Rheinisch-Westfälische Technische Hochschule Aachen
Informatik 5, Information Systems
Prof. Dr. Stefan Decker

Bachelor Thesis

An Evaluation of Similarity-Preserving Bloom Encodings in URL-Based Phishing Detection

Eduard Bursuc-Pahome

December 23, 2024

Advisors Lasse Nitz, Avikarsha Mandal
1st Supervisor Prof. Dr. Stefan Decker
2nd Supervisor Prof. Dr.-Ing. Ulrike Meyer

Abstract

Phishing attacks continue to pose a significant threat to society, leading to substantial financial and personal losses for individuals and organizations. Due to the increasing popularity of the Internet, more users are using it to perform a wide variety of daily tasks, such as connecting with friends and family, managing finances with online banking, or even online shopping. Cybercriminals deceive unsuspecting users by exploiting sensitive topics to gain personal information, often leading to personal and financial losses. It is one of the most alarming and popular forms of cybercrime. Traditional detecting methods, such as blacklists, struggle to compete with the evolving tactics of attackers. Machine learning methods have been widely used for phishing detection, with deep learning models being explored more recently. This thesis proposes a deep learning-based phishing detection system using similarity-preserving Bloom encodings to enhance privacy in a Privacy-as-a-Service context. URLs are encoded using this method to prevent leakage of sensitive information, which can be found within the URL itself. Due to their one-way nature, reversing the encodings should be a challenging task. Two encoding methods are explored: a q-gram-based approach and a feature vector-based approach. The experiments were conducted on a dataset comprising approximately 470,000 URLs, using a convolutional neural network architecture. Results show that the n-gram approach offers a strong overall performance, achieving a maximum accuracy of 95.28%. This is 2% lower than its clear text counterpart, where no encoding is performed. The feature vector approach, although less accurate (85.01% maximum accuracy), offers a more compact encoding for longer URLs, making it more scalable. These findings highlight the trade-offs associated with balancing accuracy and privacy when encoding data with similarity-preserving Bloom encodings in a Privacy-as-a-Service setting, while also potentially offering a secure and efficient phishing detection system.

Phishing attacks continue to be a harmful cyberattack that result in millions of financial losses for individuals and organizations. The idea behind this thesis came from the need for a more advanced phishing detection system, particularly focusing on the privacy-preserving aspect and application of Bloom filter encodings. This research was conducted as part of my Bachelor's Program in Computer Science at the RWTH Aachen. It provided an opportunity to explore a new field and combine privacy-preserving techniques with phishing detection. The idea stems from my two advisors, Lasse Nitz and Avikarsha Mandal. The thesis came with its challenges, from designing the overall architecture, which turned out to be more challenging than expected, to finding and testing several methods to improve the performance of the system. I would like to give my greatest thanks to my two advisors, who helped and guided me throughout this project. Their constructive feedback and overall support were extremely valuable, making it a pleasant learning experience. I would also like to thank my family and my partner for their support and patience. It is also my hope that my work contributes to the field of cybersecurity and inspires other researchers to dive deeper into this topic. This project was a great learning experience for me and I am thrilled to share my findings with the world.

— *Eduard Bursuc-Pahome, Aachen, 2024*

Contents

1 Introduction

Due to the increasing number of internet users, phishing attacks have become a significant threat to both individuals and organizations. According to the Anti-Phishing Working Group (APWG) [1], there was a 17.3 percent increase in people with internet access between June 2017 and December 2022. With the help of the internet, people are able to do almost anything, such as connecting with other users through social media, managing finances with online banking or even shopping for groceries. It has become an essential tool for people in their everyday lives. However, due to the vast amount of information that is available online, some individuals develop malicious intent, aiming to steal personal information for financial profit. As stated by [2, 3],however, it can also be for recognition or fame.

Phishing is one of the most popular forms of cyberattack on the Internet [4]. The Anti-Phishing Working Group observed a record-high number of approximately five million attacks in 2023 [5]. In the first quarter of 2024, social media platforms were the most extensively targeted sector [6]. A total of one million attacks were observed, of which the majority were executed through phishing websites. This quarter marked the lowest level of phishing activity since the fourth quarter of 2021.

Phishing is a cyberattack that aims to steal sensitive information from individuals and organizations. Examples include usernames, passwords, e-mail addresses and financial details. There are different ways phishers try to deceive users, including e-mail services, messaging apps such as WhatsApp or Telegram, via voice phishing (also known as vishing), through social media and also, websites [7].

In most phishing cases, the attacker creates a fake website and sends a spoofed e-mail to the victim, trying to make it appear as legitimate as possible. This e-mail is designed in such a way to trick the recipient into believing that it was sent by a trusted source, such as a work colleague, a friend, or a trustworthy organization. In order to cause panic and an irrational reaction, the attacker frequently refers to sensitive topics such as unusual activity on the victim's bank account or a sudden password change. This sense of urgency motivates the victim to click on the malicious link contained within the e-mail. Upon the victim's visit to the fake website, they are required to input their credentials. After submission, the phisher gains access to those credentials to use for their own benefit.

These websites typically look very similar to their legitimate counterparts, therefore making it difficult for the victim to see the difference by only looking at the content of the website. However, the Uniform Resource Locator (URL) is always going to be different, since two websites cannot have the same URL. Experienced Internet users can tell if they are real or fake by analyzing the URL, for example by checking for misspelled words or looking for random sequences of symbols [2, 3, 7].

1.1 Motivation

Despite numerous advances in cybersecurity, traditional phishing detection methods, such as whitelists, blacklists and heuristic approaches, come with their limitations. For instance, blacklisting services are slow at detecting malicious websites. According to a study by [8], blacklisting services are capable of detecting phishing attacks only twelve hours after their initial launch, whereas the majority of phishing campaigns end within the first two. In recent years, phishing attacks have become more sophisticated, making it harder for traditional methods to detect them [4].
Traditional methods also encounter difficulties in managing the huge amount of phishing websites that are generated on a daily basis. With the aid of Large Language Models (LLMs), phishers can create a lot of fake URLs with higher levels of sophistication, as shown in [9]. This further decreases the potential of traditional methods to detect them. Machine learning approaches have been extensively used for phishing detection and other cybersecurity-related applications, but as a result of growth in AI technology, deep learning methods have been explored more in recent years [10].
By focusing entirely on URL features, phishing attacks can be analyzed immediately after their initial launch, therefore enabling real-time detection, which is a crucial aspect of phishing detection systems. Because fake websites are very similar to their legitimate counterparts, it makes content-based detection challenging, as backed by [8]. Since URLs are unique, advanced methods, such as deep learning, are able to identify phishing attacks by analyzing the structure and patterns of the URL itself. This approach requires fewer resources, such as processing power and data storage, making it more suitable to implement on a larger scale. It therefore enables the analysis of millions of URLs in a short time frame [10].
When considering phishing detection in a Privacy-as-a-Service (PaaS) setting, privacy concerns may arise due to the sensitive information that may be contained in URLs. To address this concern, similarity-preserving Bloom encodings (SPBE) can be used to encode the data. This technique will ensure that similar URLs will have similar encodings, as shown in [11]. Therefore, the model should be capable of classifying encoded data

accurately. This could be relevant in a PaaS setting, where a PaaS company offers to create phishing detection systems for clients, but outsources the training to a Machine-Learning-as-a-Service (MLaaS) company due to limited resources. Encoding the data reduces the probability of leaking the sensitive information in the URLs. The PaaS setting will be discussed in greater detail in Chapter 4.
This thesis addresses the need for a sophisticated approach using advanced techniques, such as deep learning, in order to provide an accurate and efficient detection system, which can also preserve privacy with the help of SPBE.

1.2 Objectives

The primary objective of this research is to develop a deep learning-based phishing detection system that focuses on URL features and uses similarity-preserving Bloom encodings to enhance privacy. Different configurations will be tested and evaluated, with the primary focus on altering what is being encoded. The first method splits the URL into sub-strings (q-grams) and encodes them separately, as opposed to encoding the entire URL at once. The second method focuses on extracting features from the URL, creating a feature vector from those features and encoding it. Different encoding parameters will be tested to find the most optimal solutions for this use case, and to also see how the system behaves on different settings. These parameters include the length of the Bloom filter, the number of hash functions and the size of the q-gram. Common evaluation metrics such as accuracy, precision, recall, and execution time will be used to evaluate the model's performance. Afterwards, the results of these experiments will be compared with a clear text approach to conclusively evaluate the efficiency of SPBE in the context of phishing detection.

1.3 Outline

The rest of the thesis is structured as follows: Firstly, Chapter 2 will provide the most important information needed to understand the subject. It will include general information on phishing and phishing attacks, traditional methods, machine learning, features and feature extraction, deep learning, evaluation metrics, Bloom filters and finally SPBE. Chapter 3 follows, where several related studies are presented, ranging from typical machine learning approaches, to more advanced and modern deep learning systems, some of which also explore automatic feature extraction. The final paper focuses on the use case of SPBE in a domain generation algorithm detection (DGA) setting. Chapter 4 focuses on the use case of this system and the requirements that come with it, along with a threat model. Then, two candidate classifiers are presented to determine the

best choice for this system. Chapter 5 provides an overview of the concept of the system and then dives deeper in the actual methods used to develop it. Firstly, it presents the initial setup, what hardware and libraries were used, how the data was collected and pre-processed, what features were extracted and how they were adapted to fit the system, how the Bloom encoder works and then finally, the architecture of the deep learning model. Chapter 6 is the evaluation of the research, presenting the results of both approaches. Conclusions are drawn from the results to demonstrate how the system behaves under different settings, evaluating the impact of various parameters on its performance. The final chapter, Chapter 7 summarizes the entire thesis and highlights the key findings. Additionally, it provides recommendations for future work and potential ways to improve the system.

2 Background

2.1 Phishing Attacks

Phishing attacks are a social engineering technique that use various methods to lure the victim into revealing sensitive information, such as usernames, passwords, e-mail addresses, or financial information. The three core components of phishing are the medium, the vector, and the technique approach. The medium refers to the method the phisher uses to interact with the victim, which can be voice phishing, SMS or the Internet. The vector depends on the medium chosen by the phisher. E-mail, fax, social networks, websites, instant messaging applications or even publicly accessible Wi-Fi are some of the vectors in the Internet medium. The phishing technique is used by the phisher to obtain the information they are trying to steal. For example, bulk phishing refers to a large-scale attack, in which the phisher sends out the same attack to a large group of people. Spear phishing is a targeted approach against an individual or organization that uses personalized materials to improve the chances to trick the victim into doing what the phisher wants. For example, a phisher could send an e-mail pretending to be a co-worker, friend, or even family member. In order to generate personalized attacks, phishers must initially conduct research on the target, such as through social media. One sub-category of spear phishing is whaling, where the target is an individual of significant importance, such as an executive or other high-ranking employee of a company. Their position could provide the phisher with important data, which could potentially generate huge gains [7].

2.2 Traditional methods

There are several methods to detect phishing attacks, which can usually be categorized as either traditional methods or non-traditional methods. Traditional methods include user awareness, blacklists, whitelists, whereas non-traditional methods include heuristics, visual similarity and others [12].

2.2.1 User Awareness

The concept of user awareness refers to the training of users to recognize and report potential phishing threats. This is an important aspect of phishing detection, as users who have the knowledge and experience in this field are significantly less susceptible to becoming victims of a phishing attack [13]. PhishGuard[1] for instance, is a phishing simulation service primarily aimed at companies to educate and train employees on identifying phishing attempts through the sending of simulated phishing e-mails and texts [14].

2.2.2 Whitelists and Blacklists

Whitelists are databases consisting of well-known legitimate URLs, whereas blacklists consist of known phishing sites. Several well-known blacklisting services include Google Safe Browsing[2], OpenPhish[3] and PhishTank[4]. The latter two are community-based collaborative services for phishing detection, where users contribute by submitting suspicious URLs to the database, which are then reviewed and verified by the community. Google Safe Browsing is operated by Google's security team, which scans URLs daily for signs of malicious activity. When a malicious website is identified, it is added to the blacklist. Both approaches have a common disadvantage: they are slow at detecting and adding phishing websites to the respective list. According to [8], 63 percent of phishing campaigns conclude within the initial two hours of their launch, however, blacklisting services are capable of identifying the majority of phishing websites only twelve hours after their initiation.

2.3 Non-traditional methods

Some non-traditional methods for detecting malicious websites include heuristic-based detection, visual similarity, machine learning, and deep learning [12].

2.3.1 Heuristic-based approaches

Heuristic-based approaches involve the use of rules, algorithms, and techniques to identify phishing characteristics that are usually present in most attacks. For example, this could include analyzing the content of the e-mail to search for keywords that are commonly used by phishers, such as 'secure', 'login', 'banking', and others, as shown in [14, 15].

[1] https://cerebra.sa/products/phishguard
[2] https://safebrowsing.google.com/
[3] https://openphish.com/
[4] https://phishtank.org/

2.3.2 Visual Similarity

Visual similarity methods aim to identify phishing attacks based on their visual appearance. This involves examining the layout and design of a phishing website instead of analyzing the HTML code or the URL. Typically, phishing websites closely resemble their legitimate counterparts. However, in most instances, there are minor differences between them, as backed by [15]. The method proposed in [16] compares the suspected website to brands that are likely to be targeted by phishers, such as PayPal, Amazon, and eBay, by taking a snapshot of the website.

2.4 URL-based phishing detection

URL-based phishing detection focuses only on analyzing the URL of a website to determine its classification. By examining patterns and features within the URL, the machine predicts whether the website is likely to be benign or malicious.

2.4.1 URL Anatomy

It is important to understand how a URL is composed to understand its features and machine learning techniques. As defined by the RFC3986 Standard, [17] a URL typically contains four main components: the scheme, the host, the path, and the query. The scheme refers to the protocol used, such as HTTP for websites without an SSL certificate or HTTPS for sites that have one. The host identifies the resource holder; for example, in https://amazon.com, 'Amazon' is the host. The host can be further divided into three domains: top-level domain (e.g., '.com'), second-level domain (e.g., 'amazon') and subdomains (e.g., 'support.amazon.com'). Additionally, the host may sometimes include a port number [2, 17]. The path specifies a particular resource that the client wants to access, which, in the example mentioned above, could be referring to a product on Amazon. Lastly, the query contains additional information, often in the form of parameters, that the resource can use for a specific purpose [17].

2.4.2 Machine Learning

It is also possible to classify a website as legitimate or phishing by analyzing the URL, for example as shown in [14]. URL-based machine learning approaches are a safer alternative, as the user does not need to visit the malicious website. This in itself can be a risk, as some websites may contain malware. It is also possible to use machine learning in a different phishing detection setting, such as in visual similarity. In this case, the machine analyzes the context of the webpage, rather than the URL. Before being deployed and used in real-world scenarios, the machine must

undergo training on a dataset consisting of legitimate and phishing websites, as well as their classification as benign or phishing [8, 14]. Since traditional machine learning approaches cannot extract features automatically, this step is done manually by the developer prior to training.
This type of machine learning is commonly referred to as supervised learning, as it involves training the model using both input data and the corresponding output. To build a dataset, data can either be manually extracted from different websites or sourced from publicly available resources. For instance, malicious URLs are often collected from platforms such as PhishTank[5]. On the other hand, legitimate URLs can be obtained from Commoncrawl[6], for example [10].

2.4.3 Feature Extraction

Attackers frequently use diverse techniques to make their malicious website look very similar to the website they are trying to copy, rendering it challenging for users to distinguish them apart. These features are found in certain parts of the website, such as the content, the HTML code, or the URL [15].
Since the focus of this thesis will be on URL features, only those will be considered. URL features can be divided into two categories: binary and non-binary features. Binary features can have only one of two distinct values, true or false, often represented as 1 for true and 0 for false. Some binary features include the presence of an IP address or certain keywords in the URL, the use of a URL shortening service (e.g., bit.ly[7]), or the presence of HTTPS. Non-binary features can be categorized into two main types: categorical and numerical. Categorical features are those with a limited and discrete set of values. For example, the Top-Level Domain (TLD) of an URL, such as .com, .org, .net, is considered categorical. On the other hand, numerical features represent values that can technically take an infinite range of values. Examples of numerical features include the length of the URL or domain, the number of subdomains, and similar attributes. These features are widely used in phishing detection studies, such as in [3, 8, 14, 18, 19].
Certain features require the use of a specific feature extractor. For example, the Bag-Of-Words feature represents text data as a collection of words without considering grammar or the order of words. This feature is usually used for textual data, such as phishing e-mail detection. In the context of URLs, they may consist of special characters and keywords commonly found in phishing websites, like the ones mentioned in section 2.3.1, also shown in [8, 20]. Nonetheless, this approach may have certain limitations when considering URL phishing detection. As shown by [8],

[5] https://phishtank.org/
[6] https://commoncrawl.org/
[7] https://bitly.com/

the classifier may have trouble predicting new URLs with long tokens, such as paypalhelpservice.simdif.com, because these longer tokens are likely to not be present in the training dataset. Therefore, they proposed to segment the tokens first into sub-words or into q-grams, which both improved the performance of the classifier.
Another potential feature that requires a feature extractor is the entropy, which determines the randomness of URLs in order to identify obfuscation techniques. This feature was researched by [14], who calculated the entropy of non-alphanumeric characters (NAN), which refers to any character that is neither a letter nor a number. This feature improved the performance of their system on a balanced dataset of 5000 phishing sites and 5000 legitimate sites.

2.4.4 Deep Learning

Deep learning is a subset of machine learning that mainly refers to deep neural networks or deep reinforcement learning. The term 'deep' refers to the number of layers present in the network. There are many different architectures that can be used, including convolutional neural networks, recurrent neural networks, long-short term memory, artificial networks, and more, each with their distinct features. These architectures are suitable depending on the use case and the data which is used and processed [21]. One of the key characteristic properties of deep learning is that it requires a significantly higher amount of data than machine learning approaches, which can, on the other hand, perform well even if the dataset is small. Deep learning is a 'data hungry' approach, which performs better with huge amounts of data [10, 21]. Due to the intricate structure, deep learning requires additional computational resources and longer processing time. Depending on the problem and structure, it may even take months to train a classifier, whereas traditional machine learning algorithms usually train in a few minutes or, at most, a few hours. As stated in section 2.4.2, a significant drawback of traditional machine learning classifiers is that they require manual feature extraction prior to training. Deep learning methods do not require this, as they can automatically extract features from the URL without any human interaction. These techniques are widely used in the fields of malware detection, intrusion detection, and spam detection, although they have recently been explored in phishing detection as well [10, 19].

2.5 Evaluation Metrics

Evaluation metrics provide a way to measure the performance of classifiers. To understand this research, it is important to understand the more commonly used metrics in the field.

Confusion Matrix. The Confusion Matrix is depicted in 2.1. It explains the four crucial metrics needed for calculating other metrics.

Table 2.1: Confusion Matrix for Binary Classification [22]

	Actual Positive (1)	Actual Negative (0)
Predicted Positive (1)	True Positive (TP)	False Positive (FP)
Predicted Negative (0)	False Negative (FN)	True Negative (TN)

Precision. Precision (P) measures the proportion of correct positive predictions, where the model predicts true positives [22]. The formula is shown in Equation 2.1.

$$\mathrm{P} = \frac{TP}{TP + FP} \tag{2.1}$$

Recall. Recall (R), also known as Sensitivity, measures the ratio of actual positives that the model correctly predicted to the total number of actual positives [22]. The formula is shown in Equation 2.2.

$$\mathrm{R} = \frac{TP}{TP + FN} \tag{2.2}$$

Accuracy. Accuracy (A) measures the percentage of correct predictions [22]. The formula is shown in Equation 2.3.

$$\mathrm{A} = \frac{TP + FN}{TP + FP + TN + FN} \tag{2.3}$$

F1 Score. The F1 Score combines Precision and Recall into one function, which is particularly useful for evaluating the effectiveness of models on unbalanced datasets. The formula is shown in Equation 2.4.

$$\text{F1 Score} = \frac{P \times R}{P + R} \tag{2.4}$$

Error Rate. The Error Rate (ER) measures the proportion of incorrect predictions to the total number of predictions made. Essentially, it is the inverse of the accuracy score, where $1 - A = ER$. The formula is shown in Equation 2.5.

$$\mathrm{ER} = \frac{(FP + FN)}{(TP + FP + TN + FN)} \tag{2.5}$$

ROC AUC. The Receiver operating characteristic (ROC) curve visualizes a classification model's performance across various thresholds by plotting the true postive rate (TPR) against the false postive rate (FPR). These metrics are shown in in Equation 2.6 and Equation 2.7 respectively:

$$\text{TPR} = \frac{TP}{TP + FN} \tag{2.6}$$

$$\text{FPR} = \frac{FP}{FP + TN} \tag{2.7}$$

The area under curve (AUC) measures the performance of the model, where it is said that a classifier can perfectly distinguish between postive and negative classes if the value of AUC is 1. If the value is 0.5, then the classifier cannot distinguish between positive and negative classes at all. ROC AUC is useful for comparing, evaluating and selecting models and offers a clear indication for a model's ability to distinguish between binary classes.
Although ROC AUC and F1 score are commonly used in classification tasks, they were not used in this study due to the relatively balanced nature of the dataset (55% benign and 45% phishing), which is further elaborated upon in Chapter 5. As these two metrics are more useful for severely imbalanced datasets, they offer less additional value here compared to evaluating the metrics separately.

2.6 Bloom Filters

Bloom filters (BF) are space-efficient probabilistic data structure s developed for determining whether an element is a member of a set, also commonly referred to as membership queries. It was initially introduced by Burton H. Bloom in 1970 [23]. Firstly, the Bloom filter is initialized with a bit array of size m, where all bits are set to zero. When an element x is added to the Bloom filter, k hash functions $h_i(x)$ are used to generate a random number within the range $\{0, ..., m-1\}$. The bits in the array that correspond to these randomly generated numbers are then set to 1. To determine if an element y is part of the set (Bloom filter), the same k hash functions are used to produce k random numbers for y. The bits that correspond to y are checked in the filter, and, if all k bits are set to 1, then y is likely to be in the set. If any bit corresponding to y is set to 0, then y is definitely not in the set. Table 2.2 provides a summary of the symbols.
Bloom filters are used extensively in security applications, such as network security, due to their hashing nature. However, one of their major drawbacks is that they can cause false positives, which usually happens when the filter contains many elements, therefore consisting of many 1 bits. For instance, if the bits corresponding to an element x are already

set to 1 in the filter by different elements, x would be flagged as present in the set, although it is not [24].

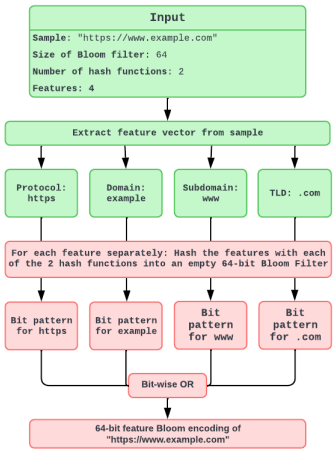
Table 2.2: Notation Table for Bloom filters

Symbol	Description
m/l	Size / Length of the Bloom filter
k	Number of hash functions
n	Number of elements to be added to the BF
q	Length of q-gram in SPBE approach
$h_i(x)$	The i-th hash function applied to element x

2.6.1 Similarity-Preserving Bloom Encodings

Similarity-preserving Bloom encodings were initially proposed by Schnell et al. in 2009 [25]. They extend traditional Bloom filters by enabling similarity queries, which can be utilized to determine if an element shares a similarity with another element. They are therefore useful for phishing detection, where phishing URLs share features with legitimate and other phishing URLs. The key difference is that, instead of hashing the full element into the Bloom filter, the element is first split into sub-strings, which are then hashed individually. This results in different bit patterns for each sub-string. By performing the binary OR function, they are combined into one Bloom filter, resulting in a Bloom encoding of that specific element. Hence, as stated by Nitz et al. [11], elements possessing decompositions will also have similar encodings.
There are several parameters that can be changed to alter the encodings, which are summarized in Table 2.2. The length of the Bloom filter l refers to the number of bits in the filter, and the number of hash functions k is the same as in normal Bloom filters. However, there is a new parameter, q, which refers to the length of the q-gram that result from splitting the input. In the study [11], which focuses on the application of SPBE in DGA detection, the input "EXAMPLE" with $q = 2$ is divided into the following 2-grams: {EX, XA, AM, MP, PL, LE}. As URLs also contain special characters, encoding this type of data may be different. The special characters can either be left as part of the URL, or removed altogether. The other option would be to extract certain features from the URL and skip the splitting step, hashing the features instead. The second approach is illustrated in Figure 2.3, where the features of an example URL are extracted and encoded.

Figure 2.3: Visualization of feature encoding inspired by [11]



3 Related Work

Phishing detection has been and continues to be a significant area of research due to the rise in phishing attacks. Diverse methods have been evaluated and implemented to identify them. In this section, some of these techniques will be discussed, including URL based phishing detection, deep learning in phishing detection and similarity-preserving Bloom encodings in a DGA setting. A summary of the methods, datasets and performance of these approaches can be found in Table 3.5.

3.1 Entropy of NAN Characters

The hypothesis of the study by Aung et al. [14] is that phishing websites tend to use more non-alphanumeric (NAN) characters, which include any character that is neither a letter or a number, than legitimate ones. However, instead of using the frequency of each character, this study calculates the entropy of ten selected NAN characters. This is done using the formula shown in Equation 3.1,

$$\text{Entropy} := -\sum_{i \in T}^{n} (P_i \log_2 P_i) \tag{3.1}$$

where i is the i-th NAN character in T, T is the list of NAN characters and P_i is the occurrence probability of character i in the input, which, in this case, is the URL.

Features. The NAN characters were selected using a Feature Importance score, which uses a Random Forest (RF) classifier to determine which characters are more important for classification. Using this method, ten of the 21 characters reversed for URIs, as defined by RFC3986 [17], were selected. These characters are shown in Table 3.1.
The experiment utilized a set of twelve features, seven of which were binary features and the rest were non-binary features. The binary features include the presence of an IP address, an executable file, sensitive words such as 'secure' or 'login', a double slash in the path, an internal link in the path, a short domain age and the usage of a free hosting service. The non-binary features include the count of dashes, at symbols, and dots, which port number is used and the entropy of NAN characters.

Table 3.1: Top 10 most frequent NAN Characters in [14]

Name	Character	Name	Character
slash	/	question mark	?
dot	.	underscore	_
dash	-	percentage	%
equal	=	ampersand	&
semicolon	;	at	@

Table 3.2: Datasets used in [14]

Dataset	Legitimate	Phishing
D_1 (balanced)	5,000	5,000
D_2 (unbalanced)	95,754	10,473

Dataset. The researchers used two datasets, respectively D_1 and D_2, which are shown in Table 3.2. The purpose of this study was to evaluate the importance of the entropy feature and compare the results of incorporating it versus not incorporating it.

Classifiers. Three classifiers were selected and compared based on their ROC AUC score, which was explained in Section 2.5. These include Gaussian Naive Bayes (GNB), Support Vector Machine (SVM) and Random Forest (RF), where RF performed best on both datasets when evaluating the ROC AUC score. This classifier was selected for the experiments in the next section.

Experiments and Results. The first experiment used the balanced dataset D_1, achieving an accuracy score of 80.68%, a precision score of 83% and a recall score of 80% without the inclusion of the NAN entropy feature. When this feature was included, the accuracy score was improved by approximately 9%, the precision score by approximately 7% and the recall score by approximately 10%.
In the subsequent experiment, utilizing the unbalanced dataset D_2, it was discovered that the entropy feature did not improve the outcomes as significantly as in the initial experiment. The accuracy rose from 92.94% to 94.05%, and both the precision and recall rose from 93% to 94% when using the NAN entropy feature.

3.2 ML detection using NLP features

This paper by Sahingoz et al. [2] presents a real time phishing detection system using seven different classification algorithms, using natural language processing (NLP) features. This proposed approach highlights

some unique aspects of the system, such as language and third-party service independence, as well as real-time execution.

Features. The features are split into two categories: NLP features and word features. There are 40 NLP features in total, some of which include brand name presence, random word detection, typosquatting indicators and other features such as URL, host and path length, known TLD check, a subdomain counter and a digit counter. The word features are a list of words that are commonly found in malicious URLs.

Dataset. The used dataset consists of 73,575 URLs from which 36,400 are legitimate and 37,175 are phishing. The legitimate samples were extracted using Yandex Search API [1] and the phishing samples were extracted from PhishTank. The dataset was also made publicly available.

Classifiers. This study compares Decision Trees, Adaboost, Kstar, k-nearest neighbours with k=3, RF, Sequential Minimal Optimization (SMO) and Naive Bayes.

Experiments and Results. The seven classifiers were compared using three different feature sets: NLP features, word vector features and a hybrid set comprising of both. Overall, the RF classifier performed best on the NLP-based feature set, achieving an accuracy score of 97.98%, with Decision Trees being a close second, achieving 97.02% accuracy on the same feature set. The results are summarized in Table 3.3.

Table 3.3: Performance of top two classifiers in [2]

Algorithm	Features	Precision	Accuracy
Random Forest	NLP Features	0.970	97.98%
	Word Vector	0.958	83.14%
	Hybrid	0.953	96.36%
Decision Tree	NLP Features	0.964	97.02%
	Word Vector	0.944	82.48%
	Hybrid	0.933	95.14%

3.3 Bag-of-Words

The main goals of this research by Tupsamudre et al. [8] is to improve phishing detection by using and improving upon lexical features. Conventional Bag-of-Words (BoW) features are, according to [8], not robust enough to detect all types of phishing attacks. Therefore, they proposed

[1] https://yandex.cloud/en/services/search-api

two techniques, Segmented Bag-of-Words (SBoW) and Bag-of-n-grams (BoN) to improve phishing detection.

Features. The main goal of this research was to compare the effectiveness of SBoW and BoN with the traditional BoW approach, which was explained in Section 2.4.3. They key difference in SBoW, compared to traditional BoW, is that instead of splitting the URL in tokens only, the tokens are further split using a word segmentation algorithm. BoN, on the other hand, uses q-grams, which were explained in Section 2.6.1, to split the tokens. The authors chose 3-grams for their experiments. They have also implemented a 'Phishy-List', which is a list of commonly used tokens in phishing domains, comprising of 105 different tokens, which is used as a binary feature. Some numerical features were also implemented, such as the length of different parts of the URL, a dot counter, a hyphen counter, the presence of an IP address and a Port, along with other typical numerical features.

Dataset. In total, 100,000 URLs were used for this experiment, which were evenly split between phishing and legitimate URLs. PhishTank was used for the collection of the phishing samples and DMOZ[2], which is a large open source directory containing over five million URLs, for the legitimate ones.

Experiment and Results. The only classifier used in this experiment was Logistic Regression (LR), which was used to compare the different setups based on the Missclassification rate (MCR) and the False Negative Rate (FNR), which is shown in Equation 3.2. MCR is also known as the Error Rate (ER), which was explained in Section 2.5. In order to get the accuracy score for these experiments, the MCR can be substracted from 100% as shown in Section 2.5.

$$FNR = \frac{FN}{FN + TP} \tag{3.2}$$

Overall, using the SBoW approach, the Phishy-List and the numerical features, the classifier achieved an error rate of 3.22%, which corresponds to an accuracy of 96.78%. Without the Phishy-List and the numerical features, an accuracy score of 95.93% was achieved using SBoW. Using BoN only, without the Phishy-List and numerical features, an accuracy score of 95.82% was achieved. These results were compared to the baseline BoW approach, which achieved an accuracy score of 94.96%.

[2]http://dmoz-odp.org/

3.4 Automatic feature extraction

This research by Opara et al. [26] proposes a phishing detection system, named WebPhish, which is based on deep learning models using URL and HTML features. The features are extracted using automatic feature extraction, therefore removing the need for manual feature engineering.

Features. In order to evaluate the performance of WebPhish, it was compared to a few approaches which use manual feature engineering. The URL features extracted here included a suspicious word counter, a digit counter, a counter for slashes, question marks, dots, hyphens, underscores, equal signs, semicolons and amperstands, number and presence of subdomains and the length of the URL along with other HTML features.

Dataset. The dataset comprised 45,373 websites which were evenly balanced between benign and phishing. The legitimate samples were taken from Alexa's top-ranked websites, while the phishing ones were taken from PhishTank.

Architecture. The architecture of WebPhish is based on a Convolutional Neural Network (CNN). The URL of the sample is embedded on character level and the HTML content is embedded on word level, which are then concatenated and fed into the neural network. The CNN architecture consists of two convolutional layers with 16 filters and eight kernels.

Results. WebPhish was compared to several state-of-the-art phishing detection systems, outperforming all of them with an accuracy of 98.1%. Then, it was compared on accuracy, precision, recall, F1 score and training time to three traditional approaches which use manual feature engineering. The results are shown in Table 3.4.

Table 3.4: Performance of WebPhish compared to traditional methods in [26]

Model	Accuracy	Precision	Recall	Training time
WebPhish	98.1%	98.2%	98.1%	491s
Kernel SVM	87.3%	83.6%	92.4%	125s
Logistic Regression	88.4%	86.5%	90.8%	65s
Random Forest	94.5%	94.2%	94.8%	90s

3.5 DL Phishing Detection System

Sahingoz et al. [21] developed a system for detecting phishing URLs by utilizing a variety of deep learning classifiers, on a data set comprising over 5.1 million websites. According to the researchers, this was the largest dataset used in phishing detection, as of the release date of this paper.

Dataset. They initially gathered their data from PhishTank and Commoncrawl for the phishing URLs and legitimate URLs respectively, resulting in a total of over 5.1 million samples. They created a small subset from this dataset, consisting of 10 percent of the websites. The dataset was then subsequently divided into 70 percent training, 20 percent validation and 10 percent testing. The dataset is available as open source on GitHub[3].

Pre-processing. Due to the varying lengths of the URLs, the researchers decided to use an URL length of 200 characters. Any URL exceeding this length was truncated, whereas any URL below it was padded to 200. They chose this length because it was considered appropriate for representing the majority of the data set.

Architecture and Results. The team evaluated their results using five different deep learning classifiers, including Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNNs), Bidirectional Recurrent Neural Networks (BRNNs) and Attention Networks (ATTs), along with some traditional machine learning techniques, including Random Forest (RF), Naive Bayes (NB) and Logistic Regression (LR). All five deep learning architectures have been tested with a single layer consisting of 128 neurons, a batch size of 7000 and 20 epochs, which served as a base scenario. These tests were conducted utilizing the smaller subset. RNN performed best here with 95.1% test accuracy. The researchers then conducted a more intricate scenario by incorporating additional layers to the network, with each layer still comprising 128 neurons. The batch size was kept the same, except for the RNN architecture, which was reduced to 3000 due to computational reasons. The training was conducted over a duration of 20 epochs. In this particular scenario, the CNN architecture had the best performance in both test accuracy, which achieved 97.5%, and also training time, which was approximately eight minutes for all 20 epochs, including pre-processing tasks.
According to [21], the CNN architecture is more appropriate for phishing detection than the other architectures they have tested. The final evalu-

[3]https://github.com/ebubekirbbr/dephides/tree/main

ation was conducted on a complex CNN architecture using the large 5.1 million dataset on 100 epochs, resulting in an accuracy score of 98.74%.

3.6 DGA Detection using SPBE

In the paper by Nitz et al. [11], the researchers proposed a Bloom encoding technique for the sanitization of training data. The proposed approach encodes the input samples in a way that preserves the similarity between the encoded samples. This approach was evaluated in the context of DGA detection.

Architecture. The input is initially decomposed into q-grams, which are then individually hashed using several hash functions. The output from the hash function yields a numerical value in the range $\{1, \ldots, l\}$, where l denotes the length of the Bloom filter in bits. After the hashing of the q-grams is completed, all the results from the hash functions are then added to an empty Bloom filter of the same length l, resulting in different bit patterns for each q-gram. The algorithm then executes a bit-wise OR with all bit patterns, resulting in the final encoded output.

Dataset and classifier. The researchers used a dataset consisting of 134,036 samples, which was evenly split between malicious and benign samples. The benign samples were taken from an university, while the malicious ones were taken from DGArchive. A total of 101,866 samples were used for training, with 5,362 serving as the holdout set and 26,808 serving as the final evaluation. The classifier used was a CNN architecture with two-stacked one-dimensional CNN layers, each with 128 filters.

Tests and Results. Five tests were conducted using different Bloom filter lengths. All tests were performed using two hash functions and the inputs were split into 2-grams. A randomization parameter f was introduced by utilizing RAPPOR [27], which adds noise to each individual encoding. Accuracy, precision, and recall were the three metrics used in the experiment to evaluate system performance. The first test was performed using a clear text baseline without any encoding, resulting in an accuracy of 99.8%. When using SPBE to encode the samples with a Bloom filter of length $l = 128$ and no randomization, the system achieved an accuracy of 96.7%. When the randomization parameter was introduced with $f = 0.4$, the model achieved an accuracy of 93.1%. Using a shorter Bloom filter length of 64 reduced the accuracy, achieving 95.1% with no randomization and 88.9% with randomization.

Table 3.5: Summary of related work

Authors	**Method**	**Dataset**	**Performance**
Aung et al. (2019)	ML-based phishing detection using the entropy of NAN characters along with 11 other features	D1: 10,000 URLs D2: 106,227 URLs	D1: 89.82% accuracy D2: 94.05% accuracy
Sahingoz et. al (2019)	ML-based phishing detection using NLP and word features, tested with 7 classifiers	73.575 URLs	97.98% accuracy using RF + NLP features
Tupsamudre et. al (2019)	ML-based phishing detection using SBow and BoN as an extension to traditional BoW + Phishy List	100,000 URLs	BoW baseline: 94.96% accuracy, SBoW + PL: 96.78% accuracy
Opara et. al (2024)	WebPhish: DL-based phishing detection using automatic URL and HTML feature extraction	43,373 URLs	Webphish: 98.1% accuracy, RF + manual features: 94.5% accuracy
Sahingoz et al. (2024)	DL-based phishing detection using neural networks, including ANNs, CNNs, RNNs, BRNNs and ATTs, evaluated on a large scale dataset	5,202,841 URLs	CNN 100 epochs: 98.74% accuracy
Nitz et al. (2023)	DGA detection using CNNs and SPBE	134,036 URLs	Baseline cleartext: 99.8% accuracy, encoding without randomization: 96.7%, encoding with randomization: 93.1%

4 Use Case and Requirements

A potential use case of this approach could be useful for a company that provides Privacy-as-a Service (PaaS). This company offers phishing detection for clients, including businesses or financial institutions, to protect themselves and their employees against phishing attacks. However, they may lack the knowledge or resources to develop and maintain such a system. The PaaS company offers to develop this system for the client. The PaaS creates and pre-processes a dataset within the client company, where the legitimate samples are websites commonly used by the employees of the client company, or websites that belong to the client company. However, the PaaS company does not have the computational power to train a model and outsources this step to a third-party Machine-Learning-as-a-Service (MLaaS) company.

For this next section, it is assumed that the MLaaS Company is honest but curious. As the benign samples that were collected within the company may contain sensitive information, the PaaS company encodes the data set with SPBE. For example, a URL may point to a resource, such as a contract, that should not be seen by anyone besides the people involved. As it is assumed that the MLaaS is honest but curious, it may try to decode the data set to get to this sensitive information. However, due to similarity-preserving Bloom encodings being one-way and hard to reverse, as backed by [11, 28], this may be difficult to achieve. The third-party MLaaS Company will receive a data set that contains binary strings and their classification, not knowing what this data is used for, therefore helping with privacy preservation.

4.1 Requirements

For the Privacy as a Service company to be successful and be able to provide phishing detection systems that preserve privacy with SPBE, some important criteria must be met. These requirements are described in Table 4.1.

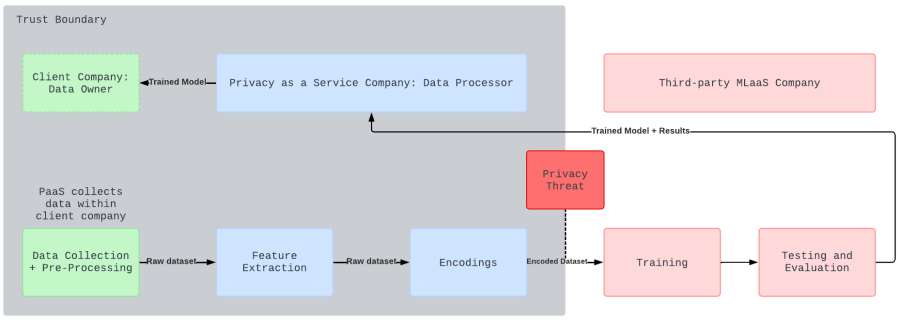
Table 4.1: Summary of Requirements

Requirement	Description
Accuracy and Precision	The system should accurately classify the URLs whilst maintaining high precision and accuracy to have a low false positive rate.
Similarity Preservation	The encodings should preserve patterns in the URL so that the system can classify accurately on the encoded samples. The accuracy rate for the encoded approach should not be significantly lower than on the clear text.
Privacy Preservation	The URLs must be encoded in a one-way fashion so that sensitive information is not accessible to the third-party organization.
Efficiency	The Bloom-encodings should be efficient and the resulting data set should not add any significant training time for the classifier.
Scalability	The system should be able to handle large data sets and increase in performance with more data.
User-Friendliness	The system should be user-friendly, as we assume the clients are not very knowledgeable in this field. The system should be easy to set up and maintain.
Compatibility	The system should be compatible with different inputs (for example, URL vs URL Features) and with different encoding parameters.

4.2 Threat Model

The threat model considers two possible scenarios regarding the adversarial party, which, in this case, is the third-party MLaaS company which trains the model. The first scenario is assumed in the use case above, where the adversarial party does not have access to any of the parameters that are used for the Bloom encodings, besides the length parameter l. This is revealed by the encoded samples themselves, since each encoded sample has the same length. Without knowing the other parameters, however, the adversarial party is very unlikely to be able to decode the data and gain access to the concealed data, which mitigates the privacy concern to a certain degree, as also backed by [11, 28]. Figure 4.2 shows a visualization of the use case based on this first scenario. The second scenario

Figure 4.2: Visualization of the Use Case



might be relevant in a setting where researchers share datasets and train classifiers, where knowing the parameters is crucial for training and evaluation. In this case, the threat is significantly larger. Using graph traversal attacks, like the approach proposed in [29], there is a high likelihood for the adversarial party to be able to decode the samples. The thesis will focus on the first scenario and not consider the second scenario, as the main focus of it lies more on testing the effectiveness of SPBE, and not on the privacy aspect of it.

4.3 Candidate Classifiers

In this section, two candidate classifiers will be compared regarding the requirements in Table 4.1. The candidate approaches were selected based on the research in [10], where the most commonly used classifiers in phishing detection were Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM).

4.3.1 CNNs

Convolutional Neural Networks are a deep learning technique which requires minimal pre-processing [21]. They consist of three types of layers: convolutional layers, pooling layers and fully connected layers [30]. They have been used in fields such as image recognition and phishing detection. According to [10], they have some strengths. CNNs are highly effective for feature extraction from raw data and can achieve high accuracy when resolving complex tasks. They are also computationally efficient, requiring less computational resources and training time than other classifiers, such as RNNs. However, CNNs require a large dataset to perform well. The researchers in [21] achieved the best results in their tests using CNNs, both in terms of accuracy and training time. Another positive aspect is that they can work with encoded inputs, as [11] used a CNN architecture for their DGA Detection system.

4.3.2 LSTM

Long Short-Term Memory is an improved version of Recurrent Neural Networks (RNNs) which excel at handling sequential data, such as sentences, for example. The main improvement over RNN is addressing the vanishing gradient problem, which can cause the model to have difficulty learning dependencies separated by many steps. They are used in various fields such as translation and speech recognition. Their structure is more complex than CNNs, generally consisting of more layers, which require more computational power than CNNs, as backed by [10, 21, 31].

4.3.3 Conclusion

Based on the comparison above, the suggested approach that will be used for this project is CNN. Although both approaches are often used in phishing detection [10], CNNs have some strengths over LSTM. They can achieve high accuracies whilst not requiring as much computational resources as LSTM [10]. Since I am working with limited computational power, this is an important aspect to consider. Although they need a large data set, that should not be an issue, as the data set that I will be using consists of approximately 480,000 samples. Since [11] have also successfully used CNNs on encoded data, this is yet another reason for choosing this architecture over the other.

5 Methodology

This chapter outlines the methodology employed for all the steps of the project, including data pre-processing, feature extraction, the Bloom encoder and the CNN Architecture. Figure 5 shows a summary of the system's conceptual architecture, providing a high-level overview of each component. Each step is explained in detail, covering the techniques used and observations made throughout the coding process. The actual results are shown in Chapter 6.

5.1 Initial setup

The first step for this project was to set up the coding environment. The entire project was executed using the current latest version of Python, Python 3.12 inside JupyterLab, a web-based IDE mainly targeted at data scientists. The reason for this choice of IDE was due to the fact that it does not require manual library installations, as most libraries needed for this project are already incorporated.

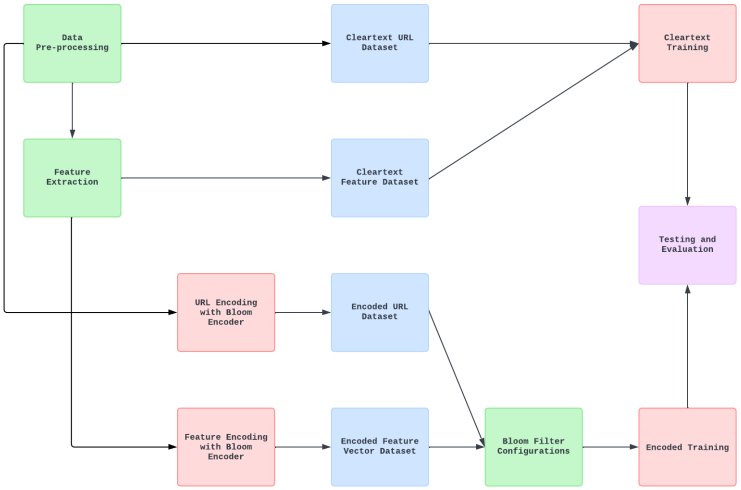


Figure 5.1: Concept of the System

5.1.1 Hardware

All experiments, data pre-processings tasks, feature extraction and encoding were conducted on my own personal computer. The specifications are listed in Table 5.2.

Table 5.2: Hardware Specifications

Component	**Specification**
Processor (CPU)	AMD Ryzen 7 3700X 8-Core Processor, 4.20GHz
Memory (RAM)	32 GB
Storage	1 TB SSD
Operating System	Windows 11 Pro

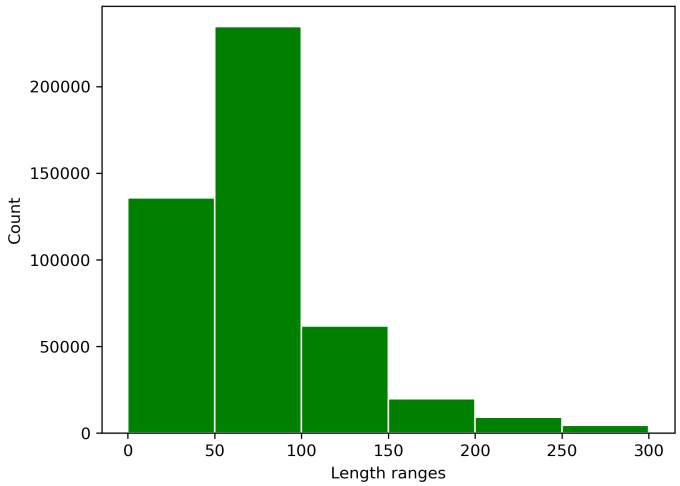
5.1.2 Libraries

Many libraries were required for this project, which are summed up in Table 5.3.

Table 5.3: List of libraries used in the coding project

Library	**Purpose**
pandas	Data preprocessing and visualization
regex (re)	Data preprocessing and manipulation
matplotlib.pyplot	Graph plotting, visualization
ipaddress	Feature extraction, ip address check
urllib.parse	Feature extraction
scipy.stats	Feature extraction, entropy
math	Mathematical operations
time	Time evaluation for training, encoding etc.
hmac	Encoding
hashlib	Encoding
nltk	Splitting URLs into q-grams
bitarray	Creating the Bloom filter
sklearn	Data splitting into training, testing etc.
tensorflow	Tokenizing raw data, sequence padding, model development, deep learning layers and metrics

Figure 5.4: URL Length distribution between 0 and 300



5.1.3 Dataset

The dataset used in this project was an open source dataset provided by [21]. It incorporates 520,285 different URLs which were collected from CommonCrawl for the benign samples and from PhishTank for the malicious samples, respectively. This is a subset of a much larger dataset incorporating 10% of it, which was used in the experiments of [21]. The dataset was already pre-split into subsets for training (70%), validation (20%) and testing (10%).

5.2 Pre-processing

The first step was to merge the three datasets into one, as I wanted to split it myself dynamically. A total of 51,944 duplicates were removed, along with one entry which had no value, which left me with 468,340 total entries. The characters were first set to lowercase, and, using a regex ASCII pattern, I checked and removed any illegal characters that are not found in the ASCII table. Afterwards, I checked the label distribution: 55% of the entries were legitimate whilst 45% were phishing, which means the dataset is relatively balanced. This is a drawback of the system, as a classifier that classifies everything as benign would already be 55% accurate, resulting in a bias. When checking the length distribution, I found out there were URLs with a length of upwards of 6,000 characters, which had to be truncated. I chose a truncation length of 200, which is also the same truncation length in [21]. Furthermore, the sample length distribution between 0 and 300 can be seen in Figure 5.4. As shown in the figure, most URLs are 50 to 100 characters long. In total, 16,213 URLs have a length of over 200, which were truncated as stated above.

5.3 Feature Extraction

To extract relevant features from URLs, several functions were implemented to capture certain characteristics of malicious URLs. They can be put into two categories, binary and non-binary features. The non-binary features are the following:

5.3.1 Non-binary features

URL Length: Measures the total character count of the URL. Usually, phishers often create longer URLs to obscure malicious content.

Host Length: Measures the character count of the host component in the URL. Long hosts may be a sign of phishing attempts.

Path Length: Measures the character count of the path component in the URL. Long paths may also be a sign of phishing attempts.

The average length of each part of the URL based on the label can be seen in Figure 5.5. As shown in the histogram, the average URL length between benign and phishing entries does not differ significantly.

Special character count: For each of the special characters defined by the RFC3986 standard [17], a count of occurrences within the URL was implemented. This included 21 different special characters, where the distribution can be seen in Figure 5.6. Special characters are often found more frequently in phishing URLs. As shown in the histogram, only four of the 21 characters have an average occurrence of one or higher, the rest being less relevant in this use case.

Digit count: This feature calculates the total number of digits found within the URL. URLs with a higher digit count are more likely to be phishing.

Keyword count: For this feature, a list of common phishing keywords was created based on [2, 8], which consists of a total of 61 keywords. The feature counts the occurance of phishing keywords: for each of the 61 keywords found in the URL, the counter goes up by one. The keyword list can be found in Table 5.7. In Figure 5.8 the distribution of keywords and digits can be observed. This feature was later changed to a binary one: if any keyword is found to be in the URL, the feature outputs a 1. This was changed to reduce the number of feature vectors that can be generated, which is explained in more detail in Section 6.2.

Entropy: The entropy of the URL measures the randomness of its character distribution. It is calculated as shown in Equation 5.1, where $p(x)$ is the probability occurrence of each character x in the URL. Usually, URLs with high entropy and therefore high randomness are

Figure 5.5: Average Length of different URL parts based on label

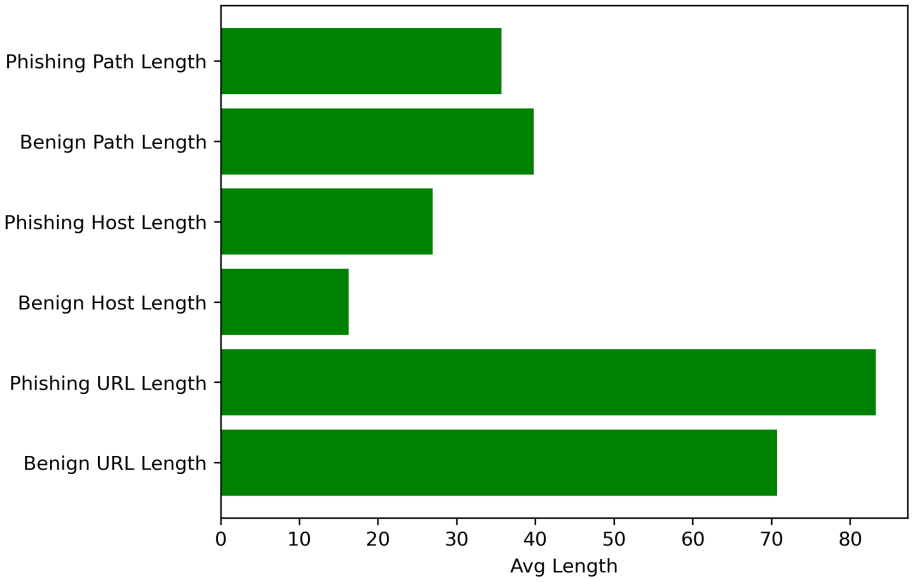
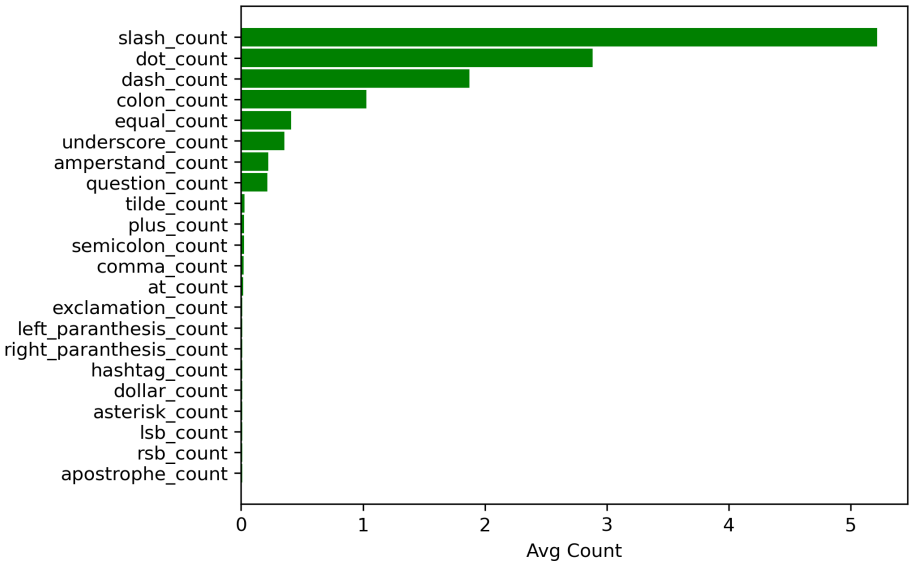


Figure 5.6: Average Count of Special Characters



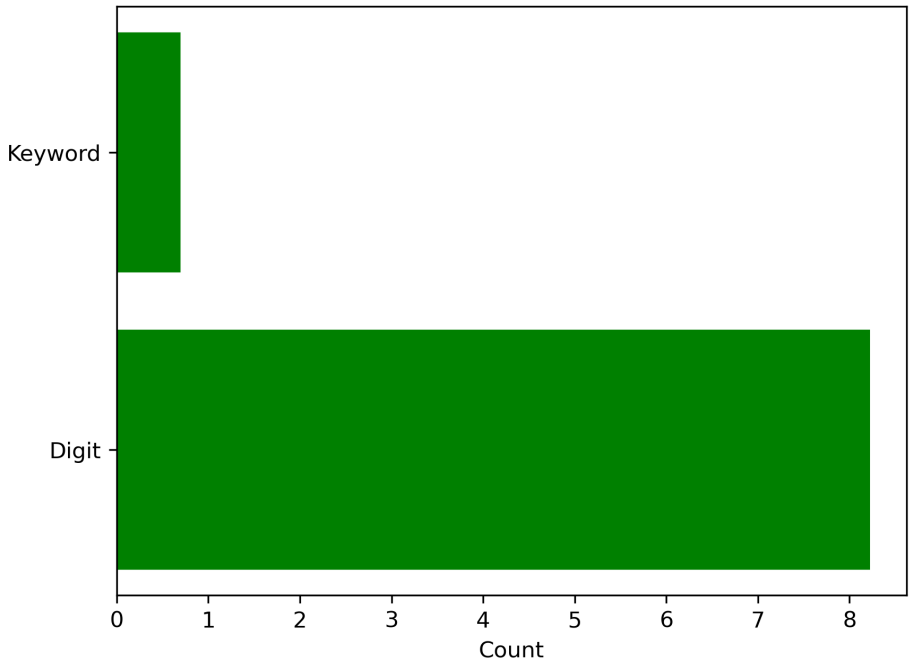
more likely to be malicious, as phishers tend to use random strings in the URLs.

$$\text{Entropy} = -\sum p(x) \cdot log_2 \cdot p(x) \tag{5.1}$$

Table 5.7: Phishing keyword list

limited	confirmation	page	signin	team
sign	access	protection	active	manage
secure	customer	account	client	information
verify	recovery	secured	business	refund
help	safe	bank	event	giveaway
card	user	device	payment	shop
banking	activity	admin	browser	billing
protect	alert	login	gift	security
confirm	info	verification	check	required
store	remove	update	configuration	support
online	provider	services	resolved	home
setup	center	summary	contact	server
solution				

Figure 5.8: Average Count of Digits and Keywords



5.3.2 Binary features

The binary features included are the following:

Presence of IP Address: This feature checks if an IP address appears anywhere in the host section of the URL. Malicious samples may contain IP addresses instead of domain names to avoid domain registration.

HTTP Protocol: Checks if the protocol used is HTTP. Phishers may use http to avoid the costs associated with SSL certificates.

Presence of executable: Checks if an ".exe" is found in the path, indicating a possible executable file. Executables may carry malware and are often found in phishing URLs.

Presence of www or com in path: Checks for the presence of "www" or "com" within the path of the URL. As these are not expected to be in the path, this may be an attempt to imitate trusted domains.

Redirect checker: This feature checks for a double slash "//" after the protocol section, which may suggest the URL is trying to redirect to a different URL. Phishers tend to use redirection to hide the true destination of their URL.

TLD Check: This features checks if the URL's TLD is among the most common ones, including "com, org, de, ru, br, net, uk, jp, it, fr". Phishers may use less common TLDs.

Shortening Service check: Uses regex to identify if the URL uses any known URL-shortening service, such as "bit.ly" or "tinyurl". Phishers often use these services to obscure the true identity of their phishing URL.

5.3.3 Interesting Findings

The feature analysis provided some insight into characteristics of phishing samples versus benign samples. These are some interesting findings:

URL Length : The average URL length is only slightly different between benign and phishing samples, whereas the expectation was a higher difference.

HTTP Protocol: A lot of recent phishing papers state that phishers have started using the HTTPS protocol more recently, making the HTTP protocol checker feature less relevant, as stated by [2, 8, 14]. However, in this dataset, 95% of phishing domains are still using HTTP, whereas only 33% of benign samples use it. This can either be seen as a limitation of this data set, or as an interesting finding: Either, the common assumption that most recent phishing websites use HTTPS is not entirely true, or, the dataset is not that recent.

Square brackets: Not one single benign entry contained any square brackets within their URLs, suggesting that the inclusion of these special characters could be a potential phishing indicator.

Dash distribution: On average, benign URLs tend to use more dashes (-), with an average of 2.71 per URL compared to 0.80 in phishing URLs.

Figure 5.9: Features without keys

url_length	has_ip	is_http	has_top_tld	slash_count	dash_count	dot_count	digit_count	keyword
80	0	0	1	2	0	2	5	0
90	0	1	0	4	0	4	15	1
60	0	1	1	6	2	2	15	1
80	0	1	1	4	4	0	0	0
90	0	1	1	4	0	2	15	0

Figure 5.10: Features with keys

url_length	has_ip	is_http	has_top_tld	slash_count	dash_count	dot_count	digit_count	keyword
ur80	ip0	ht0	to1	sl2	da0	do2	di5	ke0
ur90	ip0	ht1	to0	sl4	da0	do4	di15	ke1
ur60	ip0	ht1	to1	sl6	da2	do2	di15	ke1
ur80	ip0	ht1	to1	sl4	da4	do0	di0	ke0
ur90	ip0	ht1	to1	sl4	da0	do2	di15	ke0

Digit count: Phishing URLs use three times more digits as benign samples (twelve versus four) on average. This could indicate that phishing domains rely on numerical sequences as an obfuscation technique.

Keyword frequency: Phishing URLs contained an average of 1.15 keywords, while benign URLs only included 0.31 on average.

5.3.4 Categorization and labeling

For effective encoding, the features were categorized to reduce the total number of possible values. Without this step, the encoder would encode similar values, such as 79 and 80, differently, although the values are very similar. This also reduces the total number of possible patterns in the Bloom filter, which is important for the neural network. All features were rounded to a nearest multiple, such as three, five, ten etc. with a cut-off which was based on the 75% quartiles.
Another important step here was to add keys to each feature. This is important for the Bloom encoder to be able to distinguish between certain features. For example, if the URL has two binary features that are both set to one, the Bloom encoder would encode both the same way, although they refer to totally different features. Therefore, a key is added to help distinguish between them. A representation of this can be found in Figure 5.9 and Figure 5.10.

5.4 Bloom Encoder

The Bloom encoder is used to convert the numerical and lexical data into binary data. The first approach is done on the raw URL: Firstly, the URL is taken and split into q-grams similarly as in Figure 5.12, but instead of extracting features, the raw URL is split into q-grams. In this use case, the special characters are kept as part of the URL. Then, using a bitarray and hmac the q-grams are converted their into binary representations, as in figure 2.3, but it creates bit patterns for each q-gram in the URL and adds them together. This is done by creating a base key for each input (in this case, q-grams), which is a random hexadecimal pattern. Then, one byte is added at the end of the key to create multiple hash values for each input. For example, if k=4, we can create four different keys for each input by adding the current index of the hash functions to the key. So, h_1 would add 1 to the key, h_2 would add 2 to the key etc. The major drawback of this method is that there is a limit of hash functions that can be used, which is 16, due to the hexadecimal encoding scheme of hmac. This is also not a cryptographically secure way to generate keys, as finding one hash function for one q-gram results in finding every hash function for the specific q-gram, as the adversarial party would only need to add the index to the key. However, as this thesis mainly focuses on the effectiveness of SPBE and does not put the focus on the privacy aspect, this should not make a notable difference in this case. The baseline test uses four hash functions and a Bloom filter size of 512 bits. Figure 5.11 shows the distribution of ones in the encodings using these settings. The minimum is 12 ones, the maximum is 410 and the average is 193.
To encode the feature dataset, the features were put together in an array. Then, using the same encoding function, they are encoded into binary data representing the feature vector. The baseline approach uses a Bloom filter size of 256 and four hash functions. The distribution of the ones can be found in Figure 5.12. Here, the minimum amount of ones is eight, with the average at 66 and the maximum at 84.

5.5 Deep Learning Model

As stated in the proposal, the deep learning model chosen for this project was a convolutional neural network. The architecture of the model was kept the same for all four datasets to ensure no bias, which is described below:

Embedding Layer: This layer is only necessary for the raw URL dataset to map each tokenized input sequence to a vector space.

Convolutional Layers: Two one-dimensional convolutional layers with 128 filters and a kernel size of 3 are used to capture features and patterns within the URLs. The activation function relu is used.

Figure 5.11: Distribution of ones in raw Bloom encodings with q=2, k=4 and l=512

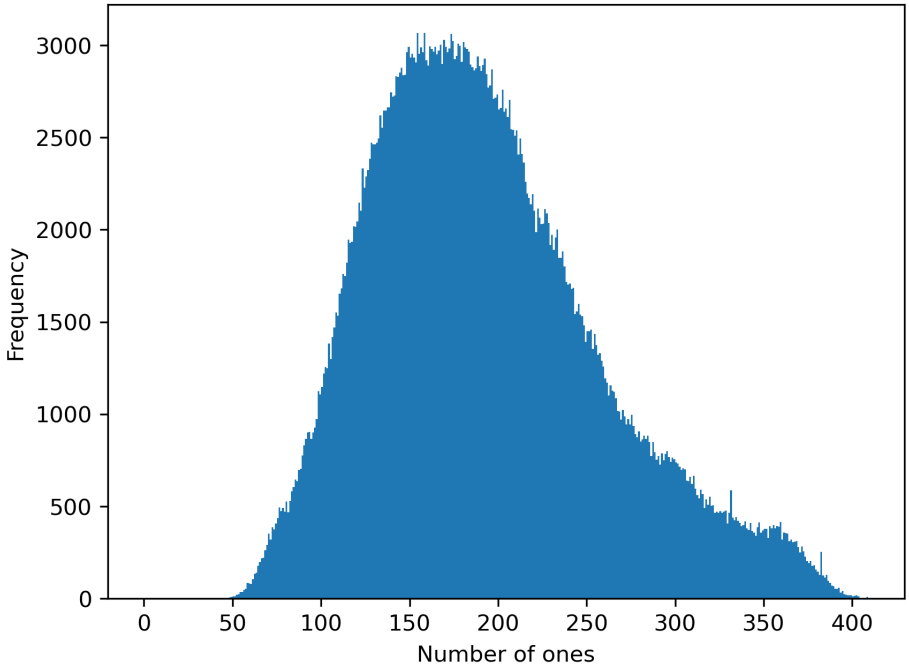
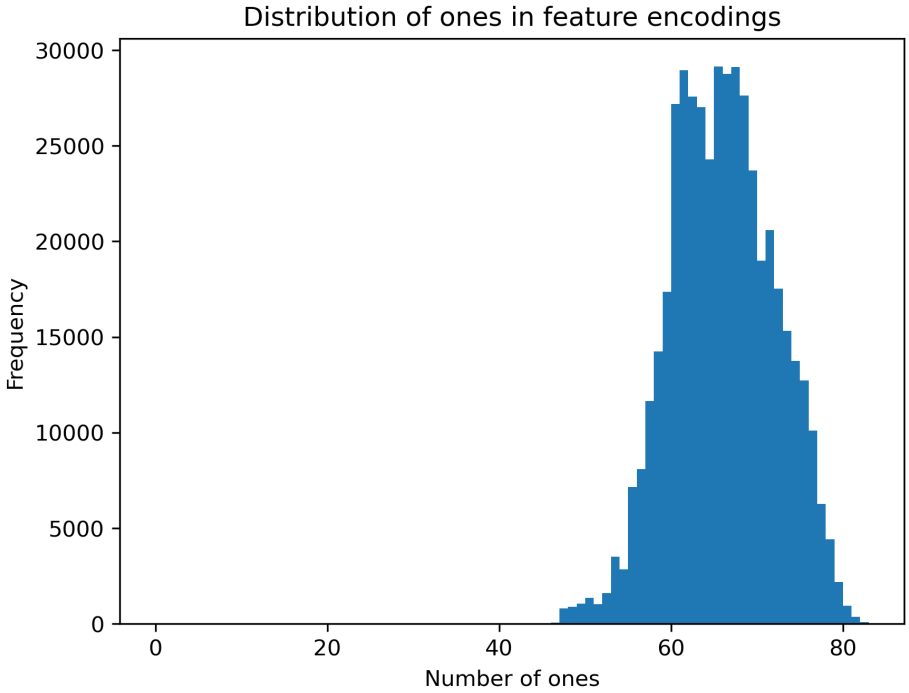


Figure 5.12: Distribution of ones in feature Bloom encodings with k=4 and l=256



Max Pooling Layers: Following each convolutional layer, a max pooling layer is used to summarize features and therefore reduce the computational load.

Dropout Layer: Following the second max pooling layer, a dropout layer with a rate of 0.2 is used to prevent overfitting.

Fully Connected Layers: After flattening the outputs, one fully connected dense layer with 64 units and relu activation is used to capture global patterns, followed by the final output layer with a single neuron and a sigmoid activation function to produce the binary output.

6 Evaluation

This chapter presents the experiments conducted and the results of those experiments. The primary objective was to evaluate the effectiveness of similarity-preserving Bloom encodings in URL-based phishing detection, using two different approaches. The first part of this chapter focuses on the first approach, where the URL is split into q-grams, while the second part focuses on the other approach, where a feature vector is extracted from the URL, which is then encoded. All experiments were executed with identical parameters on the same architecture depicted in Section 5.5. A batch size of 128 was used. The model underwent training for eight epochs and was evaluated after, using a split of 70% training, 20% validation and 10% testing. The model's performance was measured using accuracy, precision, recall, encoding and training time. The key findings are summarized at the end of this chapter.

6.1 Q-gram Approach

As outlined in Chapter 5, this approach involves dividing the URL into q-grams and encoding each q-gram separately, rather than encoding the complete URL as a whole. This technique offers the advantage of preserving similarity between URLs that have similar structures, a feat that can not be achieved if the entire URL is encoded at once.

The objective of these experiments was to evaluate the model's performance in relation to the clear-text approach, which does not involve any encoding. In the initial tests, only one parameter is modified at a time to find out the impact of each parameter on the model's performance. Further tests have changes in multiple parameters. The encoding and training time was measured using the time library in python. The findings are presented in Table 6.1, where q represents the size of the q-gram, k denotes the number of hash functions and l indicates the size of the Bloom filter, as explained in Section 2.6. It is also noteworthy to mention that splitting the URL into q-grams required approximately 7-9 seconds on average for values of $q \in \{2, 3, 4\}$.

The clear-text configuration, which involves no encoding, attained an accuracy of 97.28%. This figure is approximately 7% greater than the accuracy reported in the study [21], using the same dataset. I suspect that this

Table 6.1: Results of the q-gram Approach with different settings

Configuration	**Accuracy**	**Precision**	**Recall**	**Encoding Time**	**Training Time**	**Total Time**
Clear-text	97.28%	96.62%	97.28%	-	438s	7min 18s
Baseline (l=512, k=4, q=2)	94.25%	92.34%	94.94%	804s	1,166s	32min 50s
l=128, k=4, q=2	84.32%	81.98%	82.97%	791s	266s	17min 37s
l=256, k=4, q=2	90.70%	87.06%	92.89%	792s	522s	21min 54s
l=1024, k=4, q=2	**95.28%**	94.97%	94.38%	809s	2,246s	50min 55s
l=512, **k=2**, q=2	94.41%	93.29%	94.21%	410s	1,171s	26min 21s
l=512, **k=8**, q=2	93.41%	91.85%	93.49%	1,585s	1,198s	46min 23s
l=512, k=4, **q=3**	91.83%	89.93%	91.92%	798s	1,636s	40min 34s
l=512, k=4, **q=4**	90.01%	88.73%	88.83%	778s	1,200s	32min 58s
k=1, l=128, q=2	90.18%	88.77%	89.21%	206s	347s	9min 13s
k=2, l=256, q=2	91.72%	87.87%	94.42%	401s	693s	18min 14s
k=8, l=1024, q=2	94.86%	93.13%	95.49%	1,584s	2,448s	67min 12s

improvement stems from several factors including enhanced data cleaning, a more intricate neural network architecture and a reduced batch size.
As illustrated in the table, nearly all configurations achieved at least 90% accuracy, with the exception of the the l=128, k=4, q=2 configuration, which attained an accuracy of 84.32%. This lower accuracy is likely a result of the Bloom filter predominantly containing ones, with an average count of 104. This observation is further supported by figure 6.2, which also indicates that there are nearly 20,000 samples entirely composed of ones, from which the neural network is unable to learn anything from. When the number of hash functions k is decreased from k=4 to k=1, the accuracy increases to 90.18%. In this scenario, the average number of ones drops 48, as depicted in Figure 6.3.
The highest accuracy was achieved with the parameters l=1024, k=4 and q=2, resulting in an accuracy of 95.28%, which is precisely 2% lower than that achieved with the clear-text method. In comparison to the baseline configuration (l=512, k=4, q=2), this represents an increase of approximately 1%. However, the overall time required for encoding, training and evaluation also rose by roughly 18 minutes. When the number of hash functions k was increased from k=4 to k=8, the total time extended by an additional 16 minutes, while the accuracy decreased by about 0.5%, likely due to randomness. Generally, reducing the number of hash functions k led to improved accuracy. This suggests that the system performs better when the Bloom filters are not cluttered and contain a smaller count of ones. This suggestion is further supported by Figure 6.4, which illustrates the distribution of the l=512 dataset with different numbers of hash functions k.
Conversely, increasing the q-gram size resulted in decreased accuracy, likely due to the greater number of patterns that can be generated by the Bloom encoder. There are a total of 58 distinct characters which can occur, comprising 26 lowercase letters, 22 special characters, and 10 digits. With q=2, it is possible to generate $58^2 = 3364$ patterns. Expanding the q-gram size causes the number of patterns to increase exponentially,

Figure 6.2: Distribution of ones in raw Bloom encodings with q=2, k=4 and l=128

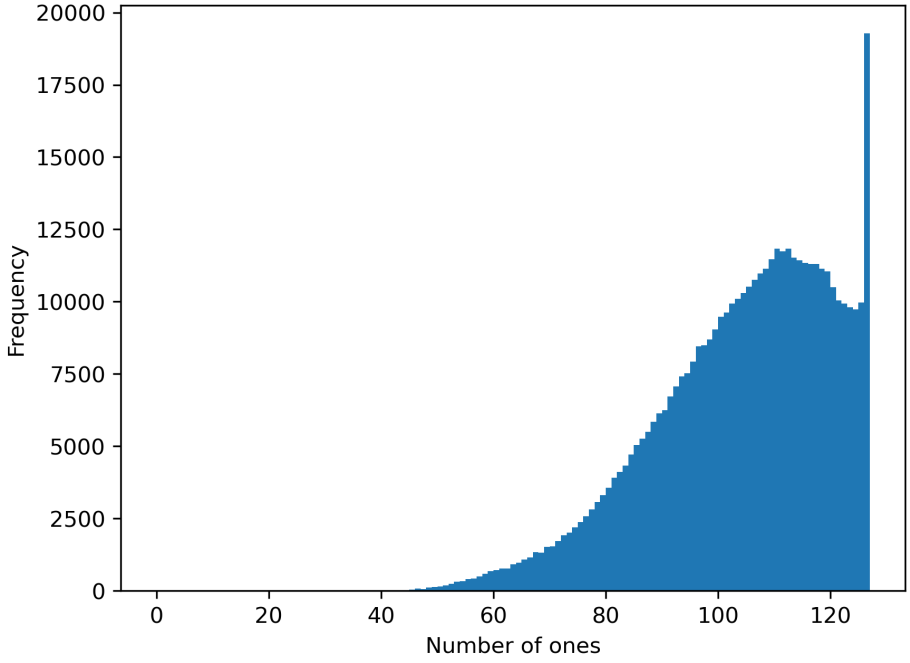


Figure 6.3: Distribution of ones in raw Bloom encodings with q=2, k=1 and l=128

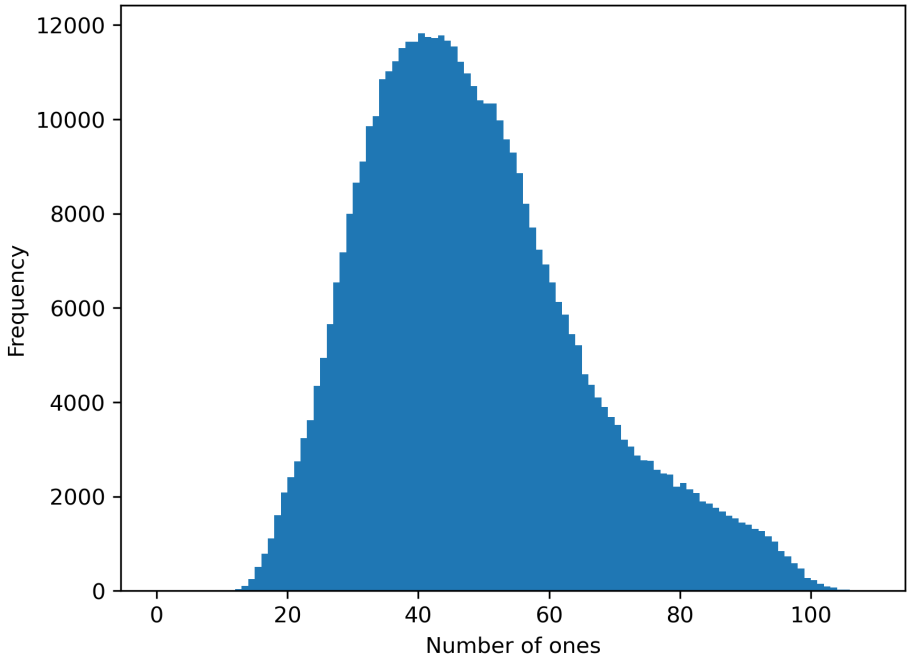
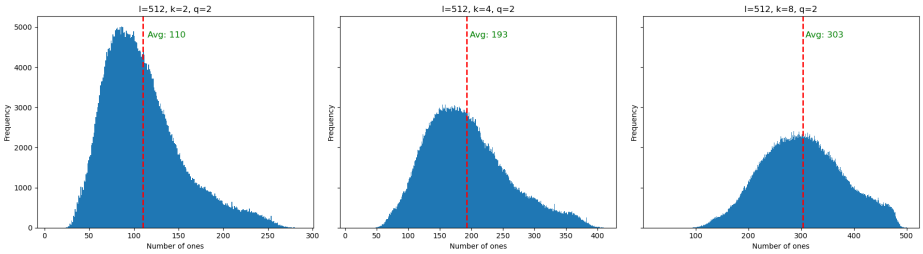


Figure 6.4: Distribution of ones in raw Bloom encodings with l=512, $k \in \{2, 4, 8\}$ and q=2



yielding $58^3 = 195,112$ for q=3 and $58^4 = 11,316,496$ for q=4. Although many of these patterns are unlikely to appear at all, the total number of patterns increases, complicating the neural network's ability to recognize and learn from them. Notably, the encoding time remains the same when adjusting l but varies with changes to k; halving or doubling k results in a corresponding halving or doubling of the encoding time. In contrast, the training time behaves oppositely: it halves or doubles with changes to l, while remaining unchanged with changes to k. These experiments suggest that as Bloom filters grow larger, the returns diminish. Although the total execution time rises significantly, the performance does not correspondingly improve.

6.2 Feature Vector Approach

This method focuses on the utilization of manually extracted features that are encoded and fed into the neural network, rather than splitting the URL into q-grams. The features employed in this approach are detailed in Section 5.3. However, utilizing all the specified features led to subpar performance of the neural network, achieving an accuracy of approximately 58%, which is only marginally better than random guessing. I suspect that this outcome is due to the number of patterns that can be generated from these features. Consequently, it was necessary to implement modifications to reduce the number of patterns. The initial measure involved removing certain features and minimizing the number of categories for each feature. Following these adjustments, only nine features remained, which included URL length, IP address verification, SSL certificate verification, Top TLD verification, Keyword verification, and counters for slashes, dashes, dots, and digits. This reduced the overall number of patterns that can be generated, however, accuracy did not see a significant improvement, reaching only about 65%. I suspect that this limitation was not only due to the quantity of patterns, but also because the feature vector had no varying lengths. The subsequent step I undertook was to not encode features that had a value of zero, thereby allowing for variability in the sizes of the

feature vector. An illustration of this method can be seen in Table 6.5.

Table 6.5: Feature Vector Example when encoding 0's versus not encoding 0's

Approach	**Feature Vector**
Encoding 0's	[ur80, ip0, ht0, re0, to1, ss0, sl3, da0, do3, di5, ke0]
Not encoding 0's	[ur80, to1, sl3, do3, di5]

The data presented in the table indicates a notable change in length following the removal of features with a value of zero. The encoding of these features add unnecessary overhead. However, the results have shown considerable improvement after their removal, as illustrated in Table 6.6. It is also crucial to highlight that the accuracy of the clear-text approach decreased from 90.30% to 86.73% subsequent to these changes. In this approach, however, adjusting the configuration did not significantly affect the accuracy. Typically, the accuracy remained just below 85%, with the highest accuracy recorded being 85.01% when using l=512 and k=8. The only configuration that yielded a lower performance was with l=128 and k=2, resulting in accuracy of 83.46%. The other configurations exhibited only slight variations in accuracy, which can likely be attributed to randomness. The accuracy metrics appear to cut off at around 85%, as no configuration attained significantly higher accuracy levels. I suspect that this is related to the selection of features rather than the encoding method itself. In these experiments, SPBE reduced the accuracy by roughly 2% compared to clear-text. A decrease in performance when performing encoding was expected, as the experiments in [11] also saw a decrease in accuracy of approximately 3% due to encoding. Consequently, it is reasonable to conclude that improving the feature selection could lead to improvements in accuracy of the encoded data. A notable advantage of the feature vector approach is its efficiency when dealing with very large samples. Larger samples require an increase in the size of Bloom filter, which in turn requires exponentially more processing time; however, feature extraction consistently produces a smaller sample, regardless of the URL size.

Table 6.6: Results of The Feature Vector Approach with different settings

Configuration	**Accuracy**	**Precision**	**Recall**	**Encoding Time**	**Training Time**	**Total Time**
Clear-text	86.73%	83.14%	87.87%	-	52s	52s
l=128, k=2	83.46%	76.82%	89.73%	224s	260s	8min 4s
l=256, k=2	84.95%	80.32%	87.63%	224s	516s	12min 20s
l=256, k=4	84.78%	80.91%	86.10%	438s	523s	16min 1s
l=512, k=4	84.80%	79.7%	88.32%	445s	1,137s	26min 22s
l=512, k=8	**85.01%**	80.34%	87.79%	876s	1,226s	35min 2s
l=1024, k=8	84.79%	79.73%	88.23%	879s	2,322s	53min 21s

7 Conclusion

This thesis aimed to assess the effectiveness of similarity-preserving Bloom encodings in the context of URL-based phishing detection, using two distinct approaches: the q-gram based approach and the feature vector approach. A variety of configurations for both methods were examined to identify the trade-offs between accuracy and time efficiency. The methodology chapter elaborates on the system's development and the steps undertaken throughout the process. It details the hardware and libraries employed, the methods of data pre-processing, the features extracted and the needed adaptations to work with the system, including categorization and key-value labeling. Additionally, several noteworthy findings are presented, providing deeper insight into the dataset and individual samples, with an emphasis on the differences between benign and malicious URLs. The Bloom encoder and deep learning model are also thoroughly explained, highlighting their development and the model architecture.

7.1 Key Findings

The evaluation chapter presents the experiments conducted with various parameters for both methods. The findings demonstrate that both approaches preserved URL similarity, as demonstrated by the accuracy achieved on encoded data compared to their clear-text counterparts.

The q-gram method attained an accuracy of 95.28% with the parameters l=1024, k=4 and q=2. Although this figure is 2% lower than the clear-text approach, which achieved an accuracy of 97.28%, it demonstrates that this method preserved the structural similarity between URLs, as it performed well. A slight reduction in accuracy on encoded samples was to be expected, as it was also experienced in the DGA Detection study [11]. Furthermore, this approach emphasizes the diminishing returns associated with increasing the Bloom filter size l or the number of hash functions k, as this leads to a significant rise in processing time.

The feature vector approach initially encountered challenges, primarily due to the excessive number of patterns it could generate and limited variety in the feature vector size. This also resulted in a lack of variety in the encodings as well. Following the changes, such as the reduction of features and categories, and removal of redundant encodings, accuracy

improved to about 85% on most configurations, the highest being achieved with l=512 and k=8 at 85.01%. Nonetheless, this approach still fell short of the q-gram encoding in terms of accuracy, although it proved to be slightly more time efficient. Additionally, the feature vector approach offers a more compact encoding. This may be advantageous when dealing with increasingly larger URLs, as the URL length does not impact the required size of the Bloom filter.
These findings reveal important observations into the trade-offs associated with similarity-preserving Bloom encodings for phishing detection. While the q-gram approach displayed stronger overall performance, the feature vector approach highlights the complexities of manual feature engineering in this use case, while also presenting a more compact solution.

7.2 Limitations and Future Work

The study yielded encouraging results. However, it was not without its limitations. Firstly, the model architecture was not adapted specifically for this application. Only one type of deep learning model was evaluated, specifically, convolutional neural networks. Also, the number of configurations tested could have been higher, as 11 configurations were tested for the method and only six for the feature vector approach. There are also several limitations regarding to the dataset, as it is not perfectly balanced, resulting in a bias as explained in Chapter 5. This dataset also showcases that a lot of phishing websites use HTTP, which could be either a contradiction to the common assumption that phishers have started using more HTTPS, as shown in [2, 8, 14], or a limitation of this dataset.
Future work could focus on optimizing the configuration, aiming to balance accuracy with time complexity. It could also involve testing a broader range of deep learning classifiers, potentially including traditional machine learning methods as well. Furthermore, the encoding process could be refined, possibly by pre-generating patterns for each q-gram and matching them against the URL. This step adds significant time to the process, as it is required to be done prior to training and testing the classifier. By pre-generating the patterns, this step could be eliminated, further improving the time efficiency. Regarding the data, this approach should be tested using a perfectly balanced dataset to remove any bias. Additionally, the system could also be evaluated against actual phishing attacks to determine its performance in a real-world scenario.

7.3 Beyond Phishing Detection

The findings from this thesis demonstrated the ability of SPBE to preserve structural similarities between URLs whilst also helping with privacy preservation. This attribute could potentially be interesting in other use

cases, where preserving patterns in the data and protecting sensitive information is crucial.
In fields such as e-mail detection and malware detection, large volumes of data are processed, where each sample can be very long (for example, full emails or code). In this case, the feature vector method becomes relevant, as it offers a compact encoding method that can deal with large data samples. In both of these two fields, privacy is an important aspect. Malware detection often involves analyzing harmful software or system logs, which should not be leaked, as it could potentially lead to security vulnerabilities. Similarly, in email detection for spam or phishing attempts, the samples may also contain sensitive information that should be kept private. For example, an e-mail could include personal details or confidential business information, therefore, preserving privacy is also of high importance in this use case.
When dealing with sensitive data, for example in healthcare research, it is crucial for researchers to have access to a patient's data, for example for developing new treatments or testing hypotheses, without leaking the individual's personal data, such as their medical history.

7.4 Final Remarks

This study has illustrated the viability of applying Bloom filters for phishing detection, highlighting the effectiveness of this system, which could be used in a PaaS context, as explained in Chapter 4. The results have been promising, however there is need for a more balanced configuration, which is both time efficient and accurate. Pre-encoding of patterns and the exploration of a wider array of classifiers are also suggestions for future work, which may be helpful for optimization. Furthermore, a perfectly balanced dataset should be used to remove any bias.

References

[1] Anti-Phishing Working Group. *APWG 1Q 2022 Phishing Reaches Record High: APWG Observes One Million Attacks Within the Quarter for the First Time in the First Quarter of 2022*. Accessed: 2024-07-29. 2022.

[2] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri. "Machine learning based phishing detection from URLs". In: *Expert Systems with Applications* 117 (Mar. 2019), pp. 345–357.

[3] H. Shirazi, B. Bezawada, and I. Ray. ""Kn0w Thy Doma1n Name": Unbiased Phishing Detection Using Domain Name Based Features". In: *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. SACMAT '18. ACM, June 2018.

[4] Z. Alkhalil, C. Hewage, L. Nawaf, and I. Khan. "Phishing attacks: A recent comprehensive study and a new anatomy". In: *Frontiers in Computer Science* 3 (2021), p. 563060.

[5] Anti-Phishing Working Group (APWG). *Phishing Activity Trends Report, 4th Quarter 2023*. Accessed: 2024-05-27. Feb. 2024.

[6] Anti-Phishing Working Group. *Phishing Activity Trends Report, 1st Quarter 2024*. Accessed: 2024-05-27. 2024.

[7] R. Alabdan. "Phishing Attacks Survey: Types, Vectors, and Technical Approaches". In: *Future Internet* 12.10 (Sept. 2020), p. 168.

[8] H. Tupsamudre, A. K. Singh, and S. Lodha. "Everything Is in the Name – A URL Based Approach for Phishing Detection". In: *Cyber Security Cryptography and Machine Learning*. Springer International Publishing, 2019, pp. 231–248.

[9] S. Gallagher, B. Gelman, S. Taoufiq, T. Vörös, Y. Lee, A. Kyadige, and S. Bergeron. "Phishing and social engineering in the age of LLMs". In: *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*. Cham: Springer Nature Switzerland, 2024, pp. 81–86.

[10] N. Q. Do, A. Selamat, O. Krejcar, E. Herrera-Viedma, and H. Fujita. "Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions". In: *IEEE Access* 10 (2022), pp. 36429–36463.

[11] L. Nitz and A. Mandal. “DGA Detection Using Similarity-Preserving Bloom Encodings”. In: *European Interdisciplinary Cybersecurity Conference (EICC 2023)*. 2023.

[12] M. Alanezi. “Phishing detection methods: A review”. In: *Technium: Romanian Journal of Applied Sciences and Technology* 3.9 (Nov. 2021), pp. 19–35.

[13] K. Jansson and R. von Solms. “Phishing for phishing awareness”. In: *Behaviour & Information Technology* 32.6 (2013), pp. 584–593.

[14] E. S. Aung and H. Yamana. “URL-based Phishing Detection using the Entropy of Non-Alphanumeric Characters”. In: *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services*. iiWAS2019. ACM, Dec. 2019.

[15] M. Khonji, Y. Iraqi, and A. Jones. “Phishing detection: A literature survey”. In: *IEEE Communications Surveys & Tutorials* 15.4 (2013), pp. 2091–2121.

[16] K.-T. Chen, J.-Y. Chen, C.-R. Huang, and C.-S. Chen. “Fighting phishing with discriminative keypoint features”. In: *IEEE Internet Computing* 13.3 (2009), pp. 56–63.

[17] T. Berners-Lee, R. T. Fielding, and L. M. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Jan. 2005.

[18] A. AlEroud and G. Karabatis. “Bypassing detection of URL-based phishing attacks using generative adversarial deep neural networks”. In: *Proceedings of the Sixth International Workshop on Security and Privacy Analytics*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 53–60.

[19] R. M. Mohammad, F. Thabtah, and T. L. McCluskey. *Phishing Websites Features*. 2015.

[20] W. A. Qader, M. M. Ameen, and B. I. Ahmed. “An overview of bag of words: Importance, implementation, applications, and challenges”. In: *2019 International Engineering Conference (IEC)*. 2019, pp. 200–204.

[21] O. K. Sahingoz, E. Buber, and E. Kugu. “DEPHIDES: Deep Learning Based Phishing Detection System”. In: *IEEE Access* 12 (2024), pp. 8052–8070.

[22] G. Naidu, T. Zuva, and E. M. Sibanda. “A Review of Evaluation Metrics in Machine Learning Algorithms”. In: *Artificial Intelligence Application in Networks and Systems*. Springer, 2023, pp. 15–25.

[23] B. H. Bloom. “Space/time trade-offs in hash coding with allowable errors”. In: *Communications of the ACM* 13.7 (July 1970), pp. 422–426.

[24] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz. “Theory and Practice of Bloom Filters for Distributed Systems”. In: *IEEE Communications Surveys & Tutorials* 14.1 (2012), pp. 131–155.

[25] R. Schnell, T. Bachteler, and J. Reiher. “Privacy-preserving record linkage using Bloom filters”. In: *BMC Medical Informatics and Decision Making* 9.1 (Aug. 2009), p. 41.

[26] C. Opara, Y. Chen, and B. Wei. “Look before you leap: Detecting phishing web pages by exploiting raw URL and HTML characteristics”. In: *Expert Systems with Applications* 236 (2024), p. 121183.

[27] Ú. Erlingsson, V. Pihur, and A. Korolova. “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS’14)*. ACM, 2014, pp. 1054–1067.

[28] L. Nitz and A. Mandal. “Bloom Encodings in DGA Detection: Improving Machine Learning Privacy by Building on Privacy-Preserving Record Linkage”. In: *Journal of Universal Computer Science* 30.9 (2024), pp. 1224–1243.

[29] W. Mitchell, R. Dewri, R. Thurimella, and M. Roschke. “A Graph Traversal Attack on Bloom Filter-Based Medical Data Aggregation”. In: *International Journal of Big Data Intelligence* 4.4 (2017), pp. 217–231.

[30] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022), pp. 6999–7019.

[31] G. Van Houdt, C. Mosquera, and G. Nápoles. “A Review on the Long Short-Term Memory Model”. In: *Artificial Intelligence Review* 53.8 (2020), pp. 5929–5955.

[32] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang. “An Empirical Analysis of Phishing Blacklists”. In: *Journal Name* (2009).

[33] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza. “Deep Learning Classifiers for Hyperspectral Imaging: A Review”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 158 (2019), pp. 279–317.