Design and Development of a Machine Learning-Based Framework for Phishing Website Detection

by

Lizhen Tang

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

April, 2022

THESIS EXAMINATION INFORMATION

Submitted by: **Lizhen Tang**

Master of Applied Science in Electrical and Computer Engineering

Thesis Title: Design and Development of a Machine Learning-Based Framework for Phishing Website Detection

An oral defense of this thesis took place on April 6, 2022 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Ramiro Liscano
Research Supervisor	Dr. Qusay H. Mahmoud
Examining Committee Member	Dr. Mohamed EI-Darieby
Thesis Examiner	Dr. Meaghan Charest-Finn

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

Design and Development of a Machine Learning-Based Framework for Phishing Website Detection

Lizhen Tang — Advisor:

Ontario Tech University, 2022 — Dr. Qusay H. Mahmoud

Phishing is a social engineering cyber attack to steal personal information from users. Attackers solicit individuals to click phishing links by sending them emails or social media text messages with deceptive content. With the development and applications of machine learning technology, solutions for detecting phishing links have emerged. Subsequently, performance optimization achieved by machine learning-based approaches were predominantly limited to the datasets used to train the model, such as few open source datasets, poorly characterized data points, and outdated datasets. This thesis introduces a framework based on multiple phishing detection strategies, which are whitelist, blacklist, heuristic rules, and machine learning models, to improve accuracy and flexibility. In the machine learning-based method, three traditional models and three deep learning models are trained and compared the performance of their test results, and concluded that the Gated Recurrent Units (GRU) model achieved the highest accuracy of 99.18%. Furthermore, in the expert-driven heuristic rule-based strategy, seven new HTML-based features are proposed. Finally, a prototype has been developed, with a browser extension to display detection results in real-time.

Keywords: Phishing Detection; Machine Learning; Heuristic Strategy; Chrome Extension

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Lizhen Tang

STATEMENT OF CONTRIBUTIONS

I hereby certify that I am the sole author of this thesis, and I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.

Results from this thesis research have been disseminated in the following publications:

- L. Tang and Q. H. Mahmoud, "A Deep Learning-Based Framework for Phishing Website Detection," in IEEE Access, vol. 10, pp. 1509-1521, 2022. doi: 10.1109/ACCESS.2021.3137636.
- L. Tang and Q. H. Mahmoud, “A Survey of Machine Learning-Based Solutions for Phishing Website Detection,” *Machine Learning and Knowledge Extraction*, vol. 3, no. 3, pp. 672–694, Aug. 2021. Doi: https://doi.org/10.3390/make3030034.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisor Dr. Qusay H. Mahmoud, for his support and guidance throughout my graduate studies. He provided me with continuous guidelines and suggestions all the time that helped me to grow as an independent researcher. I would also like to extend my gratitude to all of the course instructors for their valuable guidance in every step of my learning stage during the graduation period. Finally, I would like to thank family members, and Ontario Tech University for their encouragement and valuable support that has helped me to complete the research successfully.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

LIST OF FIGURES

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

LIST OF ABBREVIATIONS AND SYMBOLS

ASCII	American Standard Code for Information Interchange
API	Application Programmer Interface
AJAX	Asynchronous JavaScript And XML
APWG	Anti-Phishing Working Group
CSS	Cascading Style Sheets
CPU	Central Processing Unit
CNN	Convolutional Neural Networks
CORS	Cross-Origin Resource Sharing
DOM	Document Object Model
DNS	Domain Name System
DoS	Denial-of-service attack
DoD	Doc2Vec model over DOM
DNN	Deep Neural Network
ECMA	European Computer Manufacturers Association
GA	Genetic Algorithm
GRU	Gated Recurrent Units
gTLDs	Generic top-level domains
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
ID	Identity document
ISP	Internet Service Provider
IC3	Internet Crime Complaint Center
KPT	Kaggle and PhishTank
LSTM	Long short-term memory
MitM	Man-in-the-middle attack

MHSA	Multi-head self-attention
NLP	Natural language processing
QR	Quick response code
RNN	Recurrent Neural Networks
SQL	Structured Query Language
SVM	Support vector machines
SMS	Short Message Service
SSL	Secure Sockets Layer
TF-IDF	Term Frequency — Inverse Document Frequency
URL	Uniform Resource Locator
UCI	UC Irvine Machine Learning Repository
W3C	World Wide Web Consortium
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language

Chapter 1

Introduction

Internet services have brought tremendous changes to people's lives. Most online services manage users through a membership system, and individual users need to register and log in to obtain these personalized services. Due to the pandemic that started at the end of 2019, many traditional industries have shifted from offline services to online services, such as catering and retail. Users of the Internet or Netizens have much sensitive information hosted on their devices or in the cloud, such as usernames, account names, passwords, privacy questions, personal information, and credit card information. Cybercriminals obtain this information illegally and forge users to carry out illegal activities on the Internet. The wide adoption of the Internet and network attack techniques have also changed rapidly, which has brought many challenges to network security. According to the methods and forms of network attacks, cybersecurity issues are mainly divided into Denial of Service (DoS), Man-in-the-middle (MitM), Structured Query Language (SQL) injection, zero-day exploit, Domain Name System (DNS) tunnelling, phishing, and malware categories.

Phishing is a cyberattack that uses social engineering to induce people to click on phishing links to intercept sensitive information to steal funds. Social engineering is a soft skill to trick people into visiting phishing links [1]. They abuse users' kindness and trust in brands and institutions to fabricate false information and guide them to phishing websites. These phishing websites are often created by imitating regular URLs with essence elements, such as text descriptions, logos and companies' names. Attackers are best at

forging web pages that require users to submit personal data, such as login pages, payment pages, and password modification pages.

According to various professional phishing analysis reports, phishing has shown an upward trend in recent years, and the number of attacks is enormous. Phishing attacks reached an all-time high in July 2021, with APWG stating a cumulative total of 260,642 attacks that month [3]. Furthermore, the 2020 annual report from the Internet crime complaint center showed that the economic loss caused by phishing attacks was over $54 million [4]. Phish Lab's 2021 quarterly report [2] shows that phishing attackers still mainly use emails as bait, and most emails contain phishing links. The methods of staging phishing sites are still diversified, and most of them use free services and tools. However, in the third quarter, paid domain registrations increased significantly. In addition, more than 65 percent of phishing sites use Legacy gTLDs. For example, more than half of the phishing URLs are .com domains, ranking at the top. Phishing attackers usually buy people basic information from dark net markets, such as name, age, mobile phone number, and occupation. As can be seen from the report data, telecommunication and ISP data still have the highest transaction volume on Dark websites. Then they integrate basic personal information in spoof emails to gain trust.

Anti-phishing strategies involve educating netizens and technical defence. Identifying the phishing website is an efficient method in the phishing life cycle. The list-based solution is about filtering a URL with a collection of legitimate URLs or a set of phishing links. These approaches effectively prevent the reuse of the same phishing website URL, reducing the number of affected users and losses. Rule-based methods are inspired by security expert experience to determine whether a webpage is phishing with rules and

phishing characteristics extracted from URL and web page content. Machine learning-based approaches require a batch of data points to train. There are various traditional models trained with structured features, such as naïve Bayes, linear regression, logistic regression, decision tree, support vector machine (SVM), K-nearest neighbour (k-NN), and random forest (RF). In addition, deep learning-based solutions are skillful in handling unstructured data, such as text, images, etc.

The performance optimization of the machine learning-based solution is mainly to improve the quality of training data, such as mining new phishing risk characteristics, increasing the amount of data, and cross-training tests with different source data sets. Therefore, the framework presented in this thesis supports access to multiple data sources, creating datasets with mixed sources, flexibly setting the size of data sets, and updating data daily. In addition, the gated recurrent units (GRU) model obtained the highest accuracy after comparing the results of different models. Finally, a web browser extension is implemented as a client-side tool to detect phishing risks in real-time.

1.1 Contributions

The main contribution of this thesis is a framework for detecting phishing websites based on multiple strategies. The vision of this thesis is that the framework designed, implemented, and evaluated will serve as a prototype for building a real-time phishing detection tool. To this end, the contributions of this thesis are:

- Design of a framework for phishing detection based on multiple strategies.
- Implementation of a Chrome extension for capturing web pages' content and displaying phishing risk information.

- Development of a web application for receiving users' feedback and providing HTTPS service.
- Design and development of three traditional machine learning models and three deep learning models to obtain the best performance model.
- Introduction of seven new features by empirical analysis of phishing characteristics based on HTML tags applying to the heuristic method.

1.2 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 surveys the related work discussing important background information on the concepts relevant to this work. Chapter 3 describes the design of the phishing detection framework explaining the role of each module. Chapter 4 describes the implementation of the framework. Chapter 5 discusses the evaluation of the framework. Chapter 6 concludes the thesis and offers ideas for future work.

Chapter 2

Background and Related Work

This chapter provides information on the phishing background by explaining a phishing attack life cycle and several anti-phishing methods. In addition, this chapter presents the topic of detecting phishing websites, followed by various proposed solutions and frameworks for achieving high performance. Following this, several main opportunities and challenges of phishing detection are introduced.

2.1 Phishing

Phishing is a typical cyberattack performed by sending an email or a message to deceive recipients by visiting a bogus page and then collecting users' sensitive data, such as usernames, passwords, and credit card numbers, for financial gain [71].

Figure 2.1 demonstrates the phishing life cycle. First, an attacker creates a phishing website similar to a legitimate website. On the one hand, attackers used spelling mistakes, similar alphabetic characters, and other methods to forge the URL of the legitimate website, especially the domain name and network resource directory. For example, the link "https://aimazon.amz-z7acyuup9z0y16.xyz/v" (accessed on 9 May 2021) imitates https://www.amazon.com. Although users on the computer can see the URL address by moving the mouse to the clickable link, it is difficult for the average user to identify these URLs with the naked eye and memory as imitating legitimate URLs.

On the other hand, imitation of web content is also a critical point. Typically, attackers use scripts to obtain logos, layouts, and text from genuine web pages. Form

submission pages that require user input of sensitive information are most often faked by cybercriminals, such as login pages, payment pages, and find password pages.

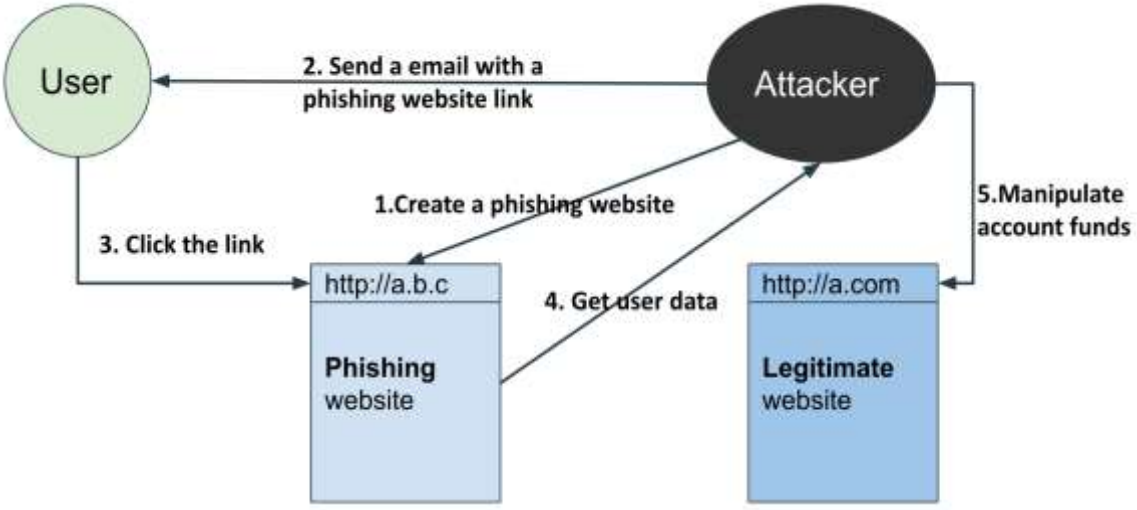

Figure 2.1: Phishing life cycle

The second step is sending an email that firmly guides readers to click on the link. The phishing links are not only sent by email but also by SMS, voice message, QR codes, and spoof mobile applications [5]. With the widespread use of smartphones and social media, the number of channels for criminals to spread false information has increased. Through these channels, text and pictures are usually used to trick readers into clicking on a link. For example, an attacker imitates a customer service representative of a telecommunications company to send an email urging users to pay to prevent downtime. Although scam emails are sent randomly, there is always a small number of users with weak defensive awareness who will be deceived. In step two, the attacker applied social engineering methods, including psychological manipulation, to trick users into making security mistakes. Perpetrators are good at building a sense of fear and urgency and gaining the user's trust via text messages. Afterward, the user clicks the link that will direct them

to open a fake website. Particularly, real URL strings are hidden before redirecting to web browsers on mobile phones.

The next step is collecting personal information on the phishing website, which looks like a real company or organization's web page, by using a similar logo, name, user interface design, and content. When users submit sensitive information to web servers that attackers build, criminals will receive all the data. The last step is stealing the user's account funds by using the victims' real information to imitate their requests on real websites. Some individuals use the same usernames and passwords for multiple websites allowing the attacker steals multiple accounts from the victim. Some phishers use stolen data for other criminal activities. Since the first phishing technique was recorded in a paper in 1987 [67], phishing methods have changed with the development of the Internet. For example, when online payment became popular, attackers began to target online payment systems. According to the 2020 Internet Crime Report, the Internet Crime Complaint Center (IC3) received 791,790 cyberattack complaints, of which phishing scams accounted for approximately 30%, becoming the most complained about the type of cybercrime and causing more than USD 54 million in losses [2].

2.2 Anti-Phishing

There are five steps before an attacker steals money from the victim's account or uses the information for other attacks (Figure 2.1). Blocking any step could stop a phishing attack. Four technologies are listed below.

2.2.1. Web Scraping

For step one, although it is hard to prevent perpetrators from creating web pages, techniques could increase their costs. Attackers will use scripts to write crawlers to obtain legal web pages' content automatically and then intercept useful information and copy it to phishing web pages. Therefore, legitimate websites could prevent web scraping by implementing several techniques concerning obfuscation, using CSS sprites to display essential data, and replacing text with images.

2.2.2. Spam Filter

Spam filtering techniques are used to identify unsolicited emails before the user reads or clicks the link. Some mainstream email services have integrated spam filtering components, such as Gmail, Yahoo, Outlook, and AOL. The initial filters relied on blacklists or whitelists and empirical rules. With the development of artificial intelligence technology, some filters also integrate intelligent prediction models based on machine learning to filter out spam that is not on the list. For example, Gmail could block approximately 100 million extra spam emails daily with the machine learning-based spam filter [6].

2.2.3. Second Authorization Verification

After the attacker obtains the user's sensitive data, the next step is to use the data to log into the legitimate website, operate the account, and steal funds. Therefore, when the website detects that the IP address and device information of the user who is logging in does not match the commonly used information, it becomes crucial to add steps to verify

the authenticity of the user. Usually, the extra verifications are dynamic and biological, such as facial movement, expression recognition, or voiceprint recognition.

2.2.4. Detecting Phishing Websites

When users visit a phishing web page that looks like a legitimate website, many people do not remember the legitimate website's domain name, particularly for some start-ups, so users do not recognize the phishing website based on the URL. Some web browsers integrate a security component to detect phishing or malware sites, such as Chrome, which will display warning messages when one visits an unsafe web page. Google launched Google Safe Browsing in 2007, and it has been integrated into many Google products, such as Gmail and Google Search. Google Safe Browsing is a security component based on a blacklist that contains malware or phishing URLs [7]. In addition, there are several web browser extensions for detecting phishing websites available. However, the blacklist or whitelist-based solutions are invalid for unknown phishing websites. Fortunately, the rapid development of artificial intelligence technology has brought new tools and solutions to detecting phishing attacks. A predictive model based on machine learning can identify phishing links that are not on the whitelist and circumvent existing rules.

2.3 Related work

This section provides details on the related work done on detecting phishing websites. The methodologies of detecting phishing websites are developed and are divided into four categories: list-based, heuristic, machine learning-based and deep learning-based methods.

2.3.1 List-Based Approaches

The list-based method includes a white-list filter and a black-list block. A white list is a collection of legitimate URLs. A blacklist is a set of phishing websites, which can be implemented in two ways: establishing a local database and crawling data from other phishing data sources; invoking a third-party list service. The list-based approach is efficient and accurate. Keeping the list up to date can effectively control multiple victims of the same phishing link attack. The drawback is it is impossible to detect zero-day phishing links.

Jain and Gupta proposed an auto-updated, whitelist-based approach to protect against phishing attacks on the client side in 2016. The experimental results demonstrate that it achieved 86.02% accuracy and less than a 1.48% false-positive rate, which indicates a false warning for phishing attacks. The other benefit of this approach is the fast access time, which guarantees a real-time environment and products [8].

2.3.2 Heuristic Methods

A rule-based heuristic strategy is about extracting features from a URL and HTML source code and creating several rules to infer phishing risks. This approach is driven by expert experiences. Tan et al. introduced a phishing detection approach named Phish WHO, which consists of three phases. First, it obtains identity keywords by a weighted URL token system and ensembles the N-gram model from the page's HTML. Secondly, it puts the keywords into mainstream search engines to find the legitimate website and the legal domain. Next, it compares the legal domain and the target website's domain to determine if the target website is a phishing website or not [9]. Chiew et al. used a logo image from the website to distinguish if the website was legal [10]. In this paper, the authors extracted

a logo from web page images by support vector machine (SVM) classifier and then queried the domain via the Google search engine with a logo as a keyword. Therefore, some researchers also called this category search engine-based approach.

2.3.3 Machine Learning-Based Solutions

Machine learning-based countermeasures are proposed to address dynamic phishing attacks with higher accuracy performance and lower false-positive rates than other methods. Consequently, the machine learning approach consists of six components: data collection, feature extraction, model training, model testing, and predicting. Feature extraction and selection are the foundation for many solutions. Table 2.1 lists the widely used feature categories as well the source of features and applicable models and techniques.

Xiao et al. [58] proposed a hybrid model to detect phishing URLs. They used a regular Convolutional Neural Network (CNN) to extract features from a URL string. Furthermore, they applied multi-head self-attention (MHSA) technology to figure out the relationship among characters, then obtain the weights of features. The experimental results demonstrated that the CNN-MHSA model obtained the highest accuracy of 99.84%. Sahingoz et al. [59] applied Natural Language Processing based features and word-level features into several machine learning models in terms of a decision tree, Support Vector Machine (SVM), k-nearest neighbour algorithm, AdaBoost algorithm, Naïve Bayes, and Random Forest. After a vectorization process, they extracted 1701 words features among 73,375 URLs. In addition, they create 40 Natural Language Process (NLP) features based on a URL string. In the NLP phase of Sahingoz et al. work, a URL was separated into several words by special characters, such as ".","/", then based on the analysis of these words, these 40 attributes were derived, such as the longest number of vocabulary

characters, the shortest number of word characters, and the number of second-level domain names. After executing a feature selection, 104 features were left. The comparable results of their research showed that the random forest model obtained the highest performance.

Table 2.1. Widely used feature categories

Category	**Source**	**Applicable model or Technology**
A sequence of characters	URL string	Deep Neural Networks
Words vectors	URL string	All[1]
Vocabulary statistical features	URL string	All[1]
Symbols and components of the URL	URL string	All[1]
Website credibility features: webpage rank, domain registration information, Google index	HTML & third-party services	All[1]
HTML tag statistical features	HTML	All[1]
Images in the webpage source code	HTML	Computer vision

All[1]: SVM, Random forest, Logistic regression, Decision tree, CNN, DNN, RNN.

In 2021, Ozcan et al. [60] used these 40 NLP features to train their hybrid model. They combined a Deep Neural Network (DNN) model and a Bidirectional Long Short-

Term Memory (BLSTM) model to expose phishing websites. Furthermore, they extracted character-level features from a URL and fed them to a character embedding layer in the LSTM architecture. Finally, they achieved a high accuracy of 98.79% with the complex hybrid deep learning model. Odeh et al. [61] extracted and selected 30 features based on URL strings and web content. The features are classified into four categories in terms of the address bar, abnormal, HTML & JavaScript, and domain-based. Some features depend on third-party services. On the testing process with a 30% data set, the AdaBoost model obtained an accuracy of 98.9%.

Liu and Fu [62] introduced a novel unsupervised machine learning algorithm to classify websites into benign and phishing. They created a web link network with nodes standing for websites and edges between nodes representing reference relationships. A reference relationship is determined via hyperlinks and similar textual content from one web page to another. Besides, they integrated the URL information of each node and used the embedding technique to transform the graphic network into a low-dimensional vector space. It finally predicted whether a web page is phishing or not by calculating the similarity in the neighbourhood.

Feng et al. [63] proposed a DOM tree structure-based method to cluster a web page as a phishing or legitimate category named DoD. The authors used a Doc2Vec model to vectorize the DOM tree and calculate the semantic similarity, then imported a hierarchical algorithm to cluster web pages. They collected a data set from PhishTank [18] and Alexa [17]. The experimental results show that DoD obtained a higher true-positive rate of 89.9% than other DOM-based clustering algorithms. However, the clustering time is much longer than other methods, even over five times.

Barraclough et al. [64] presented a novel feature list for detecting e-banking phishing websites. They combined blacklist-based, web content-based and heuristic-based methodologies to extract 3000 features. In addition, they trained five models to compare the performance with a custom data set of 30500 instances. Finally, the proposed solution achieved a high accuracy of 99.33% and a short prediction time of 0.006 seconds. However, the performance of a high error rate of 0.66% indicates that overfitting probably occurs during the training process.

In 2021, Gupta et al. developed a lightweight phishing detection approach and achieved 99.57% accuracy with the random forest algorithm [11]. The authors extracted 19,964 instances with nine lexical features from the ISCX-URL-2016 dataset published by the University of New Brunswick [12]. The ISCX-URL-2016 dataset contains more than 35,300 legitimate URLs and approximately 10,000 phishing URLs taken from an active repository of phishing sites https://openphish.com (accessed on 18 July 2021). To balance the distribution of the two classes, the authors randomly filtered 10,000 benign URLs and 9964 phishing URLs. Furthermore, the Spearman correlation algorithm and K best algorithm are applied to determine the feature importance. Based on other previous research, nine lexical features from URLs were proposed in the paper. Afterward, they cleaned the data by replacing the null and unlimited values with mean values and normalized them by scaling the values between 0 and 1. Normalization is one of the important data preprocessing procedures to guarantee that one feature is not dominated by others. In addition, they used a one-hot encoding algorithm to transfer the labels to numerical values. Once the dataset is regularized, it is divided into a training dataset and a testing dataset with eight-to-two ratios. In the process of modelling, they compared four

single classifiers with the performance and computational time. Finally, it was concluded that random forest had the highest accuracy rate and the lowest false positive rate. However, in terms of response time, SVM performed better.

2.3.4 Deep Learning-Based Solutions

Deep learning is a subset of machine learning which is built with deep structured architectures. There are some commonly used deep learning algorithms, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and long short-term memory (LSTM) networks. With the rapid development of natural language processing (NLP) and deep learning algorithms, various deep learning-based solutions are introduced for phishing detection.

Ali et al. developed an intelligence phishing detection model which combined deep neural networks (DNNs) and genetic algorithms (GAs) [13]. A DNN is a well-known deep learning technique with more than two hidden layers, an input layer and an output layer, commonly used to classify multiple labels from big data. The GAs is inspired by the biological evolution of the genes in nature and is widely used for optimization problems that aim to minimize or maximize the value of objective functions under some constraints. In this approach, the authors regarded the problem of feature selection as an optimization problem. Mathematically speaking, the objective function minimizes the number of features, and the constraint function is the accuracy of the classification model. Meeting performance requirements with minimal features reduces the model training time and could remove the noisy data. Therefore, the GA was applied to find the optimal subset of features by computing the accuracy of the DNN model in each generation. A chromosome represents a group of features, and each gene with a binary value stands for each feature,

where one is for selecting this feature and zero is not. The classification phase used the selected features as input features and the UCI dataset [66] as a training dataset to fit the DNN model. However, the GA-DNN model got a relatively low accuracy result, which was 89%. It is known that hyperparameters and the size of a training dataset significantly affect the performance of deep learning models [14].

In 2020, Aljofey et al. proposed an efficient convolutional neural network (CNN) model for phishing detection only based on URLs [15]. They extracted character-level features from the original URLs, which were collected from different phishing websites and benign websites. The experimental results showed that this model obtained an accuracy of 95.02% on their own dataset with 318,642 instances. Wang et al. introduced a fast model called PDRCNN that used the URL string as an input, extracted features by an RNN and CNN, and then classified them with the Sigmoid function [16]. The authors collected approximately 500,000 instances from Alexa.com [17] and phishTank.com [18] and extracted semantic features based on the word embedding technique, encoding the URL string to a tensor, an input of the RNN model. A bidirectional LSTM network algorithm implemented the RNN architecture to extract global features, which were the inputs of the convolutional neural network. The final one-dimensional tensor represented a group of features generated through multiple convolutional and max-pooling layers. Finally, the one-dimensional tensor was fed into a fully connected layer with a sigmoid function to classify the original input URL into the fake and phishing website. The experimental results illustrated that they achieved 95.97% accuracy.

2.3.5 Products for Detecting Phishing Websites

The goal of anti-phishing research is to prevent individual Internet users from suffering phishing attacks. With the development of anti-phishing research, phishing attackers are constantly updating their technology. Phishing links cannot be well recognized by naked eyes, and individual netizens need tools to help identify them. Many researchers naturally think of expanding on the browser.

In 2020, HR et al. built a web browser with a phishing detection component [19]. The regular web browser had two core engines: a browser engine and a render engine, which are responsible for connecting to the Internet to fetch the web page via the URL, parsing the web page by XML, HTML, CSS, JAVASCRIPT interpreters, storing cookies, etc. The proposed browser added an intelligent engine to detect phishing websites between the browser engine and render engine. When a user inputs a URL, the intelligent engine starts to predict whether the target website is a phishing website and afterward sends the result to the render engine. If the predicted result showed a phishing website, the render engine would display a warning message to the user through a browser interface. This paper used the random forest algorithm to train the model, and it obtained 99.36% accuracy and a 0.64% false-positive rate on the UCI dataset [66] with 30 rule-based features.

Furthermore, a phishing detection web browser extension is implemented easier than a comprehensive anti-phishing browser. Armano et al. introduced a real-time client-side phishing prevention solution [20]. The approach contains a built-in JavaScript frontend and a built-in Python backend. The front-end collects the web page source code and handles the user interface and interaction with the backend, analyzing the website and predicting if the page is a phishing website. The backend consists of a disputer for checking

against the whitelist, a phishing detector for predicting the website's legitimacy, and a target identifier to find the legitimate website relevant to the input URL based on the logo, keywords, and other content. The phishing detector is implemented by an existing solution that uses the gradient boosting algorithm as the classifier [21]. The authors experimented with 200 phishing websites to monitor the response time. The results showed that the response time for a phishing URL was longer than a legitimate one, which was approximately 2 s, and the appearance of the alert cost occurred in less than 500 milliseconds. In addition to the framework mentioned in academic papers, there are also several published Internet products. Several widely used products are listed in Table 2.2.

2.3.6 Solutions Comparison

Each of the four types of solutions has advantages and disadvantages. Table 2.3 shows four widely used phishing detection methods, as well as their strengths and weaknesses. The rule-based policy logic is simple, but zero-day phishing links cannot be detected, and there is no prediction function. Heuristic rule-based solutions are like experts who are familiar with phishing attacks and defences, but features and rules are prone to failure over time when an attacker cracks them. Machine learning-based approaches improve the accuracy of decision-making, but they require high quantity and quality training data. Deep learning models can effectively prevent features and rules from being cracked and exploited by attackers, but the complexity and time cost of training such models are high. Every detection tool and product applied to a real scene needs to balance accuracy, false-positive rate, and time performance. As a result, many anti-phishing solutions often combine multiple strategies.

Table 2.2. Released phishing detection products

Name	Type	Devices	Techniques	Advantages	Shortcomings	Users
Phish Detector [22]	Web browser extension	Chrome	Rule-based	Zero false-negative alarms	Only for online-banking web sites	2000+
Netcraft Extension [23]	Web browser extension	Chrome	Blacklist-based	Multiple features, including coronavirus-related cybercrime.	New phishing attacks cannot be prevented	50,000+
WOT [24]	All	Browser Mobile PC	Blacklist + machine learning algorithms	Multi-platform security service	Charged	1,000,000+
Pixm Phishing Protection [25]	Web browser extension	Chrome	Deep learning algorithm	Advanced anti-phishing solution (AI)	Charged	1000+
Sharkcop [26]	Web browser extension	Chrome	SVM algorithm	New attacks can be detected Few features are used	The project is currently on hold Feature extraction relies on third-party services, such as domain age	-
PhishFort [27]	Web browser extension	Chrome Firefox	Blacklist-based	Free	New phishing attacks cannot be prevented	2000+

In addition, there are many challenges and difficulties in moving from laboratory research to real-time production. Effective phishing detection solutions should combine new data constantly for recognizing fresh rules and training machine learning models. Phishing and anti-phishing are always in the process of confronting each other. Attackers will adjust the generation of phishing links according to the published anti-phishing rules and methods. Likewise, anti-phishing needs to optimize models and algorithms based on new phishing data. Furthermore, the performance of machine learning-based solutions highly depends on the quality of the training dataset in terms of size and validation. The published datasets are small datasets that do not satisfy the demands of deep learning approaches. According to the power law, deep learning performance keeps rising with the increase of the training data size [65]. Therefore, pulling phishing URLs and legitimate URLs from websites is recommended. However, this depends on the stability of the third-party services or websites.

Furthermore, some features and rules are extracted from URL strings depending on third-party services, for instance, the published rules [66]. Using third-party services might cost extra time and bring unstable issues. In addition, since tiny URLs do not present the real domain, resource direction, or search parameters, rule-based feature selection techniques might be useless for tiny URLs. Due to tiny URLs generated by different services, it is hard to convert them to original URLs. Furthermore, tiny URLs are short strings that are challenging for natural language processing to extract character-level features. If tiny URLs are not specially processed during data cleansing and preprocessing, they are likely to cause false or missed alarms. Internet products are also essential in terms of user experience, and users are also sensitive to false alarms of Internet security products.

Table 2.3. Phishing websites detection strategies

Phishing detection strategies	**Advantages**	**Disadvantages**
List-based	It accurately and efficiently identifies known phishing links.	It couldn't recognize zero-day phishing URLs.
Heuristic rule-based	Low time complexity; Interpretable.	Easy to be cracked by attackers.
Machine learning-based	High performance.	Performance is highly dependent on structured training data
Deep learning-based	It is not easy to be cracked by attackers; A large number of features can be extracted.	High training time and high space complexity; Less interpretable.

Rule-based models depend on rule parsing and third-party services from a URL string. Therefore, they demand a relatively long response time in a real-time prediction system that accepts a single URL string as an input in each request from a client. Phishing attacks spread to various communication media and target devices, such as personal computers and other smart devices. It is a big challenge for developers to cover all devices with one solution. Language independence and running environment independence should be taken into consideration to reduce system development complexity and late maintenance costs.

This thesis presents a phishing detection framework based on multiple strategies to improve performance of accuracy and reduce response time. In addition, optimization machine learning models can be done automatically by training with an updated dataset. A prototype has been implemented with a Chrome extension and a web application to evaluate the efficiency of multiple strategies in real-time.

2.4 Summary

In this chapter, the life cycle of phishing attacks and the characteristics of each node is introduced. Afterward, the state-of-the-art solutions for detecting phishing websites are presented in terms of list-based methods, heuristic strategies, machine learning-based solutions, and deep learning-based approaches. In addition, the pros and cons of each solution type are analyzed, and the challenges of anti-phishing products are listed. The design of the proposed framework is presented in the next chapter.

Chapter 3

Framework Design

This chapter describes the design aspects of the proposed machine learning-based framework and its architecture. To this end, the chapter first presents the overview of the architecture and describes the inspiration of the work. Furthermore, the details of data collection as the fundamental module is explained. Along with this, the details of machine learning-based method and heuristic rule-based strategy are presented. Finally, a prediction service with multiple strategies is introduced to detect phishing risk in a real-time environment.

3.1 Architecture

The major motivation behind the proposed machine learning-based framework is to develop a tool for detecting phishing websites in real-time with high efficiency, high accuracy, and low false alarms. In a real-world environment, phishing links account for a very small percentage of all network requests. This framework provides a multi-strategy converged prediction service based on a machine learning model, where the list-based strategy can be quickly filtered, reducing the average response time of requests and reducing the overall false alarm rate. In addition, the heuristic-based strategy narrows the phishing web page to a page with a confidential form, improving the overall accuracy rate. Finally, a machine learning-based model predicts phishing risks.

Figure 3.1 depicts the architecture of the proposed framework, which contains a data collection module, machine learning progress, heuristic strategy, and a prediction service. For data persistence, a database is built to store URLs collected from various data

sources. On the one hand, those data are filtered to generate blacklists and whitelists, which are used as the list-based strategy of the prediction service. On the other hand, datasets used for machine learning model training and testing are also generated from this database. The effectiveness of this framework is based on several assumptions. First, phishing is done by asking users to submit sensitive information on an active web page, and the way virus-ridden files are automatically downloaded when a connection is opened is beyond the scope of this article. Second, the ability to reduce bias in cross-training models from multiple data sources is limited to low data overlap between multiple data sources.

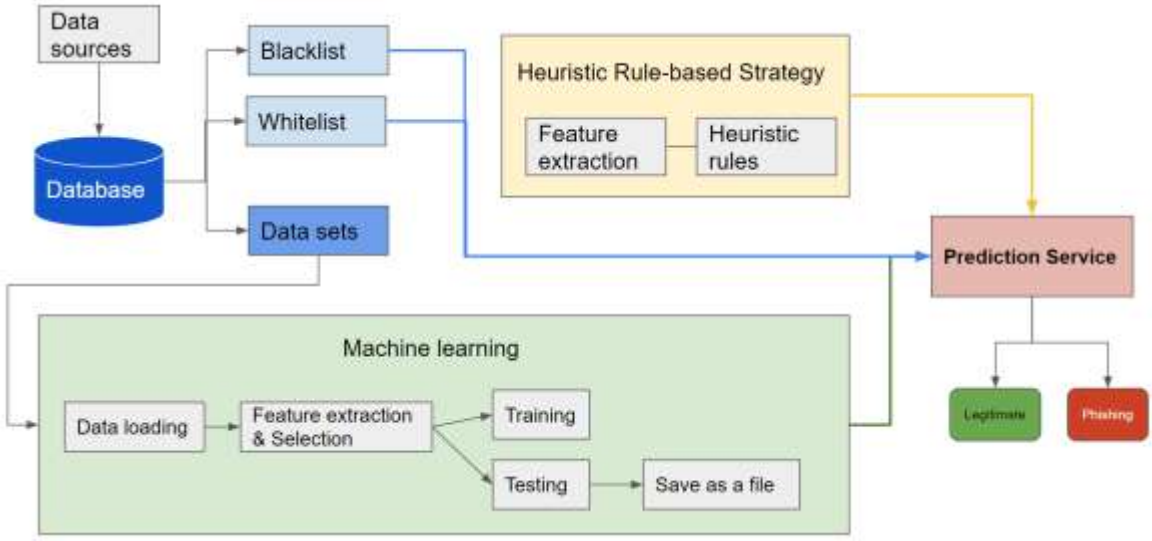

Figure 3.1: Architecture of the multi-strategy framework

3.2 Data Collection

Data is the core of the field of machine learning. The quality and quantity of data significantly impact the performance of machine learning-based modules [28]. The data collection module is the foundation of this system. Figure 3.2 presents the details of the data collection module.

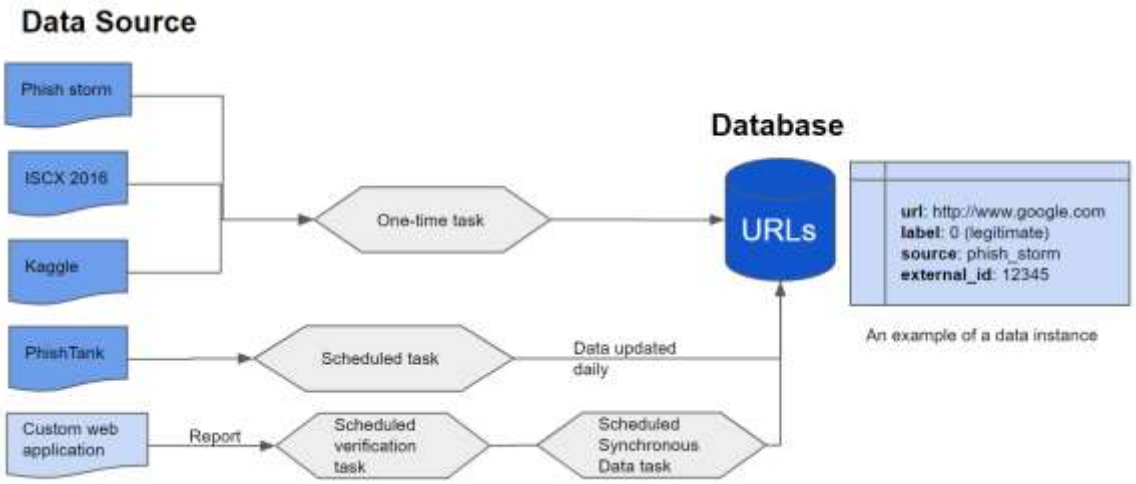

Figure 3.2: Data collection module

Data was collected from different open sources, shown in Table 3.1. The PhishStorm [29] dataset contains 96,018 URLs: 48,009 legitimate URLs and 48,009 phishing URLs. The ISCX-URL2016 [12] dataset contains 35378 legitimate URLs and 9965 phishing URLs. Data was loaded around 350,000 benign URLs from an open Kaggle project [30]. In addition, 400,000 data points were collected, and new data was grabbed from the PhishTank platform [18] every day.

Table 3.1. Data sources

Data source	Legitimate URLs	Phishing URLs
PhishStorm [29]	48,009	47,902
PhishTank [18]	0	178,495
ISCX-URL2016 [12]	35,378	9,965
Kaggle [30]	345,738	0
Total	429,125	236,362

The URL's basic structure was analyzed and extracted into several parts in terms of protocol, domain, and path. Table 3.2 presents the major fields of a table named URL. Data points are stored in a relational database, as it is flexible and efficient for providing data services by reading based on SQL. These data services can combine multiple data sets. For example, select 20,000 phishing links from PhishTank and 20,000 good links from Kaggle, and combine them into a balanced data set with 40,000 instances.

Table 3.2. Table structure for the data table named "URL"

Name	Description	Example
url	URL	https://amazom.mhmgmm.rest/mobile/
label	1: phishing; 0: legitimate	1
source	Data source	PhishTank
External id	Unique ID of the same data source	7270002
netloc	netloc	amazom.mhmgmm.rest
Gmt_created	Created date	2021-08-21 01:39:40

3.3 Machine Learning

The machine learning module is mainly responsible for model training and model testing. In this framework, the data of the training model is updated regularly, and the training and testing processes of all models are automatically and regularly triggered. The system will record each run's parameters and data collection types and save the model to a file system. It is flexible to add new models to the ML module. Figure 3.3 shows the

flowchart of the machine learning module [71]. The flowchart is divided into two branches after the feature collection process. The first branch is to extract word-level features and train traditional machine learning models, and the second branch is to extract character-level features and train deep learning models.

Phishing website detection is a binary classification problem. There are many widely used classifiers, such as Decision Trees, Random forest, k-nearest neighbours' algorithm, Bagging, Naïve Bayes, Logistic Regression, and Support vector machine (SVM). The Random forest, Logistic regression and SVM are selected in this thesis.

Many studies have shown that the random forest classifier performs better than other traditional classification models in detecting phishing networks [11], [50], [51]. A random forest is an ensemble of decision trees for classification and regression. Random forests reduce the overfitting problem by classifying or averaging the output of individual trees in training processing. Therefore, random forests generally have higher accuracy than decision tree algorithms.

Logistic regression is a statistical algorithm used to predict the outcome of a dependent variable based on previous observations. It's a commonly used machine learning model for solving binary classification problems. For example, a logistic regression algorithm can predict whether a web page is a phishing website. A logistic regression algorithm was used to solve the spam text classification problem [73] and detect phishing websites [74].

A support vector machine (SVM) is a supervised learning algorithm that classifies data points into two sections and predicts new data points belonging to each section. It is suitable for linear binary classification, which has two classes labelled, and the classifier is

a hyperplane with N dimensions relevant to the number of features. The core idea of this algorithm is to maximize the distance between the data point and the segmentation hyperplane. For example, there are two classes: phishing and legitimate, and a 29-dimension hyperplane in the UCI dataset [66] for training the SVM model. Therefore, these three traditional machine learning models are used to predict phishing URLs. The optimization of the model is accomplished by adjusting the parameters and comparing the test results.

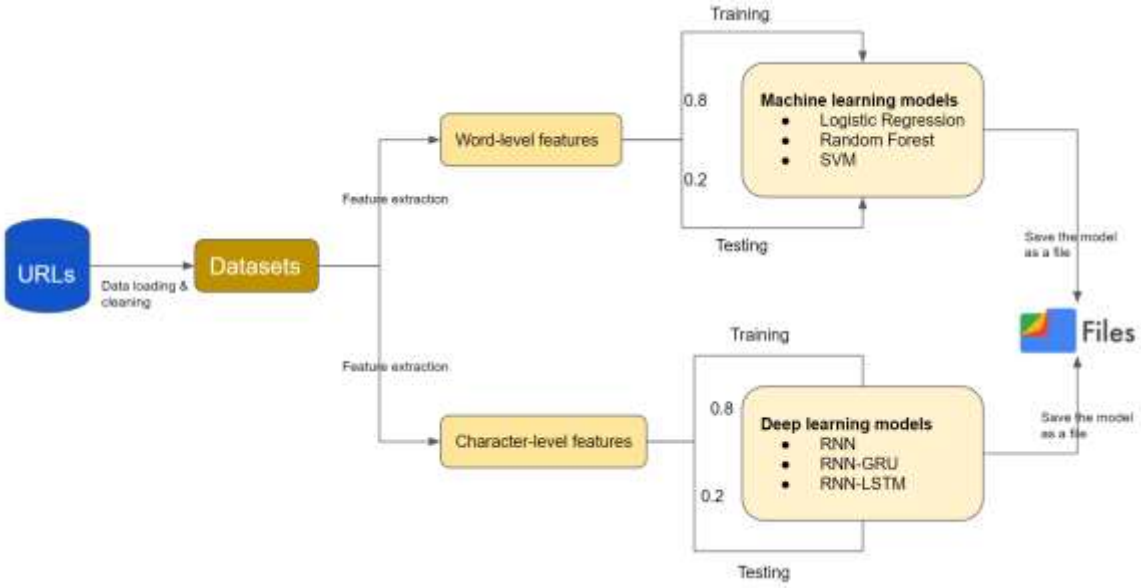

Figure 3.3: Flowchart of the machine learning module

In the deep learning model training process, URLs are converted into a set of character sequences. In deep machine learning models, the architecture of a recurrent neural network (RNN) is well suited for training data sequences. RNN is a deep neural network with an internal memory function to handle diverse length sequences of inputs, such as text. In addition, the GRU and LSTM models add a gate unit based on the RNN

architecture to control and calculate the current state. These two models can be said to be upgraded versions of the RNN structure.

Table 3.3 shows a summary of these algorithms based on the same dataset. The Big O notation is used to measure the computational complexities of machine learning algorithms. The complexity of a deep neural network depends on the architecture of the networks. Generally, it needs to compute the activation function of all neurons. Interpretability presents the difficulty of understanding how the model works. Traditional machine learning algorithms are user-friendly. In deep neural networks, it is hard to know which neuron is playing what role and which input feature contributes to the model output. A well-known drawback of deep learning models is the "black box" nature [75]. In addition, deep neural networks require more training data than other algorithms to obtain acceptable performance [76]. The significant advantage of deep neural networks is dealing with text data, such as URL strings.

Table 3.3. Machine learning algorithms for detecting phishing websites

Algorithm	**Training Time Complexity**	**Interpretability**	**Training Data Size**	**Inputs**
Logistic Regression	$O(nd)$	High	Small	Structured data
SVM	$O(n^2)$	Medium	Small	Structured data
Random Forest	$O(knd \log n)$ k = number of trees	Medium	Small	Structured data
Deep Neural Networks	Compute the activation of all neurons	Low	Large	Structured data or text data

3.3.1 Data Loading

Data are the source of each approach and prove to be a vital influence on the performance. There are two methods to collect data: loading published datasets and pulling URLs directly from the Internet. The original URL strings could be collected from websites by running open API or data mining scripts.

The dataset used for model training is obtained from the database through the data service. The data service supports the flexible selection of different data source combinations and datasets of varying data volumes. Each data instance contains a URL string and a label that signs the URL is a phishing link or a legitimate link. The label values are normalized as 1 and 0.

3.3.2 Feature Extraction

With the successful development of the natural language processing (NLP) technique, many researchers capture character-level features from URL strings based on the NLP and then feed them into deep learning models to increase the accuracy. The significant advantage of this method is not relying on third-party network services [24].

A URL string is handled as a document containing semantics and applies the Natural language processing (NLP) technology to extract features. In classical machine learning models, two methods of extracting features are used, named TF-IDF-Vectorizer and Count-Vectorizer. The TF-IDF-Vectorizer converts a collection of URLs to a matrix of TF-IDF features. TF-IDF means Term Frequency–Inverse Document Frequency. The algorithm calculates each word's TF-IDF score and then generates a matrix with those scores, which stands for the relevance of a word in the URL string. The Count-Vectorizer converts a collection of URLs to a matrix of token counts, and each token stands for one

word. Therefore, the number of features equals the vocabulary size found by analyzing the data.

In deep learning models, the tokenization process parses a URL string to a list of characters (Character-level tokens). The characters in the URL come from the ASCII character set. The most common 100 characters are selected as the character set dictionary for this study. Figure 3.4 shows all the arranged characters and the corresponding index.

Figure 3.4: Dictionary with 100 ASCII characters

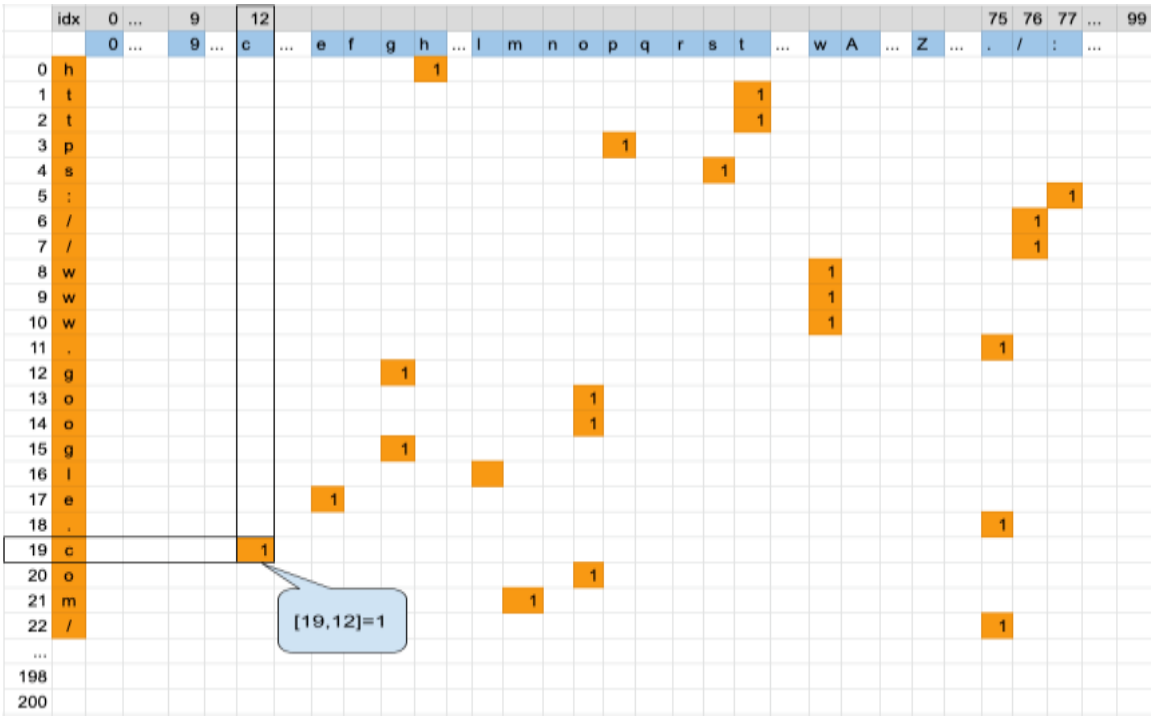

Figure 3.5: A feature matrix from a URL string and the character dictionary

The maximum length of a URL is 2083 characters [31]. Because of the calculation time of the deep learning model and the analysis of the statistical data of the existing data

set, the maximum number of URL characters is 200. Therefore, each URL can be transformed into a 200*100 matrix. The position of the dictionary corresponding to each character is marked as 1, and the remaining values are 0. Figure 3.5 shows the process of forming a matrix using Google's official website as an example.

3.3.3 Modelling

It is a solution to treat a URL as a document and use character separators to parse words as features. However, many words in URLs also lack semantics. Moreover, the analysis of word-level results in an extensive dictionary will slow the calculation time. The recurrent neural network (RNN) is a feedback neural network that stores temporary states. It's suitable for training sequence data [32]. Figure 3.6 shows a regular RNN architecture that consists of an input layer, several hidden layers and an output layer. Compared to the feedforward artificial neural networks (ANN), RNNs have a unique architecture with a connection function between neurons in hidden layers. The figure shows that the current hidden state is related to the previous hidden state and the current input. The current hidden state's functional form can be represented as Eq. (1) and (2). The tanh is a nonlinear function, W represents the weights between the neurons, and b is the bias vector of the setting. The Soft-max calculates the output value as an activation function, as shown in Eq. (3), and the model prediction value is related to the current hidden state.

$$h_t = f_w(h_{t-1}, x_t) \qquad (1)$$

$$h_t = \tanh(w_{hx} + w_{hh}h_{t-1} + b_h) \qquad (2)$$

$$Y_t = softmax\left(W_{yh}h_t + b_y\right) \qquad (3)$$

The scenario that detects the phishing link is a many-to-one task type, the input is character-level sequence data, and the output is a category. Figure 3.7 shows the structure of one hidden layer.

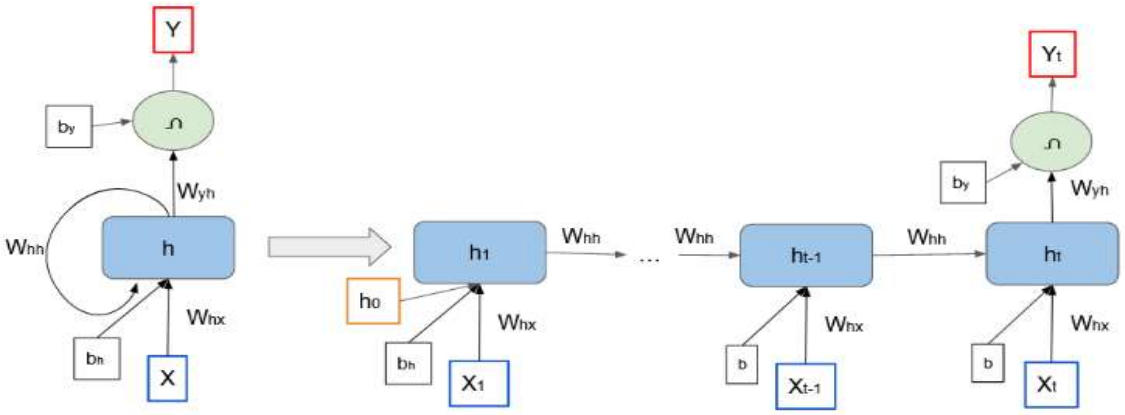

Figure 3.6: Architecture of a basic RNN

The Whx, Whh, and Wyh respectively mean the weight matrix between input and hidden layer, the weight matrix between two hidden layers, the weight matrix between hidden and output layer.

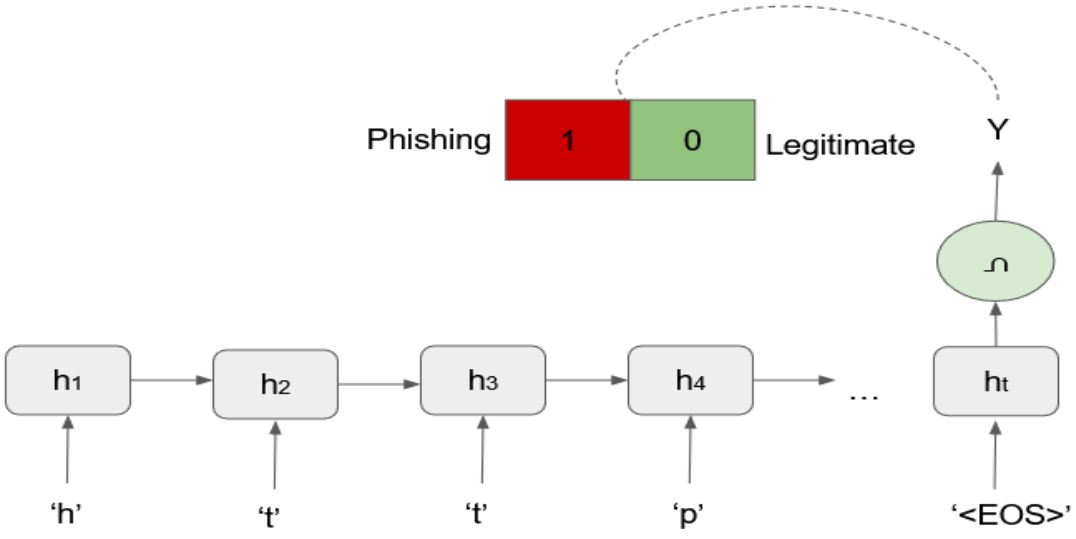

Figure 3.7: Character-level features in an RNN model

Before model training, the structure of the model is fixed, and activation function is established, so the process of model training is actually the process of optimizing the weights parameters by calculating each error. First, randomly initialize the weights matrix, then calculate the difference between the actual value and the predicted value, then use the optimized algorithm to find the optimal solution to minimize the difference, and finally adjust each weight by calculating the step each time.

Depending on the architecture of RNN and activation functions used, the basic RNN architecture does not perform well for handling inputs for long sequences because of the vulnerability to gradient vanishing or exploding problems [33]. To address these, Hochreiter and Schmidhuber introduced a gradient-based model named long short-term memory (LSTM) in 1997 [34]. They invented a long short-term memory unit instead of tanh function to compute hidden states. The LSTM unit consists of three gates and two memory cells. Cho et al. proposed a novel model with a hidden unit, which was motivated by LSTM in 2014 [35]. Figure 3.8 demonstrates the long short-term memory (LSTM) learning model architecture.

Since the hidden unit contains two gates to control and calculate the hidden state, this model is also named gated recurrent unit (GRU). Figure 3.9 shows a typical structure of GRU. It can be said that long short-term memory network (LSTM) and gated recurrent unit (GRU) are two enhanced versions of RNN. Many studies and experimental data show that for sequence data training, the LSTM and GRU architecture can achieve better performance than the basic RNN architecture [36] [37].

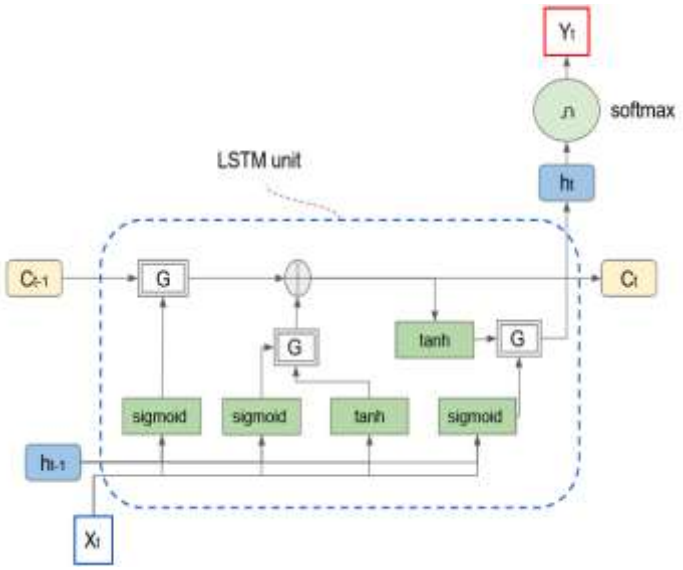

Figure 3.8: The architecture of basic long short-term memory (LSTM) cell

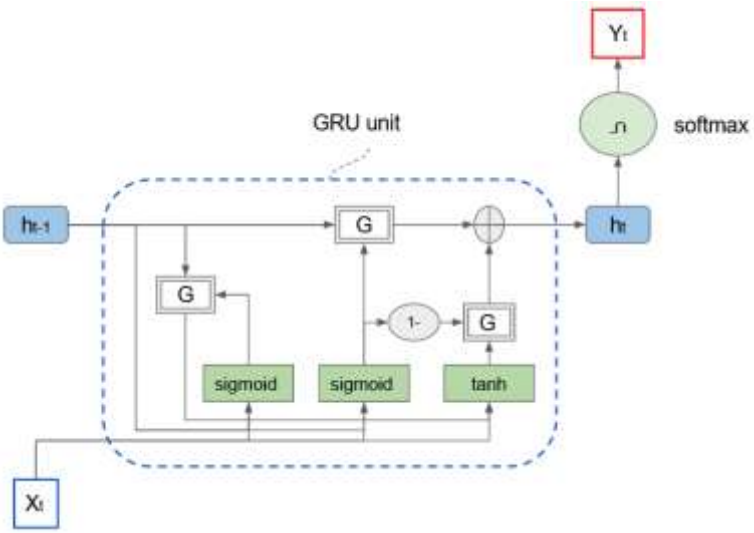

Figure 3.9: The architecture of basic gated recurrent unit (GRU)

3.3.4 Optimizer and Loss Function

In the model training process, it is also essential to choose a suitable optimizer and loss function. Among many optimization algorithms, the Adam algorithm is selected, which is a popular and effective optimization algorithm for deep learning [38]. Since the problem is a binary classification problem, the cross-entropy loss function [39] is used, which is also

called the log loss function. According to the scenario of the current problem, the output predicted value is a floating-point number between 0 and 1. Cross-entropy loss increases as the predicted probability diverge from the actual label. It is believed that it will converge quickly in the initial stage of training with the same learning rate. The loss value of each epoch is calculated the average loss value of all data points.

3.4 Heuristic Rule-Based Strategy

The core of heuristic rule-based strategy is to analyze the characteristics of phishing pages based on web page URL and web page HTML content, then extract relevant feature values and formulate a series of rules to determine whether there is a phishing risk. Phishing pages often use a non-semantic domain name, but clearly display a well-known company or brand name on the page content, and have a form to enter user sentiment information. Table 3.4 shows the attributes and corresponding rules extracted from the HTML document. The features F1, F2 and F3 judge phishing risk from a correlation between a web page content and domain name semantics. The value of the title tag is the title that describes the entire web page. Generally, the title of a normal website contains a company name or brand and the description of the current page, such as the login page of Facebook. The URL is "https://www.facebook.com/", and the text in the title tag is "Facebook – Log In or Sign Up". At the same time, the domain name part of the URL usually also contains company or brand information. Consistency can be judged by comparing the similarity of the domain name semantics and document description semantics. However, some websites did not set the title tag value. The content of the heading tag is extracted as a description of a website.

Table 3.4. Features applied to the heuristic method

Feature	Attribute	Rule
F1	The text in the <title>	If it is null, value=1, else value=0
F2	The similarity score of domain and title	If the score >threshold, value=0, else value=0
F3	The similarity score of domain and text of header tags	If the score >threshold, value=0, else value=0
F4	The number of input numbers	If the number <threshold, value=0, else value=1
F5	The number of buttons	If the number <threshold, value=0, else value=1
F6	The description of inputs has sensitive words	If it has sensitive words, value=1, else value=0
F7	Iframe tag	If it has iframe tag, value=1, else value=0

The F4, F5 and F6 determine whether there is a form that requires users to submit sensitive information. The number of inputs and buttons on the page are also very important characteristics. The input box displayed on the page can be implemented by the input tag and the textarea tag, and the input tag has 21 types, which are specified by the attribute type. Among them, there are seven types for the user to input text or numeric information, and the default is text input if there is no specified type. Buttons are implemented in a variety of ways. For example, the button tag, the input tag whose attribute value is "button", and other tags are controlled by JavaScript to submit information to the web server. Furthermore, from the analysis of normal web page functions, some form input and

submission functions are not user-sensitive information, such as commonly used search functions. Therefore, the description of input tags is captured from tags' context and attributes as features.

The tag "iframe" is used to embed another page on the current page. The embedded HTML structure cannot be obtained from the source code of the current web page. Therefore, some phishing web pages use the "iframe" tag to embed a form submission function. The F7 is used to detect abuse of iframe tags.

Each input tag is defined as a dictionary, which contains two elements: type and description. The value of the type can be obtained from the type attribute of the input tag. There are many sources of description information, including the text of other adjacent elements, the default value of the input element, and the placeholder attribute.

Confidential forms refer to forms that require users to submit sensitive personal information, such as name, ID number, password, bank card number, etc. These descriptions are usually displayed on the left, above, or in the corresponding input box. First, the similarity is calculated between the input tag description and the local sensitive vocabulary dictionary. This dictionary is created based on experience and statistical analysis of information input pages from well-known websites. Some phishing attackers try to evade detection rules by using variations of characters. For example, Figure 3.10 is a screenshot of an active phishing link "https://con_rmsubscription.com/h/y/2E7CE2C46A8733CF". In this case, one of the sensitive words, "password," is a variant. This deformation visually deceives the user, but the recognized meaning is entirely different from the visual when the computer treats it as an ordinary word.

Figure 3.10: A screenshot of a phishing website's form

A fuzzy matching algorithm is used to judge the consistency of web page content and domain name. It uses Levenshtein Distance [41] to calculate the differences between two sequences. A common method of phishing is to imitate user-submitted information pages of well-known websites, which are then deployed under the attacker's own domain name. Figure 3.11 shows the three factors used to judge consistency, namely domain name, web page content description and list of well-known website company names.

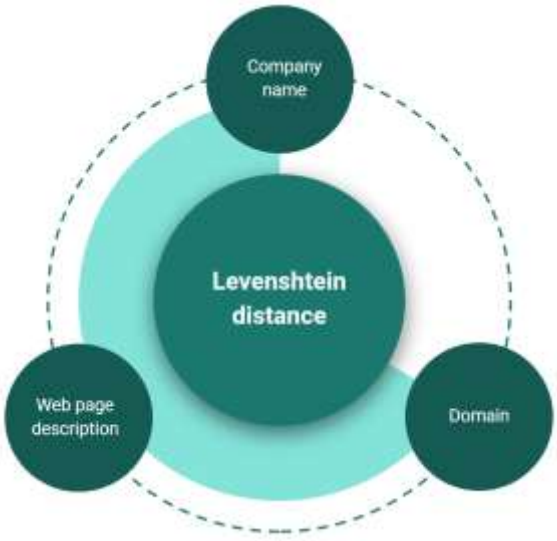

Figure 3.11: Features for consistency check

3.5 Prediction Service

The prediction service combines multiple strategies in terms of whitelists, blacklists, heuristic rules and machine learning models. Figure 3.12 presents a flowchart of the prediction service. A white list is a collection of legitimate URLs from the Kaggle dataset. A blacklist is a set of phishing websites from the PhishTank dataset. The machine learning model is trained offline described in the 3.2 section. A rule-based heuristic strategy is about extracting features from a URL and HTML source code and creating several rules to infer phishing risks. This method is driven by expert experiences.

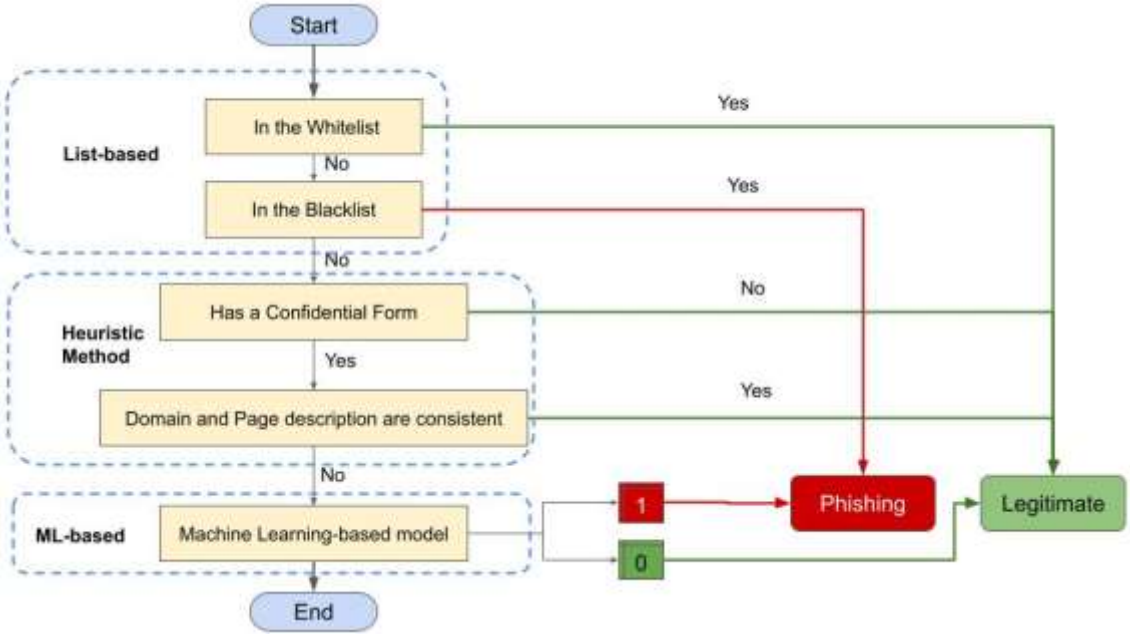

Figure 3.12: Flowchart of the prediction service

3.6 Summary

This chapter presented the design of the phishing detection framework. The architecture of the framework mainly has four components: data collection, machine learning model training, web browser extension, and a cloud application proving HTTP services. The implementation details are presented in the next chapter.

Chapter 4

Implementation

This chapter describes the implementation of the proposed multi-strategy framework. A Chrome extension is developed on the client-side to collect client data and notify of phishing risks. In addition, a web application is built for providing a prediction service with HTTPS protocol and collecting users' feedback. Finally, various libraries related to machine learning are introduced.

4.1 Prototype Overview

The prototype implementation of the entire framework is divided into three independent applications. Figure 4.1 shows a client-server architecture of the prototype. The browser extension is independently packaged and uploaded to the Chrome browser according to the extension development specifications of the Chrome browser and will be reviewed and released by the Chrome platform. Chrome browser plug-in development uses three web front-end development languages: HTML, JavaScript, and CSS. The data collection application uses Python as the main development language, using scheduled tasks to manage the collection tasks of each data source. The task fetching data from PhishTank uses the Beautiful Soup library to mining phishing URLs. Model training, prediction services and official product website are integrated into one application. This application also uses Python as the primary language and imports Flask as the web framework. Model training is managed by a timed task, which is executed once a day when client traffic is low, currently set at 10 pm. After the training is completed, the core performance indicators are written into the MySQL database in real-time, and the model is dumped into the file

system. The prediction service is a RESTful API that provides clients with real-time POST requests to obtain detection results. The core function of the official website is to accept the suspected phishing link submitted by the user and determine the link risk by manual and automatic verification.

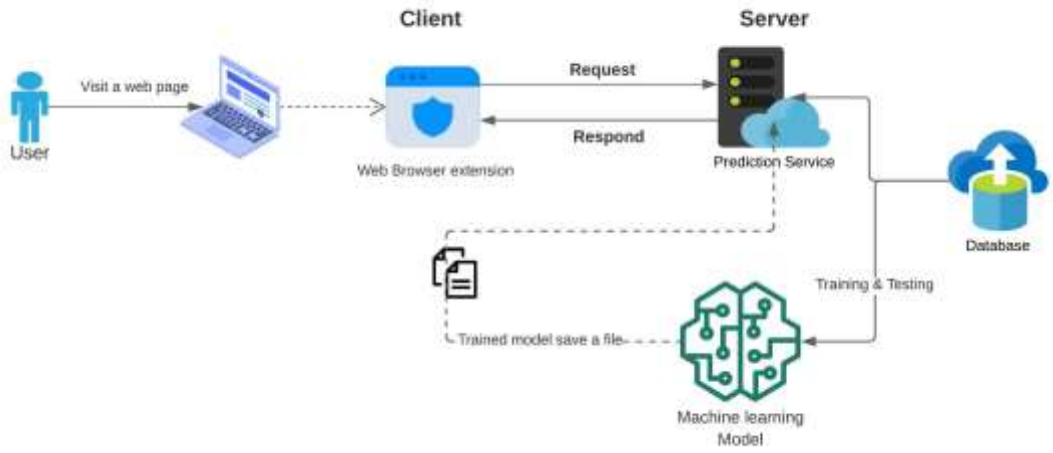

Figure 4.1 Client-Server architecture of the prototype

4.2 Chrome Extension

According to statistical analysis, the market share of Chrome browser is far ahead of other browsers in the past ten years [68]. In addition, the development documentation of the Chrome browser plug-in is complete and updated in a timely manner. Also, the Chrome plug-in is easy to install and user-friendly. Therefore, a Chrome browser extension [69] is developed as a client-side and the source code is publicly available on GitHub [72]. Since users installed the extension in the Chrome browser, the extension will automatically detect whether the newly opened URL is at risk of phishing. When the system detects that the current page is at risk of phishing, the browser plug-in will display a warning box and provide an entry to report a false alarm. Conversely, if the current page is not at risk of

phishing, the plugin provides an entry to report that the page is a phishing URL. Users will then be directed to the website to submit a report. Regarding the review process of the report, it has manual review and automatic review. The automatic review currently relies on rules. For example, if different client IP addresses submit links with the same domain name three times, the system automatically reviews them. Table 3.5 presents the structure of the report table in the database.

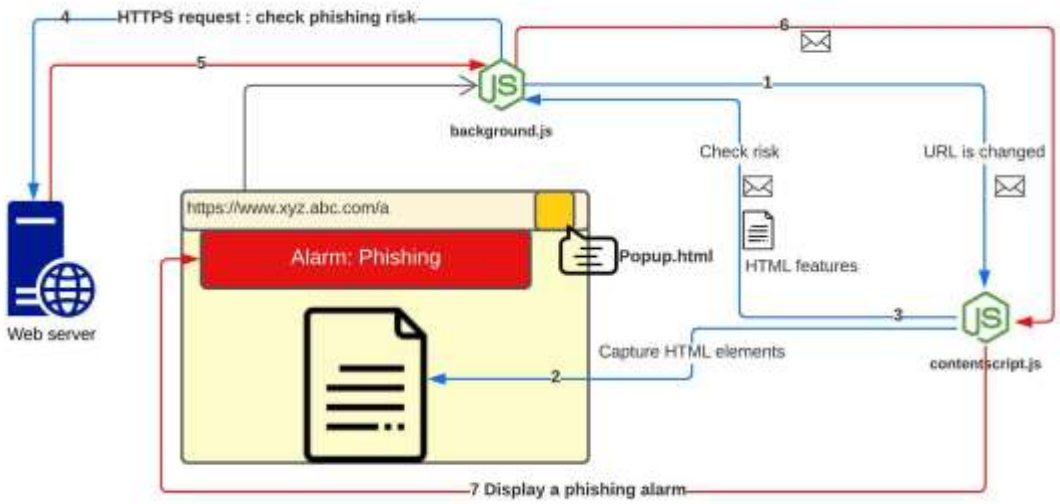

Figure 4.2: Flowchart of the Chrome extension

Figure 4.2 presents the workflow of the extension. The extension calls the prediction service for obtaining phishing risk level, with parameters in terms of the URL string and HTML elements. Since the primary purpose of phishing attacks is to steal users' information by deceiving them to fill forms on fake websites, the scope of a phishing page is narrowed to a page with form submissions and then analyzing the content that requires user input. Web pages displayed in front of users through browsers must comply with web standards, such as HTML Living Standard [40]. This standard describes the HTML language specification, tags that can be used on web pages, and related APIs. Each HTML

tag has different semantics. For example, the <a> tag represents a hyperlink. After setting the value of the “href” attribute, the click will cause the browser to visit another URL. However, there is also the ECMA standard among web standards, which supports JavaScript scripts. JavaScript is a very flexible dynamic language with the ability to change the default display and behaviour of HTML tags. For example, JavaScript can control the tag, simulating the same behavior as the <a> tag. This brings difficulty and uncertainty to HTML parsing semantics through tags and text. For example, HTML tags <h1> to <h6> represent different levels of headings, but CSS files can display content displayed with <p> tags as heading styles.

The development of the Chrome browser extension must strictly follow the development guidelines in [45]. The chrome plug-in has a configuration to contract version numbers, introduce built-in APIs, and permission control. Data interactions between modules are messaged and temporarily stored in Chrome storage. A background script listens for browser tab change events, gets the URL currently accessed, calls the back-end prediction service to get the result, and finally sends the result to a content script. The content script is primarily responsible for presenting the results on the page. In addition, a popup Html shows the details.

4.3 Web Application

The web application mainly provides HTTPS services for detecting phishing risks and processes risk reports submitted by users. Figure 4.3 shows the architecture of the web application, which includes four layers in terms of web, service, task and database. Python was used as a core language, which is a modern high-level programming language in the field of data mining and machine learning. There are various frameworks and libraries for

the Python language. In the system, data collection, data storage, model training, websites, and HTTP services are all supported by mature libraries and frameworks. In addition, the access and use of these packages are very simple and convenient.

Considering the usage scenarios and read and write performance, the data layer uses the MySQL relational database. First, the website has user management, report management, model version management and other functions which require a relational database. In addition, the data set used for model training is acquired dynamically. It is very flexible to combine different data sources and data volumes to form a new data set for model training. For example, 200,000 phishing URLs are collected from PhishTank, and 340,000 legitimate URLs are downloaded from Kaggle. A balanced data set with 40,000 URLs can be flexibly combined, including 20,000 phishing URLs and 20,000 legal URLs.

Figure 4.3: Architecture of the web application

When a false alarm or missed alarm occurs in the prediction service, the user can take the initiative to report the current falsely detected URL from the browser plug-in portal. A website is developed to receive these reports. Once the report is submitted to the

system, the system has a manual review process to confirm the risks of these URLs. In addition, there are automatic audit strategies to improve audit efficiency. Once the review is completed, these URLs will be regularly synchronized to the data collection module, and the source is reported. Table 4.1 shows the structure of a report in the database.

The system has two scheduled tasks to process these reports. First, the Validate Report task automatically reviews the report based on the rules. The current rule is that the same URL submitted by different IP addresses exceeds the threshold. In addition, the sync task will sync the verified URL to the data table of the dataset used to train the model and generate black and white lists.

Table 4.1. Table structure for the data table named "report"

Column	Data type	Description
id	int	Automatic index
Gmt_created	datatime	Create time
Gmt_modified	datatime	Verify time
url	varchar	URL string
type	int	Error type (1: phishing, 2: legitimate)
status	int	0: initial, 1: verified (legitimate), -1(phishing)
comment	varchar	Comment from user
Client_ip	varchar	user's IP address

4.3.1 Flask

The Flask is used as a web framework to provide HTTP service and maintain the official website. It is a lightweight web framework and easy to extend [42]. For example, the flask-

user package provides user authorization services. Flask is part of the categories of the micro-framework, which is a little dependent on updating and watching for security bugs. The template engine named “jinja2” is automatically installed when the Flask library is installed successfully. It is convenient to set a basic layout for pages and mention which element will change by using templates. In addition, a blueprint library is imported to organize the application into distinct components. A blueprint defines a collection of views, templates, static files and other elements that can be applied to an application.

```
from flask import Flask
import routes
app = Flask(__name__)

# # routes
app.register_blueprint(routes.routes_app)


if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Listing 4.1: Source code for create a flask application and using blueprint to manager routes

```
from flask import Blueprint
from flask import render_template

routes_app = Blueprint('routes', __name__)


@routes_app.route('/home', methods=['GET'])
def welcome():
    return render_template('/home.html')
```

Listing 4.2: Source code for defining routes in a new file by using blueprint

4.3.2 CORS

Cross-Origin Resource Sharing (CORS) is an HTTP-header-based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. When the domain name of the client's request is inconsistent with the domain name of the server interface, a CORS error will occur. An

example of a cross-origin request is that the front-end JavaScript code served from https://domain-a.com uses XMLHttpRequest to make a request for https://domain-b.com/data.json. The FLASK_CORS library is imported to handle CORS, making cross-origin AJAX possible. In addition, the interface that allows cross-origin access makes a judgment on the hostname requested by the client. Only requests made from this Chrome extension are legitimate requests.

4.3.3 APScheduler

A Python third-party library named APScheduler was used to implement the automatic verification task and data synchronization task. This library has four kinds of components: triggers, job stores, executors and schedulers. The application developer doesn't normally deal with the job stores, executors or triggers directly. Instead, the scheduler provides the proper interface to handle all the above tasks. Configuring the job stores and executors is done through the scheduler, as is adding, modifying and removing jobs. The BackgroundScheduler library is used as a scheduler to run in the background inside the application and is configured the trigger type valued "cron" to run the job periodically at a certain time of day.

```
from apscheduler.schedulers.background import BackgroundScheduler
from task.automatic_verify import automatic_verify
from task.report import sync_report_url

scheduler = BackgroundScheduler(daemon=True)
scheduler.add_job(automatic_verify, 'cron', day_of_week='mon-sun',
hour=23, minute=20, end_date='2023-12-31')
scheduler.add_job(sync_report_url, 'cron', day_of_week='mon-sun',
hour=23, minute=30, end_date='2023-12-31')

scheduler.start()
```

Listing 4.3: Source code for using background scheduler

4.3.4 Deployment

The web application was deployed on the DigitalOcean cloud platform, which provides cloud services. on the operate system is Ubuntu 20.04, the server is Gunicorn, and Nginx works as a front-end reverse proxy. Green Unicorn, commonly shortened to "Gunicorn" is a Web Server Gateway Interface (WSGI) server implementation that is commonly used to run Python web applications. Gunicorn knows how to run a web application based on the hook between the WSGI server and the WSGI-compliant web app. Therefore, a wsgi.py should be created and works as a hook (listing 4.4). In addition, the Certbot was used to obtain SSL certificates for ensuring that traffic to the server remains secure.

```
from myproject import app

if __name__ == "__main__":
    app.run()
```

Listing 4.4: Source code for creating a wsgi hook

4.4 Machine Learning

Model training is an offline task. When the data instances in the dataset are updated every day, model training can also be set to be triggered automatically by scheduled tasks. Traditional machine learning models and deep learning models are trained with the same datasets to compare performances.

4.4.1 Scikit-Learn

The scikit-learn is open-source and widely used for predictive data analysis in the machine learning field [43]. Ascikit-learn library was used to train three traditional machine learning models: Logistic Regression, Random Forest, and support vector machine. First, URLs are loaded from a MySQL database and then features are extracted by *CountVectorizer* and *fit_transform* function. There are many built-in classifiers in terms of *LogisticRegression*,

RandomForestClassifier, and *SVM*. After initializing a model, the *fit* function starts to train the model. Finally, the test dataset is applied to the *predict* function for obtaining results.

```
import seaborn as sns
from sklearn import svm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

from AI.utils.metrics_util import basic_metrics_ml
from AI.utils.model_util import data_preparation,
read_data_multi_source_limit


def ML_training(source_list, limitation=None):
    # loading data
    df = read_data_multi_source_limit(source_list, limitation)
    # # tokenizer data preprocessing
    df = data_preparation(df, r'[A-Za-z]+')
    # # feature extraction
    vec = CountVectorizer()
    # transform all text which we tokenize and stemed
    vec.fit_transform(df.text)
    feature = vec.fit_transform(df.text)
    # # split dataset for training and testing
    trainX, testX, trainY, testY = train_test_split(feature,
df.result, test_size=0.2)
    # # modelling
    clf = svm.SVC()
    clf.fit(trainX, trainY)
    # # performance metrics
    predictions = clf.predict(testX)
    cm = confusion_matrix(testY, predictions)
    sns.heatmap(cm, annot=True)
    fn_rate, fp_rate, accuracy, precision, recall, f1 =
basic_metrics_ml(predictions, testY)
    return fn_rate, fp_rate, accuracy, precision, recall, f1
```

Listing 4.5: Source code for traditional machine learning model training

4.4.2 PyTorch

The PyTorch is an open-source deep learning framework and development platform. In the deep learning models' construction process, it imported the linear layer, RNN layer, GRU

layer, an LSTM layer. The torch.cuda package is used to utilize GPUs for parallel computation [44].

```
import torch
import torch.nn as nn


class GRUClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers,
num_classes):
        super(GRUClassifier, self).__init__()
        self.num_layers = num_layers
        self.hidden_size = hidden_size
        # self.rnn = nn.RNN(input_size, hidden_size, num_layers,
batch_first=True)
        self.gru = nn.GRU(input_size, hidden_size, num_layers,
batch_first=True)
        # self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True)
        # x -> (batch_size, sequence_length, input_size)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0),
self.hidden_size)
        # c0 = torch.zeros(self.num_layers, x.size(0),
self.hidden_size)
        # out, _ = self.rnn(x, h0)
        out, _ = self.gru(x, h0)
        # out, _ = self.lstm(x,(c0,h0))
        # out: batch_size, sqe_length, hidden_size
        # out (N , 200, 100)
        out = out[:, -1, :]
        # out (N, 100)
        out = self.fc(out)
        return out
```

Listing 4.6: Source code for defining a GRU model

4.5 Summary

This chapter has presented the implementation of the presented phishing detection framework. Each software package that plays a major role in the framework has been discussed in terms of its role as well as the reasons for why it was chosen. In the next chapter, the evaluation of the implemented framework is presented.

Chapter 5

Evaluation Results

This chapter presents the details on the experiments conducted with the proposed machine learning models and the heuristic rule strategy. All the experiments were executed on a MacBook Pro 2020 running Quad-Core Intel Core i5 CPU @ 2 GHz with macOS Big Sur 11.5.2 operating system. The server has a 500 GB storage capacity.

5.1 Evaluation Metrics

The performance evaluation was carried out during the testing process. The original dataset would be divided into training data and test data, 80% and 20%, respectively. Commonly, three standard statistical metrics with accuracy, recall, and precision [46] are used to evaluate whether a machine learning model has high performance. When evaluating the classifier's behaviour on the testing dataset, there were four statistical numbers: the number of correctly identified positive data points (TP), the number of correctly identified negative data points (TN), the number of negative data points labelled by the classifier as positive (FP), and the number of positive data points labelled by the model as negative (FN) (table 5.1).

Table 5.1. Four statistical numbers of predicting results

True Labels	**Labels Returned by the Classifier in the Testing Process**	
	Positive	**Negative**
Positive	TP	FN
Negative	FP	TN

There are several broadly used metrics to evaluate performance. The classification accuracy is the ratio of correct predictions to total predictions:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

In binary classification cases, it is known that random selection has 50% accuracy. In unbalanced datasets, sometimes high accuracy does not mean that the model is excellent. For instance, among the 10,000 data, 9000 were legitimate websites, and 1000 were phishing websites, so when the prediction model did nothing, it could reach 90%. Accuracy is misleading when the class sizes are substantially different. Precision is the percentage of correctly identified positive data points among those predicted as positive by the model. The number of false-positive cases (FP) reflects the false warning rate. In real-time phishing detection systems, this directly affects the user experience and trustworthiness:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (5)$$

The recall is the portion of positive data points labelled as such by the model among all truly positive data points. The number of false-negative cases (FN) represents the number of phishing URLs that have not been detected. Leak alarms mean that users are likely to receive an attack that could result in the theft of sensitive information. Misleading users can do more harm to users than not detecting them:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (6)$$

The F-measure or F-score is the combination of precision and recall. Generally, it is formulated as shown below:

$$F_\beta = \frac{(\beta^2 + 1) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \qquad \beta \in (0, \infty) \quad (7)$$

Here, β quantifies the relative importance of the precision and recall such that $\beta = 1$ stands for the precision and recall being equally important, which is also called F1. The F-score does the best job of any single statistic, but all four work together to describe the performance of a classifier:

$$\mathrm{F1} = \frac{2 \times precision \times recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (8)$$

F1 score is used to represent the meaning of the recall and precision. In addition, in cybersecurity detection applications, false alarms can affect the user experience and trust, and leak alarms are likely to directly cause user losses. Therefore, accuracy, F1, false-positive rate, and false-negative rate are used to measure the performance of models.

$$\text{false positive rate} = \frac{FP}{FP + TN} \quad (9)$$

$$\text{false negative rate} = \frac{FN}{FN + TP} \quad (10)$$

Furthermore, Average precision (AP) is a widely used metric in evaluating the accuracy of deep learning models by computing the average precision value for recall value over 0 to 1; higher is better. Mean average precision (mAP) is the average of AP. Equation (11) shows the calculation logic. In this scene, the number of classes is two.

$$\mathrm{mAP} = \frac{1}{classes} \sum_{c \in classes} \frac{TP(c)}{TP(c) + FP(c)} \quad (11)$$

5.2 Experimental Setup and Datasets

The test data ratio is 0.2. Seven datasets are used in the machine learning training and testing process. Six machine learning models are built to compare performance to obtain a

best model with optimal parameter values. Table 5.2 lists all the datasets and the number of data instances.

Table 5.2. Datasets

Name	Number of Phishing URLs	Number of Legitimate URLs
PhishStorm	47,902	48,009
ISCX-URL2016	9965	35378
KPT-4	20,000	20,000
KPT-6	30,000	30,000
KPT-8	40,000	40,000
KPT-10	50,000	50,000
KPT-12	60,000	60,000

5.3 Machine Learning Models

Experiments are carried out in the following steps to find the optimal model quickly. First, the GRU model is chosen because its performance may be better by theoretical analysis. By comparing the results obtained from training on different datasets with GRU model, the best performance dataset is obtained. Second, by comparing the results obtained from training different models with the dataset obtained in the previous experiment, the best performance model is obtained. The last process is optimizing the model hyperparameters for the model with the dataset. The primary method is to enumerate the optional discrete values of the parameters and perform cross-combination to compare the performance of all experimental results.

5.3.1 GRU Model with Different Datasets

In this experiment, different datasets are fed to the GRU classifier. The number of epochs is 20, the batch size is 32, KPT stands for the data collected from Kaggle and PhishTank, and each KPT dataset is a balanced dataset, which consists of the same number of phishing URLs and legitimate URLs. Table 5.3 shows the core performance indicators of the GRU model with eight datasets. The ISCX dataset obtained the highest accuracy. However, the F1 score is lower than the other three KPT datasets, and the false-negative rate is high. In other words, more legitimate instances are predicted as phishing URLs during the test data process.

Table 5.3. GRU model test results with different datasets

Dataset	accuracy	F1	False-positive rate	False-negative rate	mAP
PhishStorm	0.9758	0.9748	0.0269	0.0212	0.960
ISCX-URL2016	0.9947	0.9874	0.0011	0.0191	0.986
KPT-4	0.981	0.980	0.0091	0.0285	0.948
KPT-6	0.9882	0.988	0.0089	0.0145	0.967
KPT-8	0.9898	0.9894	0.0134	0.0073	0.973
KPT-10	0.9906	0.9904	0.006	0.0132	0.984
KPT-12	0.9918	0.9915	0.0104	0.0059	0.986

Furthermore, False-positive rate and false-negative rate are used to measure efficiency. From figure 5.1, the RNN-GRU model with the KPT-12 dataset performs best. In KPT datasets, the false rate decreases linearly as the number of data increases.

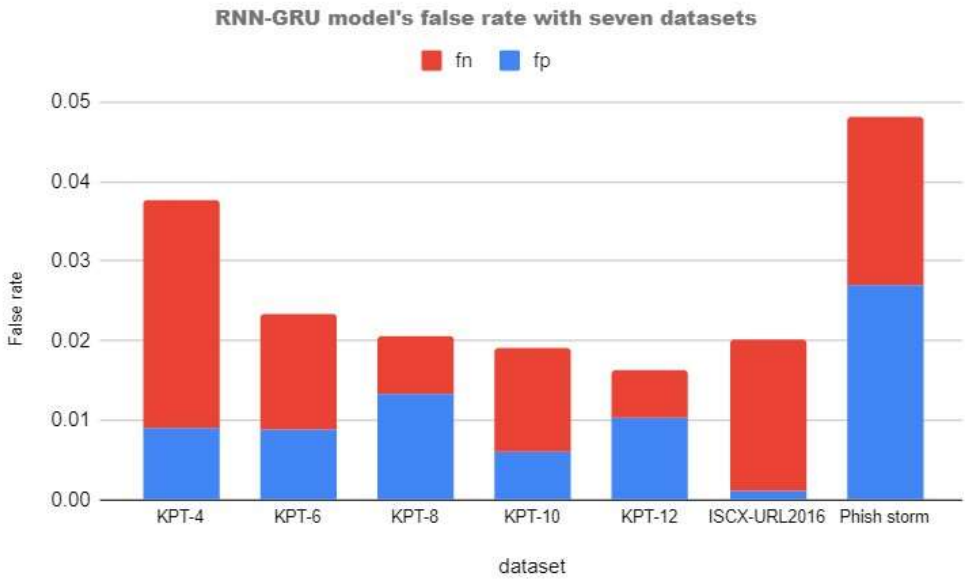

Figure 5.1: GRU model false rates with different datasets

Figure 5.2 shows the accuracy and F1 of each dataset. The KPT-12 dataset obtained 99.18% accuracy and 99.15% F1 score. To quantify how well the RNN-GRU model performs every class, the mean of average precision (mAP) is used. The mean average precision (mAP) in RNN-GRU models with KPT-12 dataset is 0.986.

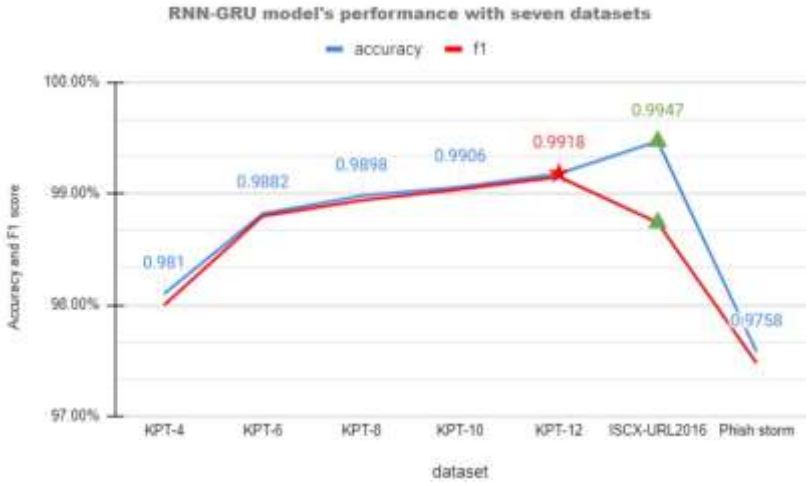

Figure 5.2: Accuracy and F1 score of the GRU model

Assessing the efficiency of a machine learning model is incomplete, depending on accuracy. In experiments, it is customary to get high accuracy in one dataset and not

perform well in another. It is also likely that models with high accuracy will not predict new data accurately in a real-time environment. These situations may be due to the fact that overfitting has already occurred [47]. Overfitting is a concept in data mining that analyzes whether a trained model can efficiently predict unknown new data [48]. In machine learning-based classification models, it is common to compare errors in the training process with errors in the validation process to see if there is overfitting, along with epoch. Figure 5.3 presents the training loss and validation loss along with epochs in the RNN-GRU model. One of the strategies to avoid overfitting is early-stopping [49]. The epoch equals 6 is the demarcation point between underfitting and overfitting.

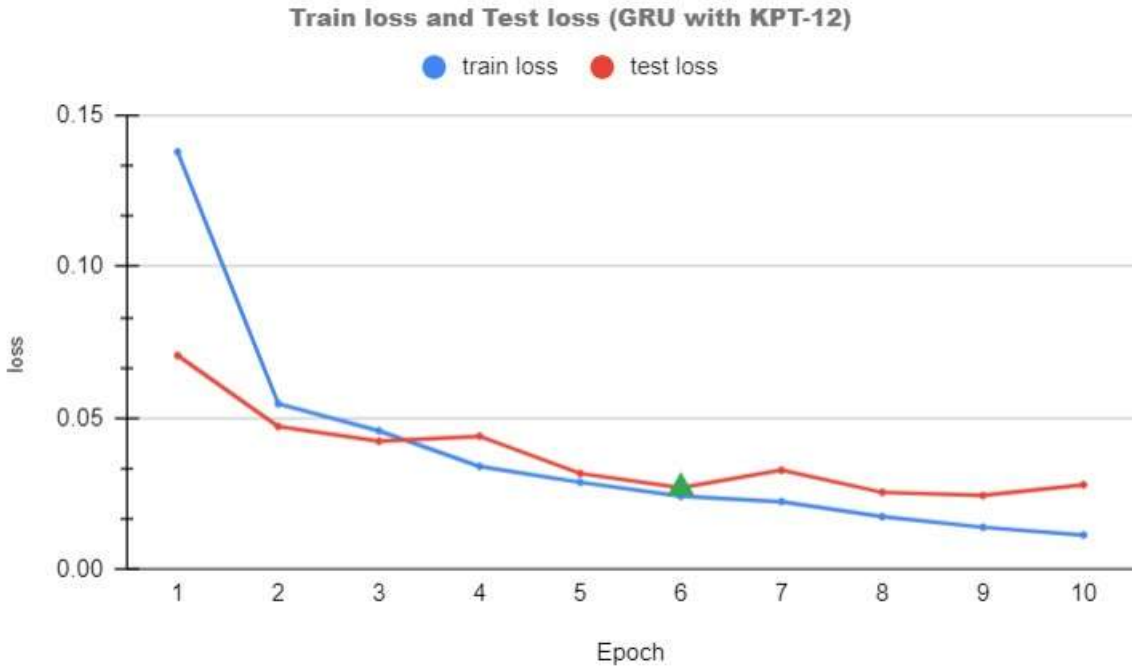

Figure 5.3: GRU model training loss and testing loss

5.3.2 Different Classifiers

The experiment is to apply the same data set to different machine learning models, from which the best performance model can be analyzed. The structure and implementation

principle of each model are different. Under the condition that the structure remains unchanged, the performance of the model will change with the change of parameters. Table 5.4 lists the parameter values applied to train traditional models. After many parameter adjustments, the performance and time-consuming are compared to find the optimal parameters.

Table 5.4. Parameters of three traditional models

Classifier	Parameters
Logistic Regression	Tolerance for stopping criteria = 1e-4 Maximum iterations = 5000
Random Forest	The number of trees in the forest = 1000 Maximum depth of the tree = 5000
SVM	Tolerance for stopping criteria = 1e-4 Maximum iterations = -1 (No limit)

Table 5.5 presents that the RNN-GRU achieved the highest accuracy of 99.18%, and the Random Forest obtained the lowest false-positive rate of 0.0047%. In this experiment, the accuracy and F1 scores of all models were very close. This performance can be seen in Figure 5.4. Because the accuracy of the underlying RNN model is less than 0.9, it is not shown in the figure. From the results data of the three deep learning models, the effects of gate unit and LSTM unit on sequence data training are explained once again.

Table 5.5. Comparison of different classifiers' performance

Classifier	accuracy	F1	False-positive rate	False-negative rate
Logistic Regression	0.9889	0.9888	0.0081	0.0141
Support vector machine (SVM)	0.9885	0.9885	0.01	0.0129
Random Forest	0.985	0.9849	0.0047	0.0253
RNN	0.7412	0.7089	0.1813	0.3372
RNN-GRU	0.9918	0.9915	0.0104	0.0059
RNN-LSTM	0.9895	0.9891	0.0118	0.0089

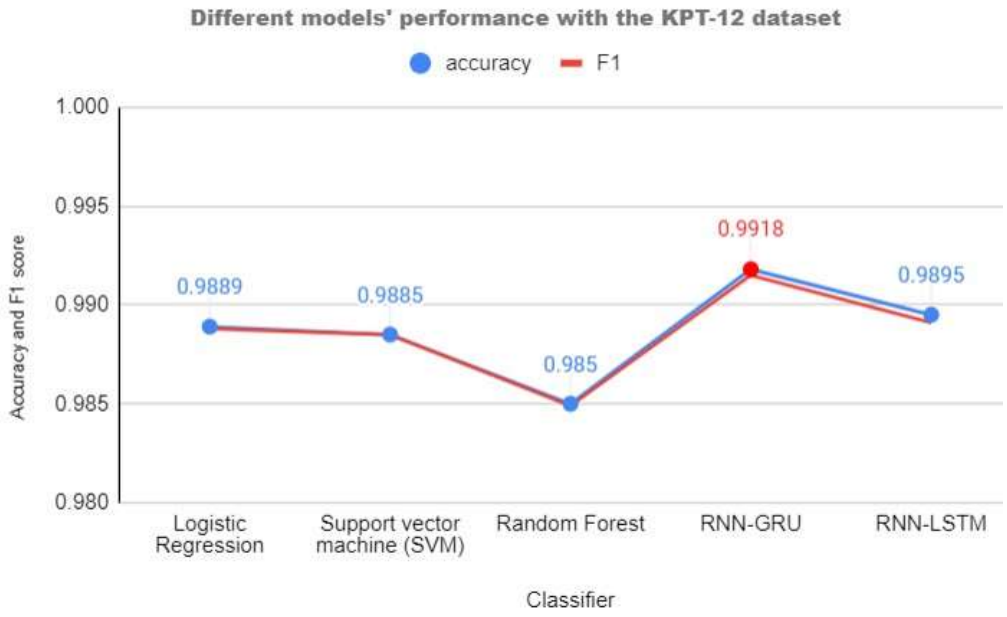

Figure 5.4: Accuracy and F1 score of different models

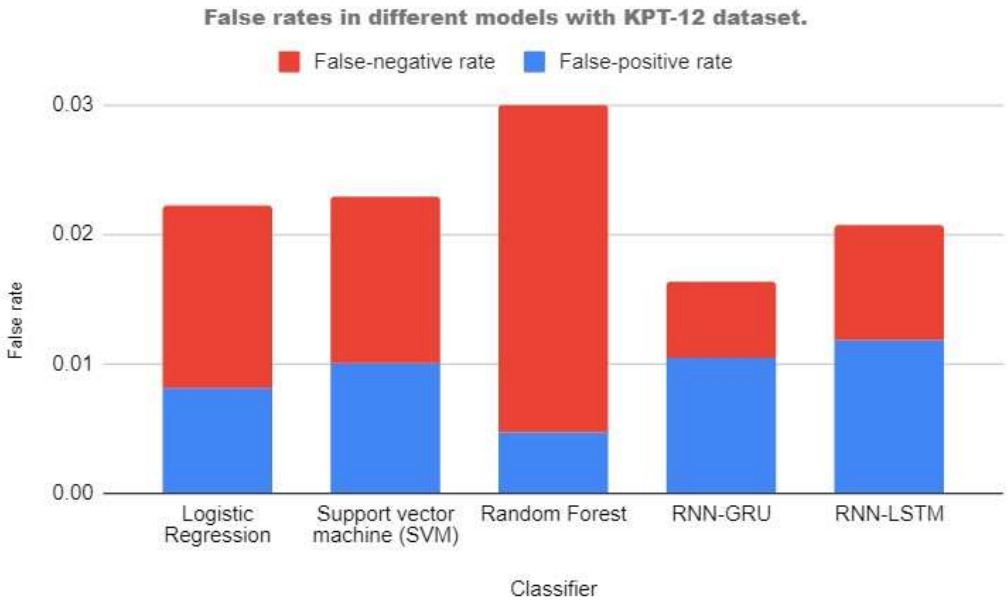

Figure 5.5: False-positive rate and False-negative rate of different models

Although the random forest model gets the lowest false-positive rate, the false-negative rate is the highest, and the sum of the two error rates is the highest.

5.3.3 Hyperparameter Optimization

From the above two experimental results, the KPT-12 dataset applied to the RNN-GRU model obtains the best performance. The third experiment is to optimize the model hyperparameters for better performance. The optional values of parameters are listed in table 5.6. A total of 162 combinations of optional values for all parameters will be performed in turn. Because the computer GPU running the experiment does not support parallel computing and it takes a long time to train a model with the KPT-12 dataset, the experiment will be performed after the system is deployed to the cloud. Access the Tensor Board tool to visualize comparison of execution results and performance metrics to get the best combination of parameters [52].

Table 5.6. Optional values of deep learning parameters

Parameter	Optional value	Default value
Batch size	[32, 64, 128]	32
epoch	[6, 10, 20]	10
shuffle	[true, false]	true
Learning-rate	[0.01, 0.005, 0.001]	0.001
Layer number	[1,2,3]	2

5.3.4 Comparison

This section compares the RNN-GRU model to existing solutions that train deep learning models to detect phishing websites. Table 5.7 shows a comparison from different dimensions, such as data collection, models, performance indicators, limitations. As for the limitations of the proposed solution implementation, since there are no short links in the data set of the training model, all current prediction services cannot accurately detect whether short links are at risk of phishing. Furthermore, only the first 200 characters of the URL are selected. A part of information will be lost when a URL consists of more than 200 characters. In addition, the process of the automatic review report is currently judged based on rules such as remote IP address, client information, and the number of times the URL has been submitted. This strategy can easily be used maliciously by phishing attackers. In the future, more data will be needed to support automatic review results, for example, by obtaining the HTML of the current URL, identifying the similarity between the logo image and the whitelisted website, and whether there is an input box in the HTML.

Table 5.7. Comparison of GRU model with other deep learning-based solutions

Model or Algorithm	**Dataset**	**Limitations**	**Accuracy**
RNN-GRU	Websites (PhishTank, Kaggle); 120,000 instances:60,000 phishing URLs, 60,000 Legitimate URLs; Character-level features based on URL string.	Short URLs are not supported; URLs of more than 200 characters will lose some of their features.	99.18 %
Transfer Learning [53]	Website (Huawei Symantec); 177,417 instances: 36,560 phishing URLs, 14,0857 legitimate URLs; 15 features based on URL string, domain, and sensitive words.	Some feature extractions rely on third-party services.	97%
Reinforcement Learning [54]	Website (PhishTank, Yandex Search engine); 73,575 instances:37,175 phishing URLs, 36,400 legitimate URLs. 14 features based on URL string, domain, and HTML.	Low accuracy; some feature extractions rely on third-party services.	90.1%
Convolutional Autoencoder [55]	Websites (PhishTank, PhishStorm, ISCX-URL-2016); 222,541 instances: 127,628 legitimate URLs, 94,913 phishing URLs; Character-level features based on URL string.	Some long URLs will lose part of their features.	97.82 %

LSTM [56]	Websites (PhishTank, Alexa); 3526 instances: 2119 phishing URLs, 1407 legitimate URLs; 18 features are extracted from URL string, third-party-based features.	some feature extractions rely on third-party services; Fail to detect phishing sites with embedded objects.	99.57 %
CNN+LSTM [57]	Websites (PhishTank, Common Crawl, WHOIS); 1 million URLs, Over 10,000 images; Character-level features based on URL string and features extracted from images.	Low accuracy; Long response time: 25s.	93.28 %

5.4 Heuristic Method

Since the features of web page content need to be extracted in scheme, the web page must be accessible to the browser. But the active time of phishing URLs is short-lived. After one or two days, the phishing URL will be identified by security products or manually reported and taken offline. Therefore, it is really hard to collect a large number of active phishing URLs in a short period of time. On the other hand, there are no available published data sets with features listed in table 3.4.

The results obtained from accessing these phishing links (Appendix B) in a browser with the detection extension installed were consistent with the labels. In addition, from the characteristics of URL links, it can be seen that some traditional characteristics are failing, such as whether the schema is HTTPS. In addition, attackers start using web hosting services, such as Weebly. Domain-related characteristics that rely on third-party services

are benign. Figure 5.6 is an example of a phishing web page with a confidential form, displaying extracted features.

This heuristic rule-based solution has three advantages in terms of efficiency, lightweight, and high user credibility. The response time from the completion of the page DOM structure rendering to the output of the predicted result does not exceed 100 milliseconds on average. It does not rely on third-party services and tools. Although some third-party APIs can obtain more information and features, there are some drawbacks to calling external APIs in real-time in a production environment. First, the response time will increase, affecting the user experience. Second, the stability and security of the service will be affected. The protocol does not require the process of model training, saving a lot of time and computing resources. The forecasting service is rule-based, improving usability through flexible control of rule thresholds, dictionary vocabulary, list of company names. In addition, the prototype implementation is simple, and features can be captured through JavaScript scripts. Although the object recognition technology of computer vision can parse the company or brand name from the picture, the technology has some drawbacks in the actual phishing detection products. Some screenshots of web pages may contain user privacy information, such as the user name of the browser log in, the default user name of the familiar URL, and so on. Therefore, highly trusted user products should avoid using web screenshot technology.

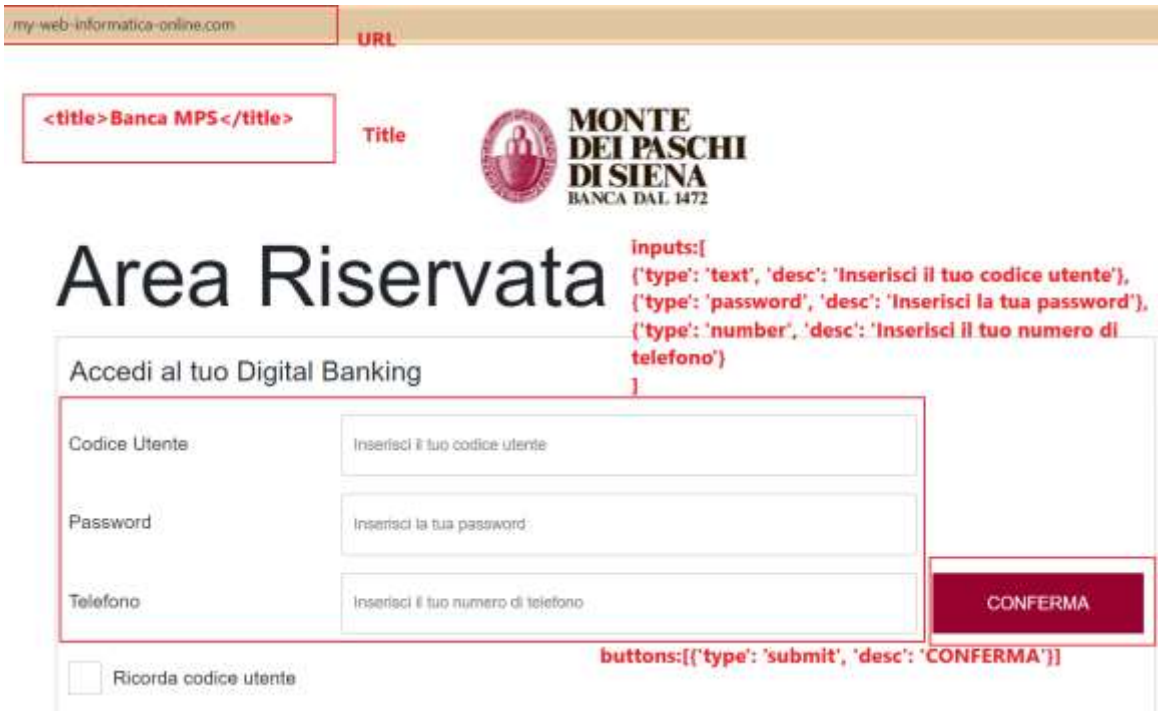

Figure 5.6: A screenshot of a phishing web page with labelled features

However, this solution also has some limitations. First, since the sensitive words dictionary and domain names are in English, the current prototype implementation only supports English websites. Suppose it has to support other languages the sensitive word dictionary should contain multi-language vocabulary, and it need an algorithm to establish a relationship between the web page content and domain name language. Second, if the form that requires the user to enter information is embedded through an iframe, this solution cannot get specific tags and attributes from the HTML source code. Therefore, using an iframe tag to embed another document is a common method by phishing attackers. Many phishing detection models include an iframe tag as an important feature. Furthermore, this method cannot obtain the real HTML source code if the rendering of the sensitive form is triggered by clicking on the page and is controlled by JavaScript. In addition, short URLs might course false alarms. Short link services can make URLs look

more concise, and the smaller number of characters can be easily spread on various social media, but short links will also lose their original semantics, especially when using shortening technology for domain names.

5.5 Chrome extension

The Chrome browser was tested for usability in two scenarios: normal URLs and phishing links. Figure 5.7 shows an example of entering a legitimate URL. In this case, the user opens the page in a web browser with no additional information. When the user clicks the plug-in button on the right side of the toolbar, the popup page is displayed with the current URL string, risk level, and other information.

Figure 5.7: A screenshot of the Chrome extension popup page (legitimate)

Figure 5.8: A screenshot of the Chrome extension displaying warning message

Figure 5.8 presents an example. When the entered URL is detected as a phishing link, a popup box with a red background appears on the page, prompting the user that the website is at phishing risk. If the user confirms that the URL is not a phishing network, they can click the false alarm button to respond to this false alarm. Figure 5.9 shows the style and content of popup pages on high-risk sites.

Compared with the extensions listed in Table 2.2, this Chrome extension has advantages in technological innovation and user experience. The prediction service is based on deep machine learning models capable of predicting dynamic phishing links, and the framework supports dynamically updating datasets and optimizing models. Secondly, the response time of the service is within 200ms, which can quickly respond to the client's situation. In addition, the extension does not store any personal privacy data of users, and does not use screen capture technology to ensure the security and credibility.

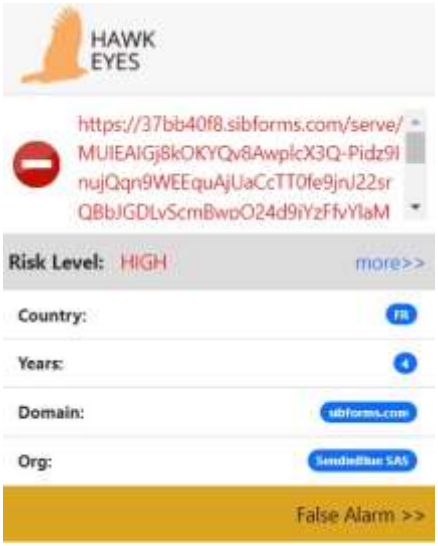

Figure 5.9: A screenshot of the Chrome extension popup page (phishing)

5.6 Summary

This chapter has presented the evaluation of the phishing detection framework, and the evaluation results have shown the presence of phishing attacks. In addition, source code and detailed descriptions of each test case have been provided, thus demonstrating the interworking of framework layers. The following chapter concludes the thesis and presents future work.

Chapter 6

Conclusion and Future Work

This thesis presented the design and evaluation of a framework based on multiple strategies for detecting phishing web pages. The prediction service combines whitelist filter, blacklist blocker, heuristic rules, and a classifier based on a deep learning model with high accuracy of 99.18%. To this end, a web application is built for providing prediction service with HTTPS protocol and processing users' reports. Furthermore, a Chrome extension is developed to capture web page content for analyzing phishing characteristics and extracting novel features used in heuristic strategies. In addition, the scheduled tasks verify reports automatically and async data to the database. Along with this, machine learning models are optimized by training with updated datasets.

In spite of promising results, there remains many goals which if met would further enhance the performance of the framework. In the heuristic strategy, the sensitive words dictionary and domain names are in English, and the current prototype implementation only supports English websites. In addition, short URLs might cause false alarms. Short link services can make URLs look more concise, and the smaller number of characters can be easily spread on various social media, but short links will also lose their original semantics, especially when using shortening technology for domain names. Furthermore, compared with state-of-the-art solutions, the performance of the RNN-GRU model needs to be improved.

For future work, in addition to supporting multiple languages and recognizing short links, Model optimization, mining new features, and building high-quality and high-data data sets will be the main work directions. First, the model training module will be independently deployed on the cloud server with NVIDIA GPUs for increasing efficiency with GPU's parallel computing power. Furthermore, creating a large mount dataset with heuristic features is an important task. Training the model with features extracted by the heuristic strategy is a solution that may achieve high performance. In addition, looking for a more comprehensive and high-quality list of company names is an important task for improving performance of heuristic strategy. In terms of data acquisition and feature extraction, some new techniques will be tried and applied. NLP technology could be used to analyze the text content of the web page to extract practical features, and text information in the picture can be identified with OCR technology.

Bibliography

[1] K. Kaushik, S. Singh, S. Garg, S. Singhal, and S. Pandey, "Exploring the mechanisms of phishing," *Computer Fraud & Security*, vol. 2021, no. 11, pp. 14–19, Nov. 2021, doi: 10.1016/s1361-3723(21)00118-4.

[2] P. by HelpSystems, "Quarterly Threat Trends and Intelligence - November 2021," *info.phishlabs.com*, Nov. 2021. https://info.phishlabs.com/quarterly-threat-trends-and-intelligence-november-2021 (accessed Jan. 27, 2022).

[3] "Phishing Attack Trends Report - 3Q 2021," APWG, Nov. 2021. [Online]. Available: https://docs.apwg.org/reports/apwg_trends_report_q3_2021.pdf.

[4] "2020 Internet Crime Report," 2020. [Online]. Available: https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf.

[5] Y. A. Alsariera, V. E. Adeyemo, A. O. Balogun, and A. K. Alazzawi, "AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites," *IEEE Access*, vol. 8, pp. 142532–142542, 2020, doi: 10.1109/access.2020.3013699.

[6] N. Kumaran, "Spam does not bring us joy—ridding Gmail of 100 million more spam messages with TensorFlow | Google Cloud Blog," *Google Cloud Blog*, Feb. 06, 2019. https://cloud.google.com/blog/products/g-suite/ridding-gmail-of-100-million-more-spam-messages-with-tensorflow.

[7] "Google Safe Browsing," *Google.com*, 2014. https://safebrowsing.google.com/.

[8] A. K. Jain and B. B. Gupta, "A novel approach to protect against phishing attacks at client side using auto-updated white-list," *EURASIP Journal on Information Security*, vol. 2016, no. 1, May 2016, doi: 10.1186/s13635-016-0034-3.

[9] C. L. Tan, K. L. Chiew, K. Wong, and S. N. Sze, "PhishWHO: Phishing webpage

detection via identity keywords extraction and target domain name finder," *Decision Support Systems*, vol. 88, pp. 18–27, Aug. 2016, doi: 10.1016/j.dss.2016.05.005.

[10] K. L. Chiew, E. H. Chang, S. N. Sze, and W. K. Tiong, "Utilisation of website logo for phishing detection," *Computers & Security*, vol. 54, pp. 16–26, Oct. 2015, doi: 10.1016/j.cose.2015.07.006.

[11] B. B. Gupta, K. Yadav, I. Razzak, K. Psannis, A. Castiglione, and X. Chang, "A novel approach for phishing URLs detection using lexical based machine learning in a real-time environment," *Computer Communications*, vol. 175, pp. 47–57, Jul. 2021, doi: 10.1016/j.comcom.2021.04.023.

[12] "URL 2016 | Datasets | Research | Canadian Institute for Cybersecurity | UNB," *www.unb.ca*. https://www.unb.ca/cic/datasets/url-2016.html.

[13] W. Ali and A. Ahmed, "Hybrid Intelligent Phishing Website Prediction Using Deep Neural Networks with Genetic Algorithm-based Feature Selection and Weighting," *IET Information Security*, Jul. 2019, doi: 10.1049/iet-ifs.2019.0006.

[14] A. Basit, M. Zafar, X. Liu, A. R. Javed, Z. Jalil, and K. Kifayat, "A comprehensive survey of AI-enabled phishing attacks detection techniques," *Telecommunication Systems*, Oct. 2020, doi: 10.1007/s11235-020-00733-2.

[15] A. Aljofey, Q. Jiang, Q. Qu, M. Huang, and J.-P. Niyigena, "An Effective Phishing Detection Model Based on Character Level Convolutional Neural Network from URL," *Electronics*, vol. 9, no. 9, p. 1514, Sep. 2020, doi: 10.3390/electronics9091514.

[16] W. Wang, F. Zhang, X. Luo, and S. Zhang, "PDRCNN: Precise Phishing Detection with Recurrent Convolutional Neural Networks," *Security and Communication*

Networks, vol. 2019, pp. 1–15, Oct. 2019, doi: 10.1155/2019/2595794.

[17] "Alexa.com," *Alexa.com*, 2016. https://www.alexa.com/ (accessed Jul. 18, 2021).

[18] "PhishTank | Join the fight against phishing," *www.phishtank.com*. https://www.phishtank.com/index.php.

[19] M. G. HR, A. MV, G. P. S, and V. S, "Development of anti-phishing browser based on random forest and rule of extraction framework," *Cybersecurity*, vol. 3, no. 1, Oct. 2020, doi: 10.1186/s42400-020-00059-1.

[20] G. Armano, S. Marchal, and N. Asokan, "Real-Time Client-Side Phishing Prevention Add-On," *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2016, doi: 10.1109/icdcs.2016.44.

[21] S. Marchal, K. Saari, N. Singh, and N. Asokan, "Know Your Phish: Novel Techniques for Detecting Phishing Sites and Their Targets," *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2016, doi: 10.1109/icdcs.2016.10.

[22] M. M. Varjani Ali Yazdian, "PhishDetector | A true phishing detection system," *PhishDetector Landing Page*, 2013. https://www.moghimi.net/phishdetector.

[23] "Netcraft," *Netcraft*. https://www.netcraft.com/ (accessed Feb. 05, 2022).

[24] W. S. Ltd, "Website Safety Check & Phishing Protection | Web of Trust," *www.mywot.com*. https://www.mywot.com/ (accessed Feb. 05, 2022).

[25] H. Shaper, "Home," *Pixm Anti-Phishing*. https://pixm.net/ (accessed Feb. 05, 2022).

[26] A. Bannister, "Sharkcop: Google Chrome extension uses machine learning to detect phishing URLs," *The Daily Swig | Cybersecurity news and views*, Oct. 05, 2020. https://portswigger.net/daily-swig/sharkcop-google-chrome-extension-uses-

machine-learning-to-detect-phishing-urls.

[27] "PhishFort Protect Anti-Phishing Cryptocurrency Browser Extension," *www.phishfort.com*. https://www.phishfort.com/protect (accessed Feb. 05, 2022).

[28] N. Gupta *et al.*, "Data Quality for Machine Learning Tasks," *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Aug. 2021, doi: 10.1145/3447548.3470817.

[29] S. Marchal, J. François, R. State, and T. Engel, "PhishStorm: Detecting Phishing With Streaming Analytics," *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 458–471, Dec. 2014, doi: 10.1109/TNSM.2014.2377295.

[30] Kaggle, "Kaggle: Your Home for Data Science," *Kaggle.com*, 2019. https://www.kaggle.com/ (accessed Feb. 05, 2022).

[31] "URL Structure [2020 SEO Best Practices]," *Moz*. https://moz.com/learn/seo/url (accessed Feb. 05, 2022).

[32] S. S., J. I. Zong Chen, and S. Shakya, "Survey on Neural Network Architectures with Deep Learning," *Journal of Soft Computing Paradigm*, vol. 2, no. 3, pp. 186–194, Jul. 2020, doi: 10.36548/jscp.2020.3.007.

[33] S. Seo, C. Kim, H. Kim, K. Mo, and P. Kang, "Comparative Study of Deep Learning-Based Sentiment Classification," *IEEE Access*, vol. 8, pp. 6861–6875, 2020, doi: 10.1109/access.2019.2963426.

[34] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[35] K. Cho *et al.*, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv.org*, 2014. https://arxiv.org/abs/1406.1078.

[36] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv.org*, 2014. https://arxiv.org/abs/1412.3555.

[37] A. Shewalkar, D. Nyavanandi, and S. A. Ludwig, "Performance Evaluation of Deep Neural Networks Applied to Speech Recognition: RNN, LSTM and GRU," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 9, no. 4, pp. 235–245, Oct. 2019, doi: 10.2478/jaiscr-2019-0006.

[38] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv.org*, 2014. https://arxiv.org/abs/1412.6980.

[39] Y. Ho and S. Wookey, "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling," *IEEE Access*, vol. 8, pp. 4806–4813, 2020, doi: 10.1109/access.2019.2962617.

[40] "HTML Standard," *html.spec.whatwg.org*. https://html.spec.whatwg.org/multipage/#toc-semantics (accessed Feb. 05, 2022).

[41] S. Zhang, Y. Hu, and G. Bian, "Research on string similarity algorithm based on Levenshtein Distance," *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Mar. 2017, doi: 10.1109/iaeac.2017.8054419.

[42] "Welcome to Flask — Flask Documentation (2.0.x)," *flask.palletsprojects.com*. https://flask.palletsprojects.com/en/2.0.x/ (accessed Feb. 05, 2022).

[43] "scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation,"

Scikit-learn.org, 2019. https://scikit-learn.org/stable/index.html (accessed Feb. 05, 2022).

[44] PyTorch, "PyTorch," *Pytorch.org*, 2019. https://pytorch.org/ (accessed Feb. 05, 2022).

[45] "Developing extensions for Chrome," *Chrome Developers*. https://developer.chrome.com/docs/extensions/mv3/ (accessed Feb. 05, 2022).

[46] V. Babel, K. Singh, S. Kumar Jangir, B. Singh, and S. Kumar, "Journal of Analysis and Computation (JAC) EVALUATION METHODS FOR MACHINE LEARNING," 2019. [Online]. Available: http://www.ijaconline.com/wp-content/uploads/2019/06/ICITDA_2019_paper_88-.pdf.

[47] H. N. A. Pham and E. Triantaphyllou, "The Impact of Overfitting and Overgeneralization on the Classification Accuracy in Data Mining," *Soft Computing for Knowledge Discovery and Data Mining*, pp. 391–431, 2008, doi: 10.1007/978-0-387-69935-6_16.

[48] IBM Cloud Education, "What is Overfitting?," *www.ibm.com*, Mar. 03, 2021. https://www.ibm.com/cloud/learn/overfitting.

[49] X. Ying, "An Overview of Overfitting and its Solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022022, Feb. 2019, doi: 10.1088/1742-6596/1168/2/022022.

[50] E. Gandotra and D. Gupta, "Improving Spoofed Website Detection Using Machine Learning," *Cybernetics and Systems*, vol. 52, no. 2, pp. 169–190, Oct. 2020, doi: 10.1080/01969722.2020.1826659.

[51] G. Harinahalli Lokesh and G. BoreGowda, "Phishing website detection based on

effective machine learning approach," *Journal of Cyber Security Technology*, pp. 1–14, Aug. 2020, doi: 10.1080/23742917.2020.1813396.

[52] "Visualizing Models, Data, and Training with TensorBoard — PyTorch Tutorials 1.9.1+cu102 documentation," *pytorch.org*. https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html (accessed Feb. 05, 2022).

[53] J. Zhang, Y. Ou, D. Li, and Y. Xin, "A Prior-based Transfer Learning Method for the Phishing Detection," *Journal of Networks*, vol. 7, no. 8, Aug. 2012, doi: 10.4304/jnw.7.8.1201-1207.

[54] M. Chatterjee and A.-S. . Namin, "Detecting Phishing Websites through Deep Reinforcement Learning," *IEEE Xplore*, 2019. https://ieeexplore.ieee.org/document/8754075 (accessed Apr. 16, 2021).

[55] S.-J. Bu and S.-B. Cho, "Deep Character-Level Anomaly Detection Based on a Convolutional Autoencoder for Zero-Day Phishing URL Detection," *Electronics*, vol. 10, no. 12, p. 1492, Jun. 2021, doi: 10.3390/electronics10121492.

[56] M. Somesha, A. R. Pais, R. S. Rao, and V. S. Rathour, "Efficient deep learning techniques for the detection of phishing websites," *Sādhanā*, vol. 45, no. 1, Jun. 2020, doi: 10.1007/s12046-020-01392-4.

[57] M. A. Adebowale, K. T. Lwin, and M. A. Hossain, "Intelligent phishing detection scheme using deep learning algorithms," *Journal of Enterprise Information Management*, vol. ahead-of-print, no. ahead-of-print, Jun. 2020, doi: 10.1108/jeim-01-2020-0036.

[58] X. Xiao, D. Zhang, G. Hu, Y. Jiang, and S. Xia, "CNN–MHSA: A Convolutional

Neural Network and multi-head self-attention combined approach for detecting phishing websites," *Neural Networks*, vol. 125, pp. 303–312, May 2020, doi: 10.1016/j.neunet.2020.02.013.

[59] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," *Expert Systems with Applications*, vol. 117, pp. 345–357, Mar. 2019, doi: 10.1016/j.eswa.2018.09.029.

[60] A. Ozcan, C. Catal, E. Donmez, and B. Senturk, "A hybrid DNN–LSTM model for detecting phishing URLs," *Neural Computing and Applications*, Aug. 2021, doi: 10.1007/s00521-021-06401-z.

[61] A. Odeh and I. Keshta, "PhiBoost- A novel phishing detection model Using Adaptive Boosting approach," *Jordanian Journal of Computers and Information Technology*, vol. 7, no. 1, p. 64, 2021, doi: 10.5455/jjcit.71-1600061738.

[62] X. Liu and J. Fu, "SPWalk: Similar Property Oriented Feature Learning for Phishing Detection," *IEEE Access*, vol. 8, pp. 87031–87045, 2020, doi: 10.1109/access.2020.2992381.

[63] J. Feng, Y. Zhang, and Y. Qiao, "A Detection Method for Phishing Web Page Using DOM-Based Doc2Vec Model," *CIT. Journal of Computing and Information Technology*, vol. 28, no. 1, pp. 19–31, Jul. 2020, Accessed: Feb. 07, 2022. [Online]. Available: http://cit.fer.hr/index.php/CIT/article/view/4899.

[64] P. A. Barraclough, G. Fehringer, and J. Woodward, "Intelligent cyber-phishing detection for online," *Computers & Security*, vol. 104, p. 102123, May 2021, doi: 10.1016/j.cose.2020.102123.

[65] T. Mitsa, "How Do You Know You Have Enough Training Data?," *Medium*, Apr.

23, 2019. https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee#:~%7B%7D:text=Computer%20Vision%3A%20For%20image%20classification (accessed Feb. 07, 2022).

[66] R. M. Mohammad, L. McCluskey, and F. Thabtah, "UCI Machine Learning Repository: Phishing Websites Data Set," *archive.ics.uci.edu*, Mar. 26, 2015. https://archive.ics.uci.edu/ml/datasets/Phishing+Websites.

[67] Jerry, F.; Chris, H. System Security: A Hacker's Perspective. In Proceedings of the 1987 North American conference of Hewlett-Packard business computer users, Las Vegas, NV, USA, 20–25 September 1987. [68]"Browser Market Share Worldwide," *StatCounter Global Stats*. https://gs.statcounter.com/browser-market-share#monthly-202011-202011-bar (accessed Feb. 25, 2022).

[69] "HAWK EYES Chrome Extension," *www.thehawkeyes.com*. https://www.thehawkeyes.com/products/ (accessed Feb. 05, 2022).

[70] L. Tang and Q. H. Mahmoud, "A Deep Learning-Based Framework for Phishing Website Detection," in IEEE Access, vol. 10, pp. 1509-1521, 2022. doi: 10.1109/ACCESS.2021.3137636.

[71] L. Tang and Q. H. Mahmoud, "A Survey of Machine Learning-Based Solutions for Phishing Website Detection," Machine Learning and Knowledge Extraction, vol. 3, no. 3, pp. 672–694, Aug. 2021. Doi: https://doi.org/10.3390/make3030034.

[72] "HAWK EYES Chrome Extension Source Code," *GitHub*, Apr. 2022. https://github.com/tanglz/extension (accessed Apr. 16, 2022).

[73] G. S. Rao et al, "Spam or Ham Text Classification using Logistic

Regression," *Turkish Journal of Computer and Mathematics Education*, vol. 12, (9), pp. 426-433, 2021. Available: http://search.proquest.com.uproxy.library.dc-uoit.ca/scholarly-journals/spam-ham-text-classification-using-logistic/docview/2623462315/se-2.

[74] K. R. Jansi and M. Arulprakash, "An Effective Model of Terminating Phishing Websites and Detection Based On Logistic Regression," *Turkish Journal of Computer and Mathematics Education*, vol. 12, (9), pp. 358-362, 2021. Available: http://search.proquest.com.uproxy.library.dc-uoit.ca/scholarly-journals/effective-model-terminating-phishing-websites/docview/2623460989/se-2.

[75] R. Shwartz-Ziv and N. Tishby, "Opening the Black Box of Deep Neural Networks via Information," *arXiv.org*, 2017. https://arxiv.org/abs/1703.00810.

[76] Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes, and Brown, "Text Classification Algorithms: A Survey," *Information*, vol. 10, no. 4, p. 150, Apr. 2019, doi: 10.3390/info10040150.

Appendix

Appendix A. Source Code

Source code files related to Chrome extension, data collection tasks, and machine learning model training.

A1. Chrome extension

The objective of this source file is to listen tab update event and extract HTML features as parameters of heuristic rules.

- background.js

- ```
  async function storeCurrentTabUrl() {
    let queryOptions = { active: true, currentWindow: true };
    let [tab] = await chrome.tabs.query(queryOptions);
    currentUrl = tab.url;
    chrome.storage.sync.set({'current_url': currentUrl});
    chrome.storage.sync.set({'data': {}});
    chrome.tabs.sendMessage(tab.id, {
          message: 'check',
          currentUrl: currentUrl,
          tabId: tab.id
    });
    return tab.url;
  }

  chrome.tabs.onUpdated.addListener(function (tabId, changeInfo)
  {
      if (changeInfo.status == 'complete') {
          storeCurrentTabUrl();
      }
  });
  chrome.tabs.onActivated.addListener(function (activeInfo) {
      storeCurrentTabUrl();
  });

  chrome.runtime.onMessage.addListener(
    function(request, sender, sendResponse) {
      if (request.message === "DOM"){
          api_url =
  "https://www.api.thehawkeyes.com/predict/ai";
          fetch(api_url, {
              method: 'POST',
              headers: {
                  'Content-Type': 'application/json',
  ```
  ```

```
                },
                body: JSON.stringify({'url':
      request.currentUrl,'num_input':request.num_input,'num_button':
      request.num_button,'title':request.title,
      'inputs':request.inputs,'buttons':request.buttons}),
              })
              .then(response => response.json())
              .then(data => {
                  chrome.storage.sync.set({'data': data});
                  sendResponse(data);
              });
          }
          return true;
        }
      );
```

- contentscript.js

```
chrome.runtime.onMessage.addListener(
  function(request, sender, sendResponse) {
    if (request.message === "check"){
        $( document ).ready(function() {
            title = $('title').text();
            placeholders=[]
            inputs = []
            buttons=[]
            $('input').each(function(index){
                type = $(this).attr('type');
                if(type==''|| typeof type ==="undefined"){
                    type="text"
                }
                valid_input_types =
['text','number','password','search','email','tel'];
                valid_button_types = ['submit','button']
                if($.inArray(type, valid_input_types)>-1){
                    val = $(this).attr('value');
                    placeholder = $(this).attr('placeholder');
                    desc = val || placeholder;
                    if(desc==''|| typeof desc === "undefined"){
                        prev = $(this).prev()[0];
                        next = $(this).next()[0];
                        if($.isEmptyObject(prev)){
                            if($.isEmptyObject(next)){
                                // other structures
                            }else{
                                desc = $(next).text();
                            }
                        }else{
                            desc = $(prev).text();
```

```
                        }
                    }
                    input = {'type':type,'desc': desc};
                    inputs.push(input);
                    num_input=num_input+1;
                }
                if($.inArray(type, valid_button_types)>-1){
                    button = {'type':type,'desc':
$(this).attr('value')};
                    buttons.push(button)
                }
            });

            $('button').each(function(index){
                btn={'type':'button','desc': $(this).text()}
                buttons.push(btn)
                num_button=num_button+1;
            });
            $('textarea').each(function(index){
                num_input=num_input+1;
            });
            chrome.runtime.sendMessage({
                message: 'DOM',
                currentUrl: request.currentUrl,
                num_input: num_input,
                num_button:num_button,
                title:title,
                inputs:inputs,
                buttons: buttons
            }, function(response) {
              if (response.phishing) {
                    chrome.storage.sync.get("exclude_url_list", ({
exclude_url_list }) => {
                        if(exclude_url_list &&
exclude_url_list.includes(request.currentUrl)){
                            return;
                        }else{
                            const URL =
"https://www.api.thehawkeyes.com/verify/add?error_type=2&url="
+request.currentUrl
                            var elemDiv =
document.createElement('div');
                            elemDiv.innerHTML =
hawk_eyes_alarmModal;
                            document.body.appendChild(elemDiv);
                            // Get the <span> element that closes
the modal
                            const closeBtns =
document.getElementsByClassName("phishing-alarm-close-btn");
                            // When the user clicks on <span> (x),
close the modal
                            if(exclude_url_list){
                                exclude_url_list.push(link);
```

```
                                }else{
                                    exclude_url_list = [link];
                                }
                                for (var i = 0; i < closeBtns.length;
i++) {

closeBtns[i].addEventListener('click', function(event){
                                        closeAction();

chrome.storage.sync.set({"exclude_url_list":exclude_url_list});
                                    });
                                }
                                const reportBtn =
document.getElementById("report-phishing-false-alarm");

reportBtn.addEventListener('click',function (event){
                                    closeAction();

chrome.storage.sync.set({"exclude_url_list":exclude_url_list});
                                    window.open(URL, '_blank').focus();
                                });
                            }
                        });
                }else{
                    if(response.source=='report'){
                       count = response.num_users
                    }

                }
            });
        });
    }
  }
);
```

A2. Data collection

This source file is used to collection data from the PhishTank website. References were made from a Python third-party library name BeautifulSoup.

```
def fetch_page(start_page, end_page):
    list_link = 'https://phishtank.org/phish_search.php'
    detail_link = 'https://phishtank.org/phish_detail.php?phish_id='
    parameters = {"page": start_page, "valid": 'y', 'Search':
'Search', 'active': 'n'}
    headers = {
        'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X
10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87
Safari/537.36',
    }
    with open('phishTank_inactive.csv', 'a+', newline='') as
```

```
csvfile:
        fieldnames = ['phish_id', 'url']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        # writer.writeheader()
        for i in range(start_page, end_page):
            list_page = requests.get(list_link, params=parameters,
headers=headers)
            soup = BeautifulSoup(list_page.content,
features="html.parser")
            table = soup.find("table")
            trs = table.findAll("tr")
            for tr in trs:
                tds = tr.findAll('td')
                if len(tds) >= 1:
                    td_id = tds[0]
                    phish_id = td_id.find('a').text
                    is_exist = is_exist_record(phish_id,
'phishTank')
                    if is_exist:
                        continue
                    try:
                        detail_page = requests.get(detail_link +
phish_id, headers=headers)
                        detail_soup =
BeautifulSoup(detail_page.content, features="html.parser")
                        url = detail_soup.find('div', {'class':
'url'})
                        div = url.find_next_sibling('div')
                        bb = div.find('b')
                        full_url = bb.text
                        URL_OBJECT = url_parse(full_url)
                        URL_OBJECT.external_id = phish_id
                        URL_OBJECT.source = 'phishTank'
                        URL_OBJECT.status = -1
                        URL_OBJECT.result = 1
                        URL_OBJECT.insert()
                        sleep(1)
                        # write to csv
                        writer.writerow({'phish_id': phish_id,
'url': full_url})
                        print('success:',phish_id)
                    except:
                        print(detail_link + phish_id)
            parameters["page"] = i + 1
```

A3. Traditional machine learning

This source file handles traditional models training and testing. References were made from source such as https://scikit-learn.org/stable/supervised_learning.html#supervised-learning.

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import train_test_split
import pickle
from AI.utils.helpers import save_ml_version, get_next_version,
get_model_file_name, get_transformer_file_name
from AI.utils.metrics_util import basic_metrics_ml,
save_confusion_matrix
from AI.ml.traditional_classifier import TraditionalClassifier
from AI.utils.model_util import data_preparation,
read_data_multi_source_limit


def feature_selection(df, feature):
    feature_new = SelectPercentile(f_classif,
percentile=30).fit_transform(feature, df.result)
    return feature_new


def ML_training(source_list, classifier_code, files_path, plot_path,
limitation=None):
    print('start---------')
    MLParameters = {
        'max_iter': 50000,
        'max_depth': 5000,
        'random_state': 0,
        'test_ratio': 0.2,
        'tokenizer_pattern': r'[A-Za-z]+',
        'feature_vector': 'cv',
        'data_size': limitation * len(source_list)
    }
    # loading data
    df = read_data_multi_source_limit(source_list, limitation)
    # # tokenizer data preprocessing
    df = data_preparation(df, MLParameters['tokenizer_pattern'])
    # # feature extraction
    if MLParameters['feature_vector'] == 'cv':
        vec = CountVectorizer()
    else:
        # collect token TF-IDF
        vec = TfidfVectorizer()
    # transform all text which tokenize and stemed
    vec.fit_transform(df.text)
```

```
    feature = vec.fit_transform(df.text)
    # # split dataset for training and testing
    trainX, testX, trainY, testY = train_test_split(feature,
df.result, test_size=MLParameters['test_ratio'])
    # # modelling
    clf = TraditionalClassifier(classifier_code,
MLParameters).classifier
    clf.fit(trainX, trainY)
    # # performance metrics
    predictions = clf.predict(testX)
    cm = confusion_matrix(testY, predictions)
    sns.heatmap(cm, annot=True)
    fn_rate, fp_rate, acc, precision, recall, f1 =
basic_metrics_ml(predictions, testY)
    # print(accuracy)
    # # save the model and transform
    # get the latest version from DB
    str_sources = ' '.join(source_list)
    cate = classifier_code + '_' + str_sources
    version = get_next_version(cate)
    model_file_name = get_model_file_name(cate, version)
    transformer_file_name = get_transformer_file_name(cate, version)
    pickle.dump(clf, open(files_path + model_file_name, 'wb'))
    pickle.dump(vec, open(files_path + transformer_file_name, 'wb'))
    # # save performance metrics : accuracy, mcc, f1
    save_ml_version(version, acc, f1, fn_rate, fp_rate, cate,
MLParameters)
    # # plot results
    save_confusion_matrix(cate, plot_path, clf, testX, testY,
version)
    # save_precision_recall_curve(cate, plot_path, clf, testX,
testY)
    print('end------')
```

A4. Deep learning

This source file handles deep learning models training and testing. References were made from source such as https://pytorch.org/text/stable/index.html.

```
import csv
from itertools import product

import numpy as np
import torch
from torch import nn
from torch.utils.data import SubsetRandomSampler
from torch.utils.tensorboard import SummaryWriter

from AI.dl.dataset_mysql import SqlDataset
```

```
from AI.dl.gru_classifier import GRUClassifier
from AI.dl.training import train, evaluate
from AI.utils.helpers import get_next_version, get_model_file_name,
save_ml_version, get_plot_info_file_name

def run_character(source_list, classifier_code, files_path,
plot_path, limitation=None):
    print('rnn start--------')
    rnn_parameters = {
        'N_EPOCHS': 10,
        'datasource': source_list,
        'data size': limitation * len(source_list)
    }
    dataset = SqlDataset(source_list, limitation)
    batch_size = 32
    test_split = .2
    shuffle_dataset = True
    random_seed = 42
    # Creating data indices for training and validation splits:
    dataset_size = len(dataset)
    indices = list(range(dataset_size))
    split = int(np.floor(test_split * dataset_size))
    if shuffle_dataset:
        np.random.seed(random_seed)
        np.random.shuffle(indices)
    train_indices, test_indices = indices[split:], indices[:split]
    # Creating PT data samplers and loaders:
    train_sampler = SubsetRandomSampler(train_indices)
    test_sampler = SubsetRandomSampler(test_indices)

    # Hyperparameters
    # parameters = dict(
    #     lr=[0.01, 0.005, 0.001],
    #     batch_size=[32, 64, 128],
    #     shuffle=[True, False],
    #     num_layers=[1, 2, 3],
    #     num_epochs=[10, 20]
    # )
    parameters = dict(
        lr=[0.005],
        batch_size=[64],
        shuffle=[True],
        num_layers=[2],
        num_epochs=[10]
    )
    param_values = [v for v in parameters.values()]
    for run_id, (lr, batch_size, shuffle, num_layers, num_epochs) in
enumerate(product(*param_values)):
        print("run id:", run_id + 1)

        train_loader = torch.utils.data.DataLoader(dataset,
batch_size=batch_size,
```

```
sampler=train_sampler)
        test_loader = torch.utils.data.DataLoader(dataset,
batch_size=batch_size,

sampler=test_sampler)
        # # initial custom model
        input_size = 100  # the number of all character
        sequence_length = 200  # the number of url characters
        # num_layers = 2  # the number of RNNs
        hidden_size = 128
        num_classes = 2
        model = GRUClassifier(input_size, hidden_size, num_layers,
num_classes)
        # # Loss and Optimizer
        learning_rate = 0.005
        criterion = nn.CrossEntropyLoss()
        # optimizer = torch.optim.Adam(model.parameters(),
lr=learning_rate)
        optimizer = torch.optim.Adam(model.parameters())
        device = 'cuda' if torch.cuda.is_available() else 'cpu'
        # # Train the model
        model = model.to(device)
        criterion = criterion.to(device)
        device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
        tb = SummaryWriter('runs/')
        # # # model graph and feature grid
        # fea, labels = next(iter(train_loader))
        # tb.add_graph(model, fea)
        # tb.close()
        best_valid_loss = float('inf')
        train_loss_values = []
        valid_loss_values = []
        train_acc_values = []
        valid_acc_values = []
        mAP_values =[]
        for epoch in range(num_epochs):
            # # training
            train_loss, train_acc = train(model, train_loader,
device, criterion, optimizer)
            # # testing
            valid_loss, valid_acc, f1, fn, fp, mAP = evaluate(model,
test_loader, device, criterion)
            # # plot result
            if valid_loss < best_valid_loss:
                best_valid_loss = valid_loss
            print(f'\tTrain Loss: {train_loss:.3f} | Train Acc:
{train_acc * 100:.2f}%')
            print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc:
{valid_acc * 100:.2f}%')
            print(f'\t mAP: {mAP:.3f} ')
            train_loss_values.append(train_loss)
            valid_loss_values.append(valid_loss)
```

```
            train_acc_values.append(train_acc)
            valid_acc_values.append(valid_acc)
            mAP_values.append(mAP)
            tb.add_scalar("Train Loss", train_loss, epoch)
            tb.add_scalar("valid Loss", valid_loss, epoch)
            tb.add_scalar("fp", fp, epoch)
            tb.add_scalar("fn", fn, epoch)
            tb.add_scalar("f1", f1, epoch)
            tb.add_scalar("Test Accuracy", valid_acc, epoch)
        # # save model
        # get the latest version from DB
        str_sources = '_'.join(source_list)
        cate = classifier_code + '_' + str_sources
        version = get_next_version(cate)
        model_file_name = get_model_file_name(cate, version)
        torch.save(model, files_path + model_file_name)
        # # save performance metrics : model_version, accuracy, f1,
fn, fp, category, parameters
        rnn_parameters['N_EPOCHS'] = num_epochs
        rnn_parameters['lr'] = lr
        rnn_parameters['batch_size'] = batch_size
        rnn_parameters['shuffle'] = shuffle
        rnn_parameters['num_layers'] = num_layers
        save_ml_version(version, valid_acc, f1, fn, fp, cate,
rnn_parameters)
        # # plot
        plot_file_name = get_plot_info_file_name(cate, version)
        with open(plot_path + plot_file_name, 'a+') as f:
            writer = csv.writer(f)
            writer.writerow(train_loss_values)
            writer.writerow(valid_loss_values)
            writer.writerow(train_acc_values)
            writer.writerow(valid_acc_values)
            writer.writerow(mAP_values)

        tb.add_hparams(
            {"lr": lr, "bsize": batch_size, "shuffle": shuffle,
'epoch': epoch, 'num_layers': num_layers},
            {
                "accuracy": valid_acc,
                "loss": valid_loss
            },
        )
    tb.close()
    print('rnn end------')
```

Appendix B. Dataset

B1. Phishing Links for Testing

This table presents ten phishing links collected from the PhishTank website on January 12, 2022.

Table A phishing URL dataset

	URL	Title	Inputs	Buttons	Label	Result
1	https://my-web-informatica-online.com/	BancaMPS	[{'type': 'text', 'desc': 'Inserisci il tuo codice utente'}, {'type': 'password', 'desc': 'Inserisci latua password'}, {'type': 'number', 'desc':'Inserisci iltuo numero ditelefono'}]	[{'type': 'submit', 'desc': 'confirm'}]	1	1
2	https://btbroadbandplc01.weebly.com/	.	[{'type': 'text'}, {'type': 'text'}]	[{'type': 'submit', 'desc': 'Sign in'}]	1	1
3	https://hwdiugeuiubweg.weebly.com/	email login page	[{'type': 'text'}, {'type': 'text'}]	[{'type': 'submit', 'desc': 'Sign in'}]	1	1
4	https://iyuy769.weebly.com/	…	[{'type': 'text'}, {'type': 'text'}, {'type': 'text'}]	[{'type': 'submit', 'desc': 'Sign in'}]	1	1
5	https://cupaie.weebly.com/	mail.yahoo.com-Home	[{'type': 'text'}, {'type': 'text'}]	[{'type': 'submit', 'desc': 'SIGN IN'}]	1	1
6	https://www.brooks-ooke.top/h-	Login/Register	[{'type': 'text', 'desc': 'Search...'}, {'type': 'text'},	[{'type': 'submit', 'desc':	1	1

	user-LoginOrRegister.html	- www.brooks-ooke.top	{'type': 'password', 'desc':}, {'type': 'text'}, {'type': 'password', 'desc':}, {'type': 'text', 'desc':}, {'type': 'text', 'desc':}]	'Login'}, {'type': 'submit', 'desc': 'Create My Account'}, {'type': 'button', 'desc': _}, {'type': 'button', 'desc': _}]		
7	https://www.brooksair.top/h-user-LoginOrRegister.html	Login/Register - www.brooksair.top	[{'type': 'text', 'desc': 'Search...'}, {'type': 'text'}, {'type': 'password', 'desc':}, {'type': 'text'}, {'type': 'password', 'desc':}, {'type': 'password', 'desc':}, {'type': 'text', 'desc':}, {'type': 'text', 'desc':}]	[{'type': 'submit', 'desc': 'Login'}, {'type': 'submit', 'desc': 'Create My Account'}, {'type': 'button', 'desc': '_'}]	1	1
8	https://www.brooks-forrunning.top/h-user-LoginOrRegister.html	Login/Register - www.brooks-forrunning.topBack To Top	[{'type': 'text', 'desc': 'Search...'}, {'type': 'text'}, {'type': 'password', 'desc':}, {'type': 'text'}, {'type': 'password', 'desc':}, {'type': 'password', 'desc':}, {'type': 'text', 'desc':}]	[{'type': 'submit', 'desc': 'Login'}, {'type': 'submit', 'desc': 'Create My Ac-	1	1

				count'}, {'type': 'button', 'desc': _}, {'type': 'button', 'desc': _}]		
9	https://att-yahoo-100130.square.site/	Home	[{'type': 'email', 'desc': 'Email *'}, {'type': 'text', 'desc': 'Pass-word *'}]	[{'type': 'button', 'desc': 'Check-out'}, {'type': 'button', 'desc': 'SIGN IN '}]	1	1
10	https://fifththirdbankonline.weebly.com/	Home	[{'type': 'text'}, {'type': 'text'}]	[{'type': 'submit', 'desc': 'NEXT'}, {'type': 'button', 'desc': _}]	1	1