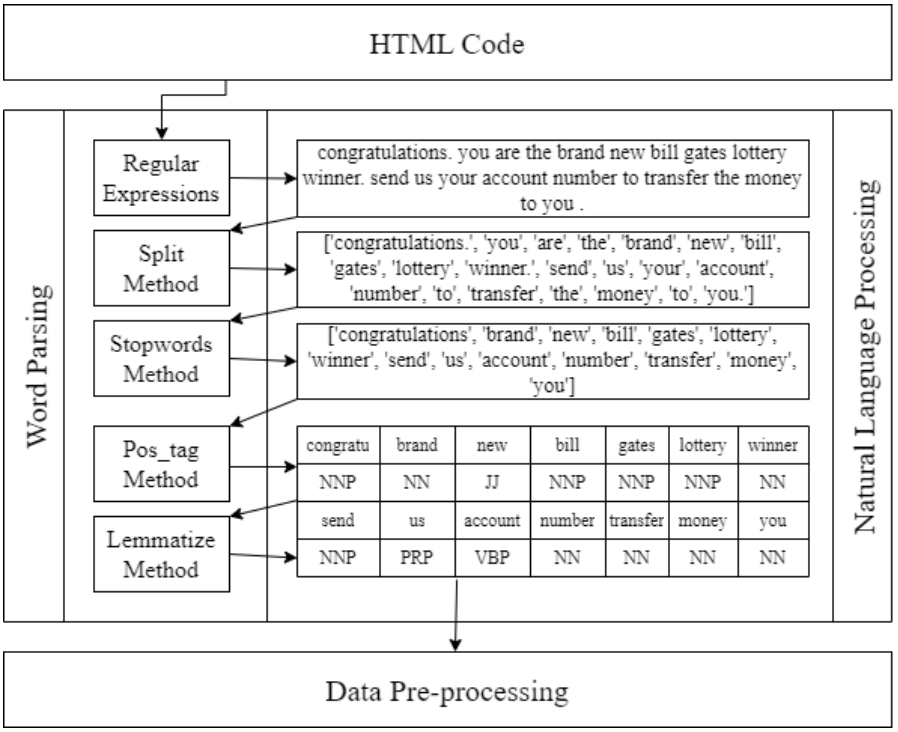


Figure 2. Word parsing sub-processes.

3.1.1. Regular Expressions

Initially, the text obtained from the web pages still contains objects such as strings, numbers, characters, and so on, which are unnecessary for the analysis. Therefore, we first transformed all letters to lowercase to continue with our analysis. Then, by regular expressions, we removed URL addresses, mentions with @ or #, HTML tags, digits, and all junk characters that need to be deleted.

3.1.2. Split Method

The split method is a string method that is used to split a string into a list of smaller sub-strings. The method takes a separator as an argument and divides the string based on the occurrence of that separator. If we execute the following code with a known phishing string:

text = "congratulations you are the brand new bill gates lottery winner send us your account number to transfer the money to you",
words = text.split()
print(words)

Then, the result will be as follows:

['congratulations', 'you', 'are', 'the', 'brand', 'new', 'bill', 'gates', 'lottery', 'winner', 'send', 'us', 'your', 'account', 'number', 'to', 'transfer', 'the', 'money', 'to', 'you']

3.1.3. Stopwords Method

Now, we cleaned the text of stopwords or empty words that do not have any meaning for our analysis because they can appear in a phishing text, as well as in a ham text, for example, the words: the, or, and, that, this, and of are deleted from the text.

Let us analyze the string obtained in the previous step with the following code:

stop_words = set(stopwords.words('english'))
words = ['congratulations', 'you', 'are', 'the', 'brand', 'new', 'bill', 'gates', 'lottery', 'winner', 'send', 'us', 'your', 'account', 'number', 'to', 'transfer', 'the', 'money', 'to', 'you']
filtered_words = [word for word in words if word.casefold() not in stop_words]
print(filtered_words)

When *stopwords Method* is executed on the example text, the following words are deleted:

'you', *'are'*, *'the'*, *'your'* and *'to'*

Hence, the set of words that stays are:

['congratulations', 'brand', 'new', 'bill', 'gates', 'lottery', 'winner', 'send', 'us', 'account', 'number', 'transfer', 'money', 'you']

3.1.4. Pos_Tag Method

The pos_tag method is used for Parts-Of-Speech (POS) tagging words in a text. POS tagging is the process of assigning a part of speech, such as a noun, verb, adjective, and so on, to each word in a text. Let us execute the following commands on the string obtained in the previous step:

tokens = ['congratulations', 'brand', 'new', 'bill', 'gates', 'lottery', 'winner', 'send', 'us', 'account', 'number', 'transfer', 'money', 'you']
pos_tags = pos_tag(tokens)
print(pos_tags)

This results in the following duos:

[('congratulations', 'NNP'), ('brand', 'NN'), ('new', 'JJ'), ('bill', 'NNP'), ('gates', 'NNP'), ('lottery', 'NNP'), ('winner.', 'NN'), ('send', 'NNP'), ('us', 'PRP'), ('account', 'VBP'), ('number', 'NN'), ('transfer', 'NN'), ('money', 'NN'), ('you', 'NN')]

These duos indicate which part of each sentence constitutes each word. However, for our syntactic and semantic analysis, we only took into account words that are nouns (start with N: NN, NNP), verbs (start with V: VBP), adjectives (start with J: JJ), and adverbs (start with R).

For this reason, the pair *('us', 'PRP')* was eliminated, leaving only the following pairs:

[('congratulations', 'NNP'), ('brand', 'NN'), ('new', 'JJ'), ('bill', 'NNP'), ('gates', 'NNP'), ('lottery', 'NNP'), ('winner.', 'NN'), ('send', 'NNP'), ('account', 'VBP'), ('number', 'NN'), ('transfer', 'NN'), ('money', 'NN'), ('you', 'NN')]

3.1.5. Lemmatize Method

The lemmatization process involves reducing words to their base or root form, which can be helpful in NLP. The base or root form of a word is called its lemma. The NLTK library provides a WordNetLemmatizer class that can be used for lemmatization. As an example, let us execute the following code:

```
words = ["cats", "running", "ate"]
lemmas = [lemmatizer.lemmatize(word, pos='v') for word in words]
print(lemmas)
```

Which produces the following output:

"cat", "run", "eat"

It can be observed that the words cats, running, and ate are lemmatized to their base forms "cat", "run", and "eat", respectively. By lemmatizing words, we can reduce the number of unique words in a text corpus, improving the accuracy and efficiency of many natural language processing tasks.

3.2. Data Pre-Processing

Once all the content of the web pages has been word parsed, it is still necessary to pre-process the data to obtain data ready to be entered into the feature representation using Keras Embedding with GloVe. For this purpose, three consecutive steps were followed, as shown in Figure 3: tokenization, encoding, and padding.

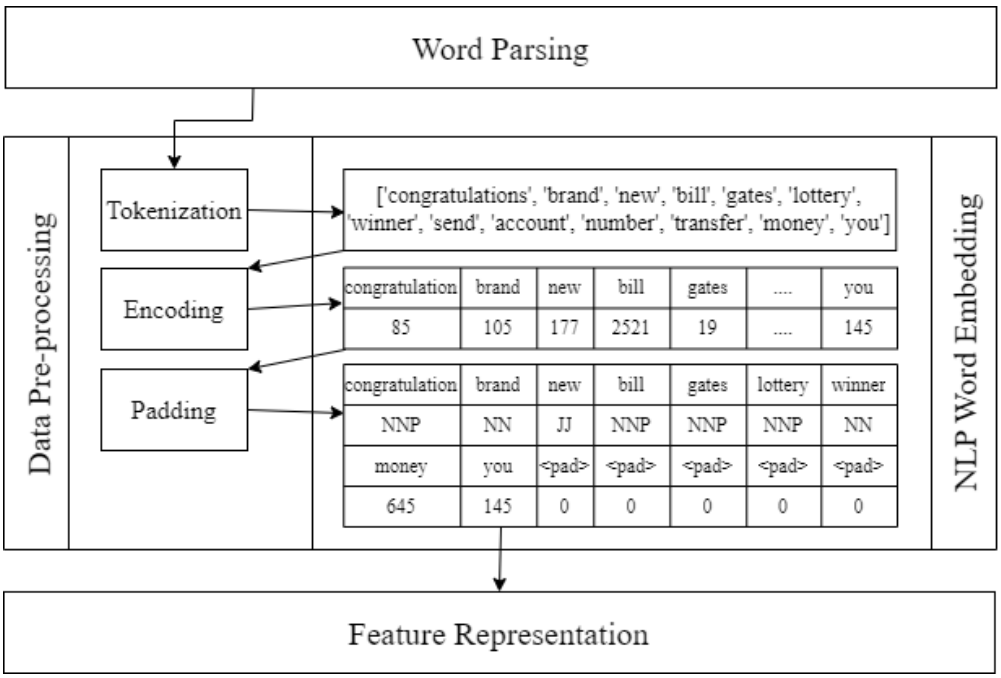


Figure 3. Data pre-processing sub-processes.

3.2.1. Tokenization

Tokenization is a necessary process for any task involving NLP. It consists mainly of vectorizing each word of text obtained previously, resulting in a sequence of meaningful words. In our case, we used the texts_to_sequences tokenization method to transform the input string into a sequence of integers. Let us run the following tokenization commands on the above string example:

words = "congratulations brand new bill gates lottery winner send account number transfer money you"
tokens = word_tokenize(words)
print(tokens)

This split each word of the input string:

['congratulations', 'brand', 'new', 'bill', 'gates', 'lottery', 'winner', 'send', 'account', 'number', 'transfer', 'money', 'you']

3.2.2. Encoding

To encode the words in the text string, we used the texts_to_sequences function, which incrementally assigns an integer to each word as it appears. This way, we convert the text data into a numeric format that the DL models can process. There are other encoding techniques, such as one-hot or TF-IDF; however, we decided to use texts_to_sequences because we wanted to highlight later in this article how Keras Word Embedding works with GloVe. The following code shows how texts_to_sequences works:

words = "congratulations brand new bill gates lottery winner send account number transfer money you"
word_encoded = tokenizer.texts_to_sequences(words)
print(word_encoded)

This produces one integer for each word:

[85, 105, 177, 2521, 19, 2118, 317, 678, 85, 654, 812, 645, 145], dtype=int32

3.2.3. Padding

By padding, we determine the maximum length *maxlen* the string entered into the DL algorithm must have. If the words on the website are longer than *maxlen*, then this string will be truncated until the length is *maxlen*. On the other hand, if the length of the website is less than *maxlen*, then the spaces in which there are no words will be filled by a PAD tag until the length *maxlen* is obtained. Let us analyze the following code:

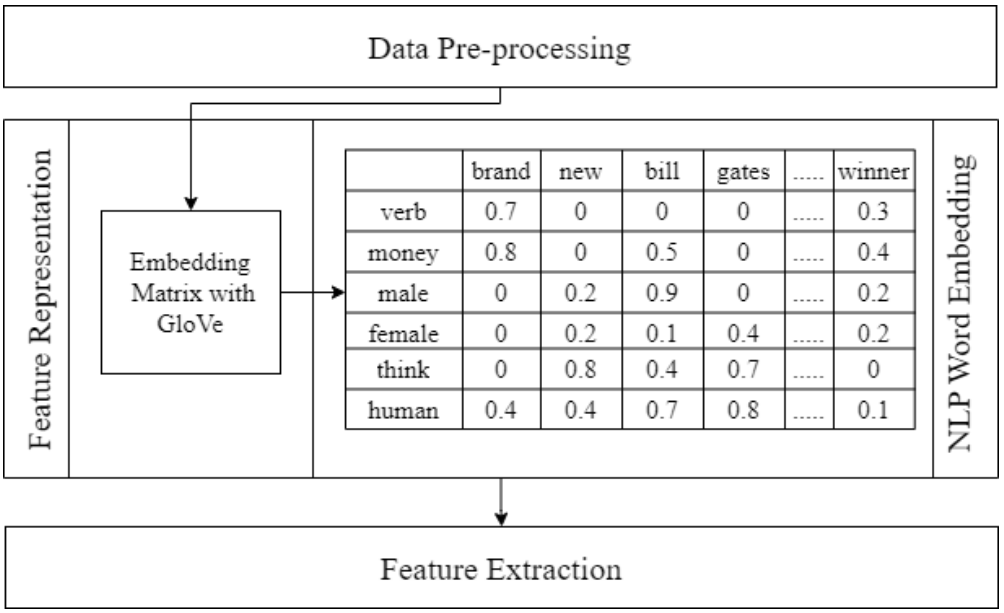
maxlen = 200
tokens = [85, 105, 177, 2521, 19, 2118, 317, 678, 85, 654, 812, 645, 145]
word_sequences = pad_sequences(tokens, padding = 'post', maxlen = maxlen)
print(word_sequences)

It will produce the following sequence until all spaces to the right of the sequence are filled with zeros. If the sequence is more than 200, then the sequence will be truncated to 200:

word_sequences = [85, 105, 177, 2521, 19, 2118, 317, 678, 85, 654, 812, 645, 145, 0, 0, 0, 0, 0, ... , 0]

3.3. Feature Representation with Keras Embedding and GloVe

Keras Embedding Layer with pre-trained GloVe word embeddings works by mapping each word in the input stream to a pre-trained vector representation, which is learned based on the distributional properties of words in a large text corpus. The main idea of our model is to obtain the semantic and syntactic importance of the text of the web pages before the DL algorithms parse it. Hence, we used Keras Embedding with the pre-trained GloVe word embeddings dataset [36]. Figure 4 shows an example of how from a list of words or their coded values, with the use of Keras Embedding and GloVe, a matrix is obtained in which values are stored that indicate the relationship that would exist between a word (of those used in our example) of a row with that of a column, considering that a value close to zero indicates that there is practically no relationship. In contrast, a value close to one indicates a high relationship. For example, if in the matrix of Figure 4, we analyze the word *gates*, we see that it has a high probability that it is related to a *thing* (0.7) or that it is related to a *human* being (0.8), but it has a very low probability that it is a verb (0).



	brand	new	bill	gates	.....	winner
verb	0.7	0	0	0	.....	0.3
money	0.8	0	0.5	0	.....	0.4
male	0	0.2	0.9	0	.....	0.2
female	0	0.2	0.1	0.4	.....	0.2
think	0	0.8	0.4	0.7	.....	0
human	0.4	0.4	0.7	0.8	.....	0.1

Figure 4. Feature representation.

When using pre-trained GloVe word embeddings with the Keras Embedding Layer, the pre-trained embeddings are loaded into memory as a dictionary in which each word is associated with a pre-trained vector representation. An embedding matrix is then created for the vocabulary by looking up the pre-trained vectors for each word. This embedding matrix is used to initialize the weights of the Keras Embedding Layer. In our study, we created the following Keras Embedding Layer with GloVe:

embedding_layer = Embedding(vocab_size, output_dim = 100, weights = [embedding_matrix], trainable = False)(deep_inputs)

where:

vocab_size = 144,236.
embedding_dim = 100.
weights = embedding_matrix obtained with the dataset glove.6B.100d.txt.
trainable = False, because it was already initialized with glove.6B.100d.txt.
(deep_inputs). This means that the input will be of type 2D and is expected to be a matrix of integers where each row represents a sequence of tokens.

3.4. DL Algorithms Execution

The final step to test our model's accuracy and mean accuracy is to input the resulting Keras embedding data into each DL algorithm, with which our model is trained and tested. As additional work in our research, to determine which algorithm best fits our model, we decided to use four DL algorithms, LSTM, BiLSTM, GRU, and BiGRU, for which we elaborated the respective code for each one in Python. We have decided to show each algorithm set up in the following chapter.

In this study, we have decided to use four DL algorithms, which are variants of RNN, considering that these algorithms have a memory, which makes them particularly suitable for processing data series with a temporal order or structure, such as words in a text. According to [37], the two outstanding RNN algorithms for dealing with time series are LSTM with 100% and GRU with 99.99%, respectively, in their evaluation of the Receiver Operating Characteristic (ROC) curve on phishing URLs. That is why we decided to study these algorithms, LSTM and GRU, but on text. We also check their respective Bi-directional approaches, BiLSTM and BiGRU. Thus, to evaluate our model and determine which of the four algorithms is the best for detecting phishing attacks, based on the content of the websites, we will use LSTM, Bi-LSTM, GRU, and BiGRU.

The main idea of using these algorithms is to take advantage of the intrinsic richness in the composition of sentences and the text itself. We consider their semantic and syntactic meaning, i.e., not to use only local feature representations but to go beyond that, using non-spatial features, such as time series. In addition, we mainly use BiLSTM and BiGRU because these algorithms reward and adjust in both forward and backward directions. This way, the analyzed text's semantic and syntactic meaning can be obtained more accurately. Also, because our issue is a binary classification problem, we use binary_cross_entropy as an optimizer. In Addition, since our dataset was unbalanced, we used K-fold cross-validation.

4. Experiment Setup

Once the model has pre-processed the dataset, it remains to run the DL algorithm on the data obtained. However, our work also aimed to determine which DL algorithm best fits the data obtained from our model. For this reason, we decided to experiment with four different DL algorithms, LSTM, BiLSTM, GRU, and BiGRU. This section details the hardware, software, and dataset used and the configuration of the four selected DL algorithms.

4.1. Hardware and Software Environment

For the execution of our experiment, we used a RIG server with Python 3.5.2 on Jupyter Notebook 6.0.2 and the libraries: Keras, NLTK, NumPy, pandas, request, scikit learn, and TensorFlow. Table 2 presents the RIG features of each component of the hardware environment.

Table 2. RIG features.

Component	Model
Processor	AMD Ryzen Threadrippe 2920X
RAM	16 GB Crucial Ballistix DDR4-3000
Video card	16 GB Phantom Gaming X Radeon VII
SSD	500 GB Crucial SSD M.2 NVMe
HD	3 TB Western Digital HDD Purple
Main board	ASUS ROG Zenith Extreme Alpha

4.2. Dataset

We used the freely available dataset Phishload [35], which was then decompressed into a SQL-like file, opened in HeidySQL, and exported to a CSV format, which is more Python-friendly.

This dataset is composed of three tables, of which we used the websites table, from which we extracted two columns:

- htmlContent column, which contains the HTML code of all the Web pages.
- isPhish column, which indicates whether a website is phishing or ham.

This dataset comprises 10,488 rows, but after deleting the rows containing null fields, the dataset was reduced to 10,373 in total, of which 9198 phishing rows and 1176 ham rows were obtained in the end. While the dataset is unbalanced, we used performance measurements for unbalanced data in the evaluation process [38]. Hence, during the execution and to demonstrate the experiment's validity, we used the K-fold cross-validation technique with K = 5, considering that the dataset was divided into 80% for training and 20% for testing.

4.3. DL Algorithms Setup

According to [20], RNN and CNN are the most used DL algorithms for phishing attack detection. Still, RNN has a different characteristic from CNN since it is developed for time-series data [39]. RNN has directional connections that allow it to compute the next step, building on previous steps [40]. Thus, RNN is widely used in NLP and fits our study

very well because our analysis is oriented to the sequence of words obtained from Web pages. A disadvantage of a simple RNN is that it cannot easily find meaningful connections which go beyond 10-time steps [28]. There are RNN-derived algorithms that can solve this problem. Therefore, we have decided to use the following four RNN algorithms for our study: LSTM, BiLSTM, GRU, and BiGRU.

4.3.1. LSTM

This is a variation of the RNN [41]. Unlike a simple RNN algorithm, which can perform computations based on a few word sequences, LSTM can calculate based on recent and non-recent words. In other words, it can determine the importance of the data that even have more than 1000 time steps between them [28]. Figure 5 shows the LSTM algorithm setup.

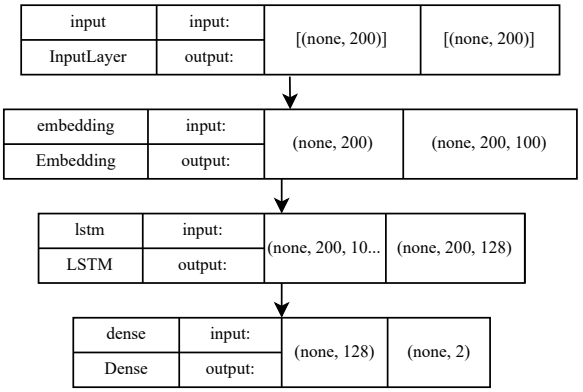


Figure 5. LSTM algorithm setup.

4.3.2. BiLSTM

This is a sequence-processing model with two LSTMs: a forward LSTM and a reverse LSTM. By this back-and-forth process, BiLSTM increases the amount of information available to the network, further improving the context of a study such as ours, where it is essential to know which word precedes and follows another word [42]. Figure 6 shows the BiLSTM algorithm setup.

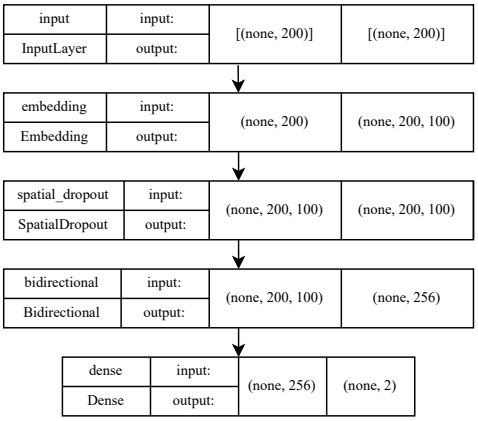


Figure 6. BiLSTM algorithm setup.

4.3.3. GRU

This is like an LSTM network, but with a forget gate. It also has fewer parameters than LSTM because it does not have an output gate. The performance of GRU networks is similar to that of LSTM when it comes to NLP [43]. Figure 7 shows the GRU algorithm setup.

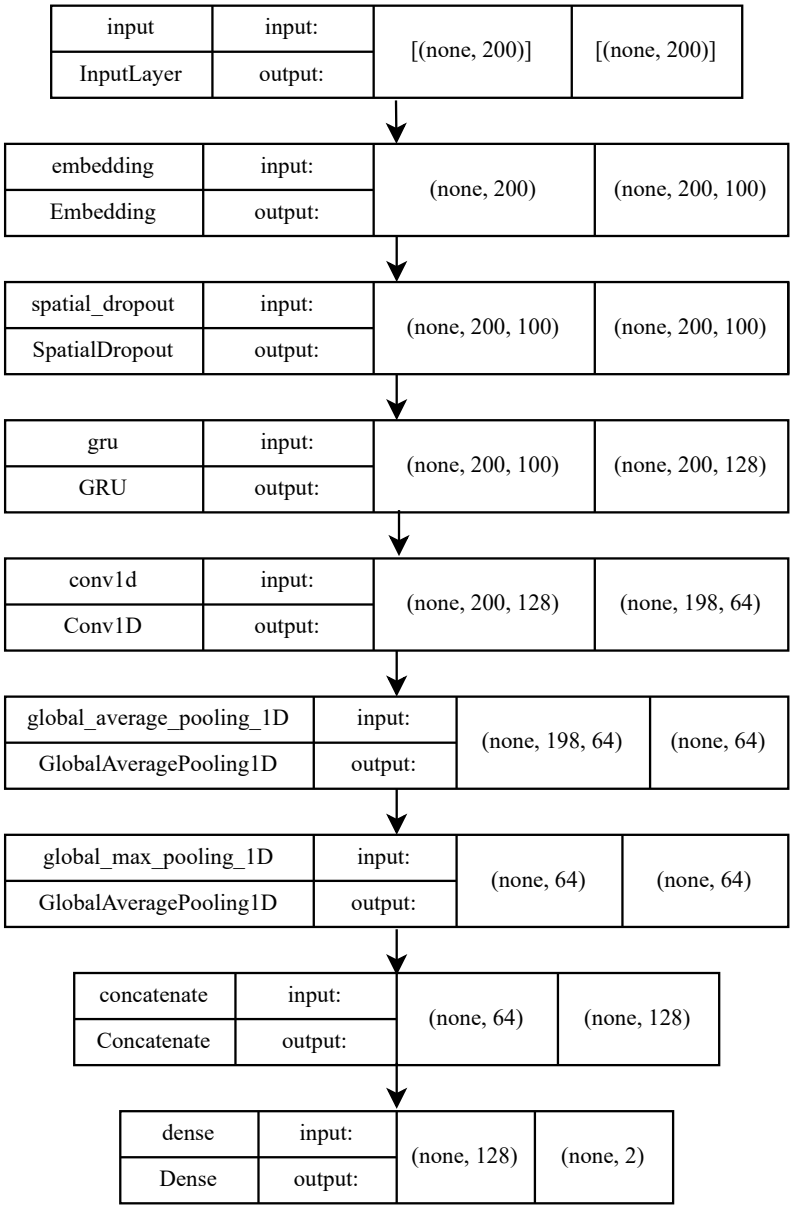


Figure 7. GRU algorithm setup.

4.3.4. BiGRU

A GRU network employs recurrence to store and retrieve information for long periods, but in practice, its performance could be better because the network only accesses past information [44]. Thus, to solve this information access problem, BiGRU has a future layer in which the data sequence is in the opposite direction. Therefore, this network uses two hidden layers to extract past and future information. These hidden layers are then connected into a single output layer [45]. These characteristics enable the bidirectional structure to assist the RNN in extracting more information and, consequently, improve the performance of the learning process. Figure 8 shows the BiGRU algorithm setup.

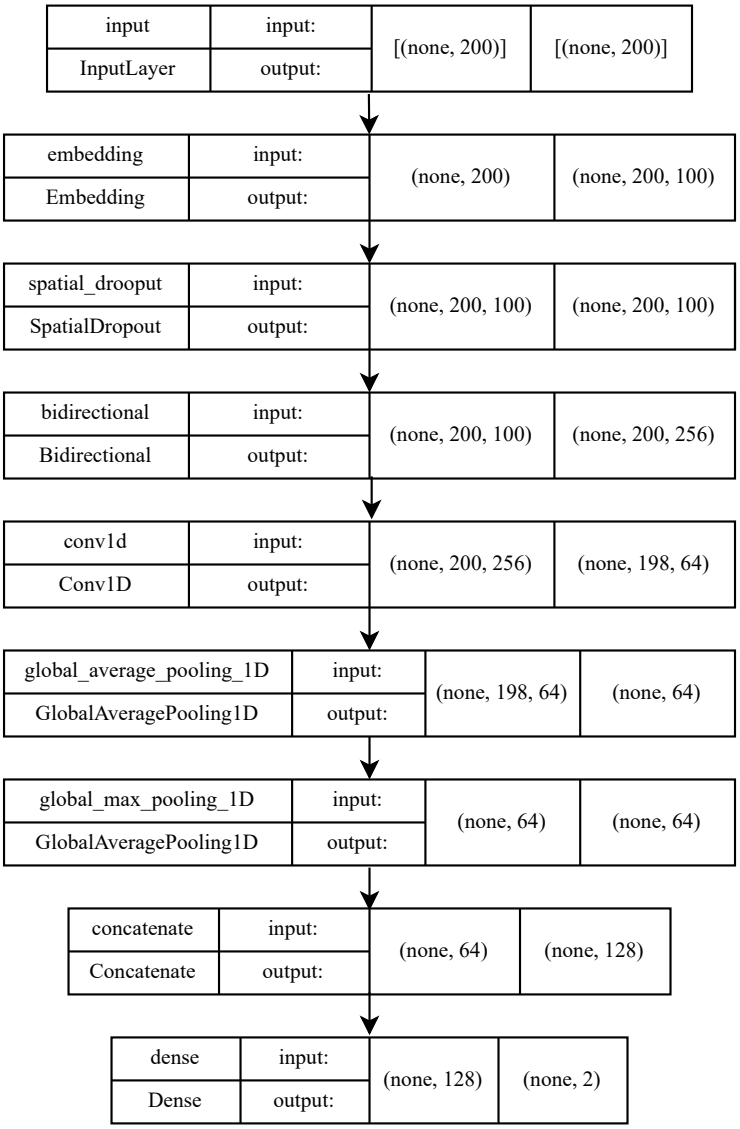


Figure 8. BiGRU algorithm setup.

4.4. Performance Metrics

We adopted commonly used quality metrics to compare the correctness of the classification of the detection model. We used the following terms for determining the quality of the classification models [46]:

- *True Positive (TP)*: the number of data items correctly classified to the positive class.
- *True Negative (TN)*: the number of data items correctly classified to the negative class.
- *False Positive (FP)*: the number of data items wrongly classified to the positive class.
- *False Negative (FN)*: the number of data items wrongly classified to the negative class.

When dealing with unbalanced data, choosing appropriate evaluation metrics that consider the class distribution in the dataset is important. We used the following metrics:

- Precision: precision measures the proportion of true positives among the predicted positives. This metric is useful for minimizing false positives or when the positive class is rare.

$$Precision = TP/(TP + FP) \quad (1)$$

- Recall: recall measures the proportion of true positives among the total number of actual positives. This metric is useful when minimizing false negatives or when the positive class is important.

$$Recall = TP/(TP + FN) \quad (2)$$

- F1-score: the F1-score is the harmonic mean of the precision and recall and is a good metric to use when there is an uneven distribution of classes in the dataset.

$$F1\text{-}Score = 2 \times ((Precision \times Recall)/(Precision + Recall)) \quad (3)$$

- Micro avg: calculates the overall average precision score over all classes, weighting each instance by its weight.
- Weighted avg: calculates the score by averaging the results for each class, weighting each result by the number of instances of that class in the dataset.
- K-fold cross-validation: since the data are not balanced, we used the K-fold cross-validation technique to calculate the mean accuracy. This technique is also helpful in avoiding overfitting the training data. This technique is a suitable metric for determining which of the four DL algorithms performs best.

5. Results and Discussion

This section shows and discusses the meaning of the results obtained. To validate it, we executed the four DL algorithms. Before executing the four DL algorithms, it is essential to determine the length of the text to be entered into the algorithms.

5.1. Analysis of the Number of Words to Enter the Algorithm

Analyzing our dataset, as shown in Figure 9, three red lines are drawn, corresponding to L = 200, L = 500, and L = 1000 words per web site, in which 9.78% of 10,373 websites had more than 1000 words. For this reason, only a max length L of 1000 words and below was taken for the NLP analysis. Thus, three values of L were defined to analyze which value works best: L = 1000 represents 90.22% of the data; L = 500, which is 81.33% of the data; L = 200, which means 58.02% of the data.

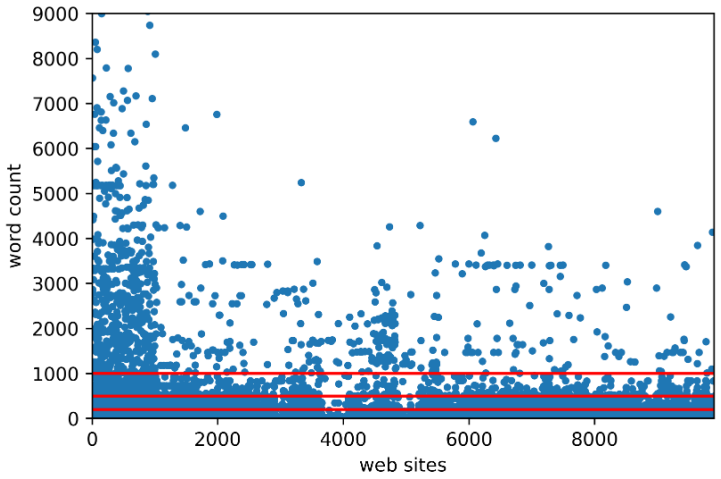


Figure 9. Word length distribution over the entire dataset.

Due to the different text lengths L obtained from each web page analyzed, the four algorithms were run with three different values of L (200, 500, and 1000) to test which of the lengths L of words performed better.

5.2. Evaluation and Results

Table 3 shows the test loss, test AUC, training loss, and training AUC obtained with each DL algorithm and each L. It was observed that GRU and BiGRU always gave better test accuracy than the other algorithms. BiLSTM showed values close to GRU or BiGRU; however, LSTM gave the worst results.

Table 3. Test loss, test AUC, training loss, and training AUC with L = 1000, 500, and 200.

Algorithm	L	Test Loss	Test AUC	Train Loss	Train AUC
LSTM	1000	0.27	0.92	0.27	0.93
BiLSTM	1000	0.17	0.98	0.12	0.99
GRU	1000	0.14	0.99	0.09	0.99
BiGRU	1000	0.15	0.96	0.07	1
LSTM	500	0.23	0.95	0.23	0.95
BiLSTM	500	0.17	0.98	0.12	0.99
GRU	500	0.14	0.99	0.09	0.96
BiGRU	500	0.14	0.99	0.07	1
LSTM	200	0.19	0.97	0.18	0.97
BiLSTM	200	0.19	0.98	0.10	0.99
GRU	200	0.16	0.98	0.09	0.99
BiGRU	200	0.17	0.98	0.07	1

Although the Area Under the ROC Curve (AUC) is not as appropriate to define the accuracy of an algorithm, it is important to compare the performance of the four DL algorithms. Table 4 shows the AUC calculated with each length L. Our model is of binary type since only two classes are evaluated (if the AUC is close to 0, indicating ham; or if the AUC is close to 1, indicating phishing). It was observed that GRU and BiGRU gave better results than LSTM and BiLSTM. Even at L = 200, GRU scored better. In addition, we also controlled the execution time of each DL algorithm in our model and observed that GRU was the one that was trained the fastest because it was trained in 240 s.

Table 4. Area Under the ROC Curve (AUC) with L = 1000, 500, and 200.

Algorithm	L	AUC = 0 (Ham)	AUC = 1 (Phishing)	Time (s)
LSTM	1000	0.74	0.72	1480
BiLSTM	1000	0.94	0.94	1660
GRU	1000	0.96	0.96	1320
BiGRU	1000	0.96	0.96	1860
LSTM	500	0.81	0.80	700
BiLSTM	500	0.94	0.94	800
GRU	500	0.96	0.96	600
BiGRU	500	0.96	0.96	840
LSTM	200	0.91	0.91	300
BiLSTM	200	0.94	0.94	320
GRU	200	0.96	0.96	240
BiGRU	200	0.95	0.95	320

Table 5 shows the computed micro avg and weighted avg metrics. In the three runs of the experiment with L = 1000, 500, and 200, it was again observed that GRU and BiGRU prevailed over LSTM and BiLSTM. In our particular case with unbalanced binary data, the most-appropriate metric to note is the weighted avg; but even for those, it was observed that GRU was equal to BiGRU. To continue our experiment with F1-Score, we have used L = 200 only because it represents 58.02% of our dataset. We do not use values below 200 words because the metrics decrease considerably.

Table 5. Micro avg, weighted avg, and F1-score.

Algorithm	L	Micro Avg	Weighted Avg	F1-Score
LSTM	1000	0.91	0.88	
BiLSTM	1000	0.98	0.96	
GRU	1000	0.99	0.98	
BiGRU	1000	0.99	0.98	
LSTM	500	0.94	0.92	
BiLSTM	500	0.98	0.97	
GRU	500	0.99	0.98	
BiGRU	500	0.99	0.98	
LSTM	200	0.97	0.95	0.93
BiLSTM	200	0.98	0.96	0.74
GRU	200	0.98	0.97	0.94
BiGRU	200	0.98	0.97	0.94

To define which of the DL algorithms worked best with the data embedded with GloVe, we performed the execution of the four algorithms using the K-fold cross-validation technique, where K = 5 and shuffle = true. Hence, the results obtained in each K-fold and the mean accuracy obtained in each algorithm can be seen in Table 6.

Table 6. Mean accuracy in percent with K-fold cross-validation with K = 1 to K = 5.

Algorithm	K = 1	K = 2	K = 3	K = 4	K = 5	Mean
LSTM	94.12	95.76	96.48	98.36	98.84	96.71
BiLSTM	94.26	95.95	97.96	98.75	99.08	97.20
GRU	95.03	95.90	98.26	98.41	99.28	97.29
BiGRU	95.22	96.53	98.55	98.84	98.84	97.39

5.3. Discussion

Based on what was stated in the previous subsection, here, we present the graphical analysis performed when executing each algorithm with an inserted Length (L) of 200 words. In general, as can be seen in Figures 10–13, the four models worked well because they were generalizing correctly. Consequently, the training accuracies' values obtained in each model were close to their respective validation accuracies' values.

5.3.1. NLP-LSTM

As can be seen in Figure 10, the test accuracy (97%) and training accuracy (97%) obtained with LSTM were acceptable. Again, it can be seen that the validation accuracy line advanced very close to the training accuracy; However, at Epoch 10, there was a drop in accuracy to less than 95%. This drop may be due to an overfitting problem. Among the four DL algorithms, LSTM was the worst performer.

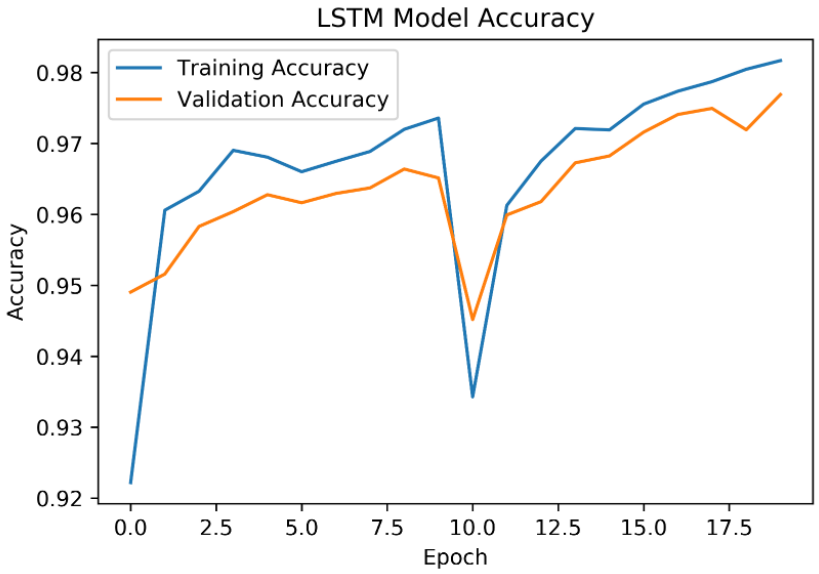


Figure 10. LSTM accuracy.

5.3.2. NLP-BiLSTM

Figure 11 shows that the training accuracy (98%) and validation accuracy (99%) obtained with BiLSTM outperformed those obtained with LSTM. Although there was also a decay at Epoch 10, this decayed slightly (to 97%). At the end of the run, at Epoch 17, there was a decrease in the validation accuracy, indicating possible overfitting.

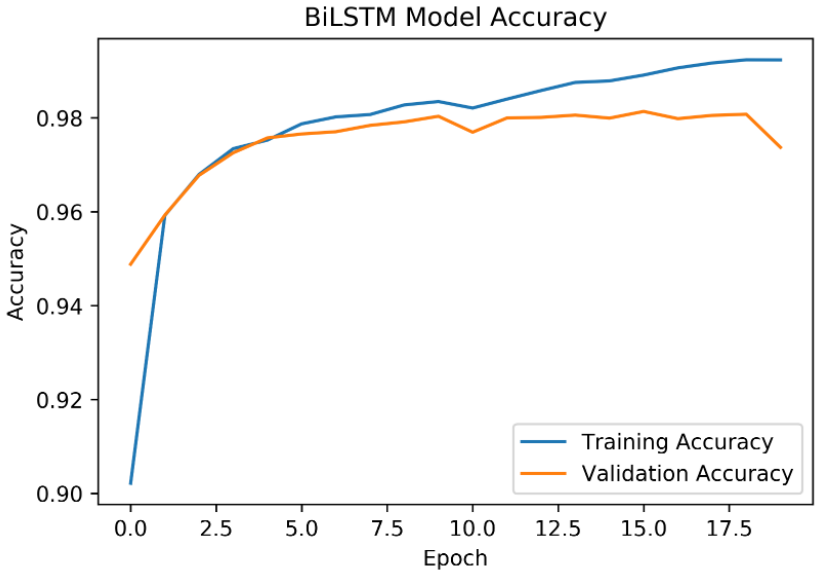


Figure 11. BiLSTM accuracy.

5.3.3. NLP-GRU

Figure 12 shows that the training accuracy (98%) and validation accuracy (99%) improve considerably. Negative slopes are not observed in the two lines, but are constant growth. In addition, although the validation accuracy declined a little at the end of the processing, compared to the validation accuracy, this decline was minimal, remaining close to the training accuracy line. Additionally, the processing time was 240 s, thus surpassing the processing time of LSTM with 300 s and BiLSTM with 320 s.

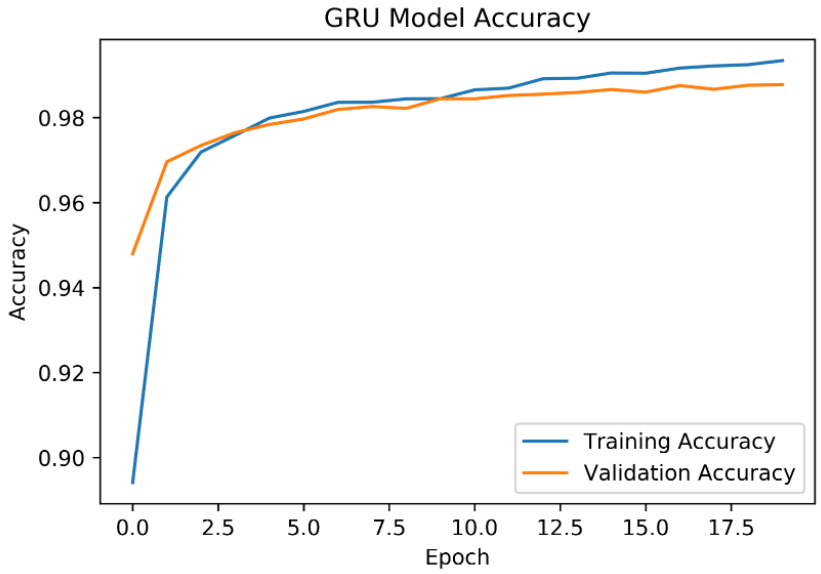


Figure 12. GRU accuracy.

5.3.4. NLP-BiGRU

Figure 13 shows that the NLP-BiGRU combination was the one that gave the best results for the training accuracy (100%). However, the validation accuracy obtained (98%) did not vary concerning BiLSTM (98%) and GRU (98%). On the other hand, the time to achieve its training execution was 320s, i.e., 33% more than the time processed by GRU.

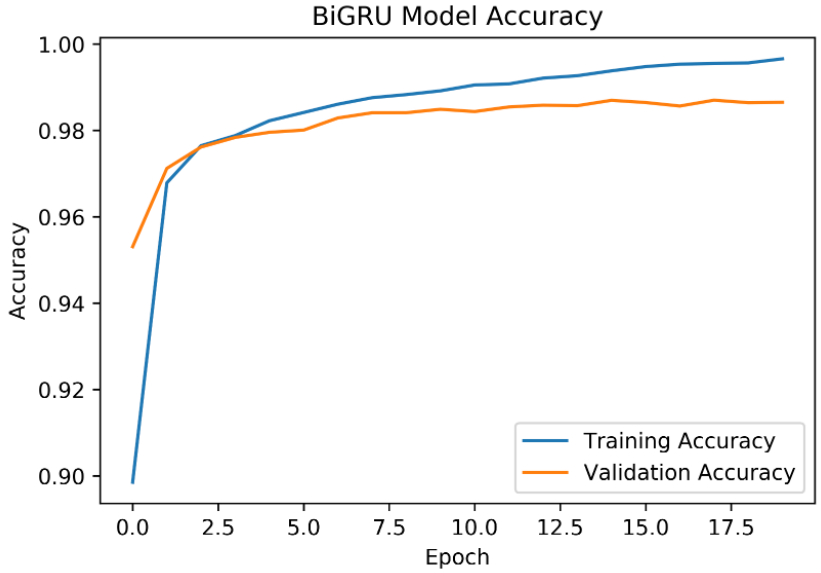


Figure 13. BiGRU accuracy.

5.3.5. Graphical Comparison of the Four Models

Figure 14 compares the accuracy obtained for each epoch in the four models. It can be seen that LSTM behaved the worst, even showing erratic behavior, with negative slopes in several periods, highlighting a very pronounced negative slope at Epoch 10, with an accuracy below 95%. The BiLSTM model gave better results than LSTM. Although the training accuracy improved notably, according to Figure 11, the same did not occur with the validation accuracy, which had a negative slope at Epoch 10 and a pronounced negative slope at the end of Epoch 20. GRU and BiGRU gave better results in training than those offered by LSTM and BiLSTM since it was observed that, in both of them, an accuracy that constantly increased from the first epoch was obtained. Analyzing GRU versus BiGRU, it

was observed that BiGRU gave better results, obtaining in the last Epoch 20 100% for its training accuracy, over the 99% obtained in GRU. Even in Figure 15, it can be seen that the model with the lowest loss was BiGRU, followed by GRU. On the other hand, of the four algorithms analyzed, the one that performed worst was LSTM.

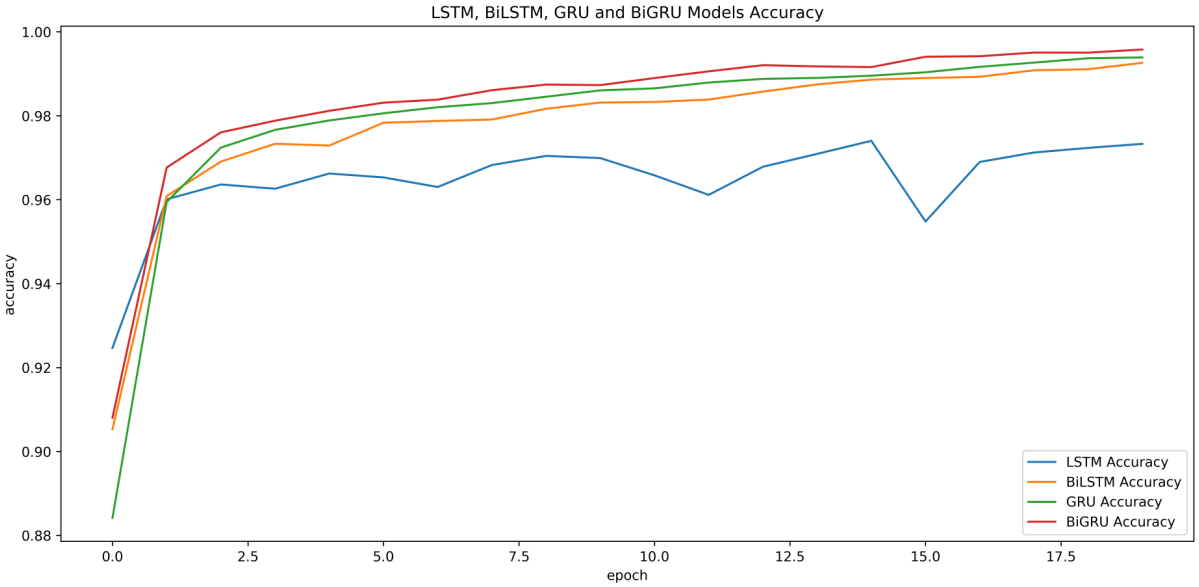


Figure 14. Accuracy comparison of the four algorithms.

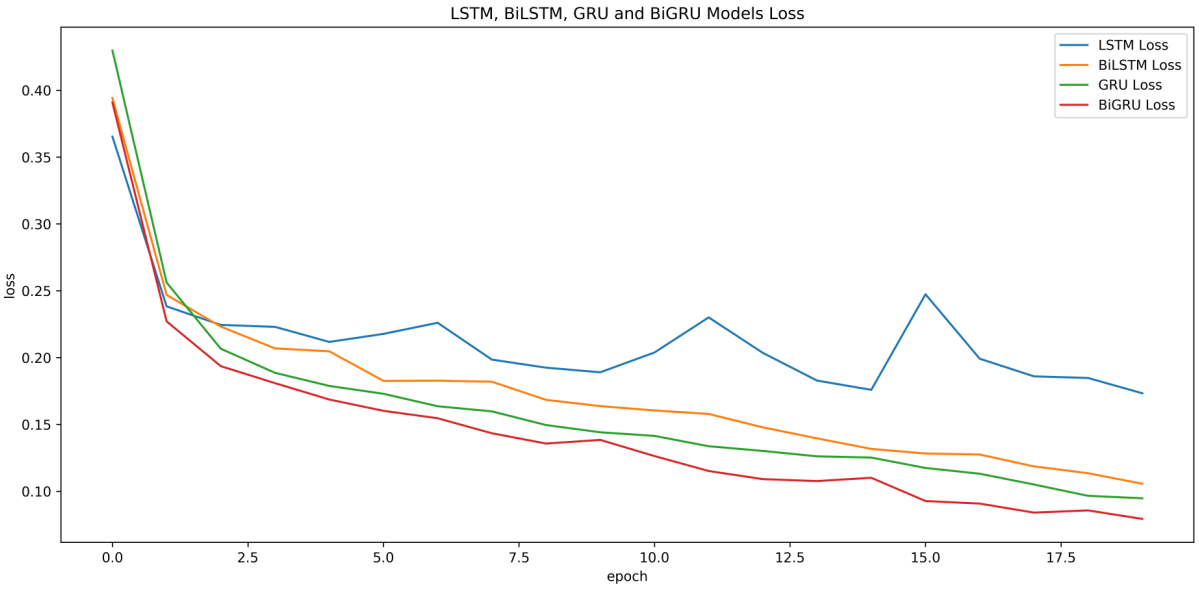


Figure 15. Loss comparison of the four algorithms.

To define which of the DL algorithms worked best with the data embedded with GloVe, we performed the execution of the four algorithms using the K-fold cross-validation technique, where K = 5 and shuffle = true. The run results in Table 6 indicate that our model is acceptable, as all four algorithms performed with a mean accuracy above 96.70%. The DL algorithm with the best results was the BiGRU algorithm, with 97.39%.

6. Conclusions

This study proposed an innovative phishing-detection model using the Keras Embedding Layer with GloVe to take advantage of the semantic and syntactic features of the web page text. Our proposed model used automatic features of the text contained in web pages,

applying word-level embedding methods to represent these features in vector form. These vectors were then input to the LSTM, BiLSTM, GRU, and BiGRU algorithms to identify phishing pages.

According to the literature review conducted in this study, there still is a research gap in web page text analysis using NLP and DL. Although several articles on combating phishing with NLP and DL were initially found, most were oriented toward combating phishing emails. As for tackling web pages, most were oriented toward working on URLs and not on the text of the web pages.

After the experiment, it can be determined that BiGRU was the DL algorithm that performed better with the data previously analyzed with NLP than the other three DL algorithms, LSTM, BiLSTM, and GRU. On the other hand, BiLSTM gave the worst results, even below those expected. The validity of our model was demonstrated using the K-fold cross-validation technique, which yielded a mean accuracy of LSTM 96.71%, BiLSTM 97.20%, GRU 97.29%, and the best, BiGRU 97.39%.

The model was tested with three text lengths obtained from web pages, L = 200, L = 500, and L = 1000. However, the results did not vary much between them, so it was determined that 200 was the optimal size to perform this analysis. Internally, we ran experiments with words below 200, but this resulted in very poor results, which is why text lengths below 200 were not taken into consideration for the experiments.

In future work, we plan to evaluate our model with other word embeddings mechanisms, such as Fast Text and Word2Vec, to evaluate their performance in the NLP domain. In addition, to improve the performance of DL algorithms, we plan to implement an attention-based DL architecture, which can differentiate which parts are more critical than others, depending on the context.

While we obtained acceptable results to face phishing attacks with NLP execution and DL algorithms, we plan to conduct a comparative study in which we can tune the parameters of both the embedding layer and DL algorithm layers for the best results. Also, when performing the SLR, we noticed that it is necessary to perform an SLR about mitigating phishing attacks only with NLP and DL. As well as it is also necessary to perform an algorithm that detects these attacks using ensemble methods of at least two DL Algorithms on various levels, such as URL, third-party information, and Web content level.

Author Contributions: Conceptualization, E.B.-A. and S.S.-G.; Methodology, E.B.-A., W.F. and S.S.-G.; Software, E.B.-A. and G.R.-G.; Validation, E.B.-A. and S.S.-G.; Formal analysis, W.F. and S.S.-G.; Investigation, E.B.-A.; Resources, D.N.-A. and G.R.-G.; Data curation, E.B.-A., W.F. and D.N.-A.; Writing—original draft, E.B.-A., S.S.-G. and G.R.-G.; Writing—review & editing, E.B.-A. and W.F.; Supervision, W.F. and S.S.-G.; Funding acquisition, E.B.-A., W.F. and D.N.-A. All authors have read and agreed to the published version of the manuscript.

Funding: The authors' funds financed this research.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Benavides-Astudillo, E.; Silva-Ordoñez, L.; Rocohano-Rámos, R.; Fuertes, W.; Fernández-Peña, F.; Sanchez-Gordon, S.; Bastidas-Chalan, R. Analysis of Vulnerabilities Associated with Social Engineering Attacks Based on User Behavior. *Commun. Comput. Inf. Sci.* **2022**, *1535*, 351–364. [CrossRef]
2. Benavides-Astudillo, E.; Tipan-Guerrero, N.; Castillo-Zambrano, G.; Díaz, W.F.; Galán, G.E.; Cazáres, M.F.; Nuñez-Agurto, D. A Framework Based on Personality Traits to Identify Vulnerabilities to Social Engineering Attacks. *Commun. Comput. Inf. Sci.* **2022**, *1535*, 381–394.
3. Macas, M.; Wu, C.; Fuertes, W. A survey on deep learning for cybersecurity: Progress, challenges, and opportunities. *Comput. Netw.* **2022**, *212*, 109032. [CrossRef]
4. APWG. Phishing Activity Trends Reports. Available online: https://apwg.org/trendsreports (accessed on 29 January 2023).

5. Lastdrager, E. Achieving a consensual definition of phishing based on a systematic review of the literature. *Crime Sci.* **2014**, *3*, 9. [CrossRef]
6. Balasubaramanian, S.; Ganesan, P.; Rajasekaran, J. Weighted ensemble classifier for malicious link detection using natural language processing. *Int. J. Pervasive Comput. Commun.* **2023**. . [CrossRef]
7. Elsadig, M.; Ibrahim, A.O.; Basheer, S.; Alohali, M.A.; Alshunaifi, S.; Alqahtani, H.; Alharbi, N.; Nagmeldin, W. Intelligent Deep Machine Learning Cyber Phishing URL Detection Based on BERT Features Extraction. *Electronics* **2022**, *11*, 3647. [CrossRef]
8. Bagui, S.; Nandi, D.; Bagui, S.; White, R.J. Machine learning and deep learning for phishing email classification using one-hot encoding. *J. Comput. Sci.* **2021**, *17*, 610–623. [CrossRef]
9. Chowdhary, K.; Chowdhary, K. Natural language processing. In *Fundamentals of Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 603–649.
10. Sutter, T.; Bozkir, A.S.; Gehring, B.; Berlich, P. Avoiding the Hook: Influential Factors of Phishing Awareness Training on Click-Rates and a Data-Driven Approach to Predict Email Difficulty Perception. *IEEE Access* **2022**, *10*, 100540–100565. [CrossRef]
11. Benavides-Astudillo, E.; Fuertes, W.; Sanchez-Gordon, S.; Rodriguez-Galan, G.; Martínez-Cepeda, V.; Nuñez-Agurto, D. Comparative Study of Deep Learning Algorithms in the Detection of Phishing Attacks Based on HTML and Text Obtained from Web Pages. In Proceedings of the Applied Technologies: 4th International Conference, ICAT 2022, Quito, Ecuador, 23–25 November 2022; Revised Selected Papers, Part I; Springer: Berlin/Heidelberg, Germany, 2023; pp. 386–398.
12. Zhang, X.; Zeng, Y.; Jin, X.B.; Yan, Z.W.; Geng, G.G. Boosting the phishing detection performance by semantic analysis. In Proceedings of the IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 1063–1070.
13. Goyal, P.; Pandey, S.; Jain, K. *Deep Learning for Natural Language Processing*; Apress: New York, NY, USA, 2018.
14. Safi, A.; Singh, S. A Systematic Literature Review on Phishing Website Detection Techniques. *J. King Saud-Univ.-Comput. Inf. Sci.* **2023**, *35*, 590–611. [CrossRef]
15. Alshingiti, Z.; Alaqel, R.; Al-Muhtadi, J.; Haq, Q.E.U.; Saleem, K.; Faheem, M.H. A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTM-CNN. *Electronics* **2023**, *12*, 232. [CrossRef]
16. Medsker, L.R.; Jain, L. Recurrent neural networks. *Des. Appl.* **2001**, *5*, 64–67.
17. Keras. Embedding Layer. Available online: https://keras.io/api/layers/corelayers/embedding (accessed on 15 November 2022).
18. Selvaganapathy, S.G.; Nivaashini, M.; Natarajan, H.P. Deep belief network based detection and categorization of malicious URLs. *Inf. Secur. J. Glob. Perspect.* **2018**, *27*, 145–161. [CrossRef]
19. Aleroud, A.; Zhou, L. Phishing environments, techniques, and countermeasures: A survey. *Comput. Secur.* **2017**, *68*, 160–196. [CrossRef]
20. Benavides-Astudillo, E.; Fuertes, W.; Sanchez-Gordon, S.; Sanchez, M. Classification of Phishing Attack Solutions by Employing Deep Learning Techniques: A Systematic Literature Review. *Smart Innov. Syst. Technol.* **2020**, *152*, 51–64. [CrossRef]
21. Salloum, S.; Gaber, T.; Vadera, S.; Sharan, K. A systematic literature review on phishing email detection using natural language processing techniques. *IEEE Access* **2022**, *10*, 65703–65727. [CrossRef]
22. Zieni, R.; Massari, L.; Calzarossa, M.C. Phishing or not phishing? A survey on the detection of phishing websites. *IEEE Access* **2023**, *11*, 18499–18519. [CrossRef]
23. Rao, R.S.; Pais, A.R. Detection of phishing websites using an efficient feature-based machine learning framework. *Neural Comput. Appl.* **2019**, *31*, 3851–3873. [CrossRef]
24. Marchal, S.; Armano, G.; Gröndahl, T.; Saari, K.; Singh, N.; Asokan, N. Off-the-hook: An efficient and usable client-side phishing prevention application. *IEEE Trans. Comput.* **2017**, *66*, 1717–1733. [CrossRef]
25. Jain, A.K.; Gupta, B.B. A machine learning based approach for phishing detection using hyperlinks information. *J. Ambient. Intell. Humaniz. Comput.* **2019**, *10*, 2015–2028.
26. Kitchenham, B.; Pearl Brereton, O.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in software engineering—A systematic literature review. *Inf. Softw. Technol.* **2009**, *51*, 7–15. [CrossRef]
27. Yen, S.; Moh, M.; Moh, T.S. Detecting compromised social network accounts using deep learning for behavior and text analyses. *Int. J. Cloud Appl. Comput. IJCAC* **2021**, *11*, 97–109. [CrossRef]
28. Ozcan, A.; Catal, C.; Donmez, E.; Senturk, B. A hybrid DNN–LSTM model for detecting phishing URLs. *Neural Comput. Appl.* **2021**, *35*, 4957–4973. [CrossRef] [PubMed]
29. Xiao, X.; Xiao, W.; Zhang, D.; Zhang, B.; Hu, G.; Li, Q.; Xia, S. Phishing websites detection via CNN and multi-head self-attention on imbalanced datasets. *Comput. Secur.* **2021**, *108*, 102372. [CrossRef]
30. Sirigineedi, S.S.; Soni, J.; Upadhyay, H. Learning-based models to detect runtime phishing activities using URLs. In Proceedings of the 4th International Conference on Compute and Data Analysis, Silicon Valley, CA, USA, 9–12 March 2020; pp. 102–106. [CrossRef]
31. Adebowale, M.A.; Lwin, K.T.; Hossain, M.A. Intelligent phishing detection scheme using deep learning algorithms. *J. Enterp. Inf. Manag.* **2020**.
32. Villanueva, A.; Atibagos, C.; De Guzman, J.; Cruz, J.C.D.; Rosales, M.; Francisco, R. Application of Natural Language Processing for Phishing Detection Using Machine and Deep Learning Models. In Proceedings of the 2022 International Conference on ICT for Smart Society (ICISS), Bandung, Indonesia, 10–11 August 2022; pp. 1–6.

33. Ariyadasa, S.; Fernando, S.; Fernando, S. Combining Long-Term Recurrent Convolutional and Graph Convolutional Networks to Detect Phishing Sites Using URL and HTML. *IEEE Access* **2022**, *10*, 82355–82375. [CrossRef]
34. Do, N.Q.; Selamat, A.; Krejcar, O.; Yokoi, T.; Fujita, H. Phishing webpage classification via deep learning-based algorithms: An empirical study. *Appl. Sci.* **2021**, *11*, 9210. [CrossRef]
35. Maurer, M.E. Phishload. Available online: https://www.medien.ifi.lmu.de/team/max.maurer/files/phishload (accessed on 16 November 2022).
36. GloVe: Global Vectors for Word Representation. Available online: https://nlp.stanford.edu/projects/glove (accessed on 15 January 2023).
37. Vinayakumar, R.; Soman, K.P.; Poornachandran, P. Evaluating deep learning approaches to characterize and classify malicious URL's. *J. Intell. Fuzzy Syst.* **2018**, *34*, 1333–1343. [CrossRef]
38. Alsufyani, A.A.; Alzahrani, S. Social Engineering Attack Detection Using Machine Learning: Text Phishing Attack. *Indian J. Comput. Sci. Eng.* **2021**, *12*, 743–751. [CrossRef]
39. Sherstinsky, A. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Phys. Nonlinear Phenom.* **2020**, *404*, 132306. . Available online: http://xxx.lanl.gov/abs/1808.03314 (accessed on 29 January 2023). [CrossRef]
40. Deng, H.; Zhang, L.; Shu, X. Feature memory-based deep recurrent neural network for language modeling. *Appl. Soft Comput.* **2018**, *68*, 432–446. [CrossRef]
41. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
42. PapersWithCode. Bidirectional LSTM. Available online: https://paperswithcode.com/method/bilstm (accessed on 21 January 2023).
43. Britz, D. Recurrent Neural Network Tutorial, Part 4—Implementing a GRU and LSTM RNN with Python and Theano. Available online: https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-4 (accessed on 29 January 2023).
44. Deng, Y.; Jia, H.; Li, P.; Tong, X.; Qiu, X.; Li, F. A deep learning methodology based on bidirectional gated recurrent unit for wind power prediction. In Proceedings of the 14th IEEE Conference on Industrial Electronics and Applications, ICIEA, Xi'an, China, 19–21 June 2019; pp. 591–595. [CrossRef]
45. Luo, X.; Zhou, W.; Wang, W.; Zhu, Y.; Deng, J. Attention-Based Relation Extraction with Bidirectional Gated Recurrent Unit and Highway Network in the Analysis of Geological Data. *IEEE Access* **2018**, *6*, 5705–5715. [CrossRef]
46. Vinayakumar, R.; Alazab, M.; Srinivasan, S.; Pham, Q.V.; Padannayil, S.K.; Simran, K. A visualized botnet detection system based deep learning for the internet of things networks of smart cities. *IEEE Trans. Ind. Appl.* **2020**, *56*, 4436–4456. [CrossRef]