# The Challenge of URL-based Phishing Detection Using In-The-Wild Data

Shai Cohen
Ben Gurion University
shco@post.bgu.ac.il

Jeremy Chew
Singapore University of Technology and Design
jeremy_chew@mymail.sutd.edu.sg

Dvir Cohen
Ben Gurion University
dvircohe@post.bgu.ac.il

Asaf Shabtai
Ben Gurion University
shabtaia@bgu.ac.il

***Abstract*—Phishing scams constitute one of the most common cybercrimes that are being committed on the Internet. As such, there is a need to proactively detect such phishing threats. While there have been several competitive work done on this very issue, most of them do not stand up to the glut of URLs that occur under real-world conditions. In this paper, we put the validity of many competitive models on doubt. By implementing and evaluating these models on data meant to simulate "real-world" conditions, we show that even though these models are able to achieve high performances on their tested data sets, they are unable to generalise well and achieve the same performance on "in the wild" data. We also propose a solution that specifically targets this issue. By using an auto-encoder trained on unlabelled URLs, we show that combining it with a classification model not only achieves competitive performance under testing conditions, but outperforms other models when evaluated against "real-world" data sets.**

## I. Introduction

Phishing URLs are used to mislead users and redirect them into malicious websites, where attackers can obtain sensitive information (e.g., usernames, passwords, credit card details, email address, etc.), or install malicious software on the users' devices. Although phishing URLs are well-known cyber-threats, many still fall prey to these scams. A report published by the FBI [13] indicates that "Phishing, Vishing, Smishing, and Pharming" attacks have had the most victims across all other internet crimes in 2019 alone. Another report published by the Anti Phishing Working Group (APWG) [3] indicates that phishing attacks rose in the third quarter of 2019 to a new high since 2016.

As phishing URLs are a major risk on the Internet, there have been much work done to identify and mitigate their risks [9], [20], [29], [27], [16], [26]. One common approach to is to create a database of black/white-listed URLs and block/approve incoming URL requests accordingly [20], [9]. However, this approach is limited. Blocking only black-listed URLs makes it still susceptible to zero-day attacks, and allowing access to only white-listed URLs could be unfair to legitimate websites, as it would not allow unpopular, yet benign, non-white-listed URLs to pass. In addition, a report from the APWG showed that the up-time of phishing pages is less than two days, thus relying on a list of a malicious domains may be flawed as the domains would become irrelevant after only a short while.

Another approach is to learn the defining characteristics of malicious/benign URLs and build an algorithm that, given a new URL, will determine if it is a phishing site. Such an approach is often implemented through the use of Machine Learning (ML) algorithms [29], [27], [26], [16]. That is, given a corpus of labelled data which contains both phishing and benign URLs, a ML algorithm tries to learn the characteristics of each group and, classify new URLs into the most similar group.

In this regard, several methods have been suggested that use only the raw URL string without collecting any additional information from the website. The information extracted are known as lexical features, and research has shown that algorithms based on only these types of features remain competitive compared to approaches that uses additional information gained from the website [8]. These methods are also much faster than other approaches, as there is no need to rely on network access and third-party services.

However, these methods share one major problem. Although these methods work well on the tested data sets, this is not the case for "in the wild" data. The test data sets are far from reflecting the "real world" environment. Most of these studies use the Alexa 1M data set as a source for benign URLs and Phishtank as a source for phishing URLs. However, these data sets include URLs that are either "very benign" or "very malicious", and hence the model does not learn how to deal with URLs that are not explicitly malicious nor benign. In addition, Phishtank contains duplicated domain names, which are not filtered out in most cases. Additionally, some of the published data sets contains a variety of problems that make it easy for the model to distinguish between phishing and benign URLs. For example, malicious URLs could always contain the URL's path, while benign URLs would never contain the path. Some of the research works also use very small data sets, which would cause inherent biases in the model and disallow it from generalising to other data.

Another common issue we found is that these models are often trained and executed only on URLs that contain the path. However, in several common scenarios listed below, this information is not present in the analyzed URL. (1) Encrypted path - According to APWG [4], almost three-quarters of phishing sites used the HTTPS protocol. Under this protocol, the path is encrypted and can not be analysed using man-in-the-middle entities, thus rendering it impossible for networking-analysis tools to detect phishing URLs using the suggested methods which require the path. (2) Data without path - Some data sets, for example, CertStream data, do not contain the path. However, these data still needs to be analyzed in order to protect against phishing attempts, which the methods that have been suggested thus far are unable to do so properly. To the

best of our knowledge, there is no study that suggests a method to classify phishing URLs when the path is not included. As such, in many cases, there is a gap between the suggested methods' assumptions and "real world" conditions.

In our study, we will put the performance of those suggested models on doubt. We will show that although there are many studies that claim that their method can successfully classify phishing URLs using only the URL, this is not completely true in a "real world" setup. This will be done by implementing state-of-the-art methods and then evaluating them on data specifically meant to simulate "in the wild" data.

We will also propose a new solution that specifically targets this issue. By first training an auto-encoder to learn a reduced representation of a URL, we show that our proposed model performs much better than other competitive models on "real world" data.

The remainder of this paper is organized as follows: First we will review the state-of-the-art methods used for phishing-URL classification, including the architectures and data sets used. Then, we will implement some of these methods and compare their results on several published data sets. Next, we will evaluate these trained models on "in the wild" data and show that they are unable to perform such classifications accurately. We then introduce our solution in detail, providing the experimental results of our solution (obtained from both published data sets as well as "in the wild" data), as well as comparing them to those of the previously implemented and tested models. Finally, the last section summarizes and concludes the paper.

## II. Related Works

We split our literature review into two main parts. Firstly, we review state-of-the-art methods that have been suggested over the years to solve the phishing classification problem. Secondly, we review studies that describe the most used data sets. This is to gain a deeper understanding of the problems present within these data sets.

### A. Phishing Classification

There have been many approaches and methods suggested to detect phishing URLs. These can be divided into three main categories as follows: (1) Black/White-list methods, (2) Additional-data extraction methods, and (3) URL-based methods. This review focuses on the third approach as it is the most relevant for our research. In addition, we will present methods that are either the most cited, the most recent, or evaluated against a large scale data set since we think those are the most relevant to our study.

*1) Black/White-list methods:* In this category, the goal is to produce a database containing as many examples of malicious and benign URLs as possible, and then use those as a check to classify new URLs.

One of the most popular research in the blacklisting domain is PhishNet [20]. The authors applied five different augmentation methods on known phishing URLs in order to create new possibly-malicious URLs. This included replacing the URL's top-level-domain, replacing brand names, etc. Then, the resulting URL are validated, checked, and added to the blacklist accordingly.

Cao et al. presents a white-list-based method as well [9]. In their paper, the authors used a Naive Bayesian classifier to maintain a white-list based on the user's browsing habits. If a user enters his credentials in some non-white-listed site, he will be alerted and made known of possible phishing attempts.

*2) Additional-data extraction methods:* The next two methods are similar as they use features extracted from the URL and website in order to augment the classification model. However, this method differs from the next as it does not just rely on the raw URL string to collect data about the website. For example, it may choose to take a snapshot of a web page, and then, by analyzing the URL with this additional information, classify the suspect URL as phishing or benign. The main disadvantage of methods belonging in this category is that they are time-consuming, since connecting to the Internet often results in communication delays to and from the host site. Furthermore, some of the features obtained from this additional data may be computationally expensive to extract and use as well.

One of the most cited methods in this category is CANTINA [29]. This method analyzes the HTML web page corresponding to a URL to extract features for a linear classifier. Firstly, term frequency–inverse document frequency (TF-IDF) is calculated on the text present in the web page to find terms that are deemed the most important in the text. The top five most important terms, along with the domain name, are fed to Google's search engine and the results are collected. Other features are also collected, such as the age of the domain, number of dots in the URL, etc.

In a later paper, the authors also present CANTINA+, an improved, two-staged method to detect phishing URLs [27]. In the first stage, two filters are applied on the HTML web page. The first filter applies SHA1 hash function on the web page and compares it to a pool of values of known phishing web pages. The second filter looks for login forms on the web page. If the hash-based filter returns no matches but the login form filter detects a form, the algorithm moves to its second stage. In this stage, a Bayesian Network model is employed on features from CANTINA as well as new ones in order to perform the classification.

Li et al. also presents one of the more recent methods for detect phishing using HTML web page features [16]. They extract features such as IP address, presence of suspicious symbols, number of external links, Word2Vec string embeddings, etc. from both the URL string as well as the HTML web page. These features are then passed into a stacked model containing variations of decision trees in order to classify the URL as phishing or benign.

*3) URL-based methods:* Unlike the previous method, URL-based methods only extract features present within the raw URL string itself. This makes them a lot more efficient and faster than the previous approach as they do not suffer from networking delays or require specific tools to execute. Additionally, these methods do not sacrifice accuracy for the sake of performance. Some researchers have claimed that URL-based models are able to obtain the results comparable to methods that rely on extracting additional information from the website [26].

| Paper | Phishing Data set | Benign Data set | Num. of records (benign\malicious) | Download Link |
|---|---|---|---|---|
| [29] | PhishTank | Alexa and Yahoo! | 100\100 | |
| [27] | PhishTank | Alexa, 3Sharp and Yahoo! | 4,740\8,000 | |
| [16] | PhishTank | Alexa | 28,320\24,789 | |
| [22] | PhishTank | Yandex | 36,400\37,175 | [1] |
| [21] | PhishTank | Common-Crawl, Alexa and Yandex | - | [2] |
| [26] | PhishTank | Alexa | 245,023\245,385 | |
| [23] | VirusTotal | VirusTotal | 16,930,453\2,137,426 | |
| [5] | PhishTank | CommonCrawl | 1,000,000\1,000,000 | |
| [17] | PhishTank | CommonCrawl | 10,604\10,604 | |
| [28] | PhishTank and Reasnable Antiphishing | Alexa and Technical challenge | 584,909\573,964 | [3][4] |
| [2] | 420K-PD | 420K-PD | 344,821\75,643 | **??** |
| [11] | PhishTank and OpenPhish | Alexa and CommonCrawl | 5,000\5,000 | [5] |
| [24] | PhishTank. Huawaei Digital and APWG | Alexa, DMOZ and private dataset | 59,547\55,857 | |
| [19] | DMOZ repository | DMOZ | 5,000\5,000 | |
| [8] | UAB Phishing Data Mine | Cyveillance | 18,990\9,506 | |
| [1] | PhishTank | Millersmiles | - | |

Some shallow learning methods have been suggested thus far. Rao et al. compares several learning models, including XGBoost, Random Forest (RF), Logistic Regression (LR), k-Nearest-Neighbors (kNN), Support Vector Machines (SVM) and Decision Trees [21]. The features that have been extracted for the models include TF-IDF features, as well as structural features such as the length of the domain, the number of underscores in the host name, etc. Among them, RFs obtain the best results.

Random Forests have also been compared against other models in [22] and obtains similarly better results. In the paper above, two modules, a "word-decomposer" module and a random-word-detection module, are applied on the URL to recognize words and random characters present within the string. Other features are also extracted from the URL, including the average length of the recognized words, the number of random words, etc.

Compared to shallow learning models, deep learning models have achieved similar or better results in many different scenarios. The task of classifying between phishing and benign URLs is no different, and there have been many papers that propose several different deep learning architectures to solve this classification problem, often with improved results. These architectures usually begin by encoding the URL characters to some vector, for example using one-hot encoding, or other embedding techniques such as word2vec, bags of characters, etc. These vectors are then passed as inputs to the deep learning network for classification. A systematic literature review of the deep learning methods is presented in [7]. We review several examples of deep learning models below.

In [26] the authors present PDRCNN, a deep neural network architecture based on Bi-Directional Long-Short Term Memory (LSTM) units and 1-Dimensional Convolutional Neural Networks (1D-CNNs). A URL is first processed and embedded using word2vec to transform it into a 64-long vector. Nine other structural features are extracted from the URL, and concatenated with the URL vector as inputs to the network. It is shown that these nine structural features have no influence on the performance of the network. Furthermore, synchronous tests were also performed to show the robustness of the model. Additionally, it is noted that even without the URL vector, the nine structural features are shown to give better results than shallow learning models like RFs.

Another example of a deep neural network is presented in [23]. In this work, the authors build a network using an embedding layer, 1D-CNNs, and several fully connected layers, which represents a larger and deeper network compared to others in the field. The authors compare their model with two baseline models. The first baseline model uses 1-5 sized n-grams while the second one extracts structural features from the URL including URL length, number of dots, etc. These features are then passed to a hash function to create a 1024-length vector that will be used as inputs to the network. In either case, the main model that the author presents outperforms either of the two baselines.

In [5] the authors propose a neural network consisting of LSTMs and fully connected layers in order to classify phishing URLs. Their model is compared to RFs using a 3-fold cross validation technique and obtains better performances (i.e., AUC, Accuracy, Recall, Precision and F1 score).

In [17] the authors propose a model based on a 1D-CNN and compared it to the LSTM-based model proposed in [5]. It is shown that the 1D-CNN-based model outperforms the LSTM model.

Another model has been proposed where a URL is embedded by averaging character embedding values [28]. These embedding values are firstly obtained using techniques such as SkipGram and are applied on the raw URL string. XGBoost is then applied on the resultant vector in order to classify the URL.

### *B. Data Sets*

In this section, we will review some common data sets used across various research works, and identify several problems present within these data sources.

Alexa 1M is a list collected by Amazon that ranks the most popular sites on the Internet. As it is unlikely that the top-ranked sites would be malicious, this data set if often used as a source for benign URLs. However, there are other rankings which are an improvement over Alexa's. [18] compared four

different rank lists (Alexa 1M, Cisco Umbrella, Majestic, and Quantcast) in the contexts of: (1) similarity - how many URLs are common across different rank lists, (2) stability - how often the ranks of each URL changes, (3) representation - to what extent does the rating reflect the level of traffic the URL's network receives, (4) responsiveness - is there a response from the URL, and (5) benignity - how many malicious domains in the list. The authors found that each of the four rank lists that they compared were very different from each other. In particular, Alexa's rankings are not stable and changes a lot from day to day. This suggests that Alexa's ranking relies on a limited view of Internet activity, and can be easily exploited by injecting false data into the ranking. As an example, tk They have found that these four ranks are very different from each other. Alexa's ranking is not stable and changes a lot between different days. This means that Alexa ranking relies on a limited view of the Internet. They showed that Alexa can be easily exploited by injecting false data into the ranking. As an example of such an attack, the authors of [14] show the feasibility of infiltrating domains into Alex'a top 100k domains with little effort. This suggests that using Alexa as a source for benign URLs may not be entirely accurate without additional verification.

Phishtank, a website where users can report phishing URLs, contains a large amount of malicious URLs dating back from 2009. As such, it is often used a data source examples of phishing URLs. However, this may not be the most updated list of phishing URLs. [6] compared three phishing blacklists (Google Safe Browsing, OpenPhish, and PhishTank) and found that Google's Safe Browsing blacklist was updated 17 times more than PhishTank and OpenPhish, implying that PhishTank may not contain as many updated and still-in-use phishing URLs as initially presented.

Some research works use data sources that are self-collated and edited and as such, it can be difficult to properly compare the suggested methods and results accordingly. A public, balanced data set is proposed in [10] for benchmarking purposes. However, this data set is limited in the fact that it only contains 30,000 URLs, and is relatively small for much work to be done on it.

## III. Background

A URL (or Uniform Resource Locator) is a human-readable text that describes the location of a desired web page/file on the Internet. In general, the URL has three main parts, presented also in Fig 1.

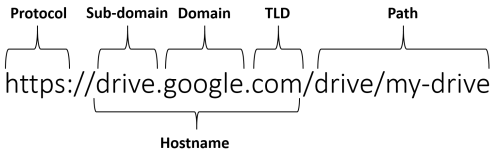


Fig. 1. URL Structure

The first part represents the protocol used, usually either HTTP or HTTPS. The second part is the hostname, which is composed of the top-level domain (TLD), the domain, and, optionally, sub-domains. The hostname describes the site name, resolved by the DNS server from the IP address of the server that holds the web page or file. The last part of the URL is the path, which describes the location of the web page/file on the host server.

TABLE I. Obfuscation types.

| Obfuscation type | Example |
|---|---|
| Benign url | legit.legitimatesite.com |
| Interchanging | legitimatesite.legit.com |
| Top-level domain | legit.legitimatesite.com.my |
| Character substitution | legit.iegitimatesite.com |
| Add subdomain | legit.legitimatesite.anothersite.com |

Phishing is a well-known fraudulent method aiming to impersonate a legitimate URL through social engineering. An attacker sends a message through email, social networks, etc., which contains, amongst other things, a malicious URL which mimics a benign one. Unsuspecting users that click on this malicious URL would be routed to a harmful web page or file, where attackers can perform cyber-attacks such as stealing login credentials, or exploiting vulnerabilities to install malicious software on the users' devices [12].

In some cases, the hostname may be a benign entity but the path may be malicious. This is mainly done by cross-site scripting (XSS) attacks. In these cases, attackers try to inject a malicious file into the benign URL. As an example, consider the following scenario: sites.google.com/site/informationfb56003/. While sites.google.com is a benign website, the attacker is using the website to host and have victims download malicious files.

In many other cases, phishing occurs in the hostname. In order to mimic a benign URL, phishing URLs disguise themselves similarly to the original URL as much as possible. This is usually achieved using URL obfuscation. Several types of URL obfuscation methods are described in Table I.

### A. HTTPS Protocol

With the development of the Internet, and the need for better and more secure protocols, there is a growing use of the Hypertext Transfer Protocol Secure (HTTPS) protocol.

HTTPS is similar to the unsecure HTTP protocol, except that almost all network traffic is encrypted under HTTPS. The only parts that are left exposed are the destination address and destination port, which is required by the networking components for successful routing of the packet to its desired destination. Other parts of the packet, such as the URL's path, are not needed for successful routing, thus HTTPS encrypts that as well.

Using HTTPS as a characteristic for the differentiation between benign and malicious URLs had some use in the past, since malicious URLs could not obtain the necessary certifications to use the HTTPS protocol. However, this is no longer the case. Today, more and more attackers are using the HTTPS protocol for their websites, and the APWG has reported that almost three-quarters of phishing URLs now use HTTPS [4]. This means that HTTPS is no longer a good defining characteristic of benign versus malicious URLs.

### B. Special Data Sources

In many scenarios, there is a need for a model that is able to generalise on any type of URLs, including those with and without the path. For example, CertStream is a data source that contains URLs that should be analyzed, in order for early detection of phishing sites. CertStream gives real-time updates of URLs whenever they apply and obtain a new security certificate. As mentioned earlier, since more and more phishing sites tend to use the HTTPS protocol, they would also appear in this data source, and hence there is a need to detect such entries as they appear. However, in this scenario, these URLs do not contain the path since the certificate is mainly for the domain alone is not path-specific.

To the best of our knowledge, there is no current research that evaluates the performance of phishing-classification models on URLs where the path is not included. As such, this represents an open gap in the research of phishing detection.

### C. NLP Methods

Natural language processing (NLP) is a sub-field in computer science, dealing with how a computer can process natural language. As URLs consist of sequences of characters usually organized in a linguistic form, a lot of recent phishing detection models use NLP techniques in order to analyze the linguistic characteristics of the URL. In many cases, phishing URLs tend to have certain linguistic characteristics that differentiates them from benign URLs. Hence, using NLP methods appears to be a straightforward way to extract information about such characteristics, and this is shown to perform relatively well on tested data.

In this subsection, we present some of the key principles and methods used in the NLP domain.

The first principle is the encoding of the alpha-numeric characters to something that is machine-usable. An obvious solution, using the ASCII values of characters as inputs, is not feasible as it there is no correlation between different ASCII values and the roles of each character in a sentence. One of the common ways to "solve" this issue is to use one-hot encoding, which converts every character to a vector that is as long as the size of the whole alphabet. This vector has a value of one in one place, corresponding to the actual character, and zeroes in the rest, so each unique character has a unique vector representation. Using this approach ensures that there is no inherent correlation between ASCII values and removes any bias that might form.

A more sophisticated method to represent characters is to use an embedding layer. In order to build this representation, a sequence is first represented as a sequence of one-hot-encoded vectors or numerical labels, and then passed to a specialised neural network that seeks to drastically reduce the input dimensions to a more usable number. The final representation is a non-human-readable nor machine-decodable vector, but it benefits from being able to be used as inputs to various neural networks. There are some ready-made embedding representations, such as word2vec, bags of words, etc.

LSTMs are a well-known deep learning module that is often used for NLP purposes. Based on Recurrent Neural Networks (RNNs), LSTMs improve upon the RNN's inability to detect long term sequential dependencies and are suited for use on sequential data. Bi-directional LSTMs consist of two LSTM cells where the both the input sequence and its reverse are used to train the model. This allows the model to better learn sequential dependencies and additional context present within the input.

CNNs are also used very often in NLP models. Although it originally saw much use in the computer vision domain, it is also used to great effect in the NLP domain. Often, while computer vision makes use of 2-dimensional (or even higher) CNNs that pass over the 2D image, CNNs used in NLP are often 1-dimensional networks that pass over the one-dimensional sequence of characters. One-dimensional filters with varying sizes are applied on the input sequence, in the hopes that the network will be able to learn and recognise sequential patterns.

Last but not least, fully connected layers are layers of neurons that connect every input node to every output node. These layers are often not too useful on their own, but are often combined with LSTM or CNN cells to allow the model to reach a decision on the classification.

### D. "In The Wild" Data

"In The Wild" data refers to data that are usually not found in common data sets such as Alexa 1M or PhishTank, and is meant to be a term that represents data that occurs under "real world" conditions. A lot of research makes claim to significant classification results on their individual tested data sets, but this may not be true on "in the wild" data. These test data sets are limited in varied ways, which would affect the models' performances when they are used on non-controlled, "real-world" data. In this section, we go through some of the limitations of these data sets.

*1) Data sources:* Many research works use the Alexa 1M data set as a source for benign URLs. However, Alexa mostly contains URLs which are short and simple with few subdomains. Hence, this is a very specific data source which only describes a subset of all benign Internet traffic. Models trained using this data set often recognise long URLs as phishing attempts, but in many cases, such a characteristic does not necessarily equate to a malicious attack. These long and complex benign URLs are more common than similar phishing URLs, so training a model with a balanced data set that contains both types of URL is likely to cause high false positive rates under "real-world" conditions.

PhishTank is another popular data source, but for phishing or malicious URLs instead, Although it is undoubtedly a large data set that contains a wide variety of phishing URLs, in many cases, some of these URLs have the same or similar domain names. Studies that fail to notice this issue, and do not choose to pre-process and clean the data set beforehand, may result in data leaking from the training set to the test set, which induces biases in the trained model.

*2) Artificial differences:* In some cases, we found differences between benign and phishing URLs which are artificial and do not arise as a result from the sequence of characters within the URL itself. The following are some examples of these artificial differences: (1) Benign URLs do not contain

the path, while phishing URLs do. (2) URLs which use the HTTPS protocol are more likely to be benign than phishing (which is not too true these days as explained above). These artificial differences do not translate well into "real world" conditions, hence models which learn these differences (either purposefully or as a result of pattern-learning) will not work as well on "in the wild" data.

*3) Small data sets:* Some research, especially the older ones, use relatively smaller data sets that are easily fitted. These data sets are likely to cause inherent biases to appear within the trained model, due to size of the data. However, "in the wild" data may contain a larger variety of URLs that are not present at all in the training and testing data set. In this case, the results of testing using the smaller test data sets are not transferable to "real world" data.

*4) Non-language oriented:* Within the common data sets, most URLs are usually in one language, even when comparing between benign and phishing URLs. Thus, this raises questions about URLs that contain words from other languages, and how the presence of such data in the real world might affect performance. However, the vast majority of proposed models do not really take into account this language difference when analyzing and extracting features from a URL.

## IV. Model Implementation

To evaluate the results of various state-of-the-art architectures on "in-the-wild" data, we have devised the following methodology. First we will select high performing architectures, that are highly cited or very recent. Then, following the papers, we implement these architectures and validate their effectiveness on the same data sets as presented in the papers. We will then perform some processing of the data sets, dropping the path from each URL (if it exists) and re-evaluate the performance of the models on this edited data set. Finally, we evaluate the models on large-scale unlabelled data sets meant to simulate "in the wild" data. The predictions of the models are then verified and the results are collated.

### A. Implemented Papers

For our research, we investigated the following papers:

*1) Classifying Phishing URLs Using Recurrent Neural Networks (CPUURNN) [5]:* This is a paper published in APWG Symposium on Electronic Crime Research, 2017. In this paper, the authors present a deep learning network architecture. A one-hot-encoding is calculated on the URL string, and passed into an embedding layer to create an 128-dimensional embedding vector. The embedding vector is then passed into a LSTM layer with a 150-step sequence, and the output is connected to a fully connected layer with a Sigmoid activation function. The network was trained to minimize binary cross-entropy with additional dropout at the fully connected layer to avoid over-fitting. It was then evaluated using a three-fold cross validation technique and to obtain an average AUC of more than 0.999. However, the authors of this paper did not discuss any pre-processing that was done on their data sets.

We chose this paper for implementation and evaluation purposes as this paper is highly cited compared to its publication date, and the data sets used contain a large number of URLs (over 2 million entries), which, when coupled with the model's performance, could give a good indication of the accuracy and efficacy of the architecture proposed.

For this paper, we implemented this model following the designs presented in the paper.

*2) eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys [23]:* This is a paper from 2017. In this paper, the authors propose a different type of deep learning architecture to classify phishing URLs based on character-level CNNs. The paper first defines a character set consisting of 87 unique URL-valid characters. Then, during the pre-processing phase, the length of each URL is set to be 200 characters long (with appropriate truncation and padding if necessary), and any characters present in a URL that doesn't exist within the character set is replaced with an <unknown> token.

The architecture of the model first uses an embedding layer to create a 32-dimensional embedding vector. This results in a $200\times32$ matrix that represents each URL. Four CNN filters of sizes 2, 3, 4, and 5 each with 256 kernels are applied on the matrixm, and the results are concatenated to form a 1024-long vector. This is then passed to three fully connected layers each with 1024 units and finally to the last layer consisting of a single neuron with a Sigmoid activation function. The network was trained to minimize binary cross-entropy with additional dropout and layer normalization units added to avoid over-fitting and speed up network training.

We chose this paper for implementation and evaluation purposes as this paper is highly cited compared to its publication date. Furthermore, the paper also used a large data set collected from VirusTotal, which is different compared to many other papers which obtained their data from Alexa 1M and PhishTank. We also note that this paper obtained very good results with the model's architecture.

For this paper, we implemented the model following the designs presented in the paper.

*3) PDRCNN: Precise Phishing Detection with Recurrent Convolutional Neural Networks [26]:* This is a journal paper from October 2019. In this paper, the authors suggest a deep learning network architecture based on CNNs to classify phishing URLs. Firstly, they define a character set that contains 59 unique URL-valid characters. Then, during the pre-processing phase, the length of each URL is set to be 255 characters long (with appropriate truncation and padding if necessary), and any characters present in a URL that doesn't exist within the pre-defined character set is replaced with an <unknown> token.

The model uses a pre-trained word2vec embedding layer to replace each character in the URL with a 64-dimensional embedding vector, resulting in a $255\times64$ matrix representation of each URL. Then, it is faded into a bi-directional LSTM with 64 units in the hidden layer. The outputs of the forward and backward pass of the LSTM are concatenated together to form a $255\times128$ matrix. Three differently-sized CNN filters ($5\times128$, $6\times128$, $7\times128$), each with 32 convolution kernels are applied on the matrix. A max pooling layer is applied on the result, so each filter reduces the matrix to a 32-dimensional vector, which are concatenated together, and connected to a

fully connected layer with a Sigmoid activation function. The network is trained with a batch size of 2048 to minimize binary cross-entropy with a dropout rate of 0.9 at the outputs of each CNN layer to reduce over-fitting. The Adam optimizer was also used for faster convergence. The model was evaluated using 10-fold cross-validation, obtaining an average AUC of more than 0.99.

We chose this paper for implementation and evaluation purposes as this paper is very recent work, and the authors used a large data set to test and evaluate their model. Furthermore, the ideas of the two models above (LSTMs and CNNs) are combined in this model. The authors had also performed a synchronous test to prove that their model is robust and can perform well in any given time period.

We received the model implementation from the paper's authors.

### B. Our Models

As we could not, to the best of our knowledge, find any research that suggest models specifically aimed at classifying URLs when the path is not included, we decided to create two different models for comparison. Even though they were designed for URLs without the path, these models still achieve competitive results when the path is included.

*1) XGBoost model:* XGBoost (Extreme Gradient Boosting) is an algorithm that is made up of an ensemble of decision trees. This algorithm achieves state-of-the-art results in many tasks due to two main concepts: (1) Boosting - a technique in which the algorithm is built from a number of weak learners, so in each iteration, a new learner is added based on the error of the algorithm thus far. (2) Gradient Descent - every added tree is added in a way such that the algorithm steps in the direction of the loss function gradient.

We chose to implement an XGBoost-based algorithm as, in contrast to the other models presented above, XGBoost is not a neural network. Furthermore, out of the many algorithms that don't rely on neural networks, XGBoost is known as one of the most robust and performant algorithms.

As inputs to our model, we extracted both structural and linguistics features from the URL. The linguistics features were extracted using a Markov Chain (MC) model. MC is a probabilistic model in which the probability of each event depends wholly on the previously-occurring event. In our context, each event is a sequence of letters, so the MC model holds the probability to switch from one sequence of letters to the next. We trained our model on data sets containing only benign URLs.

We trained five different MC models: (1) Alexa domain uni-gram - we broke up each domain name present in the Alexa 1M data set into letters and calculated the probability of switching between them. (2) Alexa URL uni-gram - we broke up each URL in the Alexa 1M data set into letters and calculated the probability of switching between them. (3) Alexa URL parts - we broke up the URLs in Alexa into different parts, i.e. top-level domain, domain, sub-domains, etc. and calculated the probability of switching between them. (4) Alexa URL bi-gram - we broke up each URL in the Alexa 1M data set into bi-grams and calculated the probability of switching between them. (5) DNS URL bi-gram - we broke up each URL in a DNS data set that we had in our lab into bi-grams and calculated the probability of switching between them. The full list of features that we have been extracted is presented in Table II.

*2) LSTM model:* We also built a deep bi-directional LSTM-based neural network since LSTMs are useful for analyzing sequential patterns.

The architecture of the network is as follows. First, we use a pre-trained word2vec embedding layer to replace each character with an embedding vector. This representation is then passed to a network consisting of three LSTM units, and a fully connected layer with two output neurons. The first two bi-directional LSTM units contains 200 hidden units, and only pass the final output to the rest of the network. The last bi-directional LSTM unit consists of 128 hidden units, and the results of each iteration are passed to the last fully connected layer. The final layer uses Softmax activation, and the whole model was trained using an input-dropout rate of 0.3, and a recurrent-dropout rate of 0.3 for each LSTM cell. The Adam optimizer was used to train the network to minimize binary-categorized cross-entropy.

### C. Evaluated Data Sets

To evaluate each model, we used five different data sets as described below (also in Table III).

*1) Benchmark data set:* This data set was taken from the paper [10]. The authors of the paper attempted to create a balanced benchmark data set which can be used to compare different anti-phishing methods. The phishing URLs in the data set were taken from PhishTank under the category of "valid phishes" and "online", meaning that the URLs point to websites that are were and verifiably malicious at the time the data was collected. Out of 15,000 benign URLs, 14,500 of them were taken from Alexa 1M, and the rest were taken from DMOZ and BOTW. This meant that the data set contained a mix of high popularity URLs (from Alexa 1M) and low popularity ones (from DMOZ and BOTW). Out of the 30,000 URLs published in this data set, 28,046 of them were live. Most of the benign URLs in this data set do not contain the path, as URLs taken from the Alexa 1M data set do not contain the path.

*2) 1M-PD data set:* This data set was taken from the paper [28]. This data set contains more than 1 million URLs, of which 587,668 are phishing and the remaining 584,909 benign. The source of the phishing URLs were from PhishTank and Reasonable Antiphishing. The Reasonable Antiphishing data source contained URLs taken from 2008 and 2015, and the data source itself does not exist anymore. The benign URLs were taken from a mix of Alexa 1M on 7 November, 2017, and from a network security technical challenge. Most of the benign URLs in this data set do not contain the path, as URLs taken from the Alexa 1M data set do not contain the path.

*3) CPUURNN data set:* This data set was taken from [5]. This data set contains more than 2 million URLs. 1,146,432 phishing URLs that were taken from PhishTank and another 1,200,000 URLs in general were taken from CommonCrawl. This data set is not public and we obtained it with permission from the paper's authors.

TABLE II. EXTRACTED FEATURES.

| Feature | Description | Component |
|---|---|---|
| is idn | Does the URL begin with "xn–" | Hostname |
| level | Number of dots + 1 | Hostname |
| length | | Hostname, Domain |
| Consonant count | Number of consonant letters | Domain, Sub-domain |
| Digit as letter count | Number of (0,1,2,5) at the URL. | Domain |
| Digit count | Number of digits at the URL. | Domain, Sub-domain |
| Digit ratio | The ratio between digits to URL length | Domain, Sub-domain |
| Hyphens count | Number of hyphens at the URL | Domain, Sub-domain |
| Hyphens ratio | The ratio between hyphens to URL length | Domain, Sub-domain |
| Vowels count | Number of vowels (a, e, i, o, u) at the URL | Domain, Sub-domain |
| Vowels ratio | The ratio between vowels to URL length | Domain, Sub-domain |
| Entropy | Let $p_s$ be the ratio of symbol $s$ at a given URL, the domain entropy is calculated by: $H(URL) = -\sum p_s * log_2(p_s)$ | Domain |
| MC Alexa unigram probabilities multiplication | Multiplication of all the probabilities associated we the specific URL. | Domain, Sub-domain, Hostname parts |
| MC Alexa unigram probabilities mean | The average value of all the probabilities associated we the specific URL. | Domain, Sub-domain, Hostname parts |
| MC Alexa unigram probabilities standard deviation | The standard deviation of all the probabilities associated we the specific URL. | Domain, Sub-domain, Hostname parts |
| MC Alexa bi-gram probabilities multiplication | Multiplication of all the probabilities associated we the specific URL. | Hostname |
| MC DNS bi-gram probabilities multiplication | Multiplication of all the probabilities associated we the specific URL. | Hostname |
| Word probability | Sum of probabilities of suspicious words to appear in a phishing URL. | Hostname |
| Tld probability | The probability of the TLD to appear in a suspicious URL. | TLD |

TABLE III. DATA SETS DESCRIPTION.

| Data set | Phishing | | Benign | |
|---|---|---|---|---|
| | Num | Source | Num | Source |
| **Benchmark** | 14,745 | PhishTank | 13,301 | Alexa<br>DMOZ<br>BOTW |
| **1M-PD** | 573,975 | PhishTank<br>R. Antiphishing | 587,909 | Alexa<br>Technical Challenge |
| **CPUURNN** | 1,146,432 | Phishtank | 1,200,000 | CommonCrawl |
| **PDRCNN** | 245,385 | Phishtank | 245,023 | Alexa |
| **Our data set** | 647,393 | PhishTank<br>R. Antiphishing<br>OpenPhish | 593,447 | Alexa |

*4) PDRCNN data set:* This data set was taken from the paper [26]. The data set contains approximately 500,000 URLs, of which 245,385 are phishing URLs, collected from PhishTank from the period of August 2006 to March 2018, and the remaining 245,023 URLs benign. In order to create the set of benign URLs, the authors first used search engines to search for the domain names that appeared in the Alexa 1M data set. Then, they collected the top 10 results for each search, and filtered out URLs that were offline. This data set is not public and we obtained it with permission from the paper's authors.

*5) Our data set:* We also created our own data set using the following method. We took 11 instances of Alexa's ranking from 2010 to 2020. If the URL appears in the top 500K at least three times then it is labelled as a benign URL. For phishing URLs we used PhishTank (PT), OpenPhish (OP) and Reasonable Antiphishing (RA). The paths are dropped from each URL. However, since some phishing URLs use XSS attacks to inject phishing content into a benign domain, dropping the path results in URLs which are labelled as phishing but are actually benign. In this case, we drop the URLs labelled phishing but whose domains appear in Alexa 1M.

## V. EXPERIMENTAL RESULTS

### A. Labelled Data Sets

In this section, we present the results of the implemented models (as described above) on the labelled data sets.

*1) URLs with path:* In order to ensure that our implementation of the various models are correct, we compare our implemented models' performances on the same data sets as mentioned in the papers. We want to make sure that we obtain the same or very similar results as presented in the literature. As the evaluation of the models was done on URLs that include the path, for a proper comparison, we have to evaluate our models on the same type of URLs. As described in IV-C, only two data sets (CPUURNN data set and PDRCNN data set) contain benign URLs that include the path, so we evaluated our models models were evaluated For both data sets, we compare the results of the corresponding papers and to the results that we obtained with our own implementations.

Data Set 1: CPUURNN data set For this data set, we obtained the data set that was presented in the paper from the authors, and implemented the described model by ourselves.

TABLE IV. URLs WITH PATH RESULTS.

| Model and Dataset | Reported Result | Accuracy | Precision | Recall | F1 | AUC |
|---|---|---|---|---|---|---|
| **CPUURNN** [5] | Paper | 0.986 | 0.986 | 0.986 | 0.986 | 0.999 |
| | Our implementation | 0.987 | 0.086 | 0.989 | 0.987 | 0.999 |
| **PDRCNN** [26] | Paper | 0.945 | 0.966 | 0.920 | 0.943 | 0.985 |
| | Our implementation | 0.956 | 0.973 | 0.937 | 0.955 | 0.989 |

Data Set 2: PDRCNN data set With respect to this data set, we obtained both the data set and the model's implementation from paper's authors.

Each of the data sets was split into an 8:1:1 training, validation, testing split. The validation set was also used for early stopping when training the model.

The results are presented in Table IV. As seen, we obtained very similar results to those presented in the paper. This suggests that our implementation matches that of the papers'.

*2) URLs without path:* As described in III, URLs without the path should also be a focus in our analysis in order to keep up with newer phishing attempts. As such, we first perform some pre-processing of the various data sets described above to remove the path (when it exists) from each URL. In order to account for potential XSS attacks (after removing the path), we remove URLs which appear in Alexa 1M. The results of each of the models are presented in Table V.

From the table, we can conclude that the benchmark data set is too small to be fitted by neural network-based models. Also, in general, the models are able to successfully fit to the data sets which do not contain the path, albeit with a slight drop in performance. Our self-collated data set appears to be the hardest data set for the models to fit to. However, the XGBoost model seems to outperform all other models on the various data sets.

### B. Unlabelled Data Sets

We discussed in III-D the potential inability for models which obtain great results on tested data sets to generalise to "in the wild" data. In this section, we present two unlabelled data sets that are meant to simulate "in the wild" data. Then, we present our evaluation and verification method as well as the results of running the implemented models on the unlabelled data sets.

*1) Description:* The unlabelled data sets that we have obtained and used are as follows:

CertStream data - CertStream is an intelligence feed that gives updates from the Certificate Transparency Log network, which tracks the issuing of certificates to websites in real time. We recorded CertStream data for eight days from 18 to 24 July 2019 (86,660,000 instances in total). Since this is a large amount of data, it will be difficult to evaluate our models on the entire data set. Hence, we randomly selected 0.02% of the data set in a uniformly distributed manner. We ended up with slightly more than 170,000 instances, uniformly distributed between each day.

As discussed in III, more and more phishing URLs tend to use the HTTPS protocol, and hence would need to register their certificate, which would be tracked by the Certificate Transparency Log network. Thus, there would be several instances of phishing URLs present within this data set. Therefore, we can use this data set to evaluate the performance of our implemented models.

Proxy data - We obtained, from a large organization, a data set that contains approximately 2 billion URL requests from the 120,000 users in the organization. The requests was recorded over a period of two weeks, from 17 July to 2 August, 2019. For each URL, we removed the path (which is mostly encrypted), and dropped duplicates, ending with slightly more than 500,000 unique entries.

For any large organization, the proxy-level requests is a good place to consider integrating a phishing detection model. Hence, it is a good idea to evaluate the performances of the various models on this data set, as this will likely be a common use-case of said models.

*2) Evaluation methods:* Precision@k is an evaluation metric that measures the number of relevant instances at the top k results of a model. In many scenarios, where a model may be suffering from a high false positive rate, this metric can be considered instead to select instances of predictions where the model is highly confident. This metric also makes it easy for experts to evaluate and verify since it is only required to check the top k results (instead of the entire data set).

In our evaluation, we choose to use the metric, assuming that if a model achieves low Precision@k results, then it will likewise achieve even lower results in other metrics (i.e. AUC, accuracy, etc.). In order to verify whether our model has predicted the label of a URL correctly, we use VirusTotal, a well-known service owned by Google that uses various anti-virus agents to analyze files and URLs. We used the readily available API to automatically check if the URLs that were classified by the models as phishing are actually phishing URLs.

Our validation method is based on the assumption that because our unlabelled data sets are relatively old (July 2019), any phishing URLs that exist within the data set would have already been reported by at least one of the anti-virus agents on VirusTotal. Furthermore, if there is a probability that the above assumption does not hold, then this probability is not larger than a certain small threshold.

*3) Results:* For our evaluation, we performed the following steps.

1) We train each of the five presented models as described above on each of the 5 presented data sets. We thus end up with 25 trained models.
2) Each of these 25 train models are then used to obtain predictions for the CertStream and Proxy data sets. We obtain 50 different sets of results.
3) For each set of results, we calculate the Precision@100 by taking the top k=100 highest scored

TABLE V. URLs WITHOUT PATH RESULTS.

| Data set | Model | Accuracy | Precision | Recall | F1 | AUC |
|---|---|---|---|---|---|---|
| **1M-PD [28]** | CPUURNN Model [5] | 0.780 | 0.806 | 0.730 | 0.767 | 0.863 |
| | CNN Model [23] | 0.770 | 0.775 | 0.753 | 0.764 | 0.857 |
| | LSTM Model | 0.769 | 0.864 | 0.633 | 0.731 | 0.852 |
| | XGBoost | 0.857 | 0.828 | 0.897 | 0.861 | 0.943 |
| | PDRCNN Model [26] | 0.761 | 0.884 | 0.594 | 0.711 | 0.848 |
| **benchmark_dataset [10]** | CPUURNN Model | 0.501 | 1 | 0.003 | 0.007 | 0.747 |
| | CNN Model | 0.610 | 0.564 | 0.973 | 0.714 | 0.811 |
| | LSTM Model | 0.717 | 0.771 | 0.618 | 0.686 | 0.788 |
| | XGBoost | 0.923 | 0.908 | 0.942 | 0.925 | 0.963 |
| | PDRCNN Model | 0.740 | 0.758 | 0.706 | 0.731 | 0.827 |
| **CPUURNN dataset [5]** | CPUURNN Model | 0.752 | 0.800 | 0.650 | 0.717 | 0.848 |
| | CNN Model | 0.748 | 0.828 | 0.605 | 0.699 | 0.849 |
| | LSTM Model | 0.746 | 0.768 | 0.682 | 0.722 | 0.843 |
| | XGBoost | 0.765 | 0.747 | 0.777 | 0.762 | 0.871 |
| | PDRCNN Model | 0.738 | 0.822 | 0.587 | 0.685 | 0.839 |
| **our_dataset** | CPUURNN Model | 0.768 | 0.803 | 0.696 | 0.745 | 0.852 |
| | CNN Model | 0.766 | 0.827 | 0.657 | 0.732 | 0.851 |
| | LSTM Model | 0.759 | 0.834 | 0.631 | 0.719 | 0.845 |
| | XGBoost | 0.812 | 0.789 | 0.838 | 0.813 | 0.912 |
| | PDRCNN Model | 0.745 | 0.795 | 0.642 | 0.711 | 0.827 |
| **PDRCNN [26]** | CPUURNN Model | 0.677 | 0.668 | 0.693 | 0.680 | 0.769 |
| | CNN Model | 0.678 | 0.626 | 0.867 | 0.727 | 0.784 |
| | LSTM Model | 0.670 | 0.645 | 0.743 | 0.691 | 0.760 |
| | XGBoost | 0.735 | 0.708 | 0.791 | 0.747 | 0.834 |
| | PDRCNN Model | 0.689 | 0.684 | 0.693 | 0.688 | 0.781 |

URLs and checked each of the URLs using the VirusTotal API.

The top 100 URLs were selected in such a way that duplicated domain names were not selected more than once. The results are presented in Table VI.

As can be seen, the results have dramatically decreased from that in Table V. The models that had excellent performance on the test data sets were unable to generalise well on data meant to simulate real-world conditions. Therefore, there is a gap between what these phishing detection models have claimed to be able to do, and the results they actually produce.

## VI. SOLUTION

In this section, we propose a new solution that specifically targets this issue. We show that our solution not only achieves competitive results on the same test data sets, but is also able to generalise well on the data sets meant to simulate real-world conditions.

Based on the idea of using "in the wild" data as a stepping stone, our model first uses the unlabelled URLs to learn a representation of a URL, which is then passed as inputs to the classification layers.

To the best of our knowledge, the idea of using "in the wild" data as a learning step to classify phishing URLs has not been done nor researched upon before.

### A. Design

Our solution comprises two different neural networks.

The first network is an auto-encoder that is trained on the unlabelled data set in order to learn an unsupervised representation of a URL. Auto-encoders are a type of neural network that is able to learn efficient data-codings by learning how to ignore the noise present in the input [15], [25]. Usually comprised of two parts, the first part, the encoding layers, seek to reduce an input vector into a smaller representation, while the second part, the decoding layers, attempt to reconstruct the original input vector from the reduced representation.

We aim to use auto-encoders to learn a "summarised" representation of a URL. As the training data is unlabelled, there will be no bias towards either phishing or benign URLs. Instead, we will use the second neural network, the classifier (described below) to detect different patterns in these representations that are otherwise not readily apparent in the original string.

The second network is a classification model made up of LSTM modules. However, instead of using the original URL as inputs, it instead takes as inputs the compressed representation of a URL as learned by the auto-encoder.

Through the transfer of learning between an unsupervised model to a supervised one, we show later that this model is able to outperform the other models on "in the wild" data.

Below, we describe in detail our model and pre-processing steps.

Firstly, we set all URLs to have a length of 38 characters. URLs longer than that are truncated while shorter URLs are padding with a <padding> token. Then, we extract character-level trigrams from the URLs. We define a "trigram-alphabet", where we take the top 3200 most common trigrams present in the entire data set (which accounts for 75% of all trigrams), and use it to encode a URL as a sequence of trigram labels. Any trigrams not present within the alphabet is denoted using

TABLE VI. UNLABELLED DATA SET WITHOUT DUPLICATES RESULTS.

| Target data set | Trained data set | Model | | | | |
|---|---|---|---|---|---|---|
| | | **CPUURNN** | **CNNModel** | **LSTMModel** | **PDRCNN** | **XGBoost** |
| **CertStream** | 1M-PD | 0.02 | 0.23 | 0.18 | 0.23 | 0.17 |
| | benchmark_dataset | 0.19 | 0.19 | 0.7 | 0.12 | 0 |
| | CPUURNN | 0.13 | 0.13 | 0.12 | 0.21 | 0.12 |
| | our_dataset | 0.09 | 0.11 | 0.22 | 0.9 | 0.35 |
| | PDRCNN | 0.04 | 0.12 | 0.05 | 0.31 | 0.04 |
| **Proxy** | 1M-PD | 0.02 | 0.02 | 0.07 | 0.08 | 0.02 |
| | benchmark_dataset | 0.01 | 0.01 | 0 | 0.01 | 0.04 |
| | CPUURNN | 0.04 | 0 | 0.07 | 0.07 | 0.05 |
| | our_dataset | 0.01 | 0 | 0.05 | 0.04 | 0.1 |
| | PDRCNN | 0.06 | 0.01 | 0.09 | 0.11 | 0.04 |
| **DNS1** | 1M-PD | 0.04 | 0.01 | 0.04 | 0.07 | 0.07 |
| | benchmark_dataset | 0.02 | 0.02 | 0.05 | 0.03 | 0.02 |
| | CPUURNN | 0.07 | 0.02 | 0.01 | 0 | 0.02 |
| | our_dataset | 0.04 | 0.08 | 0.06 | 0.08 | 0.04 |
| | PDRCNN | 0.03 | 0.05 | 0.04 | 0.05 | 0.02 |
| **DNS2** | 1M-PD | 0.11 | 0.08 | 0.23 | 0.27 | 0.09 |
| | benchmark_dataset | 0.01 | 0.02 | 0.01 | 0.02 | 0.03 |
| | CPUURNN | 0.05 | 0.06 | 0.18 | 0.17 | 0.12 |
| | our_dataset | 0.01 | 0.03 | 0.16 | 0.09 | 0.13 |
| | PDRCNN | 0.06 | 0.06 | 0.06 | 0.17 | 0.09 |

an <unknown> token. As such, including <padding> and <unknown> tokens, there are 3202 unique trigrams in our "trigram-alphabet". As each URL is exactly 38 characters long, we are able to represent each URL as a sequence of 36 trigrams.

Our auto-encoder follows an asymmetric model, with the encoding and decoding layers differing from one another.

The encoding layers comprise an embedding layer which transform numerical-labelled vectors into 10-dimensional embedding vectors. This results in each URL being represented as a $36\times10$ matrix. The matrix is then passed as a 2D vector into a unidirectional LSTM module with 4 hidden nodes. The output of this LSTM module, a $36\times4$ matrix is then taken to be the "compressed" representation of a URL.

The decoding layers take this "compressed" representation and attempt to reconstruct the original vector. This is done through three fully connected layers of sizes 144, 72, and 36 respectively.

The auto-encoder was trained on the unlabelled data set with the aim to minimize mean-squared error between the original and the reconstructed vectors. The Adam optimizer with a learning rate of 0.001 was used to perform the back-propagation.

The classification layers take as input the "compressed" representation of the encoding layers mentioned above. It consists of three fully connected layers of sizes 144, 64 and 2 nodes.

In order to train the classification layers, the auto-encoder was trained first using the methods described above. Then, the encoding layers (with its trained weights) were frozen and the "compressed" outputs rerouted to the classification layers as previously described. This new model was trained on the labelled data set with the aim to minimize binary cross-entropy with additional dropout units inserted in the linear layers to reduce over-fitting. The Adam optimizer was used with the learning rate set to 0.001 to perform the back-propagation.

### B. Results

The auto-encoder was first trained on the unlabelled data sets meant to simulate "real-world" data. In our case, this was the CertStream data set. Then, for validation, a five-fold cross-validation split was performed. Each model (using the same auto-encoder each time), was trained on five different splits of the CPUURNN data set, and the validation results were collected and averaged. The validation results on the labelled data set, along with the other competitive models, are presented in Table VII.

For evaluation, we again calculated the Precision@100 metric for each model, and averaged the results. The final results are presented in Table VIII, with our model's results bolded.

We can see that our model offers a significant improvement in terms of correctly classifying "real-world" URLs. Although our model does not produce exceptional "in-lab" results, we stress that these results *do not matter* as we are focusing on the results produced on "real-world" data. In that aspect, we are confident that our solution has merit, and may be useful in exploring new ways of utilising unlabelled "in the wild" data to augment previous phishing-URL-classification models.

## VII. CONCLUSION

In this paper, we evaluated the performance of various URL-based, phishing detection machine learning models in a "real-world" environment. We have shown that in such an environment, the URL path is usually not included during analysis, hence the results of these research papers are put on doubt. To provide concrete results, we evaluated the models' performances in this environment, by first implementing five different machine learning models (three from past research

TABLE VII. SOLUTION LABELLED DATA SET RESULTS.

| Data set | Model | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|
| CPUURNN | CPUURNN | 0.752 | 0.800 | 0.650 | 0.717 |
| | CNN Model | 0.748 | 0.828 | 0.605 | 0.699 |
| | LSTM Model | 0.746 | 0.768 | 0.682 | 0.722 |
| | XGBoost | 0.765 | 0.747 | 0.777 | 0.762 |
| | PDRCNN | 0.738 | 0.822 | 0.587 | 0.685 |
| | **Our Solution** | **0.712** | **0.687** | **0.842** | **0.756** |

TABLE VIII. SOLUTION UNLABELLED DATA SET RESULTS.

| Target data set | Trained data set | Models | | | | | |
|---|---|---|---|---|---|---|---|
| | | CPUURNN | CNNModel | LSTMModel | PDRCNN | XGBoost | Our Solution |
| **Certstream** | CPUURNN | 0.13 | 0.13 | 0.12 | 0.21 | 0.12 | **0.48** |

works, and two that were self-designed), and then evaluating them on five different labelled data sets which have been pre-processed to remove the URL path. The results show that these models are still able to perform competitively on such pre-processed data sets. To the best of our knowledge, we are the first to present such results on data sets of URLs that do not contain the path.

We have also evaluated the models on data sets meant to simulate a real world environment by using unlabelled URLs and verifying them using VirusTotal. In this scenario, we show that that these models are completely unable to match their originally-claimed performances, obtaining results that are much lower than what was originally projected. We assume that the main cause of this is due to the "polarity" of the labelled data sets used during training and testing, where URLs are either "very benign" or "very malicious". Hence, these models are unable to due with URLs that fall into the "grey area" between the two labels, which are exceedingly common during real-world usage. This represents a gap between what these models claim to be able to do, and the concrete results that we obtained.

Last but not least, we presented and demonstrated a model specifically designed to target this issue. By using an auto-encoder trained on unlabelled data sets, the model was able to learn a representation of a URL which was then passed as inputs to a classification model. This transfer learning method allowed the model to generalise inputs even on "in the wild" data, hence obtaining improved performance on those data sets.

## REFERENCES


[1] Neda Abdelhamid. Multi-label rules for phishing classification. *Applied Computing and Informatics*, 11(1):29–46, 2015.

[2] Farhan Douksieh Abdi and Lian Wenjuan. Malicious url detection using convolutional neural network. *Journal International Journal of Computer Science, Engineering and Information Technology*, 2017.

[3] APWG. *Q1 Report*, 2019.

[4] APWG. *Q4 Report*, 2019.

[5] Alejandro Correa Bahnsen, Eduardo Contreras Bohorquez, Sergio Villegas, Javier Vargas, and Fabio A González. Classifying phishing urls using recurrent neural networks. In *2017 APWG symposium on electronic crime research (eCrime)*, pages 1–8. IEEE, 2017.

[6] Simon Bell and Peter Komisarczuk. An analysis of phishing blacklists: Google safe browsing, openphish, and phishtank. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–11, 2020.

[7] Eduardo Benavides, Walter Fuertes, Sandra Sanchez, and Manuel Sanchez. Classification of phishing attack solutions by employing deep learning techniques: A systematic literature review. In *Developments and Advances in Defense and Security*, pages 51–64. Springer, 2020.

[8] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. Lexical feature based phishing url detection using online learning. In *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, pages 54–60, 2010.

[9] Ye Cao, Weili Han, and Yueran Le. Anti-phishing based on automated individual white-list. In *Proceedings of the 4th ACM workshop on Digital identity management*, pages 51–60, 2008.

[10] Kang Leng Chiew, Ee Hung Chang, Choon Lin Tan, Johari Abdullah, and Kelvin Sheng5 Chek Yong. Building standard offline anti-phishing dataset for benchmarking. *International Journal of Engineering & Technology*, 7(4.31):7–14, 2018.

[11] Kang Leng Chiew, Choon Lin Tan, KokSheik Wong, Kelvin SC Yong, and Wei King Tiong. A new hybrid ensemble feature selection framework for machine learning-based phishing detection system. *Information Sciences*, 484:153–166, 2019.

[12] Kang Leng Chiew, Kelvin Sheng Chek Yong, and Choon Lin Tan. A survey of phishing attacks: their types, vectors and technical approaches. *Expert Systems with Applications*, 106:1–20, 2018.

[13] FBI. *Internet Crime Report*, 2019.

[14] Engin Kirda. Getting under alexa's umbrella: Infiltration attacks against internet top domain lists. In *Information Security: 22nd International Conference, ISC 2019, New York City, NY, USA, September 16-18, 2019: Proceedings*, volume 11723, page 255. Springer Nature, 2019.

[15] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[16] Yukun Li, Zhenguo Yang, Xu Chen, Huaping Yuan, and Wenyin Liu. A stacking model using url and html features for phishing webpage detection. *Future Generation Computer Systems*, 94:27–39, 2019.

[17] Jakub Nowak, Marcin Korytkowski, Patryk Najgebauer, Marcin Wozniak, and Rafa l Scherer. Url-based phishing attack detection by convolutional neural networks.

[18] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156*, 2018.

[19] KV Pradeepthi and A Kannan. Performance study of classification techniques for phishing url detection. In *2014 Sixth International Conference on Advanced Computing (ICoAC)*, pages 135–139. IEEE, 2014.

[20] Pawan Prakash, Manish Kumar, Ramana Rao Kompella, and Minaxi Gupta. Phishnet: predictive blacklisting to detect phishing attacks. In *2010 Proceedings IEEE INFOCOM*, pages 1–5. IEEE, 2010.

[21] Routhu Srinivasa Rao, Tatti Vaishnavi, and Alwyn Roshan Pais. Catchphish: detection of phishing websites by inspecting urls. *Journal of Ambient Intelligence and Humanized Computing*, 11(2):813–825, 2020.

[22] Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, and Banu Diri. Machine learning based phishing detection from urls. *Expert Systems with Applications*, 117:345–357, 2019.

[23] Joshua Saxe and Konstantin Berlin. expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. *arXiv preprint arXiv:1702.08568*, 2017.

[24] Rakesh Verma and Keith Dyer. On the character of phishing urls: Accurate and robust statistical learning classifiers. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 111–122, 2015.

[25] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[26] Weiping Wang, Feng Zhang, Xi Luo, and Shigeng Zhang. Pdrcnn: Precise phishing detection with recurrent convolutional neural networks. *Security and Communication Networks*, 2019, 2019.

[27] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. Cantina+ a feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):1–28, 2011.

[28] Huaping Yuan, Zhenguo Yang, Xu Chen, Yukun Li, and Wenyin Liu. Url2vec: Url modeling with character embeddings for fast and accurate phishing website detection. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 265–272. IEEE, 2018.

[29] Yue Zhang, Jason I Hong, and Lorrie F Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on World Wide Web*, pages 639–648, 2007.

## VIII. Appendix

## Appendix

TABLE IX. UNLABELLED DATASET WITH DUPLICATES RESULTS.

| Target dataset | Trained dataset | Model | | | | |
|---|---|---|---|---|---|---|
| | | **ClassificationUsingRNN** | **CnnModel** | **LstmModel** | **PDRCNN** | **XGBoost** |
| **CertStream** | 1M-PD | 0.18 | 0.13 | 0.23 | 0.11 | 0.23 |
| | benchmark | 0.13 | 0.01 | 0.04 | 0.01 | 0.02 |
| | Classifying_Phishing_RNN | 0.18 | 0.02 | 0.12 | 0.08 | 0.26 |
| | our_data_2020-03-25 | 0.05 | 0.11 | 0.09 | 0.11 | 0.38 |
| | PDRCNN | 0.14 | 0.01 | 0.18 | 0.02 | 0.05 |
| **Proxy** | 1M-PD | 0.01 | 0 | 0.03 | 0 | 0.04 |
| | benchmark | 0.01 | 0 | 0 | 0 | 0.01 |
| | Classifying_Phishing_RNN | 0 | 0 | 0.01 | 0 | 0.03 |
| | our_data_2020-03-25 | 0 | 0 | 0 | 0 | 0.04 |
| | PDRCNN | 0 | 0 | 0 | 0 | 0 |