AN ACTIVE SELF-LEARNING MODEL FOR DECEPTIVE PHISHING DETECTION

Subhash Niroshan Ariyadasa

188077D

Doctor of Philosophy

Department of Computational Mathematics
Faculty of Information Technology

University of Moratuwa
Sri Lanka

July 2023

AN ACTIVE SELF-LEARNING MODEL FOR DECEPTIVE PHISHING DETECTION

Subhash Niroshan Ariyadasa

188077D

Dissertation submitted in partial fulfillment of the requirements for the degree
Doctor of Philosophy

Department of Computational Mathematics
Faculty of Information Technology

University of Moratuwa
Sri Lanka

July 2023

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: _ _ _ _ _ _ _ _ _ _ _ _ _ Date: _ 20-07-2023 _

The above candidate has carried out research for the PhD Dissertation under my supervision.

Name of the supervisor: Dr. K. S. D. Fernando

Signature of the supervisor: _ _ _ _ _ _ _ _ _ _ _ _ _ Date: _ 21/07/2023 _

Name of the supervisor: Prof. M. S. D. Fernando

Signature of the supervisor: _ _ _ _ _ _ _ _ _ _ _ _ _ Date: _ 21-07-2023 _

DEDICATION

This work is entirely dedicated to my respectful parents and my family. Without their constant support, this dissertation would not have been possible. They always inspire me. At the same time, my special thanks also go to my supervisors, who enlightened me with immense academic knowledge and gave me valuable advice whenever I needed it the most.

ACKNOWLEDGMENTS

In a very special way, I would like to thank the following people who have helped me undertake this research:

- My supervisors, Dr. Subha Fernando and Dr. Shantha Fernando, for their enthusiasm for this research, support, encouragement and patience.
- My progress review examiners, Dr. Chamath Keppitiyagama and Dr. Kasun De Zoysa, for their support, encouragement and feedback.
- My progress review evaluation panel, Dr. Thushari Silva, Dr. Thanuja Sandanayake, Dr. Thilini Piyatilake, Dr. C.R.J. Amalraj and Dr. Sagara Sumathipala, for their support, encouragement and feedback.
- Research coordinators at the Department of Computational Mathematics, Dr. Thushari Silva and Dr. Thilini Piyatilake, for their support when arranging my progress evaluations and other PhD program-related tasks.
- Masters program coordinator, Dr. Sagara Sumathipala, for the support received for my course-work completion.
- Dean, Assistant Registrar, and all other officials at the Faculty of Graduate Studies, the University of Moratuwa for all the support I received during my PhD program.
- Director and other officials at the Center for Information Technology Services (CITeS), University of Moratuwa, for their support in my PhD results evaluation process.
- Vice-Chancellor of Uva Wellassa University, Prof. Jayantha Ratnasekera, for allowing me full-time leaves to complete this PhD program.

- Head of the Department of Computer Science and Informatics, Uva Wellassa University, Ms. Harshani Wickramarathna, for her support and encouragement.
- All the relevant officials at the University of Moratuwa and Uva Wellassa University, for their support.
- My respectful parents, my beloved wife and my loving children, for their constant support and encouragement.
- My brothers, respectful parents-in-law, brother-in-law and sisters-in-law, for their support and encouragement.
- All of my colleagues and friends who helped me throughout this PhD program.

I am incredibly grateful for the immeasurable help and support every one of you has contributed to my program. Words alone cannot express my gratitude to you all. Thank you.

ABSTRACT

Phishing presents an ongoing and dynamic threat to Internet users, targeting personal and confidential information. Existing anti-phishing solutions encounter challenges in keeping up with the ever-changing nature of these attacks, leading to performance degradation over time. This study aims to develop an autonomous anti-phishing solution that effectively counters evolving phishing threats through continuous knowledge updates. To address the challenge of detecting the latest phishing attacks, SmartiPhish, an autonomous anti-phishing solution with continuous learning support, is proposed. Utilizing a quantitative research approach, data is collected from trusted third parties at multiple time points to create a valid dataset. The primary outcome is a reinforcement learning solution that leverages a novel deep learning model alongside Alexa rank and community decisions. The innovative use of Graph Neural Networks in the anti-phishing domain, combined with Long-term Recurrent Convolutional Networks, enables SmartiPhish to estimate a website's phishing probability using URL and HTML content features. Additionally, the study addresses a crucial research gap by developing a reliable method named PhishRepo for collecting and precisely labelling the latest phishing data. SmartiPhish exhibits positive results, achieving a detection accuracy of 96.40%, an f1-score of 96.42%, and an exceptionally low False Negative Rate (FNR) of 0.029. In real-world web environments, the solution outperforms similar solutions and demonstrates enhanced effectiveness against zero-day phishing attacks. Notably, the integration of continuous learning support facilitates a significant 6% improvement in detection accuracy after six weeks. SmartiPhish's adaptive approach integrates a systematic knowledge acquisition process, enabling dynamic updates of phishing detection features to counter the ever-evolving landscape of phishing attacks. The findings highlight its potential in strengthening cybersecurity measures and provide practical insights for dealing with phishing threats in today's digital world. Continuously updating its knowledge base, SmartiPhish stands as a strong defence, promising improved protection for Internet users.

Keywords: Cyberattack, Deep learning, Graph neural networks, Internet security, Reinforcement learning

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACMR	Absolute Cumulative Majority Relabelling
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
AIWL	Automated Individual White-List
ANN	Artificial Neural Networks
API	Application Programming Interface
ARMA	Auto-Regressive Moving Average
CA	Certificate Authority
CML	Conventional Machine Learning
CNN	Convolutional Neural Network
ComD	Community Decision
CSS	Cascading Style Sheets
Deque	Double-Ended Queue
DL	Deep Learning
DLM	Deep Learning Model
DNS	Domain Name System
DoS	Denial-of-Service
DQN	Deep Q-learning Network
FIFO	First In, First Out
FNR	False Negative Rate
GAN	Generative Adversarial Networks
GBDT	Gradient Boosting Decision Tree
GCN	Graph Convolutional Networks
GCS	Graph Convolutional Skip
GN	Generator Network
GNN	Graph Neural Network
GSB	Google Safe Browsing
HTML	HyperText Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IPR	Initial Phishing Records
KAP	Knowledge Acquisition Process
k-NN	k-Nearest Neighbour
LightGBM	Light Gradient Boosting Machine
LRCN	Long-term Recurrent Convolutional Network
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
MPN	Multilayer Perceptron Network

NLP	Natural Language Processing
NN	Neural Networks
PhaaS	Phishing-as-a-Service
pHash	Perceptual Hashing
PNN	Probabilistic Neural Network
ReLU	Rectified Linear Unit
RESTful	Representational State Transfer
RF	Random Forest
RL	Reinforcement Learning
RLM	Reinforcement Learning Model
RoF	Rotation Forest
SaaS	Software as a Service
SEO	Search Engine Optimisation
SGD	Stochastic Gradient Descent
SMO	Sequential Minimal Optimisation
SMS	Short Message Services
SSL	Secure Sockets Layer
SVM	Support Vector Machine
Tanh	Hyperbolic Tangent
TAP	Target Attack Prevention
TF-IDF	Term Frequency-Inverse Document Frequency
TLD	Top-Level Domain
URL	Uniform Resource Locator
XGBoost	eXtreme Gradient Boosting
XSS	Cross-Site Scripting

1 INTRODUCTION

1.1 Prolegomena

With the significant growth of Internet usage in the 21st century, people tend to rely more on Internet-based services. They make relationships through social networks, make payments through online banks, buy their needs and wants through online shops and share trade secrets through cloud techniques. However, the Internet is a double-edged sword where lawbreakers could also optimise its benefits to perpetrate their illegal activities online effectively. One of such illicit activities is phishing, a fraudulent attempt, usually made through email, to steal Internet users' digital assets (i.e., personal and confidential information) (Alkhalil et al., 2021). Phishing is a cyber attack that has relied on the Internet since 1996 (Alkhalil et al., 2021) and has gained a top rank in the cyber threat landscape (ENISA, 2021). Phishing is known as 'identity theft' since it impersonates one's identity in cyberspace while illegally using digital assets (Yu et al., 2008; Alkhalil et al., 2021). Therefore, many solutions (Khonji et al., 2013; Jain & Gupta, 2017; Sahoo et al., 2017; Dou et al., 2017; Opara, Chen, & Wei, 2020; Feng et al., 2020) were adopted over the years to minimise the impact of phishing threat on Internet users and protect the Internet credibility.

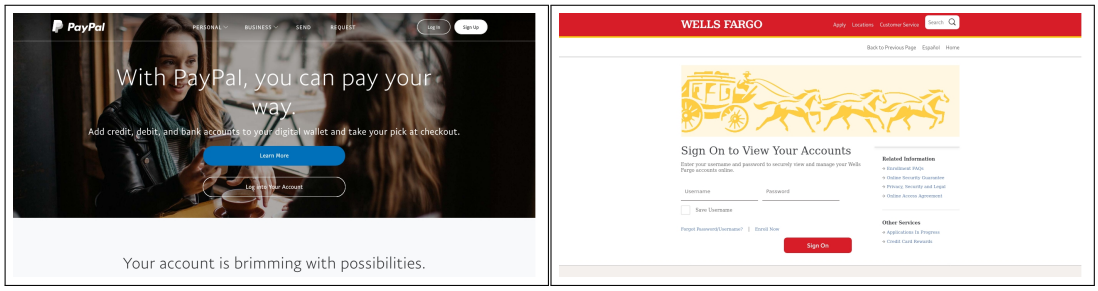
However, there is a dearth of research regarding how the changing nature of phishing should be adapted to phishing detection solutions when phishing detection features and spoofing methods are rapidly and continuously evolving - for example, the introduction of new Artificial Intelligence(AI)-based phishing kits. Therefore, this research aims to develop an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process to reduce the impact of phishing attacks on Internet users. As a result, a solution named SmartiPhish was proposed in the latter part of this study to achieve the study's aim.

This SmartiPhish is a novel phishing detection approach that recorded a 96.40% detection accuracy with a 0.029 FNR. It is more effective against zero-day phishing attacks and recorded 16.65 seconds mean detection time during actual execution. SmartiPhish used the systematic knowledge acquisition process effectively to increase its performance by 6% within six weeks.

At the beginning of this dissertation, this chapter introduces the study by first discussing the background and context, followed by the research problem, aims, objectives and questions, study's scope, significance and limitations.

1.2 Background to the study

Phishing is a social engineering crime that leads a victim to a fake website to steal personal or confidential information (H. Huang et al., 2009; Alkhalil et al., 2021; APWG, 2021b). In simple terms, a phishing attack could be a website login page (i.e., Gmail or PayPal Login) that is designed identical to the original website login page, directed through an email link that appears to be trusted, where a victim is likely to enter his valuables such as username, and password. Figure 1.1 shows two recent phishing attacks that target famous businesses, PayPal and Wells Fargo.



(a) Fake PayPal web page (b) Fake Wells Fargo web page

Figure 1.1: Examples of recent phishing attacks
The first phishing attack (a) was captured on October 03, 2021, when accessing the `https://secure03login.com/Service/` web address. It tried to impersonate PayPal's official sign-in web page. The next, (b) impersonated the popular Wells Fargo website. It was captured on October 06, 2021, when accessing the `https://cbahospitalar.com.br/002WG/well-fargo-RD528-detail/` web address.

According to the literature, the first phishing attack was reported in 1996, when it stole the confidential information of America Online (AOL) account holders (Alkhalil

et al., 2021; Gupta et al., 2016; Khonji et al., 2013). Since then, it has led to billions of financial losses for Internet users, and it has become one of the most frequent types of fraud activity on today's Internet (Alkhalil et al., 2021; ENISA, 2020). Phishing attacks are primarily motivated by financial benefits (Yu et al., 2008). However, fame and notary are also considered interesting psychological motives behind phishing (Yu et al., 2008).

Generally, the phishing process starts with a spoofed email (H. Huang et al., 2009; Alkhalil et al., 2021), and the attacker tries different tactics to get the users into their fake website. Figure 1.2 shows the typical phishing process, and those steps are further described in the following:

1. The phisher finds a target and constructs a phishing website similar to the target.
2. The phisher finds a relevant audience and distributes the phishing Uniform Resource Locator (URL) through numerous spoofed emails.
3. The victim visits the available website URL in the email and enters his sensitive information like username, password, and credit card number.
4. The victim's collected information transmits to the phisher's end.
5. The phisher uses the victim's information on the target and gets some illegal financial benefits.

Although phishing attacks have a typical information flow, the phishers constantly absorb new technologies into their attacks. Therefore, the attacking methods or techniques are continuously changing (Alkhalil et al., 2021). However, these phishing techniques could be mainly categorised as deceptive phishing and technical subterfuge (Alkhalil et al., 2021). Deceptive phishing is a social engineering-based approach and mainly works in the human layer (Alkhalil et al., 2021). In contrast, technical subterfuge-type attacks are in the technical layer and are always exploited through technical vulnerabilities in victims' environments (Khonji et al., 2013; Alkhalil et al., 2021).

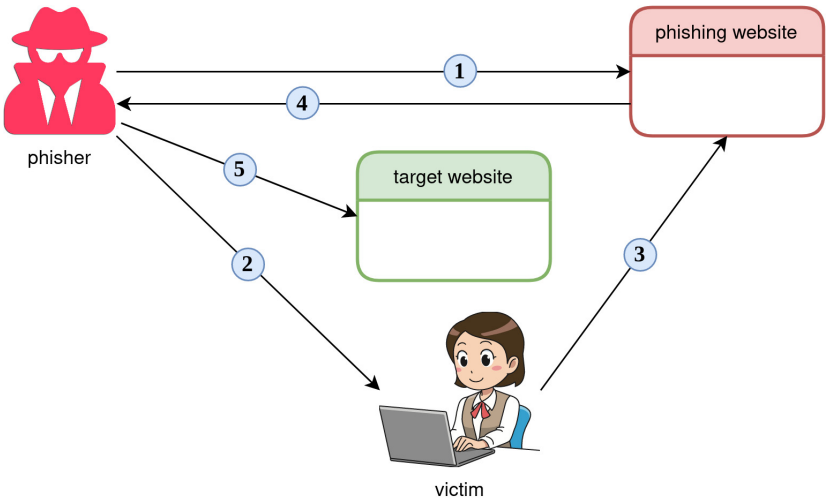


Figure 1.2: Information flow of the typical phishing attack

Phishers generally start by identifying a target website. They then send a link to this website via email. When the user clicks on the link, they are redirected to a fake or phishing website. The user unknowingly enters their personal or confidential details on the fake website, and the phishers exploit this information for malicious purposes.

Furthermore, deceptive phishing uses human psychological weaknesses, while technical subterfuge uses technological inadequacies. However, the more prominent one is deceptive phishing, and it has different attacking techniques such as spoofed website attacks, email-based attacks, phone-based attacks and social media-based attacks (Alkhalil et al., 2021). Similarly, technical subterfuge also has different techniques such as malware-based, Domain Name System (DNS)-based, content injection-based, man-in-the-middle attacks, search engine-based, and URL and HyperText Markup Language (HTML) obfuscation attacks (Alkhalil et al., 2021). These main techniques also have different forms in the current phishing nature. For example, the malware-based technique includes loggers (e.g., keyloggers), viruses, worms, spyware, adware, ransomware, rootkits, session hijackers, web trojans, and data theft (Alkhalil et al., 2021).

The number of phishing website attacks was at 150,000 limits per month in 2020 (APWG, 2021a, 2021b), whereas it was 1,609 in 2004 (Li et al., 2019). This shows that the numbers are rising day by day. Recent studies have identified that the present Coronavirus (COVID-19) pandemic worsened it by doubling the number of unique phishing website attacks in 2020 compared to previous years, and the trend is continu-

ing in 2021 as well (ENISA, 2020; APWG, 2021a, 2021b). This indicates that strong countermeasures against phishing are more essential today than in the past to decline the phishing threat to a controllable level.

However, during the past two and half decades, numerous efforts have been carried out by academia and industry to minimise this prevalent Internet threat through detectors (Khonji et al., 2013; Alkhalil et al., 2021). This results in various anti-phishing solutions in the current phishing context. These solutions could be categorised mainly under human education and software-based solutions (Khonji et al., 2013). Out of these, software-based solutions seem to be effective since human education is considered a costly and impractical approach in the rapidly changing nature of phishing (Khonji et al., 2013; Alkhalil et al., 2021).

Further, these software-based solutions are divided into four main categories based on the approach. Those are list-based, rule-based, visual similarity, and machine learning (Khonji et al., 2013). List-based solutions are more popular and are the techniques used by many modern browsers (e.g., Google Chrome and Firefox) today (Bell & Komisarczuk, 2020). Although list-based solutions are fast and straightforward, they would not be practical in today's context since maintaining lists are not simple in an environment that records more than 100,000 unique attacks per month.

However, the machine learning-based solutions have shown some success during the past few years in terms of accuracy and learning ability over the others (Dou et al., 2017; Bahnsen et al., 2017; Opara, Chen, & Wei, 2020; Feng et al., 2020). Therefore, many of the current phishing detection interests are toward machine learning, and representation techniques like deep learning are more famous in such cases (Yang et al., 2019; Jain & Gupta, 2016; Opara, Chen, & Wei, 2020; Feng et al., 2020). The main reason behind this trend is that deep learning could automatically extract essential features from raw data, removing the costly interaction of expert users in the knowledge-building and updating process (LeCun et al., 2015; Opara, Wei, & Chen, 2020).

Further, it is a better approach in the anti-phishing domain since phishing detection features are changing rapidly (e.g. Hypertext Transfer Protocol Secure (HTTPS)

features lose their priority since many of today's phishing attacks are using HTTPS) in phishing nature. Then, the abstract level feature extraction of deep learning techniques can effectively support having more prioritised features in the detection process. Although machine learning is a good technique in phishing detection, it needs more phishing examples in the learning process since data is the key for machine learning-based studies (Aassal et al., 2020). It is again considered a challenge in the machine learning-based anti-phishing domain since such data collection structures are not well established in the current anti-phishing context (Sahoo et al., 2017; Aassal et al., 2020). Therefore, the machine learning category has yet to be solved challenges when dealing with the rapidly changing nature of phishing attacks.

1.3 Research problem

Phishing attacks are among the most successful cybercrime attacks (Alkhalil et al., 2021). When controlling the impact of these attacks, phishing mitigation has been considered the most successful strategy against these attacks, which mainly depends on a successful phishing detection (Khonji et al., 2013; Alabdan, 2020; Alkhalil et al., 2021). Therefore, phishing detection is critically important to protect the digital assets of Internet users to have the credibility of information on the Internet (Alkhalil et al., 2021; Khonji et al., 2013). Numerous studies have been introducing different phishing detection approaches to mitigate phishing threats over the years (Teraguchi & Mitchell, 2004; Sheng et al., 2007; Prakash et al., 2010; Baslyman & Chiasson, 2016; Jain & Gupta, 2016; Bahnsen et al., 2017; Li et al., 2019; Opara, Chen, & Wei, 2020; Feng et al., 2020).

However, these studies primarily focused on developing anti-phishing solutions, and less attention was paid to incorporating the newer phishing detection features, which are essential against constantly changing phishing attacks (Sahoo et al., 2017; Aassal et al., 2020). As a result, the performance of these anti-phishing solutions is declining over time, and it has become a significant problem in the current anti-phishing domain (Aassal et al., 2020; Opara, Wei, & Chen, 2020; Opara, Chen, & Wei, 2020; Sánchez-Paniagua et al., 2020).

1.4 Research aim, objectives and questions

Given the lack of research regarding a systematic way of incorporating the latest phishing detection features into anti-phishing solutions, this research aims to develop an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process to reduce the impact of phishing attacks on the Internet users.

While this research aim covers the high-level context of the study, the following research objectives focus on the specific things the study plan to complete when achieving this aim.

- RO1 – To identify the phishing detection features and detection techniques utilised by the anti-phishing domain in effective phishing detection
- RO2 – To implement an effective anti-phishing solution by overcoming the identified phishing detection challenges
- RO3 – To incorporate an autonomous knowledge acquisition process to update the existing knowledge of the phishing detection features to minimise the performance loss over time

While the research objectives mentioned above describe the actions and the specific things the study plan to achieve when completing this research, the following research questions highlight the specific things this study should answer to achieve the study's aim.

- RQ1 – How can an effective anti-phishing solution be implemented while overcoming the identified challenges?
- RQ2 – How can existing knowledge of phishing detection features be automatically updated to minimise performance loss over time?

In the research questions mentioned above, the RQ1 supports achieving the RO2, and RQ2 supports the RO2's success. However, the RO1 mentioned above is primarily dependent on the literature. Therefore, this study achieves the RO1 via a comprehensive literature analysis.

1.5 Scope of the study

Phishing attacks are mainly categorised under deceptive phishing and technical subterfuge (Alkhalil et al., 2021). However, those two are different phishing levels, and the more prominent one is deceptive phishing (Alkhalil et al., 2021). Therefore, this study's scope is limited to deceptive phishing types. Under deceptive phishing also, there are different phishing techniques. These are spoofed website attacks, email-based attacks, phone-based attacks, and social media attacks (Alkhalil et al., 2021). However, this study scope is more toward spoofed website attacks since that is the dominant technique in the current deceptive phishing context. Therefore, the study's primary focus is on developing an autonomous anti-phishing solution to detect spoofed website attacks by considering spoofed website attacking characteristics, detection approaches, and challenges.

1.6 Significance of the study

This study will contribute to the anti-phishing domain by introducing a novel phishing detection method with an automated knowledge acquisition process to update phishing detection features over time. The proposed solution will help address the current shortage of research in this area and provide a real-world approach that can minimise the increasing phishing threat to have a more secure web in the future. Further, this study makes the following contributions to the research field after the successful problem-solving stage.

- An autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process.
- A novel anti-phishing solution to detect phishing attacks using URL and HTML content features.
- A novel approach to detect phishing attacks using HTML content analysis.
- A novel HTML page traverser which generates a Graph Neural Network (GNN) compatible graph from a given HTML page.

- A reinforcement learning-based collaborative phishing detection framework.
- An online phishing data repository uses to collect, validate, disseminate and archive real-time phishing data.
- A large-scale, multi-modal phishing data in raw format could be used for future research.
- A phishing-free web browser for safe browsing.

1.7 Limitations of the study

As with the many research studies, this study also includes some potential limitations when solving the identified problem in the phishing domain. The following describes those limitations on an abstract level.

- This study considered only the spoofed website attacks techniques when constructing the solution for the identified problem.
- The text/html content type HTML pages were only considered during the study.
- The significant phishing detection features used during the detection are not visible to the outside due to the used technique.
- Phishing detection features were updated only in three months, and optimal retraining time was not assessed due to infrastructure and resource limitations.
- The RL environment of the proposed solution used a delayed feedback mechanism due to the lack of resource limitations (i.e., human experts).
- This solution is built with the support of free Application Programming Interface (API) services, and a shortage of those will interrupt the proposed solution.
- This solution is vulnerable to dynamic phishing attacks (i.e., tabnapping), which could occur after loading the initial web page content.

- This solution could not consider the actual location information during the web browsing due its deployment architecture.
- This solution is only evaluated with a simulated environment due to the time constraints and risk factors (i.e., the sensitivity of phishing attacks) associated with the study.
- This solution is only tested for one type of adversarial attack called adversarial inputs in a controlled environment based on certain assumptions, and other types such as data poisoning and privacy attacks are not being tested.
- This solution needs a high-performance computer for successful execution since it uses high-end deep learning algorithms like Graph Convolutional Networks (GCN).

1.8 Structural outline of the dissertation

- **Chapter 1** introduces the background and context of the study, followed by the research problem, aims, objectives and questions, study's scope, significance and limitations.
- **Chapter 2** gives a broader view of phishing attacks by first discussing the definition of phishing and its history, followed by phishing motives, medium, process, types and techniques, mitigation, and the current state.
- **Chapter 3** gives a critical review of literature on phishing attack detection under several subheadings: data collection and labelling, feature selection and engineering, existing detection techniques, performance evaluation, and the present challenges in phishing detection. After reviewing the literature, the latter part of this chapter identifies the research problem.
- **Chapter 4** focuses on the research methodology and explains how the mentioned aim was achieved in a research context by discussing the research design, solution implementation, and methodological limitations.

- **Chapter 5** introduces phase one of the proposed implementation process, a novel phishing detection approach that uses URL and HTML features. It further discusses data collection and preprocessing, model training, and evaluation steps, followed by a result and discussion section that highlights the effectiveness of the proposed approach.

- **Chapter 6** discusses phase two of the implementation process, a novel phishing detection framework that has the flexibility to use a different set of phishing detection features. This chapter also consists of data collection, preprocessing, training, and evaluation steps followed by a result and discussion section.

- **Chapter 7** introduces PhishRepo solution, a core component of the knowledge acquisition process. It is a generalised solution to collect phishing data and is the leading labelling agent of the proposed solution. This chapter discusses its implementation under several sub-topics, followed by a diversity and effectiveness analysis to show the importance of the implemented solution.

- **Chapter 8** discusses the main steps of the proposed knowledge acquisition process, including how the learning process of phases one and two happen. Then, the final output of the study is introduced with a real-world application.

- **Chapter 9** evaluates the final output using different types of experiments. Then, the results achieved by these experiments are analysed to show the proposed solution's effectiveness against the identified research problem.

- **Chapter 10** evaluates how the defined aim was successfully achieved by attaining the study's objectives. Following that, the resolved research problem, the novelty, and contributions to the current anti-phishing domain are discussed with the study's limitations.

- **Chapter 11** concludes the dissertation by summarising the key research findings and how the knowledge developed from this study has been disseminated to the scientific community. Finally, this chapter suggests specific research directions that may be undertaken in the near future.

2 OVERVIEW OF PHISHING ATTACKS

2.1 Introduction

Chapter 1 provided the overall picture of the research presented in this dissertation. This chapter is to make the reader aware of the research domain. Therefore, this chapter gives a broader view of phishing attacks by first discussing the definition of phishing and history, followed by phishing motives, medium, process, types and techniques, mitigation, and the current state.

2.2 Phishing definition

A phishing attack is a social engineering attempt to steal Internet users' personal or confidential information by impersonating a trusted third party (Chiew, Yong, & Tan, 2018; Alkhalil et al., 2021; Gupta et al., 2016). However, the phishing definitions are not consistent in the literature due to the nature of these attacks (Dou et al., 2017; Alkhalil et al., 2021). Therefore, different studies define phishing based on the target (i.e., email filtering or website detection) and their detection approach (Dou et al., 2017). The following is a more generic definition of phishing found in Khonji et al. (2013), which considered phishing as a semantic attack.

"Phishing is a type of computer attack that communicates socially engineered messages to humans via electronic communication channels in order to persuade them to perform certain actions for the attacker's benefit" (Khonji et al., 2013).

The current study mainly focuses on deceptive websites, a fraudulent electronic communication channel that transfers phishing attacks via social engineering techniques. Therefore, the term phishing is defined in the study as a social engineering crime that leads a victim to a fake website to steal personal or confidential information (H. Huang et al., 2009; Alkhalil et al., 2021; APWG, 2021b).

2.3 History of phishing

Phishing was initially coined in 1996 with the America Online (AOL) accounts attack that stole confidential information of the AOL account holders by online scammers (Alkhalil et al., 2021; Gupta et al., 2016; Khonji et al., 2013). The term phishing originated from the well-known activity of catching fish called fishing (Khonji et al., 2013). In fishing, a fisherman uses bait to catch a fish. Similarly, phishing also uses a social engineering or technological bait to act (i.e., steal confidential information) on the Internet user (i.e., fish) to have the ultimate benefits (Chiew, Yong, & Tan, 2018; Khonji et al., 2013). However, the 'ph' in phishing that replaced the 'f' in fishing was extracted from the popular telephone hacking term called phone phreaking which was very common in the early 1970s (Gupta et al., 2016, 2017).

After the AOL incident in 1997, the media announced the evolution of the phishing attacks for the first time (Gupta et al., 2016). Initially, the attacks focused mainly on online banking and e-commerce services to earn financial gains (Khonji et al., 2013). However, over time, it has been spread into many domains (APWG, 2021a, 2021b), and the phishers also updated to have different scamming strategies to launch new attack types (Alabdan, 2020; Alkhalil et al., 2021). In 2001, the phishers started to use URLs to direct users to phishing sites, and in 2004 they passed another milestone by practising domain name system-based phishing called pharming (Gupta et al., 2016). Further, in 2005, the spear-phishing attacks that target specific individuals or organisations were first evolved (Gupta et al., 2016), and today it has become the dominant attacking technique due to the high success rate (Dou et al., 2017). Moreover, the phishers are becoming more intelligent and practising different attacking techniques through new technological support over the years, challenging new security countermeasures introduced in the past few years (Dou et al., 2017; Alabdan, 2020; H. Huang et al., 2009). Therefore, the phishing threat has a top rank in the cyber threat landscape (ENISA, 2020, 2021), and the number of attacks is rising day by day (APWG, 2021a, 2021b).

2.4 Phishing motives

The primary motivation behind phishing is financial gains. However, these financial gains can be direct or indirect because some phishers may use the stolen information for their benefit, and some may sell it to a third party (Alkhalil et al., 2021; Alabdan, 2020). However, phishers have different intentions that motivate them to exploit a phishing attack in a specific domain. The following are some of those intentions mentioned in the study of Yu et al. (2008) and Gupta et al. (2016).

1. *Steal money from bank accounts:* Spoofed emails or some other techniques are used by phishers to collect online bank account credentials and credit card details. That information is then used to get money out of the victim's account directly or indirectly (i.e., purchase goods or services).

2. *Access online services:* The phishers will launch an attack to steal login details of popular online services such as Google Accounts, Microsoft Office, Amazon or eBay. That information can be used to obtain goods or services, collect personal information or launch more phishing attacks assuming the same login credentials are used in different services.

3. *Hide the original identity:* This intention is more popular among the community who want to hide their original identity because of illegal activities such as criminal activities or buying/selling illegal goods or services carried out in cyberspace. However, there is a massive demand in the online community for stolen identities, and it motivates phishers to carry out more phishing in cyberspace.

4. *Access to confidential documents:* Phishers can launch attacks to access trade secrets of different organisations and sell that information to interested parties, like competitors.

5. *Collect personal information:* Phishers conduct attacks to collect Internet users' personal information like addresses, telephone numbers, product loyalties, or browsing patterns to sell or use directly for their benefit. The information col-

lected through such attacks can be used to shift victims from one product to another, which causes monetary loss for some businesses.

6. *Exploit security holes:* Phishers are interested in finding any security breaches in different systems or employees of an organisation. Then, the phisher can use that information to arrange further attacks like installing malware or selling those back to the phishing community for others' benefit.

7. *Fame and notoriety:* The phishers use different strategies to access personal or confidential information without expecting financial gains. This intention mainly targets the popularity among peers or the online community.

The one to six intentions mentioned above is always bound with financial gains. However, the seventh intention is a gripping psychological aspect of phishing that targets the popularity among the community.

2.5 Phishing medium

In the current phishing context, the primary medium of transferring a phishing attack to the victim's side is the Internet (Chiew, Yong, & Tan, 2018; Alkhalil et al., 2021). Besides that, voice and Short Message Services (SMS) are also popular among attackers (Chiew, Yong, & Tan, 2018). These mediums have different vectors to transfer the attack to the victim side (Alabdan, 2020). The most popular phishing medium, the Internet, has six vectors: Email, eFax, Instant messaging, Social networks, Websites and Wifi (Chiew, Yong, & Tan, 2018; Alabdan, 2020). The other two mediums, voice and SMS, have Vishing and Smishing as vectors, respectively (Chiew, Yong, & Tan, 2018; Alabdan, 2020). However, both Vishing and Smishing are migrated from email vectors, and the phisher uses a call in Vishing and a text message in Smishing (Chiew, Yong, & Tan, 2018).

2.6 Phishing process

In most cases, phishing attacks are started from emails sent to random recipients or a specific target group or individuals (Alkhalil et al., 2021; H. Huang et al., 2009).

Subsequently, the phishing process involves different steps, and understanding those steps is essential to developing successful phishing detection. Alkhalil et al. (2021) discussed the phishing process under four main phases: planning, preparation, attack conducting, and valuables acquisition.

- *Planning:* This is the first stage of a phishing attack, and it includes three main steps. First, the phisher selects a target. In most cases, it could be a financial service like an online banking interface, a retail sector business like eBay or Amazon, or internet services provider such as Yahoo or MSN. Then the second step is the construction of the fake website. The phishers build the fake website on their own or get an already designed one, or use a website from another source. After the construction step ends, the phishers collect the victim's information (i.e., name and email address), mainly via social networks.

- *Attack Preparation:* The phishers have used vulnerability scanning and medium selection as the primary foundation for the attack in the preparation stage. The medium is vital to a phisher when reaching the victims, and mainly there are three types of mediums: Internet, phone call or text message described in Section 2.5.

- *Attack Conducting:* In the third stage, the phishers select an appropriate attacking technique (*see* Section 2.7) to deliver the attack to the victim's end via the selected medium. The victim interacts with the attack in this stage, and the attacker will compromise the victim's information during this time.

- *Valuables Acquisition:* After the successful third stage, the phishers collect information and valuables from their victims manually or automatically. However, that information collection can happen during or after the interaction and is finally used for the phisher's benefit.

Although Alkhalil et al. introduced the four phases that discuss phishing attacks in a more detailed way to understand the complete phishing process, in the past literature, R. M. Mohammad et al. (2015) and Gupta et al. (2016) have presented the

phishing process differently. R. M. Mohammad et al. (2015) mentioned three main phases: planning, collection and fraud, and Gupta et al. (2016) mentioned five phases: planning and setup, phishing, infiltration, data collection, and exfiltration, which both are conceptually similar. Generally, in phishing attacks, requesting data from the victim, acquiring those data, and using the collected data for illegal purposes are common and come under any phishing process (Alkhalil et al., 2021).

2.7 Phishing types and techniques

Phishers are always keen to take advantage of human users (Dou et al., 2017; Gupta et al., 2016). They mainly use human psychology and technical vulnerabilities to execute their phishing attacks (Alkhalil et al., 2021). Therefore, the existing phishing attacks can be categorised into deceptive and technical subterfuge (Alkhalil et al., 2021).

2.7.1 Deceptive phishing

Deceptive phishing is the most common type of phishing practice that impersonate a trusted third party to deceive a victim when performing phishing activities (Alkhalil et al., 2021). In deceptive phishing, the attacker uses social engineering techniques to perform the attack by making a real scenario (i.e., account update) or using a technical aspect like images or logos (Alkhalil et al., 2021). There are different techniques of deceptive phishing attacks listed in the literature, and out of those, spoofed website attacks are more popular (Alkhalil et al., 2021; H. Huang et al., 2009). In spoofed website attacks, a phishing email or a spoofed advertisement is mainly used in the initial step, and the victims are then moved to the spoofed website through different tactics (i.e., clicking a link) (Alkhalil et al., 2021; H. Huang et al., 2009). Although spoofed website attacks are famous, some other deceptive phishing practices are also common in the phishing domain (Alkhalil et al., 2021). The following is a list of those practices based on the survey done by Alkhalil et al. (2021).

- *Phishing Email:* Email is the most common type of phishing vector in the phishing domain. Phishers utilise it to send a link leading a victim to a spoofed website. Similarly, a phishing email is also used to send malware to a potential

victim. These emails mostly claim it is from a trusted person or organisation and target a specific group or random community. Spear-phishing, which uses a target group, and whaling, which uses top-rank people, are the most common email categories other than generic phishing emails. Interestingly, phishers use legitimate or previously sent emails from trusted authorities and resend them to the same recipients or different communities with some spoofed content. It is mentioned as clone phishing, and it is again a phishing practice that comes under the email phishing technique.

- *Phone Phishing (Vishing and SMishing):* It is associated with phone calls or text messages. The phisher sends a text message with a link to the victim's phone or calls by impersonating a trusted agent to reveal the victim's personal or confidential information for their benefit.

- *Social Media Attack (Soshing):* A new deceptive practice coined with social media networks like Facebook or Twitter. The phisher uses different strategies like scams, malware distribution, account hijacking and impersonation attacks to access the victim's social network account for malicious purposes.

2.7.2 Technical subterfuge

Unlike the deceptive phishing that relies on the human layer, technical subterfuge is associated with technical vulnerabilities that exist on the technical layer. The phishers are keen to take advantage of technical vulnerabilities in the victim's environment to have more reasoning attacks (Khonji et al., 2013). However, it is not limited to the victim's environment, and it can be a vulnerability of the infrastructure the victim accessed (i.e., pharming) (Alkhalil et al., 2021). There are different phishing practices under technical subterfuge, and some of them are listed below based on the recent survey done by Alkhalil et al. (2021).

- *Malware-based Phishing:* In this technique, malicious software is downloaded to the victim's environment and produces some illegal actions. These actions intend to collect personal or confidential information of the victim through dif-

ferent strategies like logging keyboard inputs, taking screenshots, or accessing clipboard items. However, based on malware's spreading and collecting techniques, malware-based phishing has different forms: loggers, viruses, worms, spyware, adware, ransomware, rootkits, session hijackers, web trojans, and data theft.

- *DNS-based Phishing:* Pharming is the short form of this attacking technique, and it is a more dangerous attacking strategy since a legitimate link can lead a user to phishing content if the relevant DNS is hijacked. The main goal of this attack is to fill the DNS with erroneous data (i.e., add phishing page IP address over genuine IP). Then unknowingly, a user may land on phishing content and disclose his privacy.

- *Content Injection Phishing:* This technique inserts false content into a legitimate site to misdirect a user to a phishing site. SQL injection attacks at the database level, Cross-Site Scripting (XSS) attacks at the site level and compromising a web server through infrastructure vulnerabilities are some standard practices that come under this technique.

- *Man-In-The-Middle Phishing:* As the name implies, an attacker enters a communication channel and records the communications between the victim and the server. Then, that information may be used later for the attacker's benefit.

- *Search Engine Phishing:* The attacker uses Search Engine Optimisation (SEO) techniques to get a good index in the search engine for the implemented phishing site. Then, it appears to users when searching for products or services.

- *URL and HTML Obfuscation Attacks:* Many phishing attacks use link clicking methods to redirect users to a phishing site. In such scenarios, obfuscation makes a URL look legitimate (i.e., hostname obfuscation).

2.8 Mitigation of phishing attacks

Mitigation of phishing attacks could be done from different perspectives. Mainly, there are three mitigation techniques: correction, prevention and offensive defence, used by the anti-phishing domain to make phishing campaigns ineffective after a successful detection (Khonji et al., 2013). The following explain those in detail based on Khonji et al. (2013) study findings on phishing mitigation techniques.

- *Correction Technique:* The main intention of this technique is to take down phishing campaigns through reporting. After the phishing attack is detected, the information about the phishing attack could be submitted to the service providers such as hosting services, email services and social networks. Then, these service providers could act based on the reported case and make some actions such as suspension of the service or removing the malicious contents to bring the phishing campaigns down. Figure 2.1 shows how a service provider acts on reported content.

- *Prevention Technique:* This technique depends on laws. Law Enforcement Agencies introduce specific laws, and after a phishing attack is successfully detected, they will act on this through lawsuits. This process takes a long time; however, it will demotivate phishing campaigns because of the penalties caused by the lawsuits.

- *Offensive defence Technique:* This exciting technique intends to distract phishing campaigns. The primary strategy used in this technique is flooding the phishing site with fake credentials. After the phishing website is successfully detected, some software such as BogusBiter (Yue & Wang, 2008) and Humboldt (Knickerbocker et al., 2009) can produce fake credentials, access this website and submit fake login details. Then, the attacker could not find a correct login, and the campaigns became ineffective. However, the success of this technique is not adequately evaluated and, therefore, can be questioned.

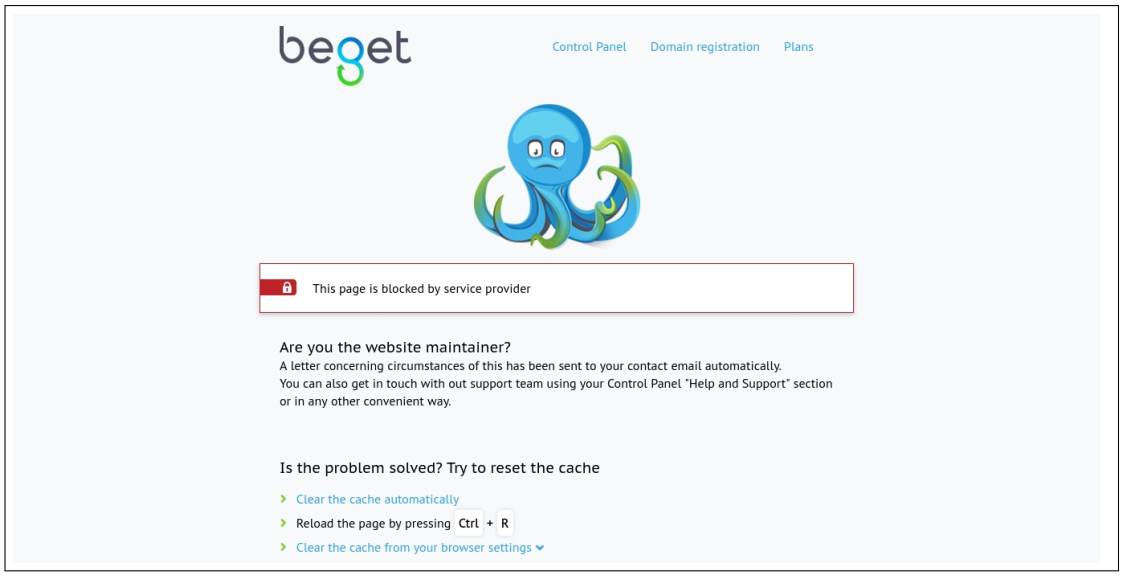


Figure 2.1: An example of how the correction technique works in phishing mitigation
This screenshot was captured on October 04, 2021, when accessing the `http://veolon.com` website. Since a trusted third party reported the relevant web page as a phishing attempt, the service provider could block the web page access to avoid future phishing attempts of the relevant phishing campaign.

2.9 Current state of phishing

The number of phishing attacks is gradually increasing over time. In 2004, the APWG reported 1,609 phishing attacks per month (Li et al., 2019), and it is now more than 150,000 attacks per month (APWG, 2021b). The Coronavirus pandemic (COVID-19) could be the key reason for such a massive number because it opened up many opportunities for phishers to further exploit their attacks (Alkhalil et al., 2021; ENISA, 2020). According to ENISA (2020), the number of phishing scams increased by 667% in only one month during the COVID-19, showing that phishers used the COVID-19 to lure the victims for their benefit. Further, APWG (2021b) mentioned that the highest number of unique phishing attacks ever reported to them increased by three times during last year, and the number of phishing attacks doubled in 2020 compared to the previous year. Figure 2.2 shows how the phishing trend has risen over the previous three years, and it has been demonstrated that the year 2021 is becoming worst than ever. Unsurprisingly, the situation is not different in the Sri Lankan context from the world. SLCERT (2020) has shown that phishing attacks have increased during the last few years. Further, Netcraft (2021) also listed Sri Lanka among the top fifty countries

based on the reported cyber incidents during the latest survey.

Moreover, the use of HTTPS in the phishing domain has also increased during the last few years. In 2017, it was around 30%, and statistics in 2021 show that it has reached more than 80% (APWG, 2021a). Therefore, HTTPS is no longer a helpful feature in phishing detection like previous anti-phishing solutions (R. M. Mohammad et al., 2013; El-Alfy, 2017; Jain & Gupta, 2018b). There are several new target industry sectors that phishers have been interested in during the last few years. One such sector is the cryptocurrency sector, and nearly 7.5% of all attacks reported during the second quarter of 2021 were against that (APWG, 2021a). Further, the phishers' interest in primary industries such as Software as a Service (SaaS) and webmail, financial institutions, eCommerce and retail, social media, payment, and telecommunication remains slightly up and down during the last five years (APWG, 2021a).

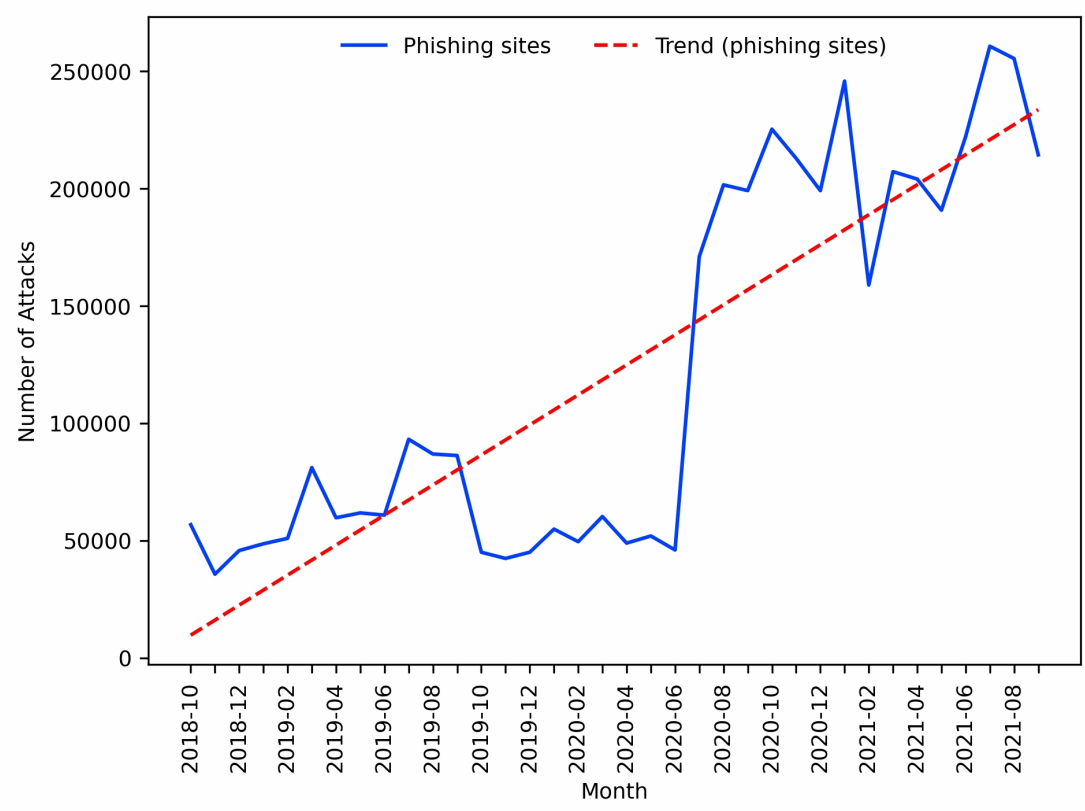


Figure 2.2: Number of phishing attacks reported during the last three years
Source: APWG reports[1] from the 4th quarter of 2018 to the 3rd quarter of 2021

[1] https://apwg.org/trendsreports/

Phishing is an acute problem since it is primarily associated with money. The literature has shown that billions of losses happened due to phishing attacks, and more recently, the Toyota subsidiary lost $37 million because of a phishing attack (ENISA, 2020). Moreover, the spear-phishing was the mainly used attacking vector in 2019, and brands like Office 365, OneDrive, Adobe, Netflix, PayPal, Apple, and Google Drive were mainly targeted by phishers (ENISA, 2020).

Interestingly, phishing has also become a service at present, and Phishing-as-a-Service (PhaaS) has been popular among phishers for the last few years (ENISA, 2020, 2021). The low cost of these solutions and some vital features like HTML character encoding and content-encryption motivates phishers to use them (ENISA, 2020, 2021). Therefore, less technically skilled people can also carry out phishing attacks comfortably at present (ENISA, 2020, 2021). However, this type of improvement in the phishing area challenges anti-phishers to have strong countermeasures against sophisticated phishing attacks in the future. Therefore, it is essential to have more research on the anti-phishing sector in the current context to minimise the impact and rising phishing trend to have a phishing-free Internet for all.

2.10 Summary

Phishing is a social engineering crime in which a victim is lured to a fake website to get personal or confidential information. If a user eats the bait, the attacker gains access to the victim's digital assets, which they can utilise. The number of attacks has increased dramatically in the last two years, while phishers' favour focuses more on deceptive phishing in today's environment. Phishing attack mitigation always relies on successful phishing detection. Therefore, the next chapter examines how phishing detection has been done in the past and what obstacles prior research has uncovered when implementing adequate phishing detectors.

3 RELATED LITERATURE

3.1 Introduction

Chapter 2 provided a broader view of phishing attacks by discussing them from different perspectives and highlighted the importance of phishing detection in successful phishing mitigation. This chapter provides a comprehensive review of the literature on phishing attack detection under several topics: data collection and labelling, feature selection and engineering, existing detection techniques, performance evaluation, and current phishing detection challenges. After completing the literature review, the latter part of this chapter identifies the research problem.

3.2 Phishing detection

The goal of phishing detection is to categorise a phishing attack. As depicted in Figure 3.1, this goal is the starting point of the phishing mitigation process which plays a critical role. Phishing detection always depends on learning (Khonji et al., 2013). A human or software uses past phishing and legitimate examples during the learning process (Khonji et al., 2013; Alkhalil et al., 2021). Therefore, correctly labelled examples are essential when implementing a new phishing detector.

Furthermore, data collecting and labelling, feature selection and engineering, detection techniques, and performance evaluation are critical aspects when developing successful phishing detectors. The following sections explain each of these aspects in greater detail.

3.3 Data collection and labelling

Phishing attacks are constantly changing due to technical advancements, effective security controllers, and public awareness (Alkhalil et al., 2021). Most phishing attacks

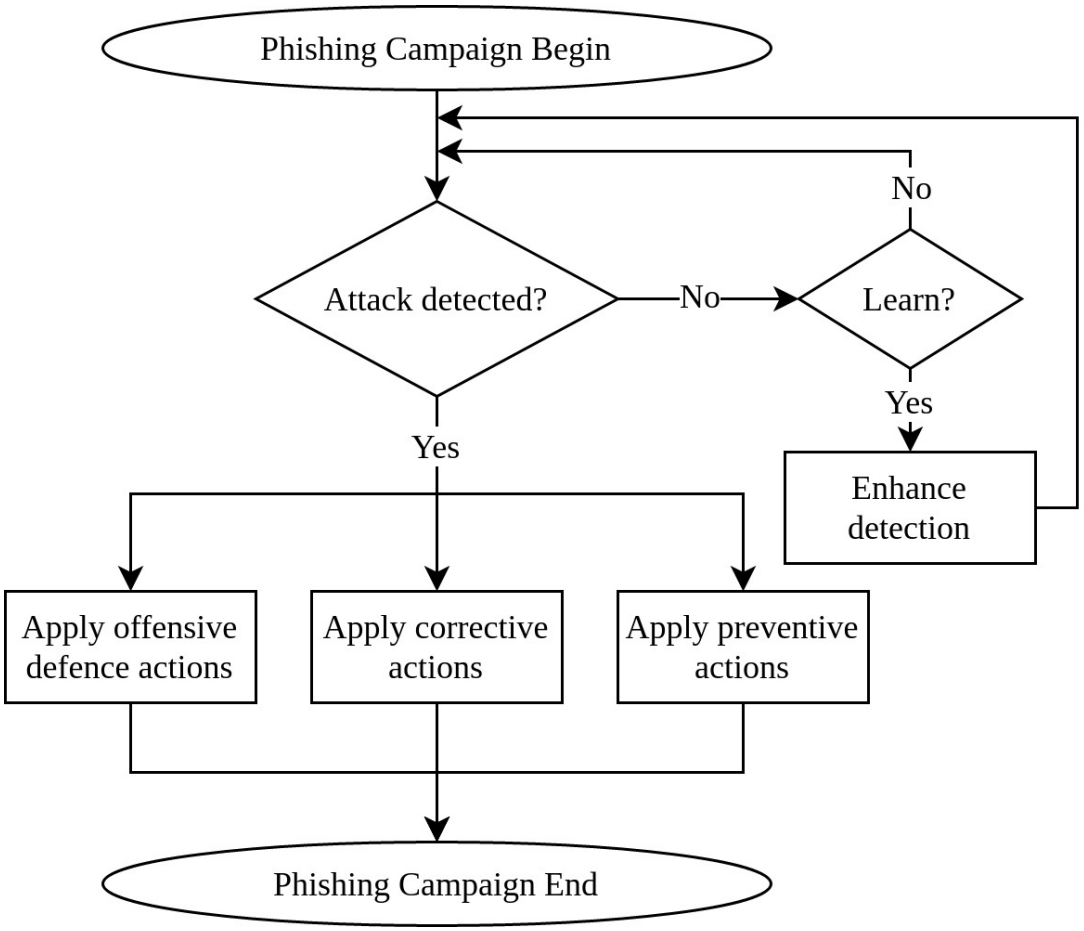


Figure 3.1: Phishing attack mitigation in terms of phishing detection
Source: Khonji, Iraqi, and Jones (2013, p. 2094)

were untargeted in the past, and phishers used a bulk attack distribution technique to find a victim (Alkhalil et al., 2021). However, that technique did not seem practical, as most untargeted attacks failed with a success rate of less than 5% (Dou et al., 2017). As a result, phishers focus on targeted attacks like spear-phishing, which has a nearly 19% success rate (Dou et al., 2017). When the phishing threat was evolving and threatening in such ways, diverse groups such as researchers, business owners, and external authorities like law enforcement agencies were more interested in these attacks (Tally, 2009). As a response, organisations such as the APWG, Phishing Incident Response Team, Phishing Report Network, and Digital PhishNet started to collect phishing-related data with different goals, resulting in different levels of data collection (Tally et al., 2006).

As a result, the Phisherman project was introduced to change the present way of collecting phishing data (Tally, 2009; Tally et al., 2006). It was a web-based system

that collects, validates, disseminates, and archives phishing data. Phisherman collects inputs from individuals and trusted third-party organisations. These inputs mainly depend on emails, and it uses several workflows to manage them due to the multi-format emails it receives from various sources. Phisherman uses a two-step verification process. The first verification is limited to a subset of submissions, while the second employs a set of heuristic rules and is available to all submissions. Phisherman allows users to download phishing URLs or a full incident report in XML format using subscriptions and by making queries during their data distribution process.

Similarly, phishing verification systems such as PhishTank[2] and OpenPhish[3] were established in the last two decades to meet phishing data needs. PhishTank is a phishing verification system that relies on public submissions and votes (Bell & Komisarczuk, 2020). It was first released in 2006, and many anti-phishers consider it one of their favourites (Wang et al., 2019; Yang et al., 2019; Butnaru et al., 2021). Further, the PhishTank collects data via public submissions, and it requires URLs and additional optional data connected with the attack, such as screenshots and WHOIS information. Subsequently, these entries are displayed to the public for voting, and PhishTank classifies them as phishing based on the community feedback. Even though PhishTank systematically collects phishing data, many researchers merely use it to obtain URL information, and other information such as screenshots or WHOIS information was not used in the literature (Wang et al., 2019; Butnaru et al., 2021; W. Chen et al., 2018).

Unlike PhishTank, OpenPhish is not free. It is based on a self-learning phishing detection algorithm developed by several researchers (Bell & Komisarczuk, 2020). It is utilised to collect phishing data in terms of URL, target brand, and screenshots and is also a popular verification system among the anti-phishing community (Zeng et al., 2020). OpenPhish services are not free, and full access to the solution requires a subscription.

As a separate phishing data collection approach, individual attempts to collect phishing data can be found in the literature. The UCI Phishing dataset is the most popular individual attempt, with a maximum of 11,055 data (Zeng et al., 2020). The UCI

[2] `https://phishtank.org/`
[3] `https://openphish.com/`

dataset is an older dataset that only provides a limited number of attributes. Further, the UCI dataset's features were preprocessed using many heuristics (R. M. Mohammad et al., 2012), limiting its use in various studies. In another study, Aassal et al. (2020) also introduced a comprehensive phishing dataset with the help of PhishTank, OpenPhish, and APWG. It resulted from a systematic data collection approach that was effectively used in the PhishBench benchmark framework.

After analysing the existing phishing data collection approaches, the study found Phisherman as the most acceptable approach, as it allows real-time phishing data collection and appears reliable in large-scale data collecting. Unfortunately, it is not available to the general public (Beck & Zhan, 2010). Then, the alternatives like PhishTank and OpenPhish collect many of the most recent phishing data, but their goals are different, such as keeping blacklists and identifying target brands. As a result, those solutions mainly focus on URLs, which are only one source of information in anti-phishing studies. Furthermore, the individual datasets mentioned earlier also have constraints, such as the number of features, size and the collection date. Because of these reasons, a reliable technique for collecting the most recent multi-modal phishing data is a timely need in the current anti-phishing domain.

Like data collection, data labelling is also essential in phishing detection. As shown in Table 3.3, many of the existing phishing detection solutions depend on labelled data. The expert labelling and the crowdsourcing approaches like PhishTank's voting method are the popular labelling approaches used in the literature. However, expert-based labelling is often costly and time-consuming in large-scale data collection (Chang et al., 2017). Crowdsourcing is the most suitable in such scenarios since it has advantages like low cost, fast labelling and diverse opinions than the expert approach (Drutsa et al., 2019). Even though crowdsourcing has many advantages, the fundamental disadvantage of it is the difficulty in acquiring quality labelled data (Chang et al., 2017; Drutsa et al., 2019). This is due to many reasons. Mainly, the poor commitment of crowd workers, uncertainty in a task, prior knowledge of the given task, and novice workers affect the quality of the crowdsourcing labelling process (Chang et al., 2017).

3.4 Feature selection and engineering

Feature selection is an essential step in phishing detection since it is highly bound to the detection accuracy of a solution (Dou et al., 2017; Aassal et al., 2020). Past studies have generated different features when having differentiated solutions against phishing attempts. Aassal et al. (2020) presented a more complex categorisation of phishing detection features using more than 250 phishing detection studies. First, Aassal et al. divided phishing detection features mainly into URL-based and website-based. Then, those two were further divided into three subclasses: lexical, network, and script. Finally, these features were analysed depending on their format and classified as syntactic, semantic, and pragmatic.

In Aassal et al. taxonomy, syntactic features are based on syntaxes like the port number and Term Frequency-Inverse Document Frequency (TF-IDF), and semantic features are based on meaning and the interpretation of the content like the presence of the target brand in URL and webpage. However, the features that do not belong to any of these two were categorised under pragmatic, and it has features like backlisted words in a URL, WHOIS information, and script loading time. In a separate study, Dou et al. (2017) categorised phishing detection features based on four characteristics primarily determined by the URL and website. It covers URL-based lexical features like URL length and the presence of HTTPS protocol, and URL-based host features such as WHOIS information. Under the website category, the content features like page rank, hyperlinks and forms, and visual similarity-based features like images and colours are listed.

In real-time phishing detection, the features that depend on third-party services such as page ranking, Alexa ranking, and age of the domain should be carefully used since those may result in increasing detection time, high development cost, and service limitations (Jain & Gupta, 2018a; Li et al., 2019; Sahingoz et al., 2019). As a result, Li et al. (2019) only used URL and HTML-based features in their solution, and Jain and Gupta (2018a) used a set of new features based on a web page's hyperlink information. However, Yang et al. (2019) stated that using multi-modal features (i.e., URL-based, content-based, third-party service features like Alexa rank) is good in phishing detec-

tion since those features may represent many of the phishing attack characteristics. Although this is a reasonable point, some key features may result in dissatisfied users due to the time required to construct and analyse them (Li et al., 2019; Sahingoz et al., 2019). Therefore, if a feature takes longer to extract, no matter how useful it is, it is not an ideal feature to utilise in real-time phishing detection (Dou et al., 2017).

In contrast to Yang et al. (2019), several studies in the explored literature rely only on URL-based features (Bahnsen et al., 2017; W. Chen et al., 2018; Wang et al., 2019; Sameen et al., 2020). Despite their high detection accuracy, as shown in Table 3.3, Sahoo et al. (2017) pointed out that URL simulation tools such as DeepPhish (Bahnsen et al., 2018) that produce malicious URLs and URL shorting services could create a long-term threat to these solutions. Further, AlEroud and Karabatis (2020) demonstrated how the Generative Adversarial Networks (GAN) could be used to evade the URL-based phishing detection solutions, raising the question of whether URL-based features alone are sufficient to detect phishing attempts in the current context. Moreover, as Sahoo et al. (2017) indicated, benign URLs may be compromised in the future to contain malicious content, which is another long-term difficulty for URL feature-based anti-phishing solutions. However, due to the lack of sufficient frameworks in the existing literature, the robustness of the features utilised in an anti-phishing solution cannot be evaluated quantitatively (Dou et al., 2017). Therefore, Table 3.1 has the most often utilised elements in past anti-phishing solutions, divided into URL-based, webpage content-based, and third-party service-based.

Similar to feature selection, feature engineering (a.k.a., feature extraction) is also essential when building a phishing detection solution because it is also linked with the detection accuracy of a solution (Dou et al., 2017). The current study identified two main feature engineering techniques practised by previous anti-phishing studies: manual and representation learning.

In manual techniques, human users actively participate in the extraction process (R. M. Mohammad et al., 2012). It can be a direct involvement or indirect involvement where the extraction can be done by a computer application built by a human expert (R. M. Mohammad et al., 2012). Also, in manual techniques, the feature extraction can

Table 3.1: Commonly used phishing detection features that come under URL, webpage content and third-party

Information source	Features	References
URL	IP address, Long URL, @ symbol, URL prefix, URL suffix, Sub-domains, HTTPS availability, Abnormal URL, Number of redirections, Domain name, Primary domain name, Path-domains, URL shortening, Redirect symbol, Domain length, Favicon, Nonstandard port, Number of dashes, URL length, Suspicious words, Position of Top-Level Domain (TLD), Embedded domains, Number of times HTTP appears, Domains count, Subdomain length, Length ratio (i.e., ratio between the length of the URL and the length of the path), Punctuation counts (i.e., !, #, $, %, and &), Number of TLDs, Port number, URL entropy, HTTPS on domain, Raw word count, Brand name on domain, Word length, Keyword count, Brand name count, Puny code, Consecutive character repeat, Known TLD, Random domain, Adjacent word count, Number of dots, Similar target brand, Embedded domain, URL with hexadecimal code, Numerical characters, Similarity index, Number of hyphens, Information entropy, Euclidean distance, Kullback-Leibler divergence, Edit distance outlier, Length of the longest work in the hostname, Longest number length of the hostname	R. M. Mohammad et al. (2013); L. A. T. Nguyen et al. (2014); El-Alfy (2017); Jain and Gupta (2018b); W. Chen et al. (2018); Pratiwi et al. (2018); Sahingoz et al. (2019); Chatterjee and Namin (2019); Li et al. (2019); Wang et al. (2019); Orunsolu et al. (2019); Yang et al. (2019); Butnaru et al. (2021); Odeh et al. (2021)
Webpage content (i.e.., HTML content and reader visible content)	Request URL, URL of anchor, Server Form Handler, Pop-up window, Hiding the suspicious links, Disabling right-click, Submitting to email, Iframe, Customised status bar, Website redirects, Number of hyperlinks, Internal hyperlink ratio, External hyperlink ratio, Null hyperlink ratio (i.e., <a href="#">, <a href="#content">, <a href="JavaScript ::void(0)">), Cascading Style Sheets (CSS), Broken link (i.e., 404 not found), Login form link, Domain name in href, Sub-domains in href, Domain name in src, Sub-domains in src, HTML content length, Hidden or restricted information (i.e., hidden codes), URL brand name in HTML code, HTML string embedding, most frequent link brand vs URL brand, title brand vs URL brand, Downloadable malicious code, Number of comments used, Number of title tag, Onmouseover	R. M. Mohammad et al. (2013); El-Alfy (2017); Jain and Gupta (2018a); Chatterjee and Namin (2019); Wu et al. (2019); Li et al. (2019); Wang et al. (2019); Yang et al. (2019); Odeh et al. (2021)
Third-party Services	DNS record, Website traffic, Age of domain, Page rank, Alexa rank, Google index, Links to page, External reports (i.e., PhishTank), Inconsistent URL, Time to live (i.e., page exist time), SSL certificate, blacklisted domain, Abnormal cookie domain, Number of DNS	R. M. Mohammad et al. (2013); L. A. T. Nguyen et al. (2014); El-Alfy (2017); Jain and Gupta (2018b); Chatterjee and Namin (2019); Wu et al. (2019); Yang et al. (2019); Odeh et al. (2021)

be done in a controlled environment, and the most significant features can be picked carefully through further analysis (Zhang et al., 2007; Joshi et al., 2008; Dunlop et al., 2010; Jain & Gupta, 2016). Although manual techniques appear to be an effective method in many domains, it is inefficient in phishing since the significant phishing detection features are frequently obsolete due to the constantly changing nature of phishing attacks, as discussed in Section 2.9. Therefore, updating the key features and selecting the best collection of features is a time-consuming and challenging operation in manual techniques (Opara, Wei, & Chen, 2020). Furthermore, features in the manual feature technique are handcrafted and visible from the outside. As a result, attackers might target these features and bypass the solutions, making the solutions useless (Opara, Wei, & Chen, 2020).

Representation learning techniques like deep learning are becoming the latest trend in the anti-phishing domain due to the limitations of manual techniques. Many of the latest anti-phishing solutions shown in Table 3.3 use deep learning instead of conventional machine learning approaches, such as Support Vector Machine (SVM), Neural Networks (NN), and Naive Bayes classifier (Chauhan & Singh, 2018). This is mainly due to the architecture of deep learning, which can automatically discover the relevant representations by traversing through several levels of abstractions (LeCun et al., 2015). Further, the deep learning techniques can learn features from raw inputs, and it gives several advantages to an anti-phishing solution over a solution that has manually extracted features (LeCun et al., 2015). One such advantage is a selection of the best set of features. In manual techniques, feature selection is subjective and dependent on experts (Bahnsen et al., 2017). However, deep learning models extract features from raw data, and the selected features are unknown to the outside world since they behave like black boxes (Bahnsen et al., 2017; Sahoo et al., 2017). Therefore, the attackers cannot pre-plan a way to bypass these solutions as they can with manual feature-based solutions (Bahnsen et al., 2017; Sahoo et al., 2017).

3.5 Detection techniques

Human errors are the most prominent factor for successive phishing attacks (Alkhalil et al., 2021). Therefore, various solutions (*see* Table 3.3) have been implemented to safeguard Internet users from phishing attacks through early detection and warning. As shown in Figure 3.2, these solutions can be classified mainly into two groups: user education and software-based solutions.

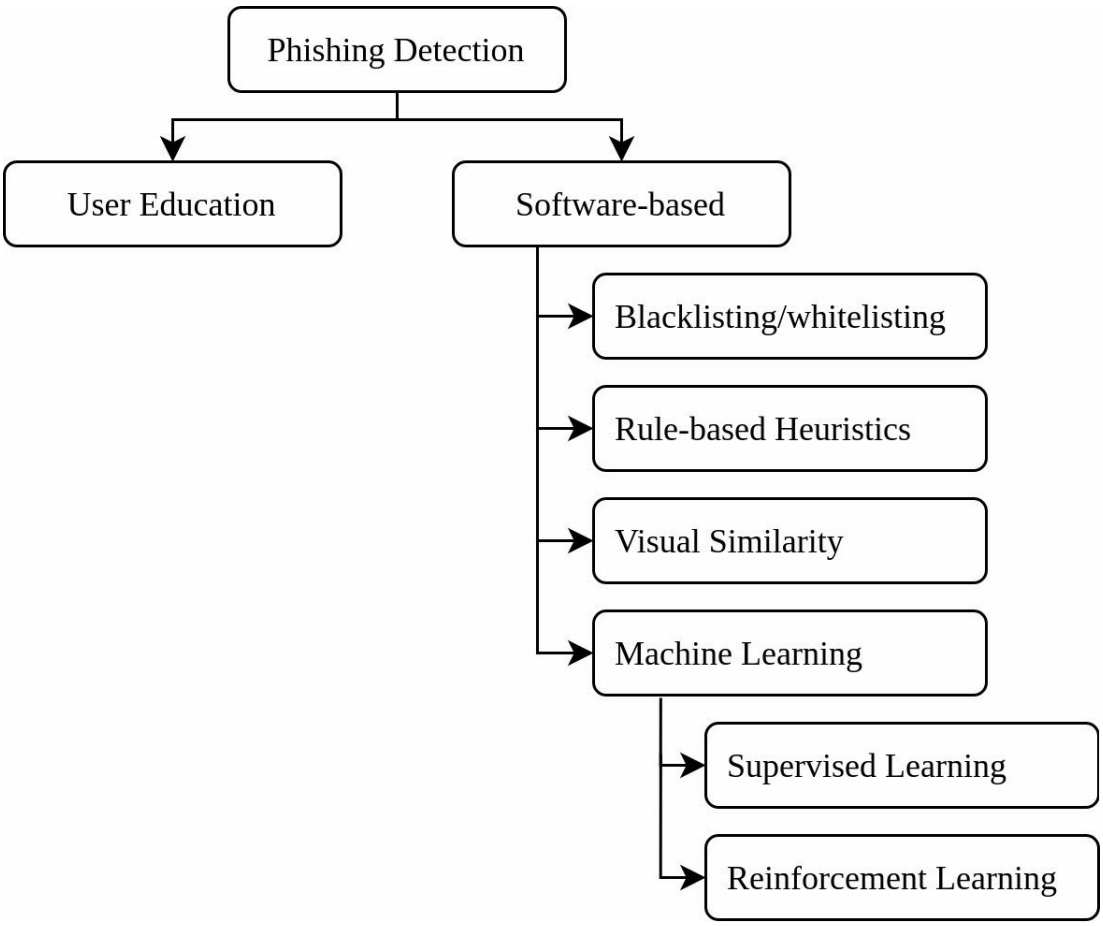


Figure 3.2: Categorisation of phishing detection solutions

3.5.1 User education

Novice people are the primary target of phishers (Khonji et al., 2013). Therefore, user education and awareness about phishing are critical in phishing detection (Khonji et al., 2013). As a result, several studies in the literature have tried to improve user education and awareness by introducing different solutions. These solutions have been

categorised under game-based education and other techniques such as online courses or seminars, phishing frameworks, and mock phishing attacks (Khonji et al., 2013; Tchakounté et al., 2020; Dixon et al., 2019).

Game-based education. Game-based education is a widespread technique applied in different studies when educating users about phishing (Tchakounté et al., 2020). It is considered an effective technique since it inherits the unique characteristics of gaming, such as fun of playing, intrinsic entertainment value, ability to ask questions from players and explore the players' ideas (Denning et al., 2013). The Anti-Phishing Phil proposed by Sheng et al. (2007) is a game-based approach focusing on phishing URLs. It has been built to practice good habits to avoid victimisation by phishing attacks. NoPhish (Canova et al., 2014) is another educational application similar to Anti-Phishing Phil, designed for Android smartphones. The main intention of this application is to provide knowledge about how to use URLs to distinguish between phishing and legitimate websites. This Android game has been designed with ten levels, and each level can be achieved based on the correct choices made by the user.

In addition, Smells Phishy (Baslyman & Chiasson, 2016) is another educational game that has been introduced in the literature. It aimed to teach users about the implications of their actions in phishing to understand the nature of phishing. Further, Smells Phishy has tried to teach the importance of SSL indicators, basic password rules, and common phishing attack characteristics to learn about the best cybersecurity practices. Similarly, What.Hack (Wen et al., 2019) is another interesting anti-phishing game that has been introduced under the user education category. It has been built with a sequence of puzzles and used to teach how a user should handle URLs, social media and attachments. According to the What.Hack experiments, this solution has increased the participants' ability to detect phishing and legitimate emails by 36.7%. Further, this study has shown that What.Hack is more effective in training than Anti-Phishing Phil, which focuses only on URLs.

Other techniques. X. Dong et al. (2008) developed a phishing framework known as a user-phishing interaction model after analysing 400 sample phishing attacks. The model was built by examining the user's selected information from an interface (i.e.,

clicking a link or giving some confidential information) and the assumptions or expectations (i.e., the system feedback) in each step. Therefore, the model discusses the interaction from beginning to end, and it illustrates where human users make mistakes when interacting with a phishing attack. Further, X. Dong et al. stated that forming an accurate perception when working with computing systems is the key to user education. However, phishing detection might not be effective only with human users, and humans need the support of others (i.e., system designers) to make the correct decisions against attacks (X. Dong et al., 2008). Moreover, Younis and Musbah (2020) implemented an interactive solution to enhance user awareness. This solution has a training awareness component to train the user and a gamification component to assess the acquired knowledge. Furthermore, they used animated videos to improve the knowledge of their trainers and game-based challenges to evaluate the trainers' knowledge.

In contrast to the techniques mentioned above, Khonji et al. (2013) mentioned that user education and awareness could be improved via online reading materials. There are plenty of online materials available on the Internet, and some of them are Microsoft's Security blog on Phishing[4] and eBay's tutorial on spoofed emails[5] (Khonji et al., 2013). Moreover, interactive notifications and warning messages are also common techniques that modern browsers have used to improve user awareness (Khonji et al., 2013). However, when these notifications and warnings are practised, the active interaction is considered superior to passive because many users often ignore passive interactions (Khonji et al., 2013; Dou et al., 2017).

Even though user education has been practised to minimise the phishing threat, it may not be a practical approach because phishing is continually developing, and new attacking strategies are being developed to respond to new security countermeasures (Alkhalil et al., 2021). Therefore, user education has become a costly strategy since the users need to update their knowledge about these attacks frequently, which needs many resources such as people, time and physical equipment (Khonji et al., 2013; Alkhalil et al., 2021). Furthermore, the user education approach also requires participants to have a minimum degree of security understanding to succeed, which is again a challenge in

[4] https://www.microsoft.com/security/blog/phishing/

[5] https://pages.ebay.com.my/education/spooftutorial/spoof_2.html

phishing (Khonji et al., 2013; Alkhalil et al., 2021). Thus, user education may not be a realistic solution to minimise the phishing impact, but it could be employed.

3.5.2 Software-based solutions

Khonji et al. (2013) categorised software-based solutions into four main categories: blacklisting/whitelisting, rule-based heuristic, visual similarity, and machine learning. Although this categorisation may seem a bit outdated, it can still be considered as valid in the anti-phishing domain. This is because when examine the current trends in phishing detection, the application of deep learning and reinforcement learning has emerged as significant techniques that were not discussed in Khonji et al. (2013)'s study. However, these two techniques fall under the broader category of Machine Learning, which has already been identified as a separate category in that study. Therefore, Khonji et al. (2013)'s categorisation remains valid in the field of phishing detection and is also adopted in this study. Furthermore, this study focuses on machine learning-based solutions within the domains of supervised learning and reinforcement learning, as the machine learning efforts explored here are predominantly centered around these two techniques (L. A. T. Nguyen et al., 2014; Bahnsen et al., 2017; Chatterjee & Namin, 2019; Wang et al., 2019; Opara, Chen, & Wei, 2020). As illustrated in Figure 3.2, this study analyses the previous software-based solutions under four categories, including two machine learning branches. The following sections discuss each of these techniques in detail, and the summary is presented in Table 3.2.

3.5.2.1 Blacklisting/Whitelisting

The blacklisting/whitelisting technique is based on the simple concept of text matching and uses a list of stored URLs when detecting phishing or legitimate websites (El-Alfy, 2017). A blacklist includes phishing URLs (Bell & Komisarczuk, 2020; Prakash et al., 2010), and legitimate URLs are handled by a whitelist (Cao et al., 2008; Jain & Gupta, 2016). In this technique, if a given URL is found on a blacklist, it is considered a phishing URL, and then the user is informed while blocking further access to the website. In contrast, if it is on the whitelist, it is considered a legitimate website which

Table 3.2: Overview of the standard phishing detection approaches

Approach	Limitations/Remarks
Supervised Learning – A model trains from known phishing and legitimate data. (Bahnsen et al., 2017; El-Alfy, 2017; W. Chen et al., 2018; Opara, Wei, & Chen, 2020; Opara, Chen, & Wei, 2020; Feng et al., 2020)	– It depends on a set of features (i.e. URL features). – A learning algorithm uses to adjust the weights of these features to achieve optimum performance. (LeCun et al., 2015; Buczak & Guven, 2016)
Reinforcement Learning – An agent is used to gather its experience in web surfing for sequential decision making. (Chatterjee & Namin, 2019)	– Agent produces an action (i.e. access/block a website) from a set of collected features from a website. – Action is measured (i.e. how good or bad) later with the correct action to be taken to make the learning happens. (François-Lavet et al., 2018)
User Awareness – A technique used in training Internet users to access the Internet services in a safe mode to protect them from phishing attacks. (Sheng et al., 2007; Baslyman & Chiasson, 2016)	– It is a machine-centric approach. – Game-based education has been found as an effective method when improving the user-awareness. – This technique expecting users to get educated about phishing is not a practical approach. (Khonji et al., 2013)
Blacklist & Whitelist – Blacklist contains a list of phishing website URLs, and a whitelist is a list of legitimate website URLs. (Cao et al., 2008; Prakash et al., 2010; Sánchez-Paniagua et al., 2020)	– List-based techniques require exact matching of the website URLs. – These techniques fail when detecting zero-day attacks since those may not include in the lists. – It is practically difficult to have an up-to-date list. (Khonji et al., 2013; Jain & Gupta, 2016; El-Alfy, 2017; Sánchez-Paniagua et al., 2020)
Rule-based Heuristics – A technique that uses a set of rules in detecting phishing attacks. (Teraguchi & Mitchell, 2004; Zhang et al., 2007; Joshi et al., 2008)	– This technique expects domain expertise when constructing high-end rules. – These rules need frequent updates to keep them alive. – The cost of updating the rules is high. (Khonji et al., 2013; Gupta et al., 2016)
Visual Similarity – A technique that uses the visual appearance of the web page in phishing detection. (Rosiello et al., 2007; Dunlop et al., 2010; Afroz & Greenstadt, 2011)	– These solutions depend on a threshold value, and it is difficult to find the optimum value. – This technique uses visual features such as text, HTML tags, CSS and images. – Maintaining an up-to-date database is challenging in these solutions; therefore, it fails to detect zero-day attacks. – Decision-making time is relatively high. (Jain & Gupta, 2017)

is safe to browse. Further, many services nowadays generate these lists for different purposes, and one famous blacklist is Google Safe Browsing (GSB) API[6] (Bell & Komisarczuk, 2020). It is a well-known blacklist that many current browsers employ to protect their users from phishing attacks (Bell & Komisarczuk, 2020).

Although this list-based technique appears simple, maintaining a list on today's Internet is challenging. The literature currently lists over 200,000 unique phishing attacks per month (APWG, 2021a), and the situation is similar on the legitimate side (Netcraft, 2021). Although many phishing websites are added, Khonji et al. (2013) mentioned that 47% to 83% of phishing URLs took twelve hours from their first appearance to enter a blacklist. It is a significant delay compared to the lifetime of a phishing website, as 63% of phishing efforts usually end within the first two hours (Khonji et al., 2013). On the other hand, maintaining these lists needs more than just collecting individual URLs since a practical list includes both reporting and confirmation, which is challenging on the present Internet (Jain & Gupta, 2016; Sánchez-Paniagua et al., 2020). Therefore, blacklisting/whitelisting is ineffective in phishing detection since it is vulnerable to zero-day attacks (Khonji et al., 2013; El-Alfy, 2017).

However, the literature has used various strategies to overcome the limitations of the blacklisting/whitelisting technique. The PhishNet tool (Prakash et al., 2010) is one such strategy which has some creative characteristics to address some of these limitations. These characteristics include a heuristic-based predictive blacklist that can generate new URLs from current ones and an approximate text matching technique that replaced the exact text matching technique of typical list-based techniques. In another study, Cao et al. (2008) introduced an Automated Individual White-List (AIWL), which is based on the Login User Interfaces. When a user enters credentials to a login interface, it first checks the whitelist. If the login URL is listed in the whitelist, this AIWL allows the user to move forward. Otherwise, it warns the users about the suspiciousness of this website. Although this AIWL seems interesting, it distracts users until it collects a list of known websites because this is a personalised whitelist.

Another whitelist called White-List maintainer (Jain & Gupta, 2016), which is

[6] `https://developers.google.com/safe-browsing`

based on multiple heuristics, has also been found in the literature. White-List maintainer primarily focuses on hyperlink properties and applies several heuristics. If these heuristics are passed, the URL is automatically added to the whitelist. However, the heuristics are not applied for each request in this solution. It first uses the whitelist as an initial filter, and if the specific URL is not listed in the whitelist, it forwards the URLs through these heuristics.

Even though these advanced solutions were attempted to improve the blacklisting/whitelisting technique, these efforts were insufficient to overcome the limitations mentioned in Table 3.2 (El-Alfy, 2017). Therefore, this technique could not eliminate zero-day attacks, which is essential for successful phishing detection.

3.5.2.2 Rule-based heuristic

Phishing heuristics are based on the typical characteristics of phishing pages (Khonji et al., 2013). These heuristics can be generalised and used as rules to detect phishing attacks (Khonji et al., 2013). SpoofGuard, a tool developed by Stanford University (Teraguchi & Mitchell, 2004), is one of several heuristic solutions available in the literature. SpoofGuard heuristics are based on domain names, URLs, links, and images, and these heuristics have been used to maintain a spoof score to make the ultimate judgement about a webpage. Another heuristic approach based on the TF-IDF algorithm is CANTINA (Zhang et al., 2007). It is a content-based solution that outperforms SpoofGuard by reaching 90% detection accuracy and a 1% false-positive rate. However, CANTINA had a significant false-positive rate of 6% at the beginning because CANTINA was mainly dependent on TF-IDF and Google search in its first implementation. Then, the research team combined simple heuristics such as the age of the domain, known images, suspicious URL, suspicious links, IP address, dots in URL, and forms with CANTINA. That strategy reduced false positives by 1% and allowed CANTINA to achieve its current accuracy.

Another heuristic solution based on the HTTP digest authentication principle is PhishGuard (Joshi et al., 2008). PhishGuard is a creative tool that hides the actual credentials, generates erroneous credentials many times, and checks the HTTP status

during a login attempt. This tool detects a phishing attempt in that process if the HTTP status is always sent 200 OK status for fake credentials or 401 UNAUTHORISED status for actual credentials. Moreover, R. M. Mohammad et al. (2014) also provided seventeen features for implementing an intelligent rule-based anti-phishing system. They thoroughly examined phishing detection features available in the literature and selected the relevant features based on popularity.

The rule-based heuristic technique is more successful than blacklisting/whitelisting, since it can detect zero-day attacks (Khonji et al., 2013). However, even though it detects zero-day attacks, more generic heuristics are prone to misclassifying legitimate websites, which is a significant drawback of this technique (Khonji et al., 2013). In addition, heuristic-based approaches have several other limitations, such as the visibility of used rules to the outside, which aids phishers in forging solutions, the validity of rules due to the rapidly changing nature of phishing, and the cost of updating rules (Khonji et al., 2013; Gupta et al., 2016).

3.5.2.3 Visual similarity

Visual similarity examines the visual look of a web page based on many characteristics (Khonji et al., 2013). There are a variety of visual similarity-based solutions, and one such solution is DOMAntiPhish (Rosiello et al., 2007). It has been considered a DOM-based method that uses a basic tag comparison when determining the similarity of two web pages. DOMAntiPhish uses three primary processes, initialisation, template computation, and coverage when identifying this similarity, and the final decision is solely based on the similarity value. Although DOMAntiPhish is a different approach, DOM obfuscation is a threat to this solution because the same website look can be achieved through multiple DOM trees in the current online context. As a result, L. D. Nguyen et al. (2014) presented an extension for DOMAntiPhish that checks the similarity of DOMs in two different methods, including the evolutionary algorithm-based DOM graph (L. D. Nguyen et al., 2014).

In contrast, PhishZoo (Afroz & Greenstadt, 2011) is a profile-based technique that detects phishing attacks using previously recorded profiles. These profiles have been

built using URLs, SSL certificates, HTML content, pictures, and scripts to achieve 96.1% detection accuracy. In another study, C.-Y. Huang et al. (2010) developed a site signature-based solution to detect phishing attempts. It is unique to a domain and is based on text and image-based elements. In their solution, the signature and domain information is cross-checked to determine the legitimacy of a website.

GoldPhish (Dunlop et al., 2010) is a browser-based plugin that detects phishing attacks by combining website logo information with Google search results. It is a viable technique for preventing zero-day attacks and superior to previous visual similarity approaches. However, the effectiveness of GoldPhish is dependent on the captured logo image and the ranking of the Google search (Jain & Gupta, 2017). This is a drawback of this solution because new businesses will struggle with GoldPhish due to their lower Google ranking (Dunlop et al., 2010; Jain & Gupta, 2017).

In addition to the visual similarity-based solutions discussed above, solutions like Phishing-Alarm (Mao et al., 2017), a wavelet hashing-based similarity mechanism (J.-L. Chen et al., 2020), pixel-level solutions (Fu et al., 2006), and many hybrid approaches (Jain & Gupta, 2017) have been proposed in the literature. Despite such approaches, visual similarity detection still faces several challenges: problems of defining a clear similarity value, maintaining databases in detection, ineffectiveness against zero-day attacks, and embedded object detection issues (Jain & Gupta, 2017). Therefore, it is not a competitive technique to fight against modern phishing attacks.

3.5.2.4 Machine learning

Machine learning-based phishing detection has begun a decade ago and has shown some promising results in the past (R. M. Mohammad et al., 2013; Bahnsen et al., 2017; Wang et al., 2019; Yang et al., 2019; Opara, Chen, & Wei, 2020). According to Table 3.3, these solutions have been implemented mainly with supervised learning and reinforcement learning techniques.

Supervised learning. The first supervised learning solution against phishing attacks was proposed by R. M. Mohammad et al. (2013). It was one hidden layer based Multi-Layer Perceptron (MLP) network that used seventeen input features to achieve

92.5% detection accuracy. Similarly, L. A. T. Nguyen et al. (2014) presented another single-layer neural network with six heuristics. It was tested against 11,660 phishing sites and 10,000 legitimate sites and achieved 98% accuracy. Furthermore, Pratiwi et al. (2018) also proposed a neural network using the phishing detection features proposed by R. M. Mohammad et al. (2013). However, it could not reach the accuracy of R. M. Mohammad et al. (2013) and could only achieve 83.38% detection accuracy.

Phish-Safe (Jain & Gupta, 2018b) is another phishing detection solution developed using fourteen URL features. These features were first evaluated using SVM and Naïve Bayes classifiers. Then, SVM was selected as the best classifier since it has achieved 90% detection accuracy. In a different study, Sahingoz et al. (2019) compared seven different machine learning algorithms, Naive Bayes, Random Forest (RF), k-Nearest Neighbour (k-NN), Adaboost, K-star, Sequential Minimal Optimisation (SMO) and Decision Tree, with three feature classes, Natural Language Processing (NLP)-based features such as average word length and keyword count, word vectors (i.e., converting words like 'online', 'protect' and 'store' into vectors) and hybrid features (i.e., NLP features and word vectors) to classify phishing and legitimate URLs. The RF algorithm with NLP-based features achieved the best performance during the experiment, and the reported accuracy was 97.98%. In a different study, Jain and Gupta (2018a) introduced an anti-phishing solution based on the HTML code of the webpage, and it achieved an overall accuracy of 98.42% using the hyperlink information available on the given web pages. However, the study only employed 2,544 phishing and legitimate web pages, relying mainly on PhishTank and Alexa.

The fuzzy logic-based data mining approach proposed by Aburrous et al. (2010) can be considered as yet another machine learning approach in phishing detection. This approach utilizes three layers and six phishing criteria: URL and domain identification, security and encryption, source code and JavaScript, page style and contents, web address bar, and human social component. The study demonstrates that among all the criteria, URL and domain identity are the essential factors in recognizing e-banking-related phishing websites. However, the features used in this solution require further analysis, as Aburrous et al. mentioned that their work is insufficient for clas-

Table 3.3: Phishing detection solutions exist in the literature

Solution	Detection Approach							Features			Feature Engineering Technique		Accuracy
	Supervised Learning	Reinforcement Learning	User Awareness	Blacklist	Whitelist	Rule based Heuristics	Visual Similarity	URL-based	Content-based	External	Manual	Representation Learning	
SpoofGuard (Teraguchi & Mitchell, 2004)						✓		✓	✓	✓	✓		*NS***
Anti-Phishing Phil (Sheng et al., 2007)			✓					✓			✓		87.0%
DOMAntiPhish (Rosiello et al., 2007)							✓		✓		✓		*NS*
CANTINA (Zhang et al., 2007)						✓		✓	✓	✓	✓		90.0%
AIWL (Cao et al., 2008)					✓	✓	✓		✓	✓	✓		100.0%
PhishGuard (Joshi et al., 2008)						✓				✓	✓		*NS*
PhishNet (Prakash et al., 2010)				✓		✓		✓			✓		*NS*
GoldPhish (Dunlop et al., 2010)						✓			✓	✓	✓		98.0%
PhishZoo (Afroz & Greenstadt, 2011)							✓	✓	✓	✓	✓		96.0%
Self-structuring MLP Network (R. M. Mohammad et al., 2013)	✓							✓	✓	✓	✓		92.5%
NoPhish (Canova et al., 2014)			✓					✓			✓		*NS*
Single-layer Neural Network (L. A. T. Nguyen et al., 2014)	✓							✓		✓	✓		98.0%
Phishing Detection using Public Key Certificates (Z. Dong et al., 2015)	✓									✓	✓		NS
Smells Phishy (Baslyman & Chiasson, 2016)			✓					✓	✓		✓		75.0%
White-List Maintainer (Jain & Gupta, 2016)					✓	✓			✓	✓	✓		89.4%

Continued on Next Page...

Table 3.3 – Continued

Phishing Detection with Rogue Certificates (Z. Dong et al., 2016)	✓									✓	✓		NS
Phishing URL Detection (Jeeva & Rajsingh, 2016)	✓							✓			✓		93.0%
Probabilistic Neural Network (PNN) (El-Alfy, 2017)	✓							✓	✓	✓	✓		96.8%
LSTM Network (Bahnsen et al., 2017)	✓							✓				✓	98.7%
Random Forest Classifier (Subasi et al., 2017)	✓							✓	✓	✓	✓		97.4%
Hyperlink-based Detector (Jain & Gupta, 2018a)	✓								✓		✓		98.4%
Phish-Safe (Jain & Gupta, 2018b)	✓							✓			✓		90.0%
LSTM Recurrent Neural Network (W. Chen et al., 2018)	✓							✓			✓		99.1%
Phishing Detection using ANN (Pratiwi et al., 2018)	✓							✓	✓		✓		83.4%
Href-based Detection (Wu et al., 2019)	✓								✓	✓	✓		89.3%
Deep RL based Detection (Chatterjee & Namin, 2019)		✓						✓	✓	✓	✓		90.1%
What.Hack (Wen et al., 2019)			✓					✓			✓		*NS*
Machine Learning based Detection (Sahingoz et al., 2019)	✓							✓			✓		98.0%
Stacking Model (Li et al., 2019)	✓							✓	✓		✓		97.3%
PDRCNN (Wang et al., 2019)	✓							✓				✓	97.0%
MFPD Model (Yang et al., 2019)	✓							✓	✓	✓	✓	✓	99.0%
HTMLPhish (Opara, Wei, & Chen, 2020)	✓								✓			✓	97.2%
PhishHaven (Sameen et al., 2020)	✓							✓			✓		98.0%
WebPhish (Opara, Chen, & Wei, 2020)	✓							✓	✓			✓	98.0%
Web2Vec (Feng et al., 2020)	✓							✓	✓			✓	99.0%
PHIBOOST (Odeh et al., 2021)	✓							✓	✓	✓	✓		98.9%

****NS** means 'Not Specified'. It is used when the solution's accuracy cannot be found.

sifying e-banking phishing websites, and additional efforts are needed to collect the most effective e-banking phishing detection features.

In addition, Jeeva and Rajsingh (2016) employed an association rule mining technique to extract rules for detecting phishing websites. Using the Apriori algorithm, they produced eighteen rules that correctly detected 93% of phishing URLs. In a separate study, Subasi et al. (2017) applied multiple learning algorithms to develop an effective anti-phishing system. The algorithms included Artificial Neural Networks (ANN), k-NN, SVM, C4.5 Decision Tree, RF, and Rotation Forest (RoF). Their findings revealed that RF performed the best in detecting phishing attacks, achieving a detection accuracy of 97.36% during the experiment.

In the realm of machine learning, Z. Dong et al. (2015) developed a real-time system for detecting phishing webpages hosted on HTTPS-enabled servers. They utilized 42 features from X.509 certificates and achieved an impressive 95.5% recall in identifying phishing websites, with an average precision of 93.7%. However, their system has a limitation—it can only detect phishing sites on HTTPS. Similarly, Z. Dong et al. (2016) also conducted experiments with machine learning techniques, using rogue certificates to detect phishing attacks. In their study, they selected several root Certificate Authorities (CA) and achieved significant results. However, in a separate study, Drury and Meyer (2019) pointed out that distinguishing between phishing and legitimate websites based on certificate information is challenging. The reason behind this argument is that phishing sites often use certificates with information similar to genuine sites, especially if both use certificates from the same issuer. Phishers can exploit compromised server certificates to make their sites look authentic, making it difficult to differentiate between them based solely on certificate details.

Bahnsen et al. (2017) proposed the first Long Short-Term Memory (LSTM)-based anti-phishing solution. It only utilises URLs and immediately feeds these URLs into the LSTM model after transferring to a machine-understandable format. This solution has achieved 98.7% detection accuracy and has been considered an effective solution compared to previous solutions. In addition, Bahnsen et al. demonstrated that the LSTM was a powerful technique for detecting phishing URLs by comparing the LSTM

performance with an RF model. The experiment used the same 2,000,000 URLs for both techniques and revealed the effect of representation learning in phishing detection. In a separate study, W. Chen et al. (2018) also used LSTM to detect phishing URLs and obtained 99.1% detection accuracy. However, W. Chen et al. used manual feature extraction during the proposed methodology and did not use the representation learning effectively.

Furthermore, W. Chen et al. (2018) showed that the LSTM is superior to the Convolutional Neural Network (CNN). They demonstrated that the LSTM could achieve more than 0.02% accuracy over the CNN in the same dataset. However, the experimented dataset was relatively limited and had only 4,000 URLs divided evenly between phishing and legitimate. Although LSTM outperformed CNN when detecting malicious URLs, Pham et al. (2018) stated that a CNN combined with LSTM could outperform each of these techniques individually. As a result, Wang et al. (2019) presented an anti-phishing solution named PDRCNN using LSTM and CNN. This study employed 500,000 data points from PhishTank and Alexa and obtained a detection accuracy of 97%. The PDRCNN detection procedure is quick, and it could detect a malicious URL within 0.4 milliseconds.

El-Alfy (2017) used PNN to develop a machine learning-based anti-phishing solution. It clustered thirty features using the k-medoid clustering technique and achieved a detection accuracy of 96.74% with a benchmark dataset. PNN used preprocessed multi-modal features, which include several third-party features. Although third-party features were employed in PNN, Li et al. (2019) mentioned that these features could increase the detection time of the solution. As a result, Li et al. (2019) introduced the first stack model-based phishing detection solution without relying on third-party service features. It used eight URL and HTML-based characteristics which can be extracted internally, including a new HTML string embedding feature. The stack model employed Gradient Boosting Decision Tree (GBDT), eXtreme Gradient Boosting (XGBoost), and Light Gradient Boosting Machine (LightGBM) in several layers to achieve 97.3% detection accuracy.

According to the literature, Yang et al. (2019) also proposed an XGBoost and

CNN-LSTM-based phishing detection solution. It used multi-modal features, including URL-based sequence features, URL-based statistical features, webpage code, and text features. By proving the previous statement made by Li et al. (2019) about detection time, the solution recorded a high detection time. As a result, Yang et al. employed a threshold value to filter the web pages that require more features during the evaluation. Although a practical threshold value seems challenging, that strategy decreased the detection time of this solution during the experiment.

In light of the literature, HTMLPhish (Opara, Wei, & Chen, 2020) was the first anti-phishing solution that relies solely on HTML analysis. It used a representation learning approach, and the raw HTML documents were fed to the model directly. This solution performed well during the experiment and achieved 97.2% detection accuracy. Moreover, WebPhish (Opara, Chen, & Wei, 2020) was an extension of HTMLPhish that combined URL features with the HTML features used by HTMLPhish. It uses raw URL and HTML and is the first of its kind. The WebPhish achieved 98% detection accuracy during the experiment. However, both HTMLPhish and WebPhish showed a noticeable performance degradation after two months, which was later regained after a successful retraining step. These two solutions demonstrated the effectiveness of the retraining process in regaining the performance of an anti-phishing model after a certain period of time.

Feng et al. (2020) also proposed an anti-phishing solution based on representation learning that employs URL and HTML in raw format. It was named Web2Vec and has outperformed all recent anti-phishing solutions (*see* Table 3.3) with a detection accuracy of 99%. Even though WebPhish and Web2Vec have demonstrated good accuracy, they have collected the experiment data from Alexa and PhishTank. However, Verma et al. (2019) and Aassal et al. (2020) stated that if a dataset collects data from Alexa and PhishTank, it might not have diverse URL lengths, resulting in misleading accuracy. Thus, the accuracies recorded by WebPhish and Web2Vec are problematic because the diversity of the used features has not been analysed in these solutions, similar to Aassal et al. (2020). PhishHaven (Sameen et al., 2020) is a different solution that looks at phishing URLs from human and AI perspectives. It has trained with

typical phishing URLs and AI-generated phishing URLs and has detected both forms of phishing URLs with an accuracy of 98%.

Reinforcement learning. This technique was not widely used against phishing attacks in the accessed literature. However, the study could find one solution under this category built by Chatterjee and Namin (2019). This solution depends entirely on URL-based features and employs dynamic behaviour. However, according to Table 3.3, the solution is not practical as it has recorded only a 90% detection accuracy. Although it has not been quite effective, the dynamic approach presented by Chatterjee and Namin has opened a different direction for future anti-phishing studies.

Despite some promising results in phishing detection (Dou et al., 2017), machine learning has specific challenges when dealing with the changing nature of phishing. Data drifting is one such challenge that is typical for all machine learning solutions (Sahoo et al., 2017; Aassal et al., 2020). However, machine learning-based anti-phishing solutions are greatly affected by this challenge because phishing attacks are constantly changing (Aassal et al., 2020). Even though a successful retraining process can address the issue of data drifting (Aassal et al., 2020), gathering a substantial amount of labeled data for the retraining process remains a challenge in the field of phishing (Sahoo et al., 2017; Zeng et al., 2020). Table 3.4 provides insights into the data quantities employed by recent machine learning-based phishing detection solutions and the corresponding machine learning algorithms utilized. It becomes evident that deep learning solutions, which have demonstrated promising results, have extensively relied on abundant data.

In addition to these, deep learning solutions are especially brittle on adversarial attacks such as data poisoning, adversarial inputs at run-time, and privacy attacks (Sahoo et al., 2017; Shirazi et al., 2019; Kashyap, 2020). These attacks can turn good-performing models into inaccurate predictions by lowering the trustworthiness of a solution (Shirazi et al., 2019). Furthermore, implementing multi-modal feature-based solutions, which are more effective in phishing detection, is also challenging in the anti-phishing domain due to the lack of data collection approaches (Sahoo et al., 2017). Therefore, machine learning techniques face various difficulties when fighting

Table 3.4: Recent advances in machine learning-based phishing detection solutions

Study	Technique	Algorithm(s)[i]	Training Size		Accuracy	KAP[ii]
			Legitimate	Phishing		
Machine Learning based Detection (Sahingoz et al., 2019)	CML	RF	36,400	37,175	97.98%	No
Remarks: The study used NLP-based features, word features and hybrid features with seven different machine learning algorithms. All these features were extracted from URLs, and NLP-based features with RF showed the highest performance.						
Deep RL based Phishing Detection (Chatterjee & Namin, 2019)	RL	DQN	36,400	37,175	90.10%	No
Remarks: The study proposed a reinforcement learning-based approach to detect phishing attacks first time in accessed literature. However, it could not achieve good detection accuracy compared to Sahingoz et al. (2019), since both studies used the same data source. The proposed study is limited to URL-based and few domain-based features. Therefore, content-based and page-based features may increase the detection rate further, as proposed by the authors.						
Href-based Detection (Wu et al., 2019)	CML	SVM	10,000	5,000	89.30%	No
Remarks: A different approach proposed by the authors depends on the 'src' and 'href' attributes in a web page. Original URL features were cross-checked with 'src' and 'href' values to determine phishing pages. The presented accuracy is comparatively low; however, the study showed low false positives and negatives compared to famous CANTINA anti-phishing solutions.						
Stacking Model (Li et al., 2019)	CML	GBDT, XGBoost and LightGBM	31,873	20,074	97.30%	No
Remarks: The proposed solution included two and six new features extracted from URL and HTML content, respectively, out of 20 used features. Further, the authors tested the effectiveness of the visual features using a different dataset that contained screenshots of web pages as a separate experiment in the study. They used a pre-trained model and a small-scale CNN in the experiment, and CNN showed 98.60% detection accuracy with the used dataset.						
A predictive model for phishing detection (Orunsolu et al., 2019)	CML	SVM and Naïve Bayes	2,500	2,541	99.96%	No
Remarks: A small dataset was used during the experiment, and both SVM and Naïve Bayes achieved 99.96% detection accuracy.						
PDRCNN (Wang et al., 2019)	DL	LSTM, CNN	500,000	500,000	97.00%	No
Remarks: The study proposed another DL approach based on URL information like Bahnsen et al. (2017). The authors highlighted that the main drawback of the proposed solution is the training time. However, it is a common problem for any deep learning-based approach. Further, the study highlighted that if a phishing URL does not have relevant semantics, the proposed solution failed to detect it.						
MFPD Model (Yang et al., 2019)	DL	CNN, LSTM, XGBoost	989,021	1,021,758	98.99%	No
Remarks: The study highlighted that multidimensional features are essential when detecting phishing attacks. The proposed solution used the CNN-LSTM algorithm to get deep URL features. Further, URL statistical features, web page code features and web page text features were used as multidimensional features. The study mentioned that nearly a million phishing and legitimate data were used during the experiment. However, only 22,445 phishing samples and 22,390 legitimate samples were used when experimenting with multidimensional features since most of the web pages in the primary dataset was not accessible.						
HTMLPhish (Opara, Wei, & Chen, 2020)	DL	CNN	47,000	4,700	93.00%	Partial
Remarks: A content-based approach that used automatic feature engineering for the first time in accessed literature. All the elements in an HTML document were considered, and a longitudinal study was carried out to find the next retraining day to handle continuously evolving issues associated with phishing detection. However, the process of new data collection and retraining was not described comprehensively in the study.						

Continued on Next Page…

Table 3.4 – Continued

WebPhish (Opara, Chen, & Wei, 2020)	DL	CNN	47,000	4,700	98.00%	Partial
Remarks: The study proposed an anti-phishing solution that concatenates raw URL and HTML content of a web page for the first time in accessed literature. Further, it is a language-independent lightweight solution and is able to detect a web page within 194μs. Even though WebPhish retrained after 2 months, the study did not discuss a systematic way to acquire the latest knowledge in the future.						
Web2Vec (Feng et al., 2020)	DL	CNN, LSTM	24,800	21,303	99.00%	No
Remarks: The study used representation approach to introduce a novel phishing detection solution. The proposed solution used URL, HTML page content, and DOM (Document Object Model) structure of web pages.						
Lightweight URL-Based Phishing Detection (Butnaru et al., 2021)	CML	RF	305,737	74,436	99.29%	No
Remarks: The proposed solution performed well compared to SVM and MLP classifiers during the experiment. Further, the study compared the proposed solution performance with Google Safe Browsing API and achieved nearly 50% more average accuracy over Google Safe Browsing. The study used a vast legitimate dataset in the experiment. The authors claimed that using an imbalanced dataset in evaluation makes a realistic scenario since legitimate traffic is high compared to phishing traffic in the natural environment.						

[i] In this column, "CML" refers to Conventional Machine Learning, and "DL" stands for Deep Learning.

[ii] "KAP" stands for Knowledge Acquisition Process, which has been utilized in this context to assess the preparedness of these recent solutions in adopting newer phishing detection features. The term "Partial" is employed for solutions that have recognized the importance of such an approach, even though they have not yet implemented it.

against phishing attacks.

3.6 Performance evaluation

Phishing detection is a binary classification problem that checks whether phishing instances are available in a mixture of phishing and legitimate instances. Therefore, phishing detection can have only four classification probabilities. These are $N_{p\to p}$: the number of correctly detected phishing instances (*referred to as* True Positive or TP), $N_{p\to l}$: the number of instances where phishing is marked as legitimate (*referred to as* False Negative or FN), $N_{l\to l}$: the number of correctly detected legitimate instances (*referred to as* True Negative or TN), and $N_{l\to p}$: the number of legitimate instances marked as phishing (*referred to as* False Positive or FP). The more descriptive presentation of these four probabilities is presented in the confusion matrix in Figure 3.3.

Furthermore, the literature has mainly used five evaluation metrics to evaluate the performance of previous anti-phishing solutions based on the given confusion matrix (Khonji et al., 2013; Dou et al., 2017; Opara, Chen, & Wei, 2020; Feng et al., 2020). The details of the evaluation metrics are listed below.

- *False Negative Rate (FNR):* This metric measures the number of phishing instances classified as legitimate to the total number of phishing instances.

$$FNR = \frac{FN}{TP+FN} \tag{3.1}$$

- *Precision:* This metric measures the number of instances where phishing is accurately identified to the total number of instances identified as phishing.

$$Precision = \frac{TP}{TP+FP} \tag{3.2}$$

- *Recall:* This metric measures the number of instances where phishing is correctly identified to the total number of phishing instances.

$$Recall = \frac{TP}{TP+FN} \tag{3.3}$$

- *f1-score:* This metric measures the harmonic mean between precision and recall.

$$f1-score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.4}$$

- *Accuracy:* This metric measures the proportion of phishing and legitimate instances accurately marked to the total number of instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.5}$$

An anti-phishing tool sees a phishing website much less frequently than a legitimate one on the Internet. As a result, the phishing to legitimate ratio is imbalanced, and the accuracy metric may be meaningless. In such instances, *f1-score* is the most influential metric utilised by earlier studies (Aassal et al., 2020; Opara, Chen, & Wei, 2020; Opara, Wei, & Chen, 2020). Even though the Internet has an imbalanced phishing-to-legitimate ratio, most of the previous anti-phishing solutions have primarily been evaluated in balanced environments, and few in an imbalanced environment or both (Aassal et al., 2020; Opara, Chen, & Wei, 2020; Opara, Wei, & Chen, 2020).

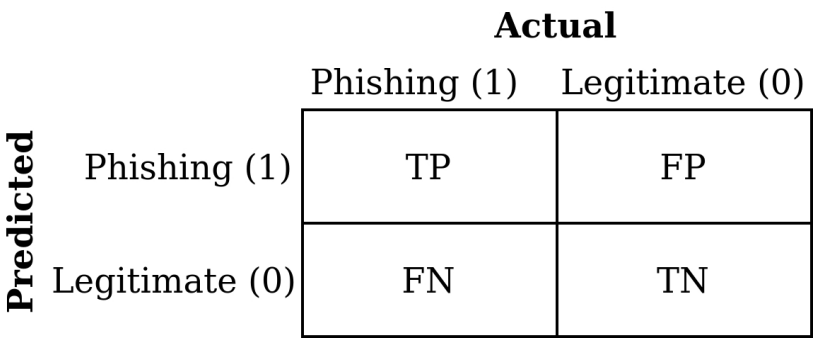


Figure 3.3: Classification confusion matrix

The TP category includes correctly predicted phishing web pages, while the TN category includes correctly predicted legitimate web pages. The categories where the incorrect classification occurs are FP and FN. In the FP, legitimate web pages are predicted as phishing, while phishing web pages are predicted as legitimate in the FN. Furthermore, this study has two class labels, 1 and 0, representing phishing and legitimate, respectively.

3.7 Present challenges

On the current Internet, comprehensive phishing detection is a timely need due to the constantly changing phishing attacks (Sahoo et al., 2017; Aassal et al., 2020; Alabdan, 2020; Alkhalil et al., 2021). However, the challenges that exist in the anti-phishing domain hinder the development of such attempts (Li et al., 2019). These challenges are from different angles, and the following is a detailed overview of the identified challenges.

1. *Emerging challenges:* The current phishing attacks are mainly developed using the support of new tools that can generate sophisticated attacks capable of bypassing some modern security controllers (Alabdan, 2020; Alkhalil et al., 2021). DeepPhish (Bahnsen et al., 2018), an AI-powered tool, is one such example designed to bypass URL-based phishing detectors by selecting the best-attacking URLs. On the other hand, phishing kits are also becoming popular among phishers. These kits remove the technical barriers that phishers face when constructing an attack and increase the complexity of modern attacks (Alabdan, 2020). Other than these, some emerging attacking techniques such as skill squatting, typosquatting, and tab-napping are also available on today's Internet (Alabdan, 2020). Complex attacks like skill squatting coupled with smart speakers are also challenging current phishing detection efforts (Alabdan, 2020). Because of these emerging challenges, phishing detection has become more complicated at present, and more robust detection techniques are required for future phishing detection success (Alabdan, 2020; Aassal et al., 2020; Sahoo et al., 2017).

2. *The difficulty of acquiring training data:* Phishing detection is based on previous examples, and good quality training data is essential for a successful training step. However, the literature has mentioned that a well-defined standard for collecting high-quality phishing data is a challenge in phishing detection (Verma et al., 2019; Aassal et al., 2020). As a result, previous researchers have conducted their studies using self-generated datasets (Bahnsen et al., 2017; Wang et al., 2019; Opara, Wei, & Chen, 2020; Sánchez-Paniagua et al., 2020; Butnaru et al.,

2021). However, constructing such datasets in a single night is also impossible due to the short lifespan of phishing pages (Bell & Komisarczuk, 2020; Zeng et al., 2020). Furthermore, previous studies have shown that many of the datasets used had several limitations, such as size, up-to-dateness, and diversity which could affect the performance of the final solution (Verma et al., 2019; Aassal et al., 2020; Sánchez-Paniagua et al., 2020). In addition, Verma et al. (2019) and Aassal et al. (2020) stated that the current anti-phishing domain lacks multi-modal feature-based reliable phishing datasets, which is a current requirement. Even though these issues have been discussed in the literature, there is still a lack of a well-established systematic approach for collecting large-scale phishing data for future anti-phishing initiatives.

3. *Lack of latest phishing data:* A well-labelled dataset is vital in anti-phishing research. Similarly, up-to-date phishing data is also essential to enhance real-world detection. In an anti-phishing study, if the training step uses old examples, the ultimate goal of the study will be useless since it may not perform well at present. As a solution to this problem, some previous studies have used (Wang et al., 2019; Yang et al., 2019; Orunsolu et al., 2019; Butnaru et al., 2021) phishing verification systems like PhishTank. Although these systems are a good choice, they only provide limited information (e.g., URL information), as explained in Section 3.3. Therefore, as Sahoo et al. (2017) and Aassal et al. (2020) highlighted, the lack of a reliable source for collecting the latest phishing data is an early challenge for any anti-phishing study.

4. *Data drifting leads to model decay:* Phishers are moving with the technology. As a result, they are keen to use new technologies to change their attacking strategies (Aassal et al., 2020; Alkhalil et al., 2021; Alabdan, 2020). However, such attempts negatively impact the existing models because these new attempts modify the significant phishing detection features employed in phishing attacks (Aassal et al., 2020; Sánchez-Paniagua et al., 2020). For example, consider the use of HTTPS in modern attacks. In 2017, only 30% of phishing attacks used HTTPS in their attacks. Therefore, earlier solutions (R. M. Mohammad

et al., 2013; El-Alfy, 2017; Jain & Gupta, 2018b) used HTTPS as a significant phishing detection feature. However, the use of HTTPS in phishing attacks has reached an 80% level at present and almost tally with HTTPS usage on legitimate websites (APWG, 2021b). As a result, the importance of the HTTPS feature in phishing detection is degraded presently. This indicates that the constantly changing phishing attacks change the significant features that could easily be used to detect phishing attacks. It causes a problem for existing anti-phishing solutions because these changes reduce their performance (Aassal et al., 2020; Opara, Wei, & Chen, 2020; Opara, Chen, & Wei, 2020; Sánchez-Paniagua et al., 2020). Therefore, frequent updates of significant phishing detection features are critical to maintaining the performance of these anti-phishing solutions. Even though this has been identified, it is a challenge for anti-phishing solutions due to a lack of effort in systematic knowledge acquisition processes (Sahoo et al., 2017; Aassal et al., 2020).

5. *Issues in third-party feature representation:* According to Yang et al. (2019), multi-modal features are critical for an effective anti-phishing solution because it reflects phishing attacks from different perspectives. However, many of the previous solutions (Bahnsen et al., 2017; Wang et al., 2019; Opara, Chen, & Wei, 2020; Feng et al., 2020; Butnaru et al., 2021) have relied heavily on URL and HTML features since third-party features such as domain age, Alexa ranking, and Google ranking mainly cause service delays and increase the detection time (Li et al., 2019; Sahingoz et al., 2019). Therefore, if a solution is planned to design on third-party services to get the support of multi-modal features for effective phishing detection, the designers should consider a better strategy to handle these service delays to satisfy their end-users.

6. *Adversarial attacks:* Adversarial attacks are mainly associated with machine learning-based solutions (Sahoo et al., 2017; Shirazi et al., 2019; Kashyap, 2020). These attacks could be categorised under privacy attacks, adversarial inputs at runtime, and data poisoning (Kashyap, 2020). Furthermore, these attacks can convert well-performing models into incorrect predictions and lower

their trustworthiness (Shirazi et al., 2019). Therefore, adversarial attacks are challenging for current machine learning-based anti-phishing solutions.

3.8 Problem definition

Phishing is a type of fraud that primarily targets personal or confidential information, damaging Internet credibility (Alabdan, 2020; Alkhalil et al., 2021). It is one of the most successful forms of cybercrime on today's Internet and has grown from thousands to hundreds of thousands over the last several years (Alabdan, 2020; Alkhalil et al., 2021; APWG, 2021b; ENISA, 2021). According to the literature, phishing mitigation has been considered the most successful strategy against these attacks, which mainly depends on a successful phishing detection (Khonji et al., 2013; Alabdan, 2020; Alkhalil et al., 2021). Therefore, academics and the industry have worked together to develop successful phishing detection solutions for many years (Teraguchi & Mitchell, 2004; Sheng et al., 2007; Prakash et al., 2010; Baslyman & Chiasson, 2016; Bahnsen et al., 2017; Li et al., 2019; Jain & Gupta, 2016; Opara, Chen, & Wei, 2020; Feng et al., 2020). These efforts have been categorised mainly into user education and software-based solutions, and out of these, software-based solutions have shown some promising results, especially machine learning-based solutions (Khonji et al., 2013; Bahnsen et al., 2017; Li et al., 2019; Opara, Chen, & Wei, 2020; Feng et al., 2020; Alkhalil et al., 2021).

However, existing anti-phishing solutions are inadequate for detecting new phishing attacks when significant phishing detection features emerge rapidly on the Internet (Aassal et al., 2020; Opara, Wei, & Chen, 2020; Opara, Chen, & Wei, 2020; Sánchez-Paniagua et al., 2020). As a result, the performance of these anti-phishing solutions is declining over time, and it has become a significant problem in the current anti-phishing domain (Aassal et al., 2020; Opara, Wei, & Chen, 2020; Opara, Chen, & Wei, 2020; Sánchez-Paniagua et al., 2020).The primary reason for this problem is the insufficient focus that existing anti-phishing solutions have received when integrating newer phishing detection features into their systems (Aassal et al., 2020; Opara, Wei, & Chen, 2020; Opara, Chen, & Wei, 2020; Sánchez-Paniagua et al., 2020). This ob-

servation becomes evident when examining the "KAP" column in Table 3.4. It reveals that only a few solutions (Opara, Chen, & Wei, 2020; Opara, Wei, & Chen, 2020) have recognized the necessity of such an approach to ensure the accuracy of the anti-phishing solution. However, none of these solutions have actually implemented this approach to address the issue of declining performance.

Even though the literature identified the leading cause of the main problem, the anti-phishing domain currently lacks a systematic way of incorporating the latest phishing detection features into anti-phishing solutions due to various challenges, as discussed in Section 3.7. Therefore, this research aims to develop an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process to reduce the impact of phishing attacks on Internet users.

3.9 Summary

In the phishing mitigation process, phishing detection is an essential step. User education and software-based solutions are the two most common phishing detection approaches, and out of these, machine learning solutions which are classified under software-based solutions, have attracted a lot of phishing detection interest. It is mainly due to the learning ability of these machine learning solutions, especially deep learning solutions, which can extract features from raw data. Even though these solutions exist on the Internet, they are unprepared for newer phishing attacks due to many challenges. Therefore, the study aimed at an anti-phishing solution with continuous learning support to automatically update the effective phishing detection features. As the starting point, the next chapter discusses the proposed methodology and explains the essential stages to be followed when achieving the aim of the study.

4 RESEARCH METHODOLOGY

4.1 Introduction

Chapter 3 provided a comprehensive review of the literature on phishing detection, which is the core area of this study. It also defined the research problem and justified the aim. As mentioned in Chapter 3, this study aims to develop an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process. This chapter introduces the methodology as the initial step in that direction. It discusses the research design, solution implementation, environment setup, and methodological limitations to illustrate how this research attained its goals.

4.2 Research design

This study was primarily motivated by the increasing trend of phishing attacks noticed over the last few years. Phishing attacks primarily target the digital assets of Internet users and directly affect the credibility of the Internet. As a result, phishing mitigation is a timely need, and it always depends on successful phishing detection. Therefore, the goal of this study was to develop a phishing detection approach that could reduce the impact of phishing attacks.

Phishing detection is generally a classification task where the detection knowledge depends on previous examples. According to Villiers (2012), the research on phishing detection follows the positivism research philosophy since the detection knowledge can be reproduced. Following this philosophy, previous works on phishing detection were first analysed to understand the problem domain. This analysis discovered that existing phishing detection solutions had performance losses over a brief period (two months) due to a lack of attention dedicated to adopting newer phishing detection

features into their solutions. However, some of these solutions have regained their usual performance after updating their phishing detection knowledge with the latest phishing examples. According to these findings, if a successful knowledge acquisition process can be integrated with an anti-phishing solution, it can retain the phishing detection ability for a more extended period. Therefore, this research aims to develop an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process to reduce the impact of phishing attacks on Internet users.

This study followed a deductive reasoning since the proposed solution depends on the existing anti-phishing solutions. Therefore, these solutions were first analysed to identify the effective techniques employed in previous solutions. As a result, several distinct phishing detection techniques were identified in Chapter 3. In recent years, machine learning techniques have attracted most of the interest in phishing detection mainly due to their ability to learn and their previous success in detecting phishing attacks. Therefore, machine learning was selected as the appropriate technique for achieving the mentioned goal.

Since this research came under a quantitative research approach and followed an experimental research strategy, the study's experiments were conducted in controlled simulated environments with data. Therefore, the data collection step of the methodology was essential. Also, this research aimed to update the existing phishing detection knowledge through an automatic process in different time frames. As a result, time-series data was required to update the existing phishing detection knowledge and evaluate the proposed architecture. Since the data were collected at multiple time points, the time horizon was longitudinal.

A machine learning-based research has a set of standard steps: data collection, pre-processing, model building and evaluation (Kamiri & Mariga, 2021). Thus, these steps were effectively utilised in the proposed implementation process, which used three phases, as shown in Figure 4.1. Section 4.3 discusses these three phases in detail by justifying the design artefacts employed. A more detailed overview of the implementation process is divided into chapters since each phase included several sub-steps,

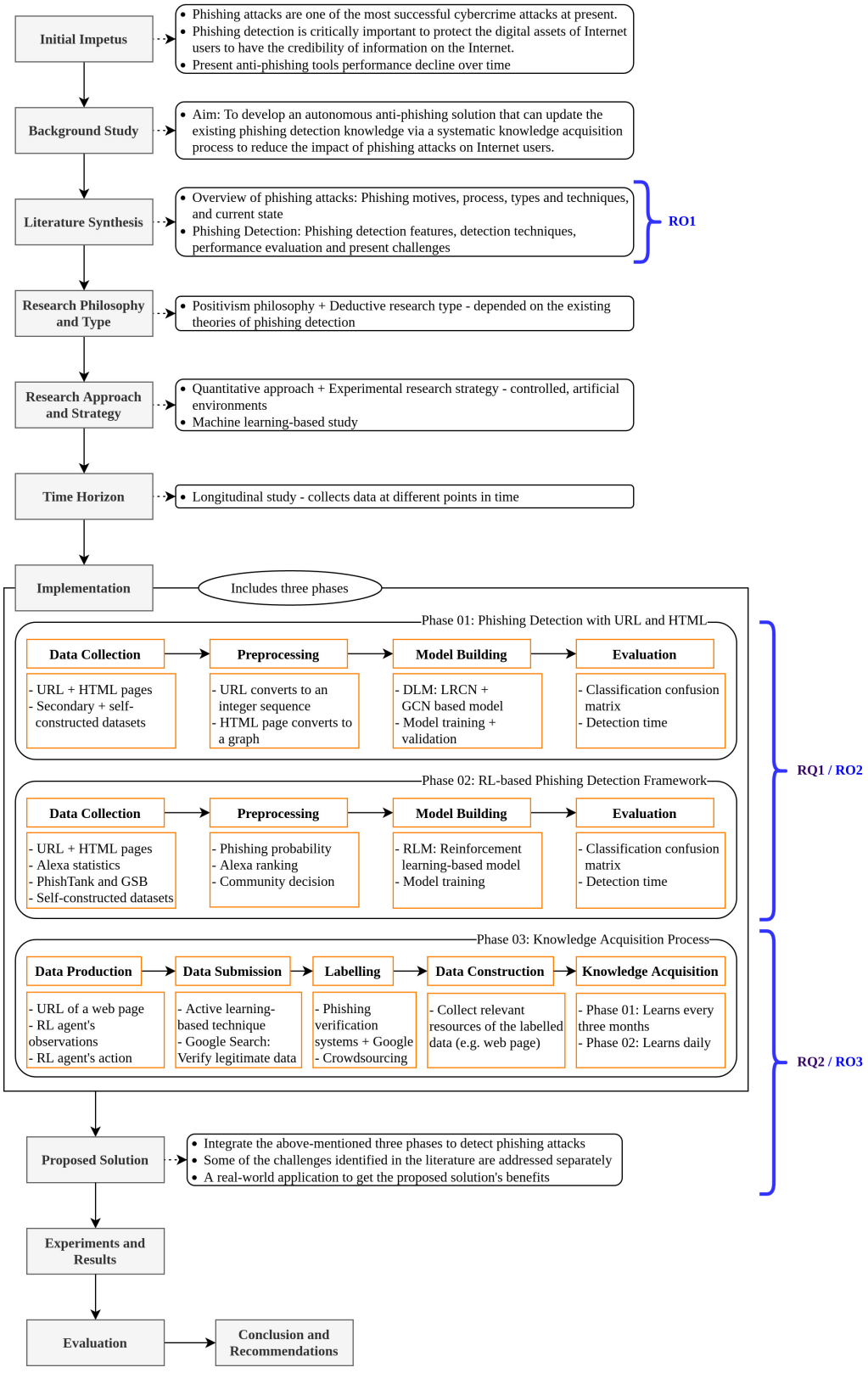


Figure 4.1: Research methodology

as shown in Figure 4.1. Therefore, Chapters 5, 6, 7, and 8 explain each phase of the implementation process and the final solution in detail.

The literature review identified a critical challenge after the solution was fully implemented. It was the impact of adversarial attacks, which was discussed in Section 3.7. According to that, these attacks generally come under three categories: privacy attacks, adversarial inputs at runtime, and data poisoning. Even though the privacy attacks could be managed through solid security practices like strong passwords and a secure physical setup, adversarial inputs at runtime were identified as a threat to the proposed solution. As a solution to this threat, a Generator Network (GN) was implemented with the support of a deep learning-based generative model called GAN to generate some possible adversarial inputs. These inputs were then used to train the proposed solution to detect these attacks in future. This GAN technique is explored in detail when introducing the GN architecture used in the final solution in Chapter 8. However, the third adversarial attacking type is minimal in the proposed solution because the used knowledge acquisition process primarily depends on self-generated data created after a systematic verification step. Therefore, these attacks were not considered during the implementation process.

The final solution was constructed after connecting the components mentioned above (i.e., phase two outcome, knowledge acquisition process and GN). This solution was then deployed as a Representational State Transfer (RESTful) web service, and was integrated with a web browser environment implemented by this study. After that, the proposed solution was evaluated using a simulated browser environment, and the results were thoroughly analysed and discussed. The conclusion of the final solution was discussed in terms of the aims and objectives to justify how the identified problem was solved through the proposed solution in the latter phase. Finally, the research was concluded by making a few future recommendations that could improve the solution performance further to detect more complex phishing attacks in the future.

4.3 Solution implementation

According to Section 4.2, the solution was implemented in three phases: Phase 1: Phishing detection with URL and HTML, Phase 2: RL-based phishing detection framework, and Phase 3: Knowledge acquisition process. The following sections discusses the implementation process for each phase by justifying the design considerations used.

Phase 1 implementation: Phishing detection always depends on the significant features employed to detect a phishing attack. Therefore, in Chapter 3, these features were analysed and clustered under three main categories: URL-based, HTML-based, and third-party (*see* Table 3.1). Even though the features from all these three categories are essential for better phishing detection, various previous solutions depended on URL-based and HTML-based phishing detection features due to limitations existing with third-party features, such as the detection time and service cost.

According to the analysis shown in Table 3.1, the internal features (i.e., URL-based and HTML-based) are the most vulnerable to phishing attacks because phishers have more control over them. Therefore, these internal features need to be retrained occasionally to detect the latest phishing attacks to overcome the first and fourth challenges mentioned in Section 3.7. Therefore, this phase one outcome was limited to URL-based and HTML-based phishing detection features, and this was one design constraint used in the initial design of the proposed solution.

As mentioned previously, the current study was driven by the machine learning technique. Therefore, manual or representation learning techniques were suitable for extracting significant phishing detection features. However, according to the study's aim, manual techniques were inappropriate because they required human involvement. Therefore, a representation learning technique called deep learning was selected to construct the phase one phishing detection solution. Various machine learning research have used deep learning, as shown in Chapter 3, although these has mostly been limited to the URL-based category. Therefore, a deep learning-based phishing detection solution with raw HTML content could not be found in the initial stage.

As a result, a novel approach was constructed in this study to analyse raw HTML

content to detect phishing attacks. It was GCN-based and the first method to phishing detection to employ GNN. Although deep learning was used in the implementation process, it is a black-box technique that does not allow for interpreting the significant phishing detection features employed in this solution. However, this constraint added a plus point to the proposed solution because these crucial features are hidden from the outside; therefore, the phishers could not easily bypass the proposed solution. After forming the design level decisions, the phase one solution was implemented with two deep learning architectures to manage a given website's URL and HTML content. Here, the URL part of the website was overseen by an LRCN architecture and HTML content analysis was performed by a GCN architecture. This implementation included several steps, as shown in Figure 4.1. Chapter 5 discusses these steps in detail when introducing the complete implementation of the phase one solution.

Following the successful implementation of the phase one solution, an anti-phishing solution with 96.4% detection accuracy and a 0.036 FN rate was produced. Even though it performed well during the experiment, the performance of the phase one solution declined by 9.35% for one year time. It indicated that the phase one solution performance was also affected over time, similar to the previous anti-phishing solutions. It implied that the phase one solution was affected by the research problem. Although retraining could improve the performance of this solution, this step demanded an enormous number of data because deep learning models require a large amount of data for a good training phase. However, according to the second and third challenges identified in Section 3.7, collecting a large amount of data in a shorter period is difficult in the phishing domain. As a result, phase two was proposed to minimise the performance drop until a retraining step was met.

Phase 2 implementation: All neural network solutions, in general, involve function approximation in their learning curves (Smadi, 2017), and these solutions could be regarded function approximation machines meant to accomplish statistical generalisation (Goodfellow et al., 2017, p. 169). However, when finding the optimal behaviour, the network depends on the preliminary information available in the training dataset (Smadi, 2017). Although this scenario is expected in the supervised learning paradigm,

it becomes a problem for neural network solutions that function in environments like phishing, where frequent changes and the latest examples are limited (Smadi, 2017; Sahoo et al., 2017; Aassal et al., 2020). In such cases, the Reinforcement Learning (RL) approach was proposed (Smadi, 2017). Since RL uses the trial-and-error concept, it is more appropriate to overcome the challenges of finding more latest data since it can learn through experience (Smadi, 2017; Sutton & Barto, 2018). Therefore, the phase one solution was upgraded to an RL-based phishing detection framework during phase two.

RL is an active machine learning approach that is based on the interaction between an intelligent agent and the environment (Sutton & Barto, 2018). In RL, an agent is responsible for learning optimal behaviour for a certain situation by interacting with the environment and gathering knowledge (Sutton & Barto, 2018). The main disadvantage of RL systems is the high computational costs associated with achieving the optimal policy (Tizhoosh, 2005; Smadi, 2017). Therefore, the current work selected only three observations to have an uncomplicated RL environment to work with less execution power. These observations are the phishing probability produced by the phase one solution, the popularity of a website and the present blacklist knowledge.

The main problem with the phase one solution was the performance degradation over time due to constantly changing phishing attacks. It might be due to the significant phishing detection features employed by this solution because the phase one solution relies entirely on internal features. Therefore, attackers have more freedom to change their strategies to create newer attacks. In contrast, the phase two solution used two external factors alongside the phishing probability generated by the phase one solution. As a result, this second phase solution may slightly alter the phase one decision depending on its experience with the other two aspects because this second solution is an RL environment. For example, if the phishing probability of a phishing website was 0.49, the phase one solution identified this website as a legitimate website. However, if the RL has past phishing website experience with a non-popular website with a 0.49 phishing chance, the RL may identify this website as a phishing website within the RL environment. This example scenario is possible in phase two because the phase one

solution is dependent on prior knowledge, whereas RL works in the real world using its experience.

Phase two produced a phishing detection framework. According to the results of the experiments, phase two output achieved 94.11% detection accuracy, whereas phase one achieved only 87.82% detection accuracy with the same experiment. It has been demonstrated that the phase two implementation achieved its objective. However, the entire implementation process cannot be discussed here because it includes several steps, as illustrated in Figure 4.1. As a result, Chapter 6 describes the steps followed when implementing the phase two output, including experiments and results.

Although the phase two solution addressed the performance drop problem identified in phase one, retraining these two solutions (i.e., phase one and phase two outputs) was required when addressing the fourth challenge presented in Section 3.7. As a result, phase three was critical in achieving the study's aim. This phase was primarily implemented to collect data for the phase one solution's retraining process and provide feedback to the phase two solution to continue its learning process. However, it followed a systematic approach from data collection to knowledge acquisition, as described in Chapter 8.

Phase 3 implementation: According to A. H. Mohammad and Al Saiyd (2010), a successful knowledge acquisition always depends on the entity that can provide the related knowledge. In phishing detection, this entity is phishing or legitimate web pages. However, legitimate web pages are easy to collect because these are typical websites available on the Internet. A simple Google search can provide more than a hundred thousand legitimate web pages to acquire knowledge for phishing detection. However, phishing data collection is challenging due to the second and third challenges mentioned in Section 3.7. Therefore, a systematic approach to collect and label phishing websites was an initial requirement, and it was identified as an existing gap in the anti-phishing domain. As a result, a gap-filling solution named PhishRepo, which collects, verifies, disseminates, and archives real-time phishing data, was proposed in phase three. Although this solution is used for phishing data collection, it has an interactive phishing data labelling process, which can be used to label both phishing and

legitimate data. Chapter 7 discusses more details about this solution.

The main objective of this knowledge acquisition process was to provide the latest data to update the existing phishing detection knowledge of the phase two solution. In that case, the recent phishing web pages are the perfect source to update the existing phishing detection knowledge (Khonji et al., 2013; Alkhalil et al., 2021). On the other hand, the phase two solution continuously interacted with the real environment when detecting phishing attacks. Therefore, the implemented RL environment in phase two was considered the appropriate place to collect recent phishing and legitimate data. Although the required data could be collected from this RL environment, proper labelling of these data was vital in the quality learning process.

However, in real-world execution, the RL environment often receives a large number of web page requests, and labelling each web page becomes a hard, time-consuming, and costly task. Therefore, the concept of active learning was selected to overcome the labelling bottleneck (Settles, 2009). Active learning is a subset of machine learning that queries an information source to label new data due to the difficulties of labelling every data point collected by a solution (Settles, 2009). There are several active learning techniques (Settles, 2009), and out of those, a pool-based active learning technique was selected to label unlabelled web pages.

As shown in Figure 4.2, active learning depends on an Oracle. Therefore, the current study selected PhishRepo as Oracle. Even though this PhishRepo could provide the Oracle service, sending more legitimate data made some problems in the PhishRepo process. As mentioned in Chapter 7, the PhishRepo labelling process consists of two parts, the first of which is automated and the second of which is dependent on crowd workers. The main intention of the first step was to reduce the workload of crowd workers. As a result, if a web page is labelled in the first stage, it will not proceed to the second step. Further, this first verification step depends on two primary phishing verification systems since PhishRepo intended to collect phishing data. Therefore, if a legitimate web page comes to this verification process, this web page ultimately goes to the crowd workers. It was considered an overload for the crowd workers because simple services like Google search ranking can verify the legitimacy of a web page

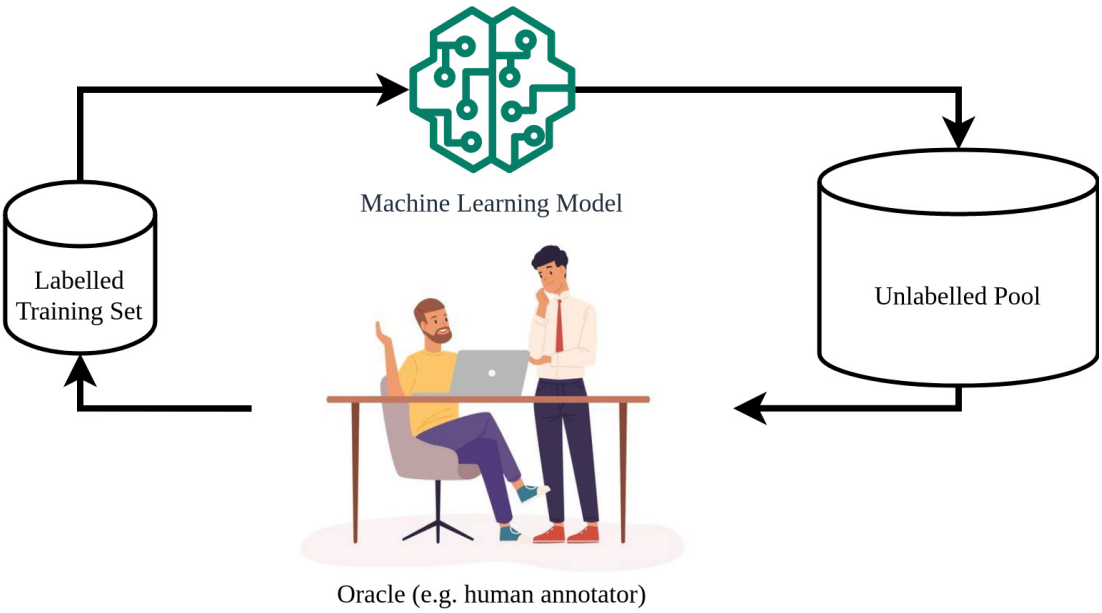


Figure 4.2: The active learning cycle
Source: Settles (2009, p. 4)

Active learning is a machine learning approach that involves iteratively selecting and annotating the most informative or uncertain data samples for training a model. By actively selecting the most informative data samples for annotation, active learning can achieve higher model performance with fewer labeled data points compared to traditional supervised learning approaches.

within a few seconds. Although this Google ranking service has some drawbacks (*see* Chapter 3), the implementation process decided to use this service only to verify the high probability legitimate websites marked by the RL agent to reduce the number of legitimates that were sent to Oracle. Therefore, when selecting the active learning strategy, a hybrid of expected model change and uncertainty sampling strategies were considered to label the unlabelled data (Settles, 2009).

The expected model change strategy generally labels the data item, which is more critical when changing the model behaviour. Therefore, with this strategy, all the phishing web pages marked by the RL agent were sent to Oracle because phishing examples are vital when updating the existing phishing knowledge. The other selected strategy depended on the uncertainty. Since all the phishing data were selected with the first strategy, this strategy only applied to the legitimate web pages. Therefore, high uncertainty legitimate web pages were also sent to Oracle for correct labelling. The uncertainty was measured by a formula derived from Shannon's entropy (Shannon,

1948):

$$E_w = -\sum_{i=0}^{N-1} p_i \, log_2 \, p_i \tag{4.1}$$

where E_w is the entropy for a given web page, N is the number of possible labels (i.e., phishing, and legitimate), and p_i denotes the legitimate and phishing probabilities of the web page, which were output by the phase one solution.

However, the disqualified legitimate web pages were separately verified using Google Search. It was merely a verification phase, and only the potential web pages were checked. Since the number of legitimates was higher than phishing in real execution, the labelling strategy did not introduce further problems. The implemented knowledge acquisition process included several steps, and Chapter 8 presents more details about the proposed knowledge acquisition process.

4.4 Environment setup

A successful environment setup is essential in a machine learning-based study to produce a good output. Therefore, the following considerations were made during a successful implementation process.

1. **Programming language: Python**

   The most used programming language in machine learning-based studies is Python (Kamiri & Mariga, 2021). Therefore, this study selected Python version 3.7 since it was the latest Python version available in the initial stage of this study. The python environment was installed from *Anaconda3-2019.03-Linux-x86_64* distribution.

2. **High-level library: Keras**[8]

   Keras and Apache MXNet are the two popular high-level neural network APIs used in Pythonic environments (Ramasubramanian & Singh, 2018). However, Keras is a widespread library with good community support (Ramasubramanian & Singh, 2018). Therefore, Keras was selected as the high-level library for the implementation tasks.

[8] `https://keras.io/`

3. **Backend engine: TensorFlow**

   There are several backend engines like TensorFlow, Theano, and CNTK in the current dataflow programming industry (Ramasubramanian & Singh, 2018). However, TensorFlow is the most efficient (Ramasubramanian & Singh, 2018), and this study also depended on TensorFlow version 2.1, which was the latest in the initial stage of this study.

4. **Hardware capability: CPU based machine**

   The solution was implemented in a CPU-based machine since it was affordable and readily available for this study. However, this machine was used only to function the final solution. The primary implementation process required a high-end machine, which is described in point five of this section. The basic machine was Intel(R) Xeon(R) CPU @ 2.30GHz virtual machine with one core and 8 GB of memory. It had 52GB of hard disk capacity.

5. **Cloud infrastructure: Google cloud**

   This study used high-end deep learning architectures like GCN. It was not obtained from the local environment due to resource constraints. Therefore, as an alternative way, this study decided to use cloud infrastructure since it was the most affordable and flexible solution due to time and budget constraints. Therefore, Google cloud was selected based on its attractive packages and cost-effectiveness. This cloud platform provided Intel(R) Xeon(R) CPU @ 2.20GHz based deep learning virtual machine with eight cores and 192 GB of memory. It further had 100GB of hard disk capacity.

6. **Database engine: MySQL**

   This study selected a free and open-source database engine called MySQL due to budget constraints. However, this solution did not use any high-end database processes. Therefore, MySQL had all the features to fulfil the database requirement of the implementation process.

7. **GNN library: Spektral**[9]

[9] `https://graphneural.network/`

Spektral is a Python library for deep graph learning based on the Keras API and TensorFlow. It is a simple and flexible framework for creating GNNs. This study used Spektral version 0.3 during the implementation process since it was the latest in the initial stage of this study.

4.5 Methodological limitations

As with most studies, the current study's design is subject to limitations as follows.

1. **The text/html content type HTML pages were only considered.**
   The HTML pages nowadays have different content types. For example, application/pdf, where the content is delivered as a PDF file, and text/html, where the content is delivered as a generic HTML page. However, all these content types cannot be used with this study's graph generation process due to specific structural differences compared to a generic HTML page. Therefore, text/html content type HTML pages were only considered during this study by filtering them through the content-type meta tag.

2. **The significant phishing detection features used during the detection are not visible to the outside due to the used technique.**
   This study used deep learning as it is an intelligent technology in situations where automatic feature extraction is frequently needed. However, deep learning technologies are black-box types, and the internal structure is not visible to the outside. Therefore, the lack of interpretability of the used feature in the proposed phishing detection approach is a limitation of this study.

3. **Existing phishing detection knowledge was updated only in three months, and optimal retraining time was not assessed due to infrastructure and resource limitations.**
   This study used a knowledge acquisition process to update the existing knowledge of the phishing detection features and provide feedback to the implemented RL environment. However, updating the current phishing detection features involved high-end deep learning techniques like GCN, requiring high-end compu-

tational power. Therefore, due to the infrastructure limitations, the performance loss was examined after three months, and regular check-ups were not performed to determine the optimal retrain duration. Furthermore, this study discovered a lack of up-to-date data to conduct a retraining process when time is limited. Therefore, a three-month interval for the retraining was defined and used several times.

4. **The RL environment of the proposed solution used delayed feedback mechanism due to the lack of resource limitations (i.e., human experts).**
   The proposed solution has an RL environment responsible for getting the final decision on a given web page. It is based on actions and rewards. However, the rewards depend on the feedback given by the external party, and it was incorporated with the implemented labelling process. Since the labelling process started two days after the submission, the feedback for action was delayed for at least two days. Therefore, after two days of delay, the agent learned about its past actions.

4.6 Summary

This research was motivated by the increasing number of phishing attacks in cyberspace. Phishing detection is a successful strategy against phishing attacks that entirely depends on previous theories. Although there are numerous anti-phishing solutions, they are ineffectual against continually evolving phishing attempts due to their lack of readiness. Therefore, this study proposed a new phishing detection approach that involved a systematic knowledge acquisition process. However, implementing this solution was not straightforward due to its complexity. Therefore, this proposed implementation was planned in three subsequent phases organised in several chapters. The next chapter discusses the first phase of the implementation process, named phishing detection with URL and HTML.

5 PHISHING DETECTION WITH URL AND HTML

5.1 Introduction

Chapter 4 discussed the research methodology followed by the study when achieving the research aim. It also justified specific design artefacts with the methodological limitation the solution experienced during the research design. Further, the methodology chapter highlighted that the proposed solution was implemented in three phases. This chapter focuses on phase one of these three, implementing a phishing detection solution with URL and HTML features. This chapter first discusses the overall design of the solution. Then, data collection and preprocessing, model training, and evaluation steps are discussed. Finally, a result and discussion section highlight the effectiveness of the phase one implementation when achieving the study's aim.

5.2 Overview of the solution

In the initial stage of this study, there was no deep network model[10] in the phishing detection domain that used representation learning to extract features simultaneously from URLs and HTML contents. However, several solutions effectively utilised deep learning with URLs (Bahnsen et al., 2017; Wang et al., 2019; Sahingoz et al., 2019; Butnaru et al., 2021). Therefore, the study planned to have two separate deep network models to handle URL and HTML features and combine the knowledge at the end to have a collective decision. For referencing simplicity, these models were given meaningful names. The model that handled URL features in the detection process was named URLDet, and the HTMLDet was the other one that handled the HTML

[10]The first model that uses representation learning to extract features simultaneously from URLs and HTML contents was introduced in November 2020. This study output was also presented to the scientific community at that time. However, due to the publication procedure, this study could not get the credit of the first model.

features. Then, the complete model means that the concatenation of URLDet and HTMLDet was named Deep Learning Model (DLM). However, the DLM building logic was first tested using a hybrid approach in the study's initial stage. Section 5.2.4 describes the used hybrid approach in detail, and the following sections (Section 5.2.1 to Section 5.2.3) describe the followed design process when implementing the DLM.

5.2.1 URLDet

Malicious URL detection through deep learning is a popular topic in the anti-phishing domain (Sahoo et al., 2017). This study first analysed different architectures used by previous studies. Then, the LRCN deep learning architecture with a CNN layer followed by an LSTM network (Donahue et al., 2015) was selected due to its past performance (Pham et al., 2018; Yang et al., 2019). In LRCN architecture, the CNN could work as a front layer to extract useful local features from raw URLs, and then LSTM extracts contextual features (Yang et al., 2019). Hence, the URLDet model was designed based on LRCN architecture. As shown in Figure 5.1, the URLDet model has three primary parts: preprocessing, feature extraction, and classification.

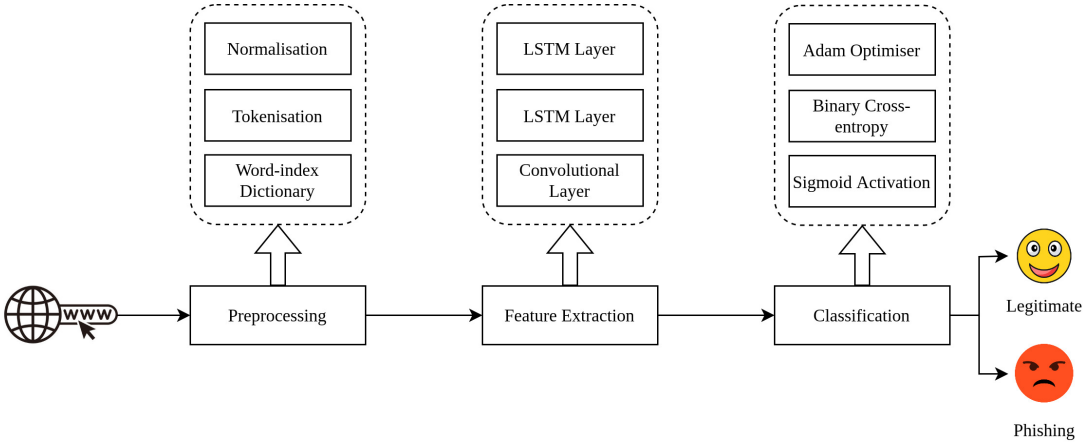


Figure 5.1: Workflow of the URLDet model

5.2.1.1 Preprocessing

A typical website URL looks like Figure 5.2. However, the feature extraction will be done by the deep learning architecture in deep learning approaches. Therefore, the

URL needs to be in machine-understand language and not in Figure 5.2 format. In neural network architecture, the inputs need to be in a vector of numbers to perform mathematical operations (Wang et al., 2019). Hence, the study wanted to preprocess the collected URLs to have a numeric format as the first step.

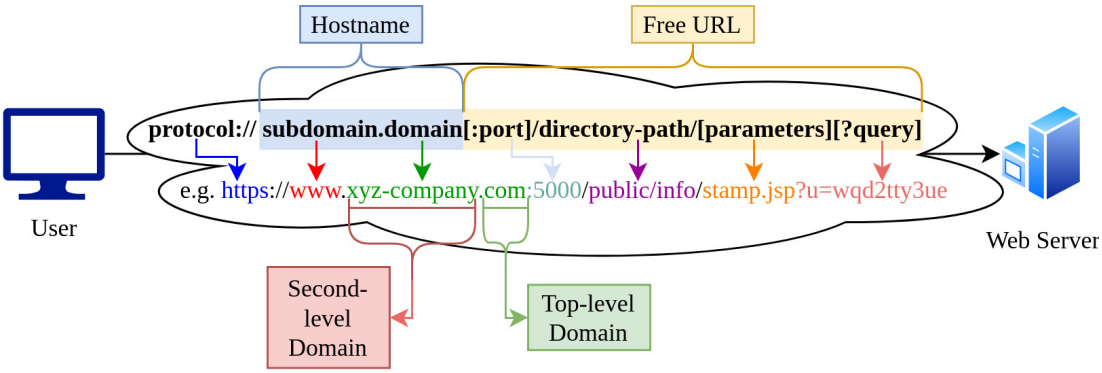


Figure 5.2: The typical structure of a URL
Source: Yang et al. (2019, p. 15198) and Sahingoz et al. (2019, p. 348)

At first, each character of a URL is considered a word, and the vocabulary index was created based on word frequency using the *text_tokenizer* and *fit_text_tokenizer* methods from the Keras library to have a word-index dictionary. The URL corpus for the dictionary was the URLs available in the training set of the classic dataset (*see* Section 5.3). Then, the *texts_to_sequences* Keras method transformed each character in the URL into a sequence of integers. In that process, each character in the URL was replaced with the corresponding integer value from the word-index dictionary to have a numeric format for a given URL.

As shown in Figure 5.3, the input layer of the URLDet was a tensor, and the tensor must have the same shape throughout the training process. Therefore, the study generated the character length distribution on the URLs available in the classic dataset. The distribution looked like Figure 5.4, and the URLs did not have the same size to produce the same shape. Therefore, as the second step of the preprocessing task, the URLs were normalised to have the same character lengths and a maximum length of 150. That value was selected based on the URLs' character length distribution, and as shown in Figure 5.4, most of the URLs were fully represented when it came to the 150th character. Thus, in the normalisation process, the URLs with more than 150

character lengths were chopped at the 150th character, and if the URL did not contain 150 character length, then the remaining character length was filled with 0s to have the same input shape. The Keras *pad_sequence* method was used during the normalisation process by setting the padding argument to 'post' to fill the zeros at the end when necessary. Algorithm 1 shows the preprocessing procedure for a 150-size sequence to feed the URLDet model.

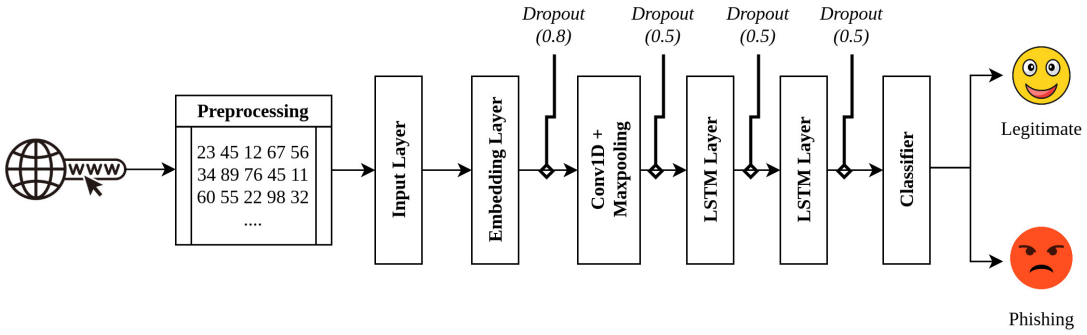


Figure 5.3: The URLDet architecture

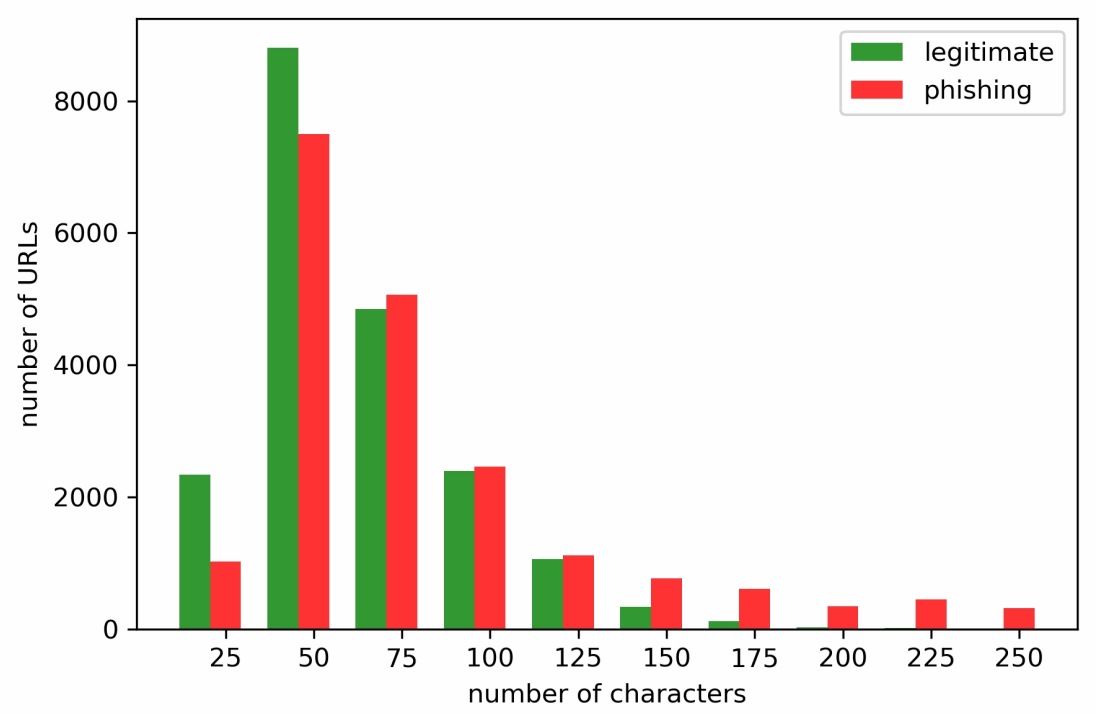


Figure 5.4: Character length distribution of the URLs

Algorithm 1 URL preprocessing procedure

```
1: procedure URLPREPROCESS(url, wordindexdic)
2:     maxlen ← 150
3:     padding ← post
4:     url_int_tokens ← wordindexdic.texts_to_sequences(url)
5:     url_normalised ← pad_sequences(url_int_tokens, maxlen, padding)
6: return url_normalised
```

5.2.1.2 Feature extraction

After the preprocessing step, the URLs were ready for the feature extraction. The LRCN was responsible for this task, and Figure 5.3 shows the proposed LRCN architecture for the URLDet. When designing the proposed architecture, the study followed the evolution approach that used random values to fine-tune hyper-parameters and network structure (Smadi, 2017). Therefore, the study used different CNN and LSTM layers with different hyper-parameter values. Then, the presented architecture was selected as the reasonable architecture. However, due to the time constraints of the study, all the different type of combinations with hyper-parameters and different network structures was not evaluated during the implementation process. Hence, the proposed model was built with the most influential architecture (Figure 5.3) found through the experimented architectures.

First, the preprocessed URLs were passed to the input layer in feature extraction. Then the input layer passed that input to the embedding layer to have a vector representation. The embedding layer was configured to have a vocabulary size of 100, 256 dense embedding dimensions, and a 150-length input sequence with 1e-5 valued L2 regulariser function. The result of the embedding layer was then inputted to the 1D convolutional and Maxpooling layers for the feature extraction, and the layers collectively extracted local deep correlation features from the embedding matrix. The window size of the 1D convolutional layer was set as three, and it had a 256 output size with the Rectified Linear Unit (ReLU) activation function. Further, the 'he uniform' initialiser with 'zeros' bias was used in the 1D convolutional layer, and the L2 regulariser function was also applied with the value of 1e-5. Then the result of the pooling was inputted into the first LSTM layer.

The LSTM network was responsible for carrying out the representation learning task, especially capturing the context of the URL sequence using the local features extracted from the CNN layer. In that process, the LSTM layers used a Hyperbolic Tangent (Tanh) activation function in a 32 output space with a 1e-5 valued L2 regulariser function. Further, the last moment output of the first LSTM layer was inputted to the second LSTM layer, and it further extracted the contextual features from the URL sequence. Finally, the last moment of the LSTM neural network was sent to the classifier for the classification task. Further, as shown in Figure 5.3, the dropout strategy was applied in several places to prevent overfitting in the feature extraction process.

5.2.1.3 Classification

After the feature extraction step, the classifier was responsible for the final decision about the URL. Since the URL is legitimate or phishing, the study built the classifier using a dense layer with the sigmoid activation function. Further, the study selected the binary cross-entropy loss function as the target loss function since this was a binary classification problem. However, when selecting the optimising strategy for the target loss function, the study underwent different optimisation strategies and found that Adaptive Moment Estimation (Adam), which is an improvement of the Stochastic Gradient Descent (SGD) algorithm, was the perfect selection for the current scenario (Yang et al., 2019). Therefore, it used Adam optimiser to minimise the target loss. Figure 5.5 shows the summary of the URLDet model and the full implementation of the model presented in Appendix A.1 for further reference.

5.2.2 HTMLDet

HTML is the primary language used when building web pages. As shown in the literature (R. M. Mohammad et al., 2013; Subasi et al., 2017; Pratiwi et al., 2018; Li et al., 2019; Opara, Wei, & Chen, 2020; Opara, Chen, & Wei, 2020; Feng et al., 2020), the HTML content of the web page has provided some essential features when detecting phishing attacks. However, in the initial stage of this study, there was no representation

Layer (type)	Output Shape	Param #
main_input (InputLayer)	[(None, 150)]	0
embedding (Embedding)	(None, 150, 256)	25600
dropout (Dropout)	(None, 150, 256)	0
conv1d (Conv1D)	(None, 150, 256)	196864
max_pooling1d (MaxPooling1D)	(None, 37, 256)	0
dropout_1 (Dropout)	(None, 37, 256)	0
lstm (LSTM)	(None, 37, 32)	36992
dropout_2 (Dropout)	(None, 37, 32)	0
lstm_1 (LSTM)	(None, 32)	8320
dropout_3 (Dropout)	(None, 32)	0
output (Dense)	(None, 1)	33

Total params: 267,809
Trainable params: 267,809
Non-trainable params: 0

Figure 5.5: The URLDet model summary

learning approach[11] to extract significant HTML features automatically. Although the representation learning was not practised, manual feature extraction was highly used to collect HTML features since it was an essential source when detecting phishing attacks (R. M. Mohammad et al., 2013; Subasi et al., 2017; Pratiwi et al., 2018; Li et al., 2019).

However, when it comes to representation learning, any input needs to be constructed in a machine-understandable way to perform some task (Wang et al., 2019). Therefore, it was a challenge in the initial stage of the study since there was no clue on how to convert an HTML page into a machine-understandable way. Thus, the study

[11] Opara, Wei, and Chen (2020); Opara, Chen, and Wei (2020); Feng et al. (2020) came in latter part of the study.

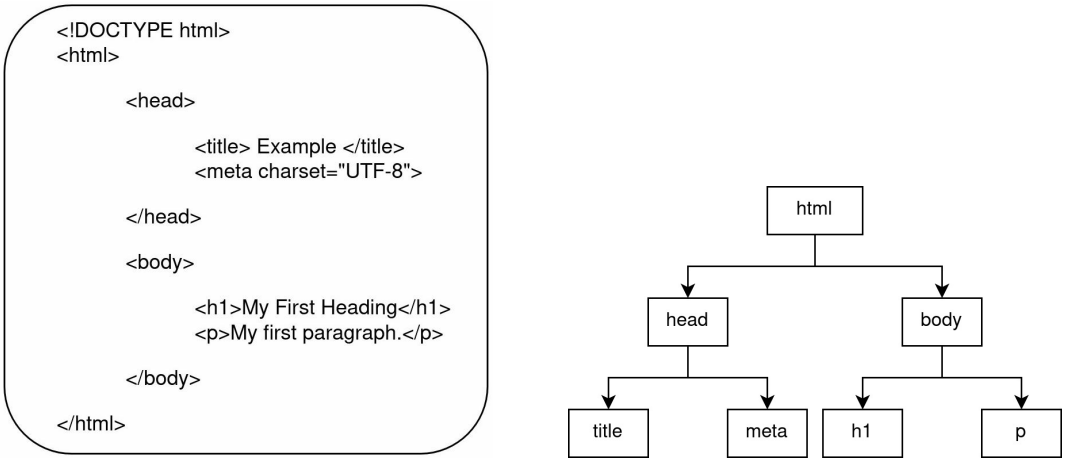
first considered doing a thorough literature analysis of different deep learning architectures used in different studies. Then, this study's interest was focused on an interesting novel architecture called GCN (Kipf & Welling, 2016). GCN is a compelling neural network architecture that operates on graphs (Kipf & Welling, 2016). In general, graph (G) is a pair of nodes (V) and edges (E) which is denoted as G(V, E) (Scarselli et al., 2009). A GCN takes an input feature matrix (X) and graph structure (A). The input feature matrix is an N $\times$ D matrix, where N is the number of nodes, and D is the number of input features for each node (Kipf & Welling, 2016). Similarly, the graph structure is an N $\times$ N adjacency matrix representation (Kipf & Welling, 2016).

Further, Bianchi et al. (2021) proposed a new filtering mechanism over the traditional polynomial filters called Auto-Regressive Moving Average (ARMA) filters to improve the GCN architecture. The experiment shows that ARMA achieves higher mean accuracy and lower standard deviation than the traditional GCN approach in graph classification (Bianchi et al., 2021). Furthermore, Bianchi et al. defined a Graph Convolutional Skip (GCS) layer in that work and proposed GCS with ARMA filters for better performance in the GCN architecture.

Technically, the HTML content of a web page contains HTML tags called elements, and elements have been attributed to providing additional information about the element. Further, an attribute usually comes in name/value pair like name = 'value'. As shown in Figure 5.6, the HTML content has a tree structure, and the study identified that the HTML tree structure could be used to generate a graph. Since the HTML content can be extended to a graph structure and GCN architecture is effective with graphs, the study selected GCN as the deep learning architecture to process the HTML content. However, it could not be done directly. Therefore, as shown in Figure 5.7, the HTML content also had to go through three primary parts: preprocessing, feature extraction, and classification.

5.2.2.1 Preprocessing

GCN architecture requires two inputs: adjacency matrix (A) to describe graphs structure and feature matrix (X) to describe node features. The HTML content was first



(a) Sample HTML code (b) Tree view of the given HTML code

Figure 5.6: Example of an HTML page in a tree view

converted into a graph in an adjacency and feature matrix. However, the study could not find any systematic way to convert HTML content directly into a graph. Therefore, it had to decide a way to do it. In the process of doing this, initially, the HTML elements were selected as graph's nodes, and the attributes of those elements became the node features. However, the HTML tag did not carry an equal number of attributes. As a result, the feature matrix was in different sizes, making it unacceptable in GCN architecture since GCN requires the same set of features for all nodes (Kipf & Welling, 2016). Thus, the study had to reconsider the graph generation process.

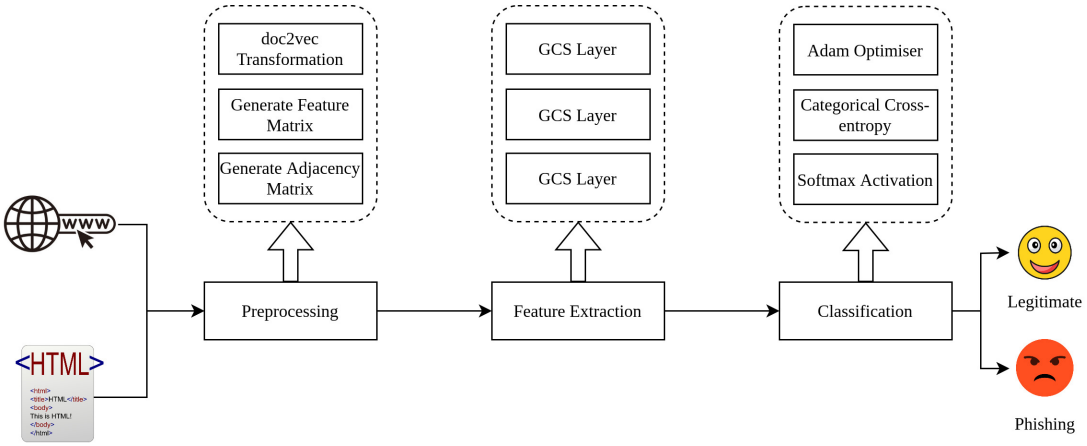


Figure 5.7: Workflow of the HTMLDet model

In the second approach, the study selected HTML elements and those elements'

attributes as the graph's nodes, and the node features were node labels, which means tag or attribute name, and values, which means the value of the tag or attribute. The HTML elements do not contain any values; those contain only the labels. However, the attributes usually have labels and values. In the second approach, the feature matrix or the X became N $\times$ 2 matrices since each node has only two features: label and value. Then the problem of having different feature matrix sizes was solved with the second approach. Therefore, the study selected it to extract nodes and features to construct the graph.

The graph was generated based on Algorithm 2 in the preprocessing step. It gets two inputs: the URL of the website and the HTML content, and outputs adjacency matrix and feature matrix. The algorithm first selects the graph nodes by selecting the elements and attributes available on the page. Then, the node's features are filled with element or attribute names and any value associated with the attribute. However, the element node's value feature is always empty since it has only the name and does not carry any value. Then, based on the parent-child relationship, the graph's edges are decided hierarchically. For example, consider the following HTML code example to understand how Algorithm 2 process this code to construct a graph.

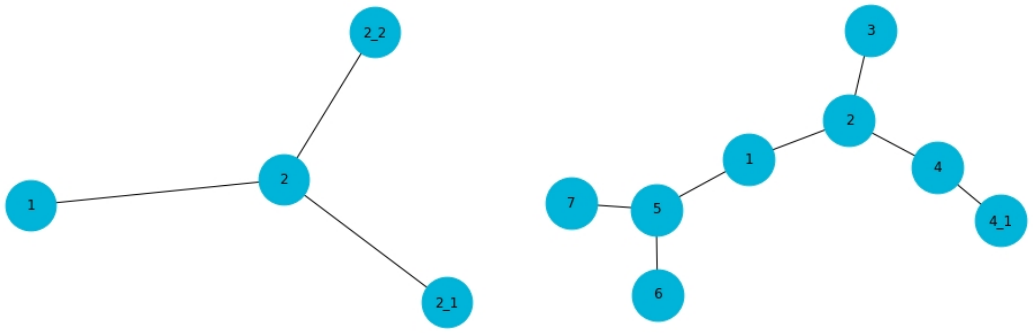
```
<p>
    <img src = 'a.jpg' alt = 'example-image'/>
</p>
```

In the above example, the HTML element, 'p' and 'img', and 'img' element attributes, 'src' and 'alt', are considered nodes. Therefore, the example has four nodes: p, img, src, and alt. First, the Algorithm label these nodes with a unique identification number. It starts from the top element and hierarchically goes to the bottom and gets an integer value starting from one. The attributes of an element also get an integer value starting from one, but it has a specific format: first, element label, then underscore character, and finally, the relevant integer value. For example, consider 'img' element 'src' attribute. The algorithm label it as 2_1 since two is the 'img' element identification number, and one is the identification number of the 'src' attribute. Therefore, in the above example, nodes get 1, 2, 2_1 and 2_2, respectively. After the labelling

Algorithm 2 Graph generation procedure

```
1:  procedure GRAPHGENERATION(url, HTMLContent)
2:      node ← integer
3:      node_from ← integer
4:      attr_node ← integer
5:      nodes_list ← list object
6:      attr_list ← list object
7:      name ← string
8:      value ← string
9:      separator ← string
10: initialisation:
11:     node ← 0
12:     name ← label
13:     value ← value
14:     separator ← _
15: extraction:
16:     node_from ← node
17:     node++
18:     for element in HTMLContent do
19:         attr_node ← 1
20:         nodes_list ← tuple(node_from, node)
21:         attr_list ← concatenate(node, name, element_name)
22:         attr_list ← concatenate(node, value, empty_string)
23:         for attribute in element do
24:             attr_node_label ← concatenate(node, separator, attr_node)
25:             nodes_list ← tuple(node, attr_node_label)
26:             attr_list ← concatenate(attr_node_label, name, attribute_name)
27:             attr_list ← concatenate(attr_node_label, value, attribute_value)
28:             attr_node++
29:         goto extraction.
30:     nodes_list remove o^th element
31:     attribute_list remove 1^st element
32:     attribute_list insert 1^st element ← concatenate(1, value, url)
33: generation:
34:     graph ← node_list
35:     adjacency_matrix ← graph
36:     attr_list ← doc2vec(attribute_name, attribute_value in attr_list)
37:     feature_matrix ← attr_list
38: return adjacency_matrix, feature_matrix
```

step, the edges list is generated based on the relationship of those two nodes. Here, the edges list is (1, 2), (2, 2_1), and (2, 2_2). Then, these nodes and edges construct an adjacency matrix. Figure 5.8 presents two graph representations that came through Algorithm 2. Graph A is relevant to the given example HTML code, and graph B shows the constructed graph for Figure 5.6 HTML code through Algorithm 2.



(a) Graph A (b) Graph B

Figure 5.8: Example graphs constructed from the graph construction process

The subsequent output that Algorithm 2 produced was the feature matrix. It is also generated simultaneously to the edges list. However, the feature values were converted to a machine-understand format using the doc2vec transformation technique (Q. Le & Mikolov, 2014) when generating the feature matrix. A domain-specific doc2vec model was first constructed before the experiment using the classic dataset web pages corpus, and the trained doc2vec model was used to transform feature values into a vector format. Table 5.1 shows the feature matrix of the above example. It shows values in a human-readable format; however, values are in vector format in actual execution.

Further, as additional information, the study decided to use the URL of the website under the 'html' element by introducing a feature called 'url'. It is not a common attribute that comes with 'html' elements but was intentionally added because some studies have shown that the URL of the website has a direct relationship with several html elements (i.e., a, img, script, and link) in a webpage (Jain & Gupta, 2018a; Chiew et al., 2019). Therefore, as supporting information, the study decided to add the URL of the website for the first node of each graph generated through Algorithm 2.

Table 5.1: Input X values

Node / Feature	Label	Value[a]
1	p	
2	img	
2_1	src	a.jpg
2_2	alt	example_image

[a]The values are transformed to a vector format in real execution using doc2vec.

5.2.2.2 Feature extraction

The feature extraction process of the HTMLDet model consisted of an input layer and three GCS layers with pooling layers. The reason to use GCS layers was the Bianchi et al. (2021) study. They experimentally showed that GCS with ARMA filters is better in terms of performance in GCN architecture. However, the model required three inputs with GCS layers: the adjacency matrix, the feature matrix, and segment ids. Then the requirement of three tensors was originated to capture those inputs. Mainly, it wanted a sparse tensor to capture the adjacency matrix. However, the preprocessing step could only generate two inputs: adjacency matrix and feature matrix. Therefore, the third input, segment ids, was needed before the input layer. However, segment ids depend on adjacency and feature matrix based on Spektral documentation. Thus, this study used Spektral inbuilt method to generate the third input.

Like in URLDet, the HTMLDet also used an evolution approach when building the GCN architecture. Therefore, a different number of GCS layers with different hyper-parameters were exercised. Then, Figure 5.9 architecture was selected based on the performance. However, all combinations were not experimented with during the study due to time constraints, and the best structure and hyper-parameter values were selected based on performed experiments. All GCS layers consisted of 32 channels in the selected structure and used ReLU as the activation function. Further, all the GCS layers used the L2 regulariser function with the value of 1e-3 and ‘he uniform’ initialiser with ‘zeros’ bias.

Similarly, all the MinCutPool layers also used ReLU as the activation function, L2 as the regulariser function with the value of 1e-3 and ‘he uniform’ initialiser with ‘zeros’ bias as kernel and bias initialisers. Further, the k value of the MinCutPool layer

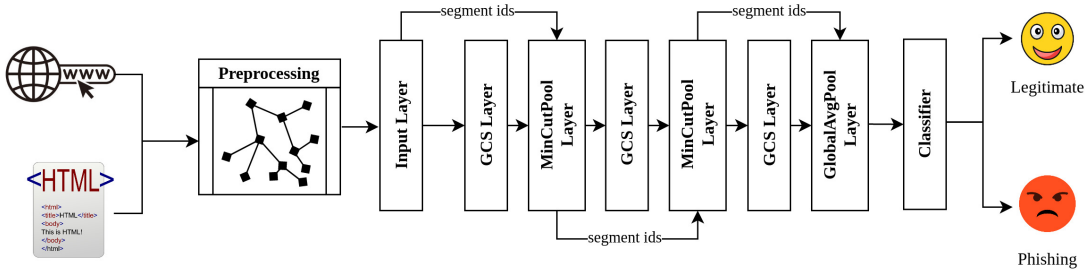


Figure 5.9: The HTMLDet architecture

was decided using the number of average nodes. However, the last layer, GlobalAvgPool, was used by the study with the default settings since the layer generates a feature map for the classification task.

First, the three inputs were passed to the input layer during the feature extraction process. Then, the input layer converted those into tensors, and two of those, the adjacency matrix and feature matrix, were inputted to the first GCS layer to carry out graph nodes' feature representation. Then, the output of GCS was passed to the MinCutPool layer. The MinCutPool layer was responsible for constructing the aggregated features. Therefore, initial segment ids were passed to the MinCutPool to construct the aggregated features with relevant adjacency and segment ids.

Then the aggregated features were outputted by the first MinCutPool layer and were passed to the second GCS layer for more abstract feature representation. The GCS and MinCutPool processed the node features in the same way as the previous and outputted relevant feature matrix, adjacency matrix, and segment ids. Then the second-level feature matrix was again passed to the third GCS layer for a more deep feature representation. After the third GCS layer completed the feature extraction process, its output was inputted to a GlobalAvgPool layer and generated one feature map for the classification task.

5.2.2.3 Classification

Once the feature map was ready, the next task was to classify the input based on the constructed feature map. It was done through a dense layer with a softmax activation function. The study used the softmax activation function since it had shown bet-

ter performance than the sigmoid activation function in several experiments done at the design stage of the model. The study selected the categorical cross-entropy loss function since softmax was used as the activation function to adjust the network's weights. However, when selecting the optimising strategy for the target loss function, the study followed the same criteria explained under URLDet model implementation (Section 5.2.1.3) and selected Adam optimiser to minimise the target loss. Figure 5.10 shows the summary of the HTMLDet model and the full implementation of the model presented in Appendix A.2 for further reference.

Layer (type)	Output Shape	Param #	Connected to
X_in (InputLayer)	[(None, 2)]	0	
input_1 (InputLayer)	[(None, None)]	0	
graph_conv_skip (GraphConvSkip)	(None, 32)	160	X_in[0][0] input_1[0][0]
segment_ids_in (InputLayer)	[(None,)]	0	
min_cut_pool (MinCutPool)	[(582, 32), (582, 58	19206	graph_conv_skip[0][0] input_1[0][0] segment_ids_in[0][0]
graph_conv_skip_1 (GraphConvSki	(582, 32)	2080	min_cut_pool[0][0] min_cut_pool[0][1]
min_cut_pool_1 (MinCutPool)	[(291, 32), (291, 29	9603	graph_conv_skip_1[0][0] min_cut_pool[0][1] min_cut_pool[0][2]
graph_conv_skip_2 (GraphConvSki	(291, 32)	2080	min_cut_pool_1[0][0] min_cut_pool_1[0][1]
global_avg_pool (GlobalAvgPool)	(None, 32)	0	graph_conv_skip_2[0][0] min_cut_pool_1[0][2]
dense (Dense)	(None, 2)	66	global_avg_pool[0][0]

Total params: 33,195
Trainable params: 33,195
Non-trainable params: 0

Figure 5.10: The HTMLDet model summary

5.2.3 Deep learning model (DLM)

The primary task of the DLM was to classify whether a given website is phishing or not based on the URL and HTML content of the website. After constructing the URLDet and HTMLDet models, the next challenge was combining those outputs to make a collective decision. In that task, the study followed the concept of transfer learning

that aims to improve the performance of the final model. During this process, the two models, URLDet and HTMLDet, were separately trained and saved (*see* Section 5.4). Then, the classification parts of the models were removed, and both feature vectors were concatenated to form the final feature vector $y_c^k(x)$, as in equation 5.1. It was a 64-length vector since both URLDet and HTMLDet feature vectors were 32-length.

$$y_c(x) = Concat(y_c^k(x))\ k\ \varepsilon\ \{1,2\} \tag{5.1}$$

Then a dense layer with two neurons was added as the output layer of DLM, and the softmax activation was enabled to distinguish between phishing and legitimate websites. Further, DLM used Adam optimiser with a categorical cross-entropy loss function to optimise the difference between actual and predicted outputs. As shown in Figure 5.11, the final model was implemented to get URL and HTML content as inputs and output phishing and legitimate probability bypassing the inputs through the URLDet and HTMLDet models. Figure 5.12 shows the plotted DLM model graph and the full implementation of the model presented in Appendix A.3 for further reference.

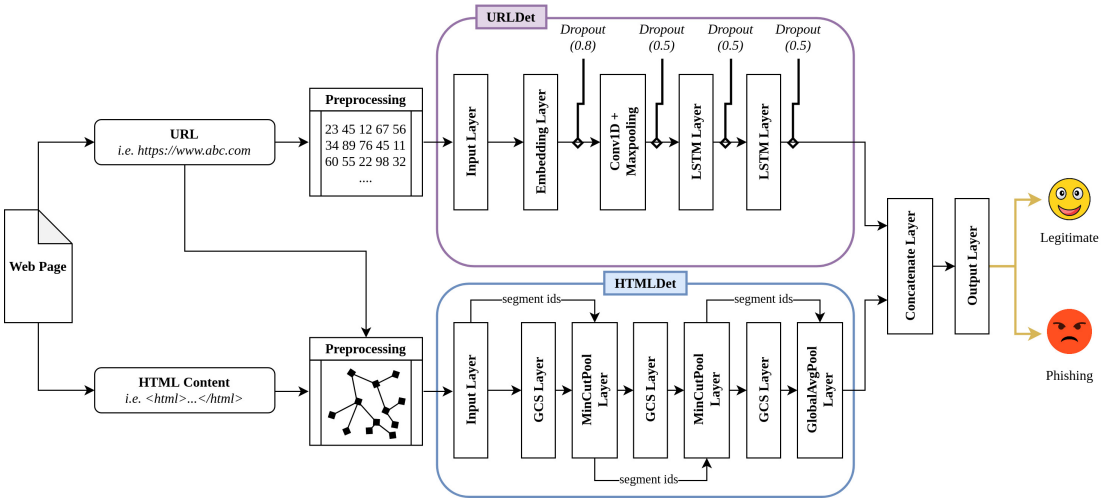


Figure 5.11: The DLM architecture

5.2.4 Hybrid DLM

The purpose of the Hybrid DLM was to appraise the proposed DLM building logic mentioned above, and some of the design considerations exist in the literature (i.e., the

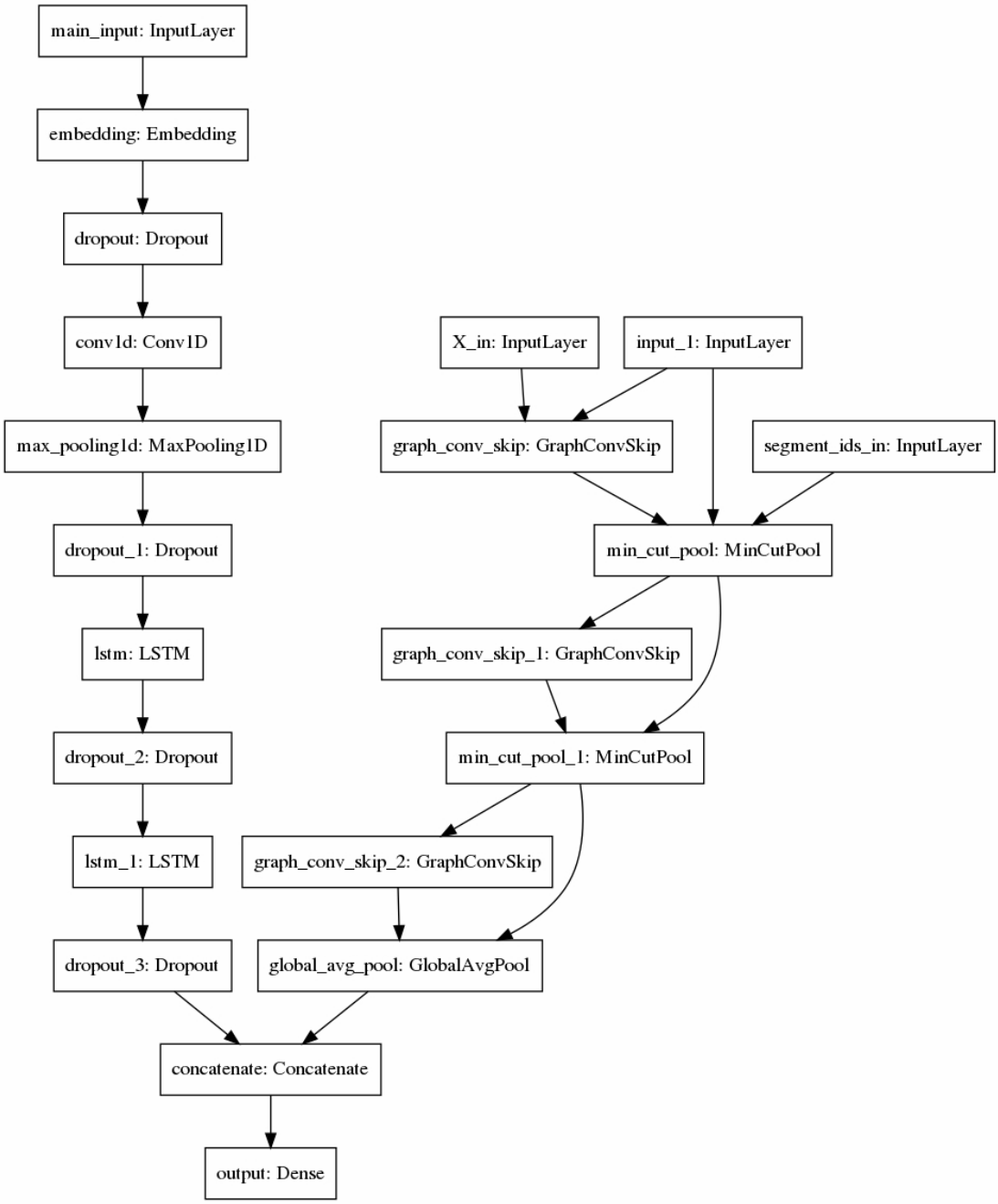


Figure 5.12: A plot of DLM model graph

use of LRCN architecture in malicious URL detection) with the classic dataset. Since a representation learning-based HTMLDet model has not existed in the literature, the study first planned to implement a phishing detection solution with the existing model-building knowledge. It confirmed that the proposed approach was good from two sides. First, the experiment results confirmed that the classic dataset not being used before

was compatible with phishing detection tasks. Secondly, it confirmed that the proposed primary logic and some design considerations were reasonable within a short time. For example, with the Hybrid DLM, this study would check whether the LSTM or LRCN works well in malicious URL detection. In literature, Pham et al. (2018) mentioned that LRCN is a better approach. So, with the support of Hybrid DLM, this study proved that Pham et al. argument is correct with evidence.

Further, the logic used to combine the outputs of URLDet and HTMLDet models to produce a combined decision was also evaluated with Hybrid DLM. Since Hybrid DLM performed well during the experiments (*see* Table 5.4), this study concluded that the proposed logic was reasonable when detecting phishing attacks. Therefore, Hybrid DLM played an essential role during the study by acting as a prototype solution to validate some of the used design aspects.

When building Hybrid DLM, the URLDet model used the same deep learning architecture proposed during the design of DLM (Section 5.2.3). However, the Hybrid DLM's URLDet model used only one LSTM layer, and when converting URLs into the numeric format, it used Python's printable class in the string package instead of Keras's inbuilt tokeniser due to its simplicity. Figure 5.13 shows the architecture of the Hybrid DLM's URLDet model, and Appendix A.4 presents the model implementation.

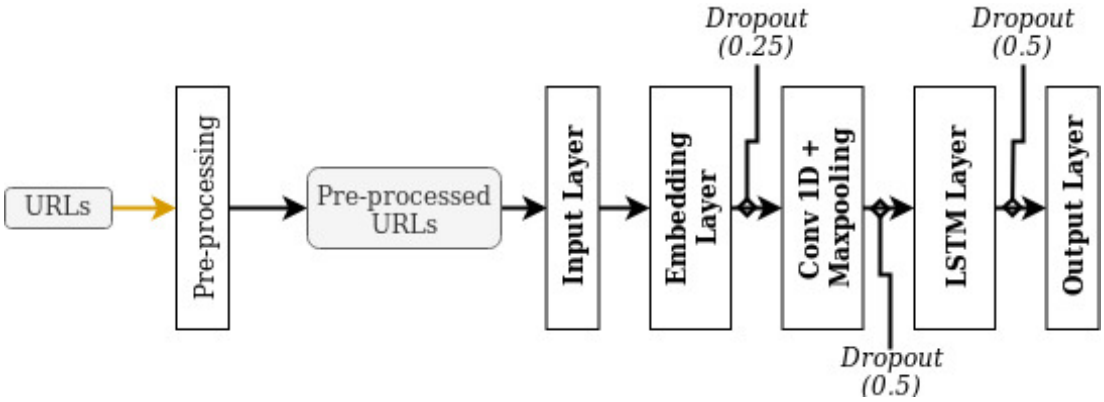


Figure 5.13: The URLDet architecture of the Hybrid DLM

However, the HTMLDet model in Hybrid DLM was a manual feature extraction-based deep learning architecture since there were no representation learning-based feature extraction solutions. Therefore, fifteen features were manually extracted and

passed to a 1D convolutional network. All these features were already used in the previous solutions. The study selected those based on the importance and relevancy by referring to the relevant literature. The features are as follows:

1. **Number of hyperlinks** (Jain & Gupta, 2016): Number of 'href' attributes relevant to <a> in a web page.

2. **Number of null pointers** (Gu et al., 2013; Jain & Gupta, 2016): Number of 'href' attributes with the value empty or '#' on a web page.

3. **External link ratio** (Gu et al., 2013; Jain & Gupta, 2016; Li et al., 2019): Ratio between the total number of available hyperlinks and external links.

4. **Personal data forms** (Li et al., 2019; Gupta et al., 2016): Binary value was used to check whether a <form> tag with one or more <input> child tags is available on a page.

5. **Length of the HTML page** (Li et al., 2019): HTML code taken as a string and its length calculated.

6. **Number of <script> tags** (Li et al., 2019): The number of <script> tags used in the web page.

7. **Number of <link> tags** (Li et al., 2019): The number of <link> tags used in the web page.

8. **Number of <!–> tags** (Li et al., 2019): The number of comments used on the web page.

9. **External resource ratio** (Li et al., 2019): Ratio calculated using HTML tags like <img> , <script> , and <link> .

10. **Favicon** (Chiew et al., 2019): Binary value was used to indicate whether a web page had a favicon and loaded from the same domain.

11. **Internal form ratio** (Chiew et al., 2019): Ratio between the available <form> tags and the number of form's action attribute had the same domain or relative path.

12. **Abnormal form ratio** (Chiew et al., 2019): Ratio between the available <form> tags and the number of form's action attribute contained a '#', 'about:blank' or an empty string.

13. **External form ratio** (Chiew et al., 2019): Ratio between the available <form> tags and the number of form's action attribute contained a URL from an external domain.

14. **Title tag** (Chiew et al., 2019): Binary value was used to check whether the <title> tag was used one time on the page inside the head area.

15. **Title tag and brand name** (Li et al., 2019): Binary value was used to check whether the <title> tag contained the URL brand name.

The study used a self-generated script (*see* Appendix B - HTML feature extractor) to extract the features mentioned above. Like in the original HTMLDet, Hybrid DLM's HTMLDet also used HTML content and the website URL as inputs to produce the values of these features. After extracting these features, the features were passed to a 1D convolutional network. It had an input layer, two 1D convolution layers, a pooling layer, a flatten layer, a dense layer, and an output layer. First, the inputs were converted to a floating-point value. Then, those values were shaped and passed to the convolution layers, which used the ReLU activation function. Then the pooling and flatten layers further processed the input and extracted local features. Those were then passed to a dense layer with 32 neurons and ReLU activation function. Then the output of the dense layer was fed to the output layer of the model. It was also a dense layer with one neuron. Since phishing detection is a binary classification task, the sigmoid activation function with binary cross-entropy loss function was used with Adam optimiser for the optimisation task. Further, the dropouts were used after each convolution layer, as shown in Figure 5.14, to avoid over fittings. Appendix A.5 presents the implemented Hybrid DLM's HTMLDet model.

After implementing URLDet and HTMLDet models, the outputs of both models were added to produce a combined output for a given website URL and HTML content. However, the final layer of the Hybrid DLM was a single neuron-based dense

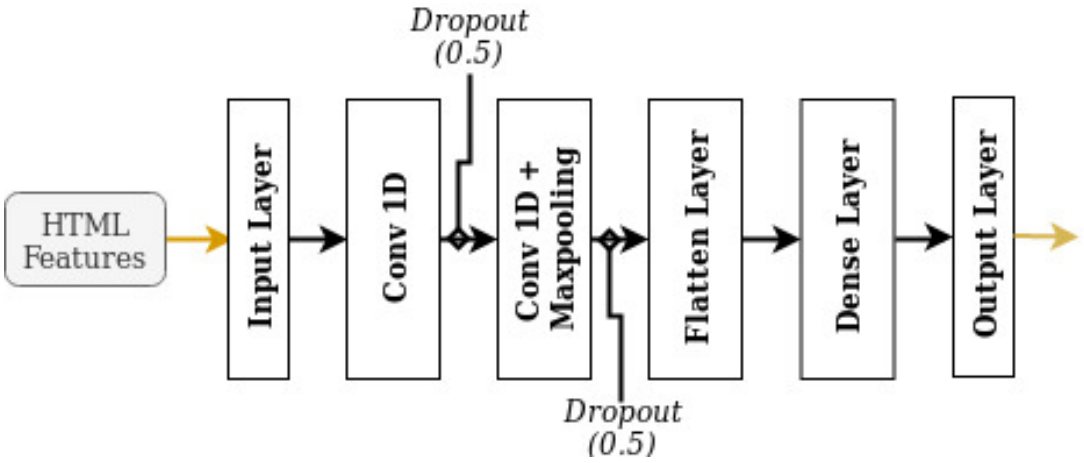


Figure 5.14: The HTMLDet architecture of the Hybrid DLM

layer with a sigmoid activation function since the output was either phishing or legitimate. It used binary cross-entropy loss function with Adam optimiser when finding the optimum mapping. Figure 5.15 shows the plotted Hybrid DLM model graph and the full implementation of the model presented in Appendix A.6 for further reference.

5.3 Data collection and preprocessing

Phishing detection was considered a classification task containing two classes: legitimate and phishing. Therefore, the data collection process included the collection of legitimate and phishing data to construct a dataset. Suppose a dataset has S amount of data points where each data point consists of three parts: website URL (u), an HTML content of the web page (w) and the label (y). A data point can be represented as u_i, w_i, y_i, where i indicates the index of the data point. Then, $y_i \; \varepsilon \; \{0, 1\}$; $y_i = 1$ corresponds to a phishing website, and $y_i = 0$ represents a legitimate website.

Data collection is a challenging task for any phishing detection solution since most phishing websites are available online only for a few hours (Zeng et al., 2020). The study had to collect URLs and the relevant web pages to support the implementation process, and it made the task more challenging since there was no central place to collect both at the same time (Sahoo et al., 2017; Aassal et al., 2020). Therefore, this study had to collect these data at different times to construct a more reliable dataset. As a result, the study used two primary self-constructed datasets. Those were named

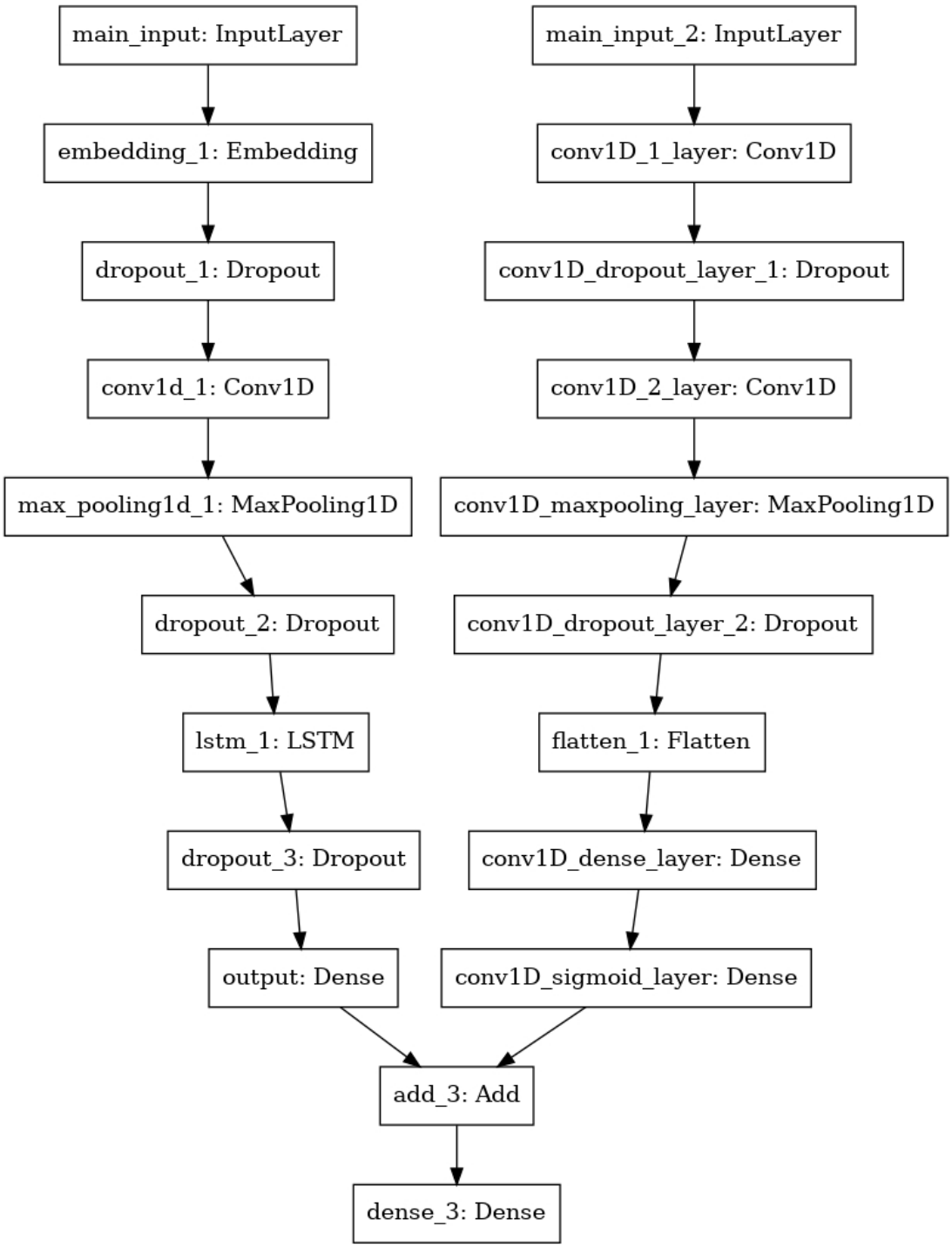


Figure 5.15: A plot of Hybrid DLM model graph

classic dataset and modern dataset. The classic dataset contained old data collected before September 2019, and the modern had the latest data collected from December 2020 to October 2021. Other than these two, the study had used a separate dataset to

compare the performance of the implemented solution with a well-performed similar solution in the state of arts. It was named a benchmark dataset and collected from Feng et al. (2020) study recently done in the anti-phishing domain that shows a reasonably high detection accuracy compared to existing phishing detection solutions. These three datasets were used in different scales during the experiments to have the training, testing and validation tasks, and Table 5.2 presents the details of those datasets in each task. After the datasets were collected, those datasets were preprocessed according to the procedures mentioned in Section 5.2.1.1 and 5.2.2.1 to generate inputs for URLDet and HTMLDet models.

Table 5.2: Details of the used datasets

Dataset	Sub-dataset	Legitimate	Phishing	**Total**
Classic [40,000]	Training Set (Tr_C)	13,895	14,105	**28,000**
	Validation Set (Val_C)	2,006	1,994	**4,000**
	Testing Set (Te_C)	4,099	3,901	**8,000**
Modern [50,000]	Training Set (Tr_M)	17,500	17,500	**35,000**
	Validation Set (Val_M)	2,500	2,500	**5,000**
	Testing Set (Te_M)	5,000	5,000	**10,000**
Benchmark [46,096]	Training Set (Tr_B)	17,360	14,906	**32,266**
	Validation Set (Val_B)	2,480	2,130	**4,610**
	Testing Set (Te_B)	4,960	4,260	**9,220**

5.3.1 Classic dataset

As the name implies, the classic dataset contained data downloaded early in the study. Therefore, all the data contained in the classic dataset were downloaded before September 2019. The classic dataset contained 40,000 data in an equal amount of phishing and legitimate. The legitimate data were downloaded with the support of Google. Since the Google search engine rank was used by several studies to select legitimate websites (Zhang et al., 2007; Dunlop et al., 2010; El-Alfy, 2017; Odeh et al., 2021), the study assumed that top-ranked Google search results belong to the legitimate category. Therefore, the study selected a set of keywords (*see* Appendix B - Google search keywords) from the Internet and passed those to the Google search engine. Once the results were received from Google, the study selected the top ten URLs. However, this

study did not select any already selected URL and always checked whether the URL domain was listed in the selected list maximum of ten times to have a diverse dataset (Aassal et al., 2020). The study used a self-generated script to perform legitimate data download tasks, and it is available in the project source repository for further reference. After the list of URLs was collected, this study fetched those URLs to download the relevant web pages. Finally, the URLs and web pages were saved appropriately under the classic dataset.

The phishing data was constructed with the support of PhishTank and a public phishing website dataset (Chiew, Chang, et al., 2018). This public dataset contained 15,000 phishing website URLs with relevant web pages, which were downloaded from March 20, 2016, to April 30, 2016, with the support of PhishTank. However, the study did not use all of those since a preprocessing step was initially carried out to have a quality dataset. In that step, first, all URLs were verified either by PhishTank or GSB. Suppose one of those systems did not validate any URLs, then those were not carried forward during the dataset construction. After the verification was done, the verified URL's webpages were again checked to remove some noisy data items like webpages with 404 error status and some different content like hosting servers' home page due to the unavailability of the requested webpage. This study got the support of the Beautiful Soup library[12], and a self-generated script (*see* Appendix B - Noisy data remover) was used during that task. After the preprocessing step, this study could collect 5,060 amounts of data from this public dataset.

However, based on the previous studies (R. M. Mohammad et al., 2013; El-Alfy, 2017; W. Chen et al., 2018; Sahingoz et al., 2019), this study intended to collect 20,000 phishing data. Therefore, to fill the remaining numbers, this study used the PhishTank phishing URL list. With the support of PhishTank API[13], this study downloaded phishing URLs and fetched those content to get the webpage. However, if the URL was not accessible, those were removed, and only accessible URLs were saved under the classic dataset with relevant webpages. Although a set of criteria were used when downloading PhishTank data, the study again checked the downloaded webpages to

[12] `https://pypi.org/project/beautifulsoup4/`
[13] `https://phishtank.org/api_info.php`

remove noisy data items, as mentioned before, to maintain the quality of the dataset. After several iterations, the study collected 20,000 phishing data to construct the classic dataset. Then, to have a balanced dataset, the study selected 20,000 legitimate data items alongside phishing data items to finalise the classic dataset.

After constructing the classic dataset, the 40,000 data items were divided randomly into three sub-datasets: training, validation, and testing. 70% of the data was used for the training, while the remaining 10% and 20% were used for validation and testing, respectively. The amounts of legitimate and phishing data available in those sub-datasets are mentioned in Table 5.2.

5.3.2 Modern dataset

Phishing attacks are constantly changing. Therefore, this study wanted to have data on the latest phishing attacks and incorporate that knowledge with the implemented model. The modern dataset was constructed with that intention, and it contained 50,000 data in an equal amount of phishing and legitimate. The legitimate data was downloaded with the support of the Ebbu2017[14] dataset and Google search engine. Ebbu2017 dataset contained legitimate and phishing URLs. However, the study considered only legitimate URLs since phishing data were old. First, the legitimate URLs were selected and fetched from the Internet. If the page was not accessible, then that URL was skipped. Otherwise, it was saved with the relevant webpage. However, the study could collect 12,731 legitimate data from the Ebbu2017 dataset. Then, the remaining data items were collected using the Google search engine following the same procedure followed during the classic dataset construction process mentioned in Section 5.3.1.

Phishing data was mainly collected from PhishTank and OpenPhish. Since the modern dataset was constructed to get the latest data, two scripts were primarily implemented to download the latest phishing URLs from PhishTank (*see* Appendix B - PhishTank data extractor) and OpenPhish (*see* Appendix B - OpenPhish data extractor). The scripts were executed every hour to communicate with PhishTank and Open-

[14] `https://github.com/ebubekirbbr/pdd/tree/master/input`

Phish APIs to download the latest phishing URLs to minimise 404 errors experienced with phishing pages due to their short-lived nature. The modern dataset construction was done from December 1, 2020, to October 31, 2021.

After constructing the modern dataset, it was mainly split into two sub-datasets as training and testing in a 4:1 ratio, and 10% of the training data were used for the validation task during the model training. The experiments used a balanced dataset during the training process. Therefore, equally legitimate and phishing data were listed under all sub-datasets of the modern dataset, as displayed in Table 5.2.

5.3.3 Benchmark dataset

The purpose of the benchmark dataset was to compare the model's performance with an anti-phishing solution introduced by Feng et al. (2020) that has shown some interesting performance in phishing detection during the study's period. Therefore, the benchmark dataset was constructed using a publically shared dataset[15] that was used by Feng et al. (2020) to evaluate their model. The benchmark dataset contained 24,800 legitimate data items and 21,296 phishing data items. Since the Feng et al. dataset had URLs and relevant web pages, there was no need to fetch the web pages separately. However, seven phishing pages had some issues when extracting the downloaded dataset. Therefore, the benchmark dataset from the original dataset excludes those seven data items. In the Feng et al. dataset, the phishing data were collected from September 2019 to November 2019. Although those data were old compared to the study's modern dataset, it was used as the benchmark dataset since a similar solution that performed well over others had used this dataset. Hence, to have an accurate comparison, this dataset was required. Further, during the experiments, the benchmark dataset was split into three sub-datasets: training, validation and testing, the same as the modern dataset. The amounts of legitimate and phishing data available in those sub-datasets are mentioned in Table 5.2.

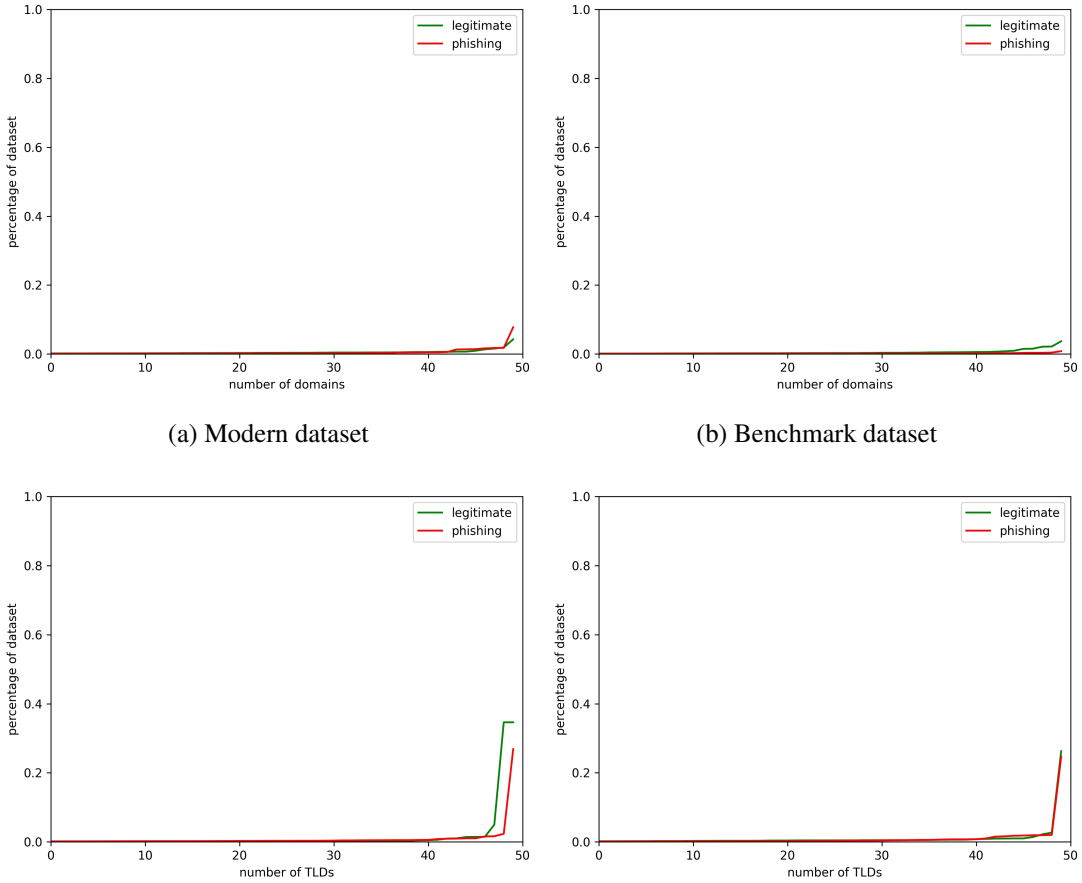
[15] `https://github.com/Hanjingzhou/Web2vec`

5.3.4 Diversity of datasets

Out of those three datasets, the classic dataset was used only in the initial stage of the study to train the URLDet model, HTMLDet model and DLM. The performance of the proposed model and the benchmark task were done with modern and benchmark datasets. Therefore, those datasets were analysed to check for their diversity before being used for performance evaluation or benchmark tasks. Aassal et al. (2020) mentioned that the diverse dataset in an anti-phishing study is essential for a more generalised trained model. However, Aassal et al. (2020) mentioned that there is no widely accepted method to check the diversity of a dataset, and they have used the number of different domains and the number of different TLDs when checking the diversity of their dataset.

The current study used two distribution graphs, URL length and secure and non-secure representation, alongside Aassal et al. (2020) criteria to analyse the diversity of the used datasets. Those two distribution graphs were used because of two reasons. First, the literature has shown that HTTPS in phishing attacks has an increasing trend (APWG, 2021a). Therefore, the study wanted to check whether the collected datasets presented this trend to confirm the reliability of the dataset. Next, Verma et al. (2019) mentioned that the URL length distribution is essential to have unbiased, accurate model training since if a feature like URL length is not well distributed, it may become a prominent feature and leads to a biased model at the end. Therefore, the study used the number of different domains, the number of different TLDs, the distribution of HTTPS, and the distribution of URL length when analysing the diversity of modern and benchmark datasets.

When analysing the domains and TLDs distribution, this study inherited Aassal et al. (2020) procedure. First, a list of unique domains and TLDs were collected separately from each dataset and used in calculating those frequencies. Then, the top fifty domains and TLDs were selected and used in calculating the percentage of those proportionally to the size of the relevant dataset. Then those were plotted to have the domains and TLDs distribution as shown in Figure 5.16. In URL length distribution analysis, the number of characters of a URL was considered the URL's length (Li et



(a) Modern dataset

(b) Benchmark dataset

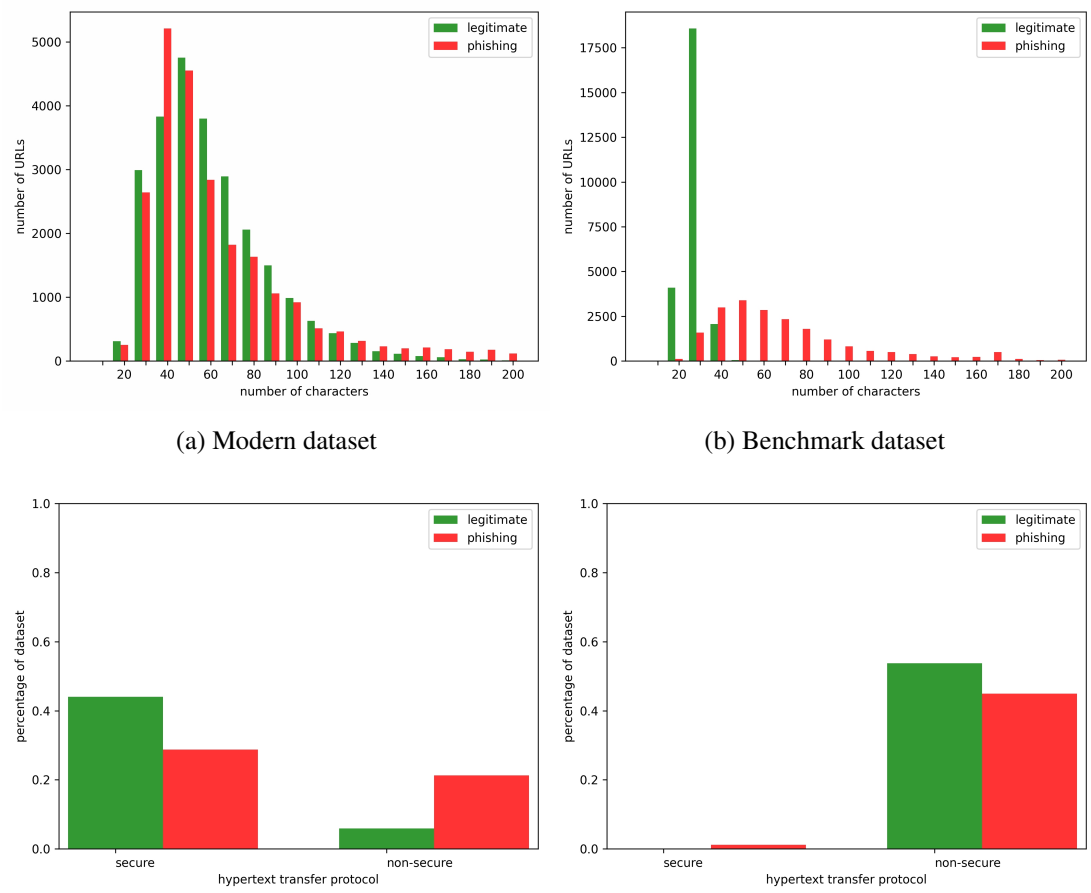
(c) Modern dataset

(d) Benchmark dataset

Figure 5.16: Distributions of domains and TLDs in the modern and benchmark datasets

al., 2019). Then the length was calculated, and the URLs were divided into several intervals to have more meaningful insight into the distribution. This study selected 20 as the interval after several intervals since it was the more descriptive and convenient presentation. Figure 5.17(a) shows the URL distribution of the two datasets. Then, the HTTPS analysis was mainly based on whether the URL used HTTPS or not. Therefore, the URLs were divided into two categories based on the HTTPS appearance of the URL. After that, the number of URLs that came under each category was counted to draw Figure 5.17(b) distribution.

The results obtained through domain and TLDs analysis show that both datasets were not biased to a specific set of domains or TLDs. However, the URL character length and HTTPS distribution (Figure 5.17) interpreted a different viewpoint of these two datasets. Figure 5.17(b) shows that the benchmark dataset's legitimate URL



(a) Modern dataset

(b) Benchmark dataset

(c) Modern dataset

(d) Benchmark dataset

Figure 5.17: Distributions of URL length and HTTPS in the modern and benchmark datasets

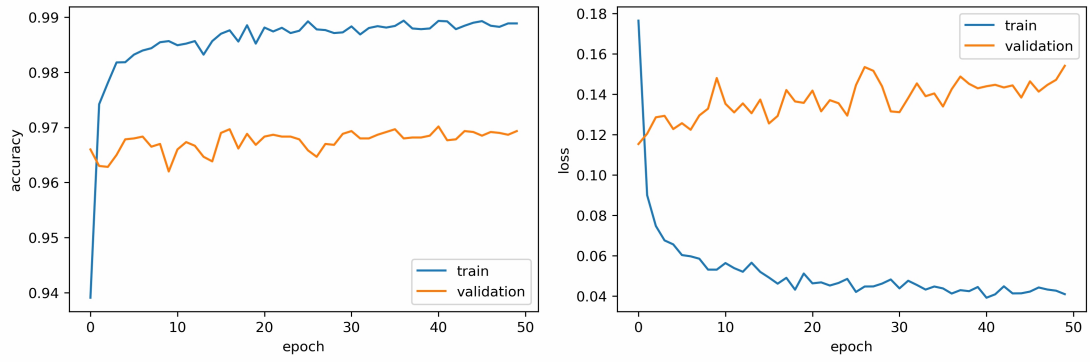
character lengths were positively skewed, and most URLs had come under the HTTP category. However, the modern dataset had well-distributed data under both these criteria. Therefore, the modern dataset was comparatively more diverse than the benchmark dataset. Thus, the study had used the benchmark dataset only for benchmark tasks, and the evaluation was done with the modern dataset since it presented the real phishing attacking nature.

5.4 Model training

As described in Section 5.2, the study mainly had two models: Hybrid DLM and DLM. Therefore, the model training section discusses these two separately since those two had different intentions.

5.4.1 Hybrid DLM training

The hybrid DLM was mainly trained with the classic dataset. However, it was trained with the modern dataset once the benchmarking task was performed in the DLM model evaluation. During the training, first, the URLDet and HTMLDet were separately trained and saved to the disk using the h5py package (*see* Appendix A.4 and 5). Then the complete solution, Hybrid DLM, was trained for 50 epochs with a 0.001 learning rate. Figure 5.18 shows the Hybrid DLM's accuracy and loss curves. However, the performance graphs indicated a performance loss of the validation dataset once the epoch exceeded ten. The study identified it as an overfitting scenario. Then, the early stopping technique was applied and trained again. The Hybrid DLM was trained and evaluated three times to avoid data biases that could exist in the selected data, and the selection was made with the support of the model selection function in Python's scikit-learn library.



(a) Accuracy curve for train and validation set (b) Loss curve for train and validation set

Figure 5.18: Hybrid DLM performance curves

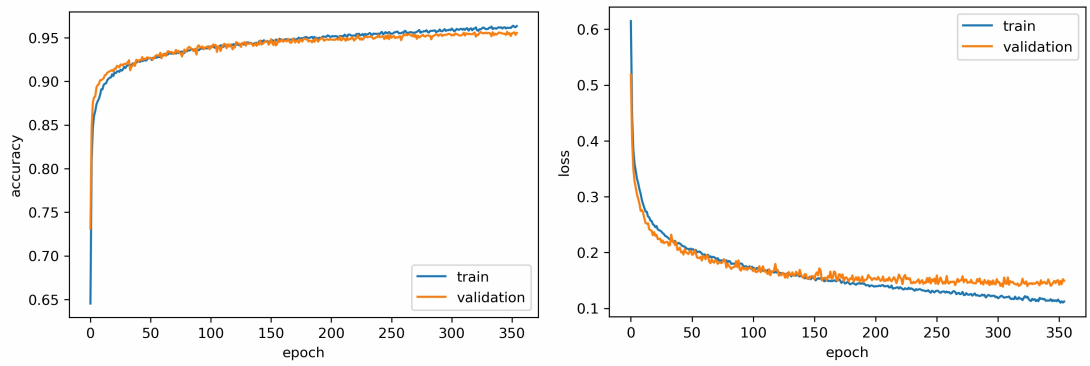
5.4.2 DLM training

The DLM model was trained in two phases. In the first phase, the model was trained with the classic dataset. Then, in the second phase, it was retrained from the modern dataset since the study wanted to have a realistic model before having the final performance evaluation of the DLM. However, due to the nature of modern and benchmark datasets, the study had retrained phase one DLM again with a benchmark dataset to

have an accurate performance comparison with Feng et al. (2020) solution.

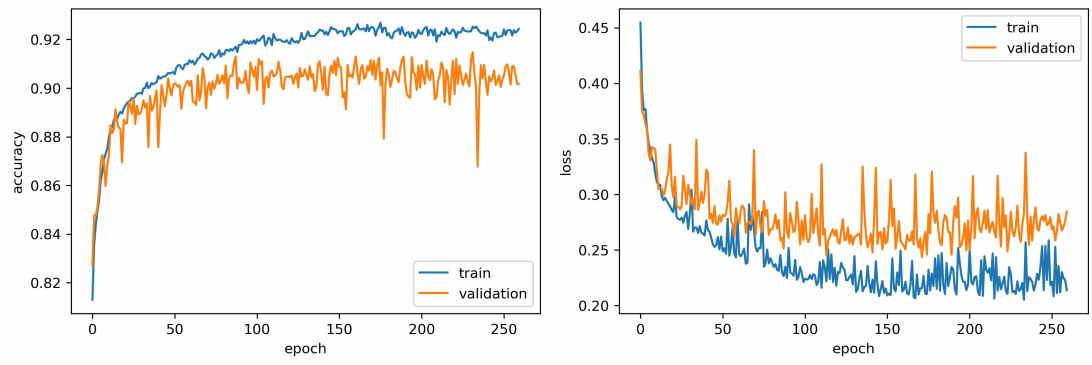
5.4.2.1 Training with the classic dataset

The modern dataset construction process took nearly a year since it collected most of the data daily. Thus, the initial level of the study used the classic dataset. Therefore, the classic dataset was used to train the sub-models: URLDet and HTMLDet and the DLM model in three separate steps. First, the URLDet model was trained with 64 batch sizes and a learning rate of 1e-4 for 500 epochs. Since the early stopping technique was used, the training process was stopped at the 355th epoch. Then, the trained model was saved and given the name 'URLDet.h5'. Figure 5.19 shows the accuracy and loss curves of URLDet for both training and validation data. Next, the HTMLDet model was trained using a batch size of one since one was a requirement in ARMA filters and a learning rate of 1e-3. Here, the early stopping technique was applied to avoid over-fitting, and the training was stopped in the 272nd epoch, as presented in Figure 5.20. The trained model was again saved to the disk with the name 'HTMLDet.h5'. Finally, DLM was trained with a batch size of 1 and a learning rate of 1e-5. In DLM training, the training process was stopped at the 79th epoch due to the applied early stopping criteria. Figure 5.21 illustrates the DLM training process's accuracy and loss curves. Training and validation data were selected based on Table 5.2 in all these steps.



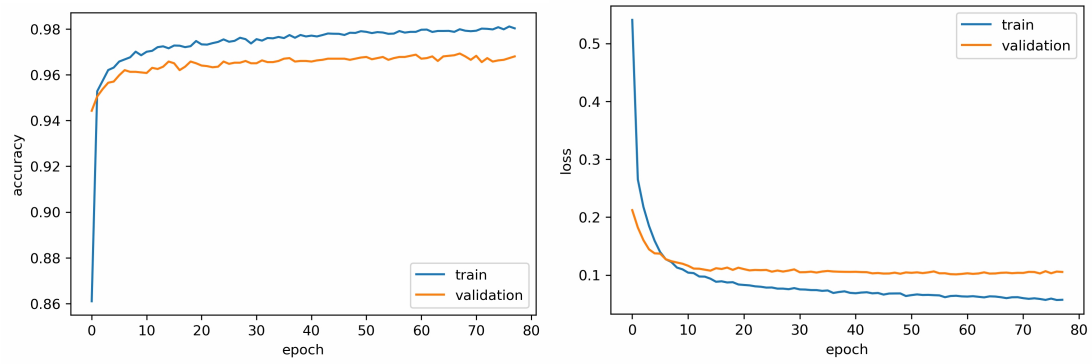
(a) Accuracy curve for train and validation set (b) Loss curve for train and validation set

Figure 5.19: URLDet performance curves



(a) Accuracy curve for train and validation set (b) Loss curve for train and validation set

Figure 5.20: HTMLDet performance curves



(a) Accuracy curve for train and validation set (b) Loss curve for train and validation set

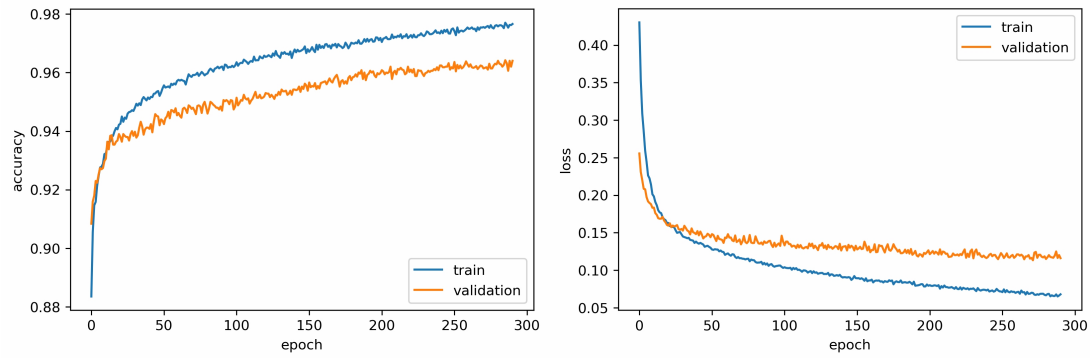
Figure 5.21: DLM performance curves in phase one training

5.4.2.2 Training with the modern dataset

After a successful phase one training, the DLM was retrained with the modern dataset in phase two to have a realistic performance evaluation process since the phishing nature is rapidly changing and the classic dataset was comparatively old. The retraining used a batch size of 1 and a learning rate of 1e-5 like in phase one and completed training within 292 epochs. Figure 5.22 shows the phase two training process's accuracy and loss curves. After phase two training, the DLM was ready for evaluation. Table 5.2 was referred to when selecting training and validation data for the retraining task.

5.4.2.3 Training with the benchmark dataset

After a successful diversity analysis stage, the study concluded that the benchmark dataset was not well diverse compared to the modern dataset. Therefore, as explained



(a) Accuracy curve for train and validation set (b) Loss curve for train and validation set

Figure 5.22: DLM performance curves in phase two training

in Section 5.4.2, the DLM that was retrained with the modern dataset was not suitable for carrying out the benchmark task since it was more biased in specific criteria. Therefore, as a solution, the phase one DLM was retrained with the benchmark dataset to have model compatibility with the benchmark dataset to have an accurate comparison. The same batch size and learning rate exercised in phase two training were used in the benchmark training phase as well. After the successful training phase, the trained DLM was used to compare its performance with the Feng et al. (2020) solution.

5.5 Model evaluation

Like in training, the evaluation process was also conducted separately for Hybrid DLM and DLM models.

5.5.1 Hybrid DLM performance

Hybrid DLM was used only to confirm whether the planned architecture effectively detects phishing attacks. Therefore, the study performed an overall performance check with the Hybrid DLM, and also it was benchmarked with different architectures to validate the effectiveness of the selected architecture. Table 5.3 shows the overall performance of the Hybrid DLM with the classic dataset, and Table 5.4 presents the detection accuracies of different architectures with a selected test dataset.

Table 5.3: Hybrid DLM performance evaluation

	Accuracy	Precision	Recall	*f1*-score
Experiment 1	98.49%	98.23%	98.75%	98.44%
Experiment 2	98.46%	98.74%	98.18%	98.41%
Experiment 3	98.07%	98.39%	97.77%	98.02%
Average	**98.34%**	**98.45%**	**98.23%**	**98.29%**

Table 5.4: The Hybrid DLM comparison with different architectures

Architecture	**Feature Set**	**Accuracy**
LSTM only	only URLs	88.67%
LSTM + 1D Conv	only URLs	96.20%
1D Conv	only 15 HTML Features	91.70%
Hybrid DLM	15 HTML Features and URLs	97.74%

5.5.2 DLM performance

The DLM was the main output of phase one of the implementation process. Therefore, it was evaluated under four main criteria: overall performance, benchmarking, zero-day attack detection, and detection time. The following sub-sections describe those in detail.

5.5.2.1 Overall performance

After the initial training of the DLM, the solution was evaluated on Te_C and Te_M datasets. Then, after phase two training was concluded, the model was re-evaluated with the Te_M dataset. Table 5.5 shows the results for the specified performance metrics during these evaluations.

Table 5.5: DLM performance evaluation

Dataset	Phase	Accuracy	Precision	Recall	*f1*-score
Te_C	Phase 1	96.64%	97.59%	95.80%	96.69%
Te_M	Phase 1	87.29%	86.35%	88.58%	87.45%
	Phase 2	**96.42%**	**96.40%**	**96.44%**	**96.42%**

5.5.2.2 Benchmarking

DLM was benchmarked in two ways. First, it was used with three benchmark models: URLDet, HTMLDet, and Hybrid DLM. Then, it was evaluated with the benchmark dataset to compare the model's performance with a similar solution that outperformed all state of art solutions.

All the benchmark models were initially trained with the modern dataset in the first experiment. Then, the performance was evaluated using the Te_M dataset. Table 5.6 shows the obtained results of the experiment. In the second experiment, the phase one DLM was trained on the benchmark dataset and evaluated the before and after performance of the DLM using the Te_B dataset. Table 5.7 presents the results achieved by the second experiment.

Table 5.6: The DLM comparison with selected phishing detection models

Model	Feature Set(s)	Accuracy	Precision	Recall	*f1*-score
URLDet	URL Only	94.81%	95.57%	93.98%	94.77%
HTMLDet	HTML Only	89.87%	91.18%	88.28%	89.71%
Hybrid DLM	URL + HTML*	96.95%	96.98%	96.28%	96.51%
DLM	URL + HTML	96.42%	96.40%	96.44%	96.42%

*Manually extracted HTML features

Table 5.7: The DLM performance evaluation with the benchmark dataset

	Accuracy	Precision	Recall	*f1*-score
Before Train with Tr_B	86.65%	94.58%	75.42%	83.92%
After Train with Tr_B	**99.57**%	**99.41**%	**99.65**%	**99.53**%

5.5.2.3 Zero-day attack detection

A zero-day attack, essentially a new or not yet reported attack detection, is a vital feature to be considered in any anti-phishing solution. Therefore, DLM was evaluated against zero-day attacks through a specific testing procedure exercised in previous studies (Thakur & Verma, 2014; Butnaru et al., 2021) with the support of a famous GSB blacklist.

During the experiment, the PhishTank was continuously monitored through a self-generated script (*see* Appendix B - PhishTank data extractor). The script was executed every hour and collected verified phishing URLs submitted in the last hour. Then, the collected URLs were submitted to DLM and GSB API simultaneously. Once the results of those two were available, those were separately recorded. After a while, the second attempt was carried out using the URLs that were not marked as phishing by Google during the first attempt. However, the study recorded a more than 24 hours delay between the first and second attempts since 47%-83% of phishing URLs are added to the blacklists after 12 hours (Khonji et al., 2013). Then, if a submitted URL did not receive a phishing flag from Google in the first attempt but received it in the second attempt, it was considered a zero-day attack. Finally, the correct decisions made by DLM over the zero-day attacks were calculated to get a conclusion about DLM's zero-day attack detection ability. The experiment was done for three days, and the results can be seen in Table 5.8.

Table 5.8: Results of the zero-day attack detection experiment

1st Attempt Date*	2nd Attempt Date	f_1	f_2	f_3	f_4	f_5
20-11-2021	22-11-2021	53	49	16	**04**	**04**
21-11-2021	23-11-2021	217	188	101	**18**	**17**
22-11-2021	24-11-2021	135	128	85	**06**	**06**

*URLs submitted to the PhishTank on this date were only used

f_1 - Number of phishing websites downloaded from PhishTank

f_2 - Number of phishing websites correctly detected by the DLM

f_3 - Number of phishing websites correctly detected by Google Safe Browsing API

f_4 - Number of zero-day attacks recorded by the experiment

f_5 - Number of zero-day attacks correctly detected by the DLM

5.5.2.4 Detection time

The average detection time for a given website was calculated using 3,000 randomly selected data from the Te_M dataset. The achieved results have shown that DLM takes 1.8 seconds average time when responding to the given requests, as shown in Figure 5.23.

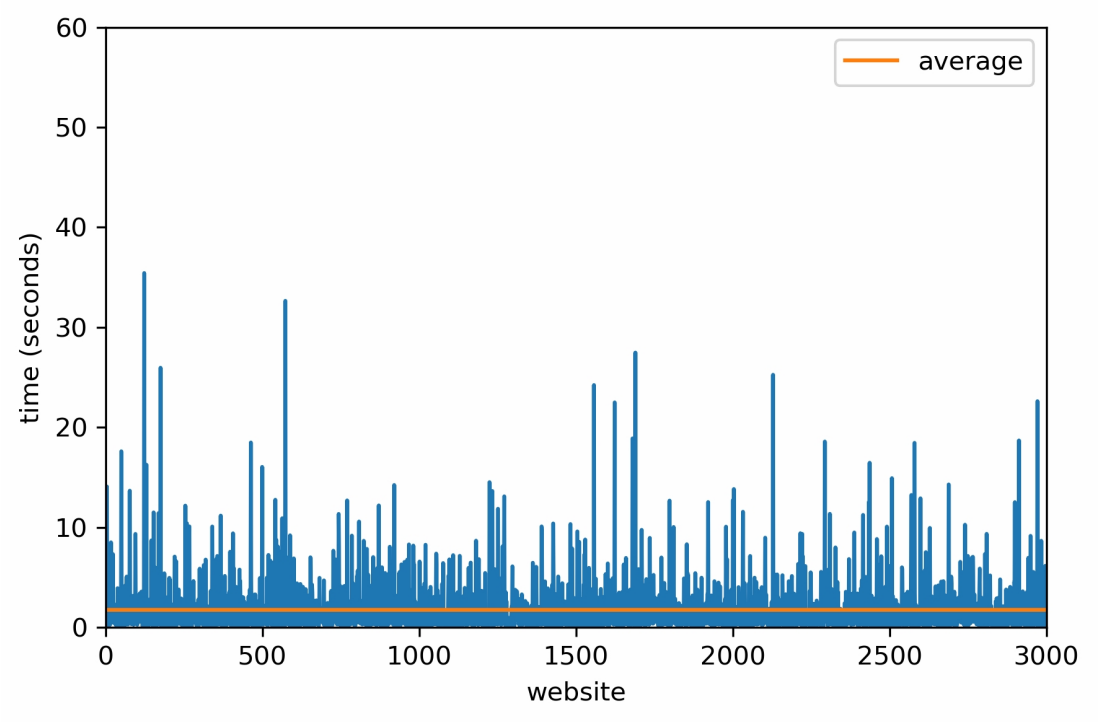


Figure 5.23: DLM detection time curve

5.6 Results and discussion

As mentioned in the design stage, the two models: Hybrid DLM and DLM, were evaluated for different purposes. The Hybrid DLM acted as the prototype model to evaluate the proposed primary architecture of the solution and validate the classic dataset's compatibility when training a deep learning network. Table 5.3 showed 98.34% average detection accuracy in the Hybrid DLM. It indicates that extracting URL and HTML features separately and combining the significant features to make the final decision works well when detecting phishing attacks. Table 5.4 illustrates it more precisely by ranking the proposed architecture well over the other architectures on the same dataset. As shown in Table 5.4, URLDet and HTMLDet achieved 96.20% and 91.70% detection accuracy, respectively, and after combining those in the proposed way, the result was 97.74%. Therefore, based on the Hybrid DLM results, it is clear that the proposed primary architecture of the phase one solution is effective in phishing attack detection. Further, it proved that the LRCN architecture that was suggested by the literature in malicious URL detection was superior compared to the LSTM architecture since the

LRCN achieved 7.53% more detection accuracy compared to the LSTM.

Furthermore, Section 5.5.1 experiment also proved the effectiveness of the classic dataset in deep learning tasks. Since there is no standard mechanism to check the quality and quantity of the training dataset in a successful deep network training, the study used Section 5.5.1 experiment to show its effectiveness. The experiment used three different portions of the training, testing, and validation dataset, and all the experiments achieved more than 98% detection accuracy. It illustrates that the training model always sees enough phishing and legitimate examples from the classic dataset to select significant phishing detection features to improve detection accuracy.

The DLM was introduced from Hybrid DLM as the original solution. It mainly addresses the part of the RQ1. The DLM is a brand new idea that uses robust GNN architecture. It has experimentally shown that it can detect phishing attacks with 96.42% accuracy. The DLM performs well against zero-day attacks, and it is more practical in real-time since it takes only 1.8 seconds to take a decision on a single request. Further, the DLM was benchmarked with the benchmark dataset to compare the performance with the Feng et al. (2020) solution. As shown in Table 5.7, the DLM achieved 99.57% detection accuracy on the benchmark dataset, and it is well above Feng et al. (2020) achieved accuracy.

However, the accuracy is not a good metric to compare two machine learning-based phishing detection models. The results available in Table 5.5 and 5.7 illustrates that more precisely. According to the experiment, the DLM accuracy with the modern dataset was 96.42%, but it achieved 99.57% with the benchmark dataset. According to Aassal et al. (2020), it is because of the diversity of the dataset. The study has clearly shown that the benchmark dataset is not diverse since it failed two used criteria during the diversity analysis. However, the modern dataset is diverse and reliable. It indicates that the accuracy is not a perfect metric when comparing two detection models if both are not using the same dataset. However, it also highlights the importance of sharing the used dataset with the research community once a detection model is built. Then, as this study did, the others can also use the dataset to benchmark and compare models. However, the study had identified that the benchmark dataset was biased. Therefore,

the more realistic detection accuracy of the model is considered 96.42%.

Furthermore, the FNR of the DLM was calculated at 0.036 with the support of Table 5.5. A low FNR is a good achievement of the DLM since FNR is critical in a problem like phishing since marking a phishing website as a legitimate one has a high impact in the anti-phishing domain. Further, the benchmarking results presented in Table 5.6 proved that the used approach is better in phishing detection since it outperformed both URLDet and HTMLDet models. However, the Hybrid DLM achieved marginally high detection accuracy compared to the DLM. It may be due to manually selected features in the Hybrid DLM. However, it will not be an effective solution when achieving the research aim since it requires expert support when updating significant phishing detection features.

In any phishing detection solution, zero-day attack detection is crucial. The DLM achieved it more successfully. According to the zero-day attack detection experiment, the DLM has correctly detected 27 attacks out of 28 attacks. It is proportionally 96.43%. Further, the solution's overall accuracy was 90.12% during the experiment, and it was well above the GSB API, which had only 49.88% detection accuracy. Like zero-day attacks, detection time also plays a vital role in real-time phishing detection since the high detection time always creates inconvenienced users in the end (Dou et al., 2017). The DLM was successful in that case also since it recorded only 1.8 seconds average detection time during the experiment. It is less compared to the Yang et al. (2019) study since, after several tries, they only could achieve 3.5 seconds average detection time in their solution. Therefore, in terms of the phishing detection accuracy, detection time and effectiveness against zero-day attacks, the DLM has shown good performance in real-time phishing detection.

Although the DLM is an effective anti-phishing solution, Section 5.5.2.1 experiment highlights that the detection ability of the model declined by 9.35% during a one-year time. This declining model performance over time was identified as the research problem at the initial stage of this study. At this point, the study now experimentally shows the identified problem's reality because DLM performance decreased by 9.35% within a year. However, after retraining on the modern dataset, Table 5.5 shows that

the DLM had reclaimed its accuracy. It again proves the effectiveness of the retraining process that was already proved in the literature. It also guarantees that the retraining that updates the existing phishing detection knowledge of the model could improve the model performance again to a reasonable level. Then, the remaining issues are how to accomplish this retraining process systematically and how to minimise the performance loss of this solution until the model reaches a retraining point, as retraining is a difficult task as identified in Section 3.7 due to the difficulties in collecting training data in phishing domain. The future chapters of this dissertation will discuss how this study counters these challenges to achieve the proposed research aim.

5.7 Summary

This chapter delivered a phishing detection solution named DLM with 96.42% detection accuracy and a 0.036 FN rate with the help of LRCN and GCN deep learning architectures that used URL and HTML features. However, the proposed solution achieved 99.57% detection accuracy with a benchmark dataset, moving the solution to the top of similar solutions. The DLM recorded a detection time of 1.8 seconds per web page, and the proposed solution was very effective against zero-day attacks. However, the DLM suffered from performance degradation like similar solutions. Therefore, the subsequent phases proposed in Chapter 4 are critical for successfully resolving the identified problem. As a result, the next chapter discusses phase two of the proposed methodology.

6 REINFORCEMENT LEARNING TO ENHANCE PHISHING ATTACK DETECTION

6.1 Introduction

Chapter 5 introduced the first phase of the proposed implementation. It was a representation learning-based phishing detection approach that simultaneously used URL and HTML features. According to the proposed methodology, the next step of the study was implementing phase two, a reinforcement learning-based phishing detection framework. Therefore, this chapter first discusses the technology followed during the phase two implementation. Then, the phase two solution elements, the proposed solution, data collection and preprocessing, the solution training, and evaluation steps are discussed. Finally, a result and discussion section highlight the importance of this framework when achieving the proposed aim.

6.2 Reinforcement learning (RL)

RL is a branch of machine learning, and it learns through trial-and-error experience (Sutton & Barto, 2018). The agent and the environment are core elements of an RL framework, and a policy, a reward signal, and a value function are sub-elements of it (Sutton & Barto, 2018). The following explains those in brief:

1. *Agent:* The agent is the learner and the decision-maker in the RL framework (Sutton & Barto, 2018). In a typical RL problem, the agent observes the environment and acts on a situation presented by the environment (Sutton & Barto, 2018; François-Lavet et al., 2018). The agent decides it based on a policy and receives a pleasure or pain value called reward to estimate the produced decision (Sutton & Barto, 2018).

2. *Environment:* The environment is what the agent interacts with. It includes everything other than the agent (Sutton & Barto, 2018). The environment produces states for the agent, and once the agent takes action, the environment updates the states to next.

3. *Policy (π):* Policy defines how an agent act at a given time and the core of an RL agent (Sutton & Barto, 2018). It is a mapping between the perceived state and the action (Sutton & Barto, 2018). For example, if an agent follows policy π at time *t*, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. It further defines how an agent should behave at a given time (Sutton & Barto, 2018; François-Lavet et al., 2018). Therefore, it can be considered the brain of the agent. Since the agent aims to achieve many rewards over the long run, the agent should find the optimal policy that depends on an optimal action-value function.

4. *Reward signal:* This defines the goal of an RL problem. The agent observes the environment and acts accordingly to receive a reward (Sutton & Barto, 2018; Alkhalil et al., 2021). A reward is a numerical value used to encourage or discourage the agent's action in RL (Sutton & Barto, 2018; Alkhalil et al., 2021). Since the agent is goal-directed and the goal is to maximise the expected cumulative reward over time, the agent tries to learn the best sequence of actions to maximise the total amount of rewards in the long run (Sutton & Barto, 2018; Alkhalil et al., 2021). In that process, if an agent receives a low reward for a specific action, the agent may change the action through policy change to get a high reward in the future. Therefore, the reward signal is the primary basis when altering an agent's policy (Sutton & Barto, 2018).

5. *Value function:* The value function produces the long-term action effect for a given state (Sutton & Barto, 2018). The value is the total amount of reward an agent expects to collect over the future, and it is not immediate like rewards (Sutton & Barto, 2018). Those are secondary since there are no rewards, meaning there are no values. However, the RL always seeks the highest values regardless of the highest rewards since the value is the most important when selecting

an optimal policy (Sutton & Barto, 2018).

Figure 6.1 illustrates a typical RL framework. According to Figure 6.1, at each step *t*, *t* = 0,1,2,3..., the agent observed an environment's state, $s_t \in S$, where S is a set of possible states. Then the agent acts on the observed state, s_t, and selects the best action at $a_t \in A(s_t)$, where $A(s_t)$ is the set of actions available for the state s_t. As a result of the previous action, the agent receives a reward, $r_{t+1} \in R$, and the next observable state, $s_{t+1} \in S$, at a one-time step later. Each time step, the agent receives a reward relevant to the previous state, r_t, and defines whether the agent's previous action is good or bad (Sutton & Barto, 2018).

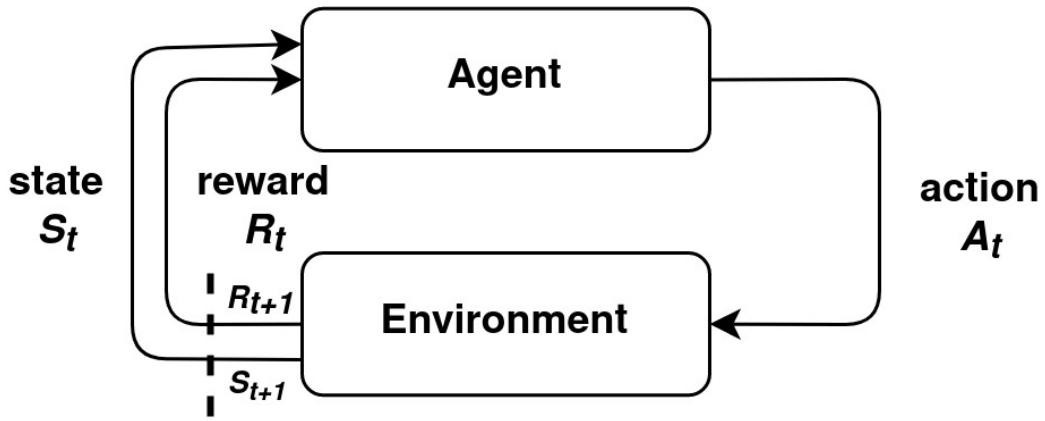
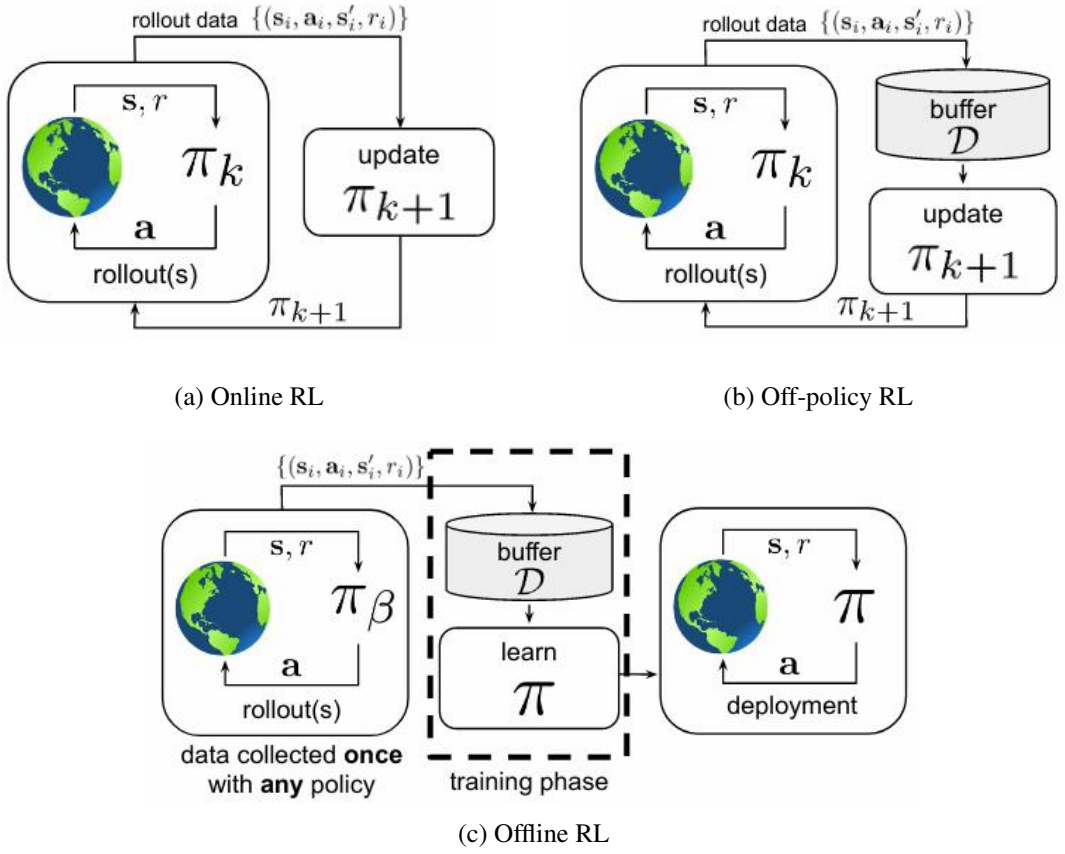


Figure 6.1: The agent–environment interaction in RL
Source: Sutton and Barto (2018, p. 54)

Generally, RL learns to control dynamical systems, which can be fully defined by a Markov Decision Process (MDP) (Levine et al., 2020; Sutton & Barto, 2018). MDP is built on top of the Markov property mentioned as the next state mainly depends on the current state and the action taken at that state (Sutton & Barto, 2018). Generally, MDP denotes a tuple (Smadi, 2017; Levine et al., 2020): (S, A, T_r, R) where S is the state space for all possible states and A is the action or control space. Then, $A(S_t)$ becomes the possible actions set for a given S at time *t*. Furthermore, T_r defines the state transition function, and R is the reward function. The reward maximises when it is closer to the goal and reduces when it is away from the goal (Smadi, 2017).

The goal of RL is to learn a policy (Levine et al., 2020). It can be achieved through different ways such as policy gradients, approximate dynamic programming

(e.g. Q-learning and actor-critic) and model-based reinforcement learning (Levine et al., 2020). RL is fundamentally in the online learning paradigm (Levine et al., 2020). However, online learning is not always practical due to the cost of collecting data and the dangers of using online data directly in a learning process (Levine et al., 2020). Furthermore, online learning is unsuitable if the domain is complex and some generalisation techniques are essential when using data (Levine et al., 2020). Therefore, the literature suggests different RL paradigms: online reinforcement learning, off-policy reinforcement learning, and offline reinforcement learning (Levine et al., 2020). Figure 6.2 shows the different architectures of those paradigms.



(a) Online RL

(b) Off-policy RL

(c) Offline RL

Figure 6.2: Different types of RL architectures
Source: Levine, Kumar, Tucker, and Fu (2020, p. 2)

6.3 Reinforcement learning model (RLM)

As mentioned in Section 4.3, reinforcement learning was the appropriate technology to enhance the phase one solution. Therefore, the study was planned to implement an RL environment called RLM during the enhancement step to decide whether a given website is phishing or not. The RLM was implemented by adding the required elements mentioned in Section 6.2. Figure 6.3 shows the implemented RLM.

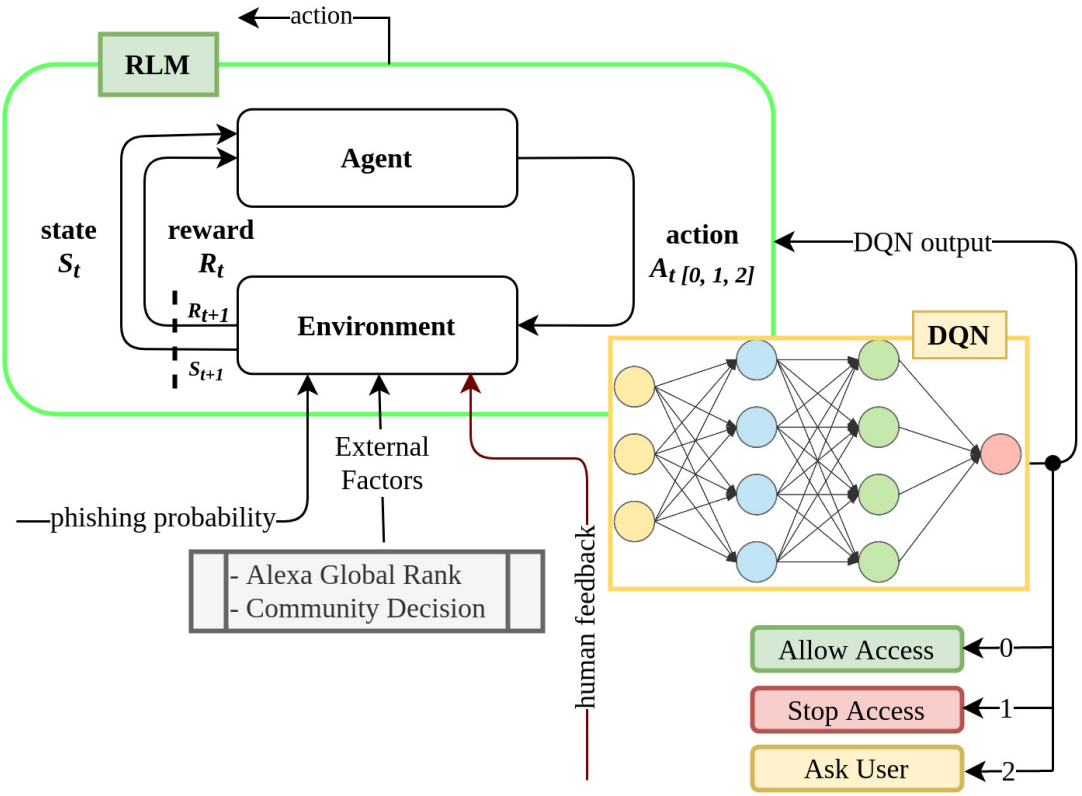


Figure 6.3: The proposed RLM architecture

6.3.1 Environment

Generally, an agent interacts with the environment by gathering observations from a given state (Sutton & Barto, 2018; François-Lavet et al., 2018). Therefore, the study decided the state's observations to support an effective agent action. First, the output of DLM was considered. The DLM produces two primary outputs: legitimate probability and phishing probability. Those two probabilities are interrelated, and the sum of both becomes one always. Therefore, one of those was decided to select, and since the

study's aim is related to phishing detection, the phishing probability was selected as the first observation value to produce the state of the RLM.

Since the DLM represents the internal phishing detection features, the study decided to use external phishing detection features alongside phishing probability during the RLM implementation, as mentioned in Chapter 4. However, the literature found that the external features negatively impacted the solution by increasing the detection time when using external features. Therefore, the study carefully selected only two external factors initially to form an effective detection process in the RLM.

The first one, community decision, indicates whether the phishing community has already identified a given web page as a phishing attack. It was derived from the blacklist feature listed in previous studies (El-Alfy, 2017; Bell & Komisarczuk, 2020). The community decision was generated with popular phishing verification systems: PhishTank and GSB. These two systems collaboratively work in the RLM, and if one of the services previously recorded a given web page as a phishing attack, that information would be passed to the RLM environment with the value of one or zero. One shows if either PhishTank or GSB already identified the requested page as a phishing attempt, and zero indicates that it was not detected as a phishing attempt. Even though the community may not always identify the attacks before RLM, if they are identified, then that information makes the RLM process fast and more accurate since the agent could observe it. Therefore, the fundamental theory behind the community decision was to transfer the existing knowledge available in the phishing community about phishing websites to the RLM agent to support the correct decision-making process. The study only depended on PhishTank and GSB since those provide free API services, but the RLM could use other specific services with little engineering effort if required.

The second external factor was Alexa global rank. It is identified as an essential factor in the literature to measure the popularity of the requested web page (L. D. Nguyen et al., 2014; El-Alfy, 2017; Yang et al., 2019). The Alexa global ranking system[16] is an Amazon service that ranks millions of websites in popularity. The Alexa service provides the popularity of a website starting from one, and one indicates the most pop-

[16] `https://www.alexa.com/`

ular website over the past three months. If a given web page ranking was not available in the Alexa service, the RLM made the ranking zero. Otherwise, the rank provided by the service would be used as it is in the proposed solution. Since most phishing attacks are reported from non-popular sources[17], the popularity provides additional information to the agent to make correct decisions.

In general, Alexa rank provides a sense of popularity about a website. It provides a global rank ranging from 1-100,000 and higher. Therefore, the global rank factor may intrinsically influence the agent more due to its value since other two factors are in the range of 0-1. To avoid such a situation and converge gradient descents more quickly, the normalisation process generated a normalised Alexa rank by bringing the range into 0-1. Equation 6.1 was used in the normalisation process, and A_r denotes the Alexa global rank for a given website. However, as mentioned previously, the Alexa rank was zero if the Alexa service failed to provide a global rank for a submitted website address.

$$Alexa\ rank = \frac{1}{A_r} \tag{6.1}$$

In a general RL problem, the agent starts interacting with the environment by gathering an initial observation in a given state (Sutton & Barto, 2018; François-Lavet et al., 2018). In RLM, a state was a web page request; therefore, the agent-environment interactions continued without a limit since the solution was constantly getting requests in the real world. In the proposed architecture, the state was a three-dimension web page request that overviews a web page regarding phishing probability, community decision and Alexa global rank. Therefore, the web page request or the state in RLM was a size three vector of three inputs that the environment perceived from outside. In there, the phishing probability was generated by the DLM. Therefore, the phishing probability was directly inputted into the environment from the DLM. However, to minimise the impact of the external factors over the detection time of the solution, the services were asynchronously called while DLM produced the phishing probability.

[17]Most of the phishing attacks recorded in PhishTank and OpenPhish have non-popular domains. However, some recorded attacks are from some famous web hosting domain like `sites.google.com`, `weebly.com` and `000webhostapp.com`.

Furthermore, the proposed RLM environment was built with the popular RL toolkit called Gym[18]. Since the environment perceives three factors as described previously and the agent produces three actions (*see* Section 6.3.4), the observation space and action space of the RLM were set into three. The minimum implementation of the RLM environment is available in Appendix A.7 for further reference.

6.3.2 Policy

The final goal of RL is to find the optimal policy. Therefore, policy plays a vital role in the RL framework. As shown in Figure 6.2, the RL solutions are mainly categorised into three based on how the policy will update when finding the optimal policy. Therefore, initially, the study wanted to select an appropriate architecture to construct a policy for the agent to work in the RL environment.

According to Levine et al. (2020), the online RL environment updates the policy with data collected by the same policy. In such cases, if a reasonable policy already exists, it can be used effectively to implement the RL environment. However, there was no such policy to start the learning process in this study. Therefore, online reinforcement learning was considered an imperfect option initially. Then, the off-policy approach was considered. In the off-policy approach, the solution can collect data from a different policy (e.g. ε-greedy) and train a policy in the future (Sutton & Barto, 2018; Levine et al., 2020). Since that was more appropriate for the study's perspective, the study selected off-policy RL architecture to build the RLM.

After selecting the architecture, the study wanted to select an appropriate RL algorithm to optimise the RL objective. Then a simple off-policy algorithm called Q-learning was found. Since Q-learning is simple and widely used (Sutton & Barto, 2018; Levine et al., 2020), the RLM implementation was planned with the Q-learning algorithm. Q-learning is a dynamic programming approach that accurately estimates the state-action value function when finding the near-optimal policy. In any RL problem, the agent's goal is to achieve many rewards over the long run. Therefore, the agent should find the optimal policy that depends on an optimal action-value function.

[18] `https://gym.openai.com/`

The optimal action-value function or in Q-learning optimal Q-function ($\bar{q}$) is defined as,

$$\bar{q}(s,a) = max_{\pi} q_{\pi}(s,a) \tag{6.2}$$

for all $s \in$ S and $a \in$ A(s) (Sutton & Barto, 2018; Levine et al., 2020). In simple words, Equation 6.2 gives the highest expected return that an agent can get by policy π for each possible state-action pair. The $\bar{q}$ further satisfies the Bellman optimality equation (Sutton & Barto, 2018; Levine et al., 2020),

$$\bar{q}(s,a) = E[R_{t+1} + \gamma max_{a'} \bar{q}(s',a')] \tag{6.3}$$

where R_{t+1} is the expected reward for the state, s, by the following action, a, at time t; γ is the discount factor, and (s, a) is the next state-action pair. Algorithm 3 shows the Q-learning algorithm.

Algorithm 3 Q learning

1: Initialize $Q(s, a), \forall s \in S, a \in A(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
2: **for all** episode **do**
3: Initialize S
4: **for all** step **until** S is terminal **do**
5: Choose A from S using policy derived from Q (e.g., ε-greedy)
6: Take action A, observe R, S'
7: $Q(S,A) \leftarrow Q(S,A) + \alpha\ [R + \gamma\ max_a\ Q(S',a) - Q(S,A)]$
8: $S \leftarrow S'$

The Q-learning was recently combined with a deep convolutional network and implemented a Deep Q-learning Network (DQN) to get the benefits of deep learning when finding the optimal policy in RL (Mnih et al., 2015; Mousavi et al., 2017). Therefore, the DQN was preferred in the current study when finding optimal state-action pairs. The following section discusses the DQN implementation carried out during the RLM implementation.

6.3.3 DQN

The DQN of the proposed solution was implemented, as shown in Figure 6.4. It has several parts like prediction network, target network, and replay memory. The following discusses the functions of these parts in detail.

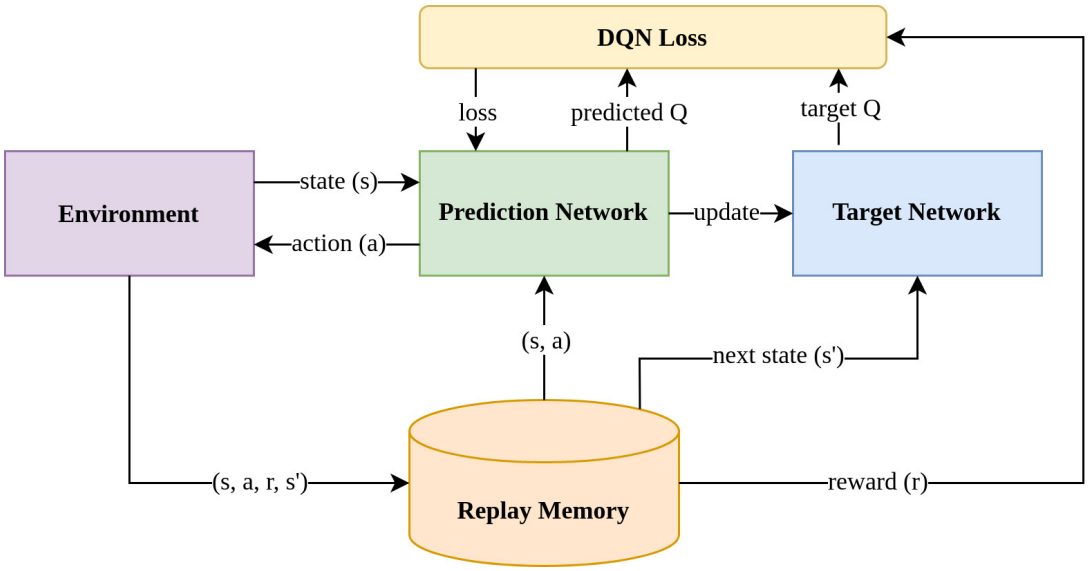


Figure 6.4: DQN architecture

First, the prediction network was implemented as a Multilayer Perceptron Network (MPN) with two hidden layers of 128 units and ReLU activation. Figure 6.5 shows that the network absorbs three inputs from the RL environment and produces one output with the support of the linear activation function. However, the study followed an evolutionary approach when selecting the presented prediction network. Therefore, different sets of prediction models were configured with different hidden layers, units, and hyperparameters (i.e., loss function, activation function). Then, those were trained and evaluated based on the maximum cumulative rewards each achieved. After that, Figure 6.5 architecture was selected based on its effectiveness (*see* Appendix B). However, the study did not evaluate all the combinations of networks built with different aspects due to time constraints.

Although the DQN is effective when selecting action-value pairs, it results in unstable or diverged RL when similar observations are seen for a certain period (Mnih et al., 2015). In the phishing context, the identified situation was more prominent since

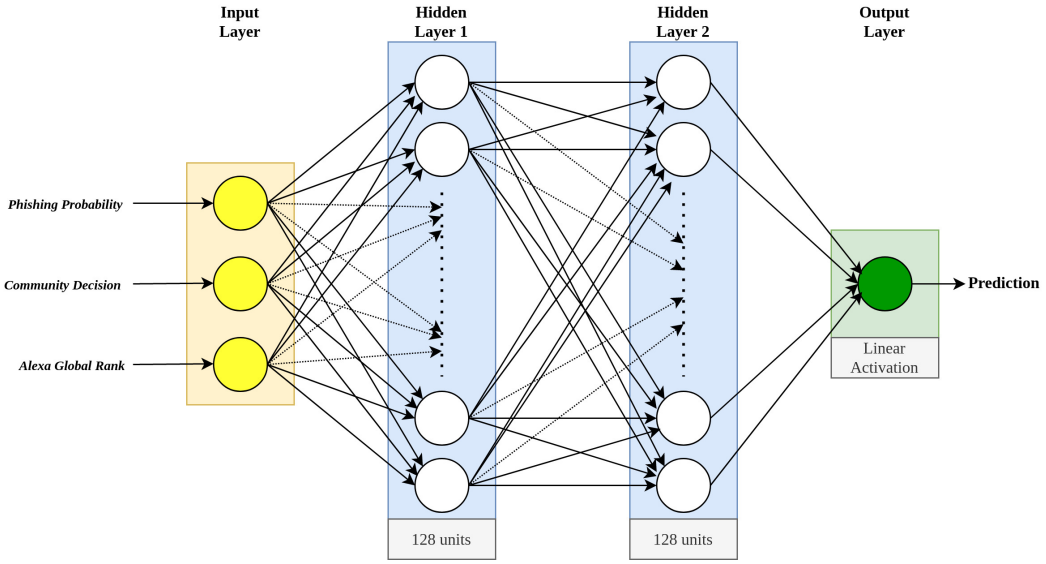


Figure 6.5: DQN prediction model

a phishing page is a rear case than the number of legitimate cases seen by an agent (Aassal et al., 2020). Therefore, two essential techniques proposed by Mnih et al. (2015) were considered to overcome such situations to avoid the unstable or diverged RL.

The first one was experience reply. It solves correlations present in the sequence of observations and allows the agent to learn more from individual tuples multiple times (Mnih et al., 2015; François-Lavet et al., 2018). A large buffer of experience and a randomised training data sample (Mnih et al., 2015) was used. The current study incorporated this technique into implemented DQN environment by introducing a fixed-size replay memory. Then a minibatch from the buffer was selected when training the DQN, allowing it to pass the same tuple multiple times to the DQN. Algorithm 4 explains it in detail. The second essential technique proposed by Mnih et al. (2015) is the target network. This technique further reduces the RL's correlation effects and makes it more stable (Mnih et al., 2015; François-Lavet et al., 2018). Therefore, in RLM, the agent has two similar Q-networks called prediction and target network in the same structure. Traditionally, the target network is a mirror network of the prediction network. The prediction network is used in training, but the target network is not trained. Those were periodically synchronised with the weights of the prediction network. However, the Q-values of the target network are used only to improve the training of the pre-

diction network. Algorithm 4 further explains the experience reply and target network techniques used during DQN implementation to improve the stability of the training.

Algorithm 4 DQN procedure used in the RLM

Initialise prediction network Q
Initialise target network $\bar{Q}$
Initialise replay memory D
Initialise the *Agent* to interact with the Environment
$\varepsilon \leftarrow$ setting ε value
$\varepsilon_{\text{d}} \leftarrow$ setting a ε-decay value
$\varepsilon_{\text{min}} \leftarrow$ setting a ε-minimum value

for new state s **do**
    Choose an action a using policy ε-greedy(Q)
    Agent takes action a, observe reward r, and next state *s'*
    Store transition (*s,a,r,s',done*) in experience replay memory D

    if *enough experience* **in** D **then**
        Sample a random *minibatch* of N transitions from D
        for *every transition* (s_i,a_i,r_i,s'_i,$done_i$) **in** *minibatch* **do**
            if $done_i$ **then**
                $y_i = r_i$
            else
                $y_i = r_i + \gamma \, max_{a' \in A} \bar{Q}(s'_i, a')$
            Calculate the loss $L = \frac{1}{N} \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$
            Update Q using the Adam algorithm by minimising the loss L
            Every C step copy weights from Q to $\bar{Q}$

            if ε is **not in** ε_{min} **then**
                $\varepsilon \leftarrow$ decrease ε by ε_{d}

6.3.4 Agent

The RLM used a DQN-based agent, as mentioned previously. The agent is responsible for selecting a suitable action for a given state from three primary actions: 'allow access', 'stop access' and 'ask user'. The allow access was used when the agent saw a secure web page. The stop access action was used if a given web page was more toward the phishing side. However, if the existing knowledge was not enough for the agent to decide on a specific web page, the agent requested some additional support from the outside; that is, the user who requested the webpage. The ask user action was

used in that situation. Then the user could submit his feedback about the web page for further action.

In RL, the agent's goal is to maximise the total number of rewards it receives. Sutton and Barto (2018) highlighted that the balance between exploration and exploitation is essential in RL to get a good reward from the existing knowledge or achieve a better one through exploration. The RLM uses three parameters to balance the exploration-exploitation dilemma. Those are epsilon (ε), a numerical value used to decide whether to explore or exploit, an ε decay value to decrease ε to maintain the trade-off between exploration-exploitation when the agent is getting more knowledgeable, and ε minimum to have a minimum exploration ability with the agent in the long run. Further, the agent uses an undirected exploration technique called ε-greedy (François-Lavet et al., 2018) to select an action during the exploration time. The ε-greedy balances exploration and exploitation by choosing between exploration and exploitation randomly. In the initial stage of the agent training, the current study was more toward exploration since the agent did not have any prior experience with states. However, once the agent learned, more control was taken by the prediction network by reducing the involvement of the random action by minimising the ε value. Algorithm 4 further describes how the agent used the DQN to get compelling predictions, and the full implementation of the RLM agent is available in Appendix A.8 for further reference.

6.3.5 Reward function

A reward is a single number sent by the environment to the agent in each time step, depending on the agent's action (Sutton & Barto, 2018; François-Lavet et al., 2018). It is the only way to alter a policy, and it produces evaluative feedback to the selected action at a time step (Sutton & Barto, 2018; François-Lavet et al., 2018). In RLM, the reward function depends on human feedback on the agent's behaviour, and it used three equations relevant to three specific scenarios. Further, the RLM evaluated whether the agent got an easy task or a difficult task and accordingly, the reward was generated. The easiness or difficulty was decided based on an entropy value (Ew) calculated us-

ing Shannon's entropy defined in Equation 4.1. If the entropy or uncertainty was very low for a given task, it indicated that the agent got an easy task. Then the RLM expected a correct action from the agent. If the agent failed to do so, the environment produced a very high negative reward since the agent failed in a simple task. However, if the uncertainty was very high, it was considered a difficult task and a correct action resulted in a very high positive reward for the agent. However, the correctness was decided by the feedback the RLM gets through human participation.

The RLM used three reward functions to handle different scenarios. The following described those in detail.

$$r = int(E_w * 100) \tag{6.4}$$

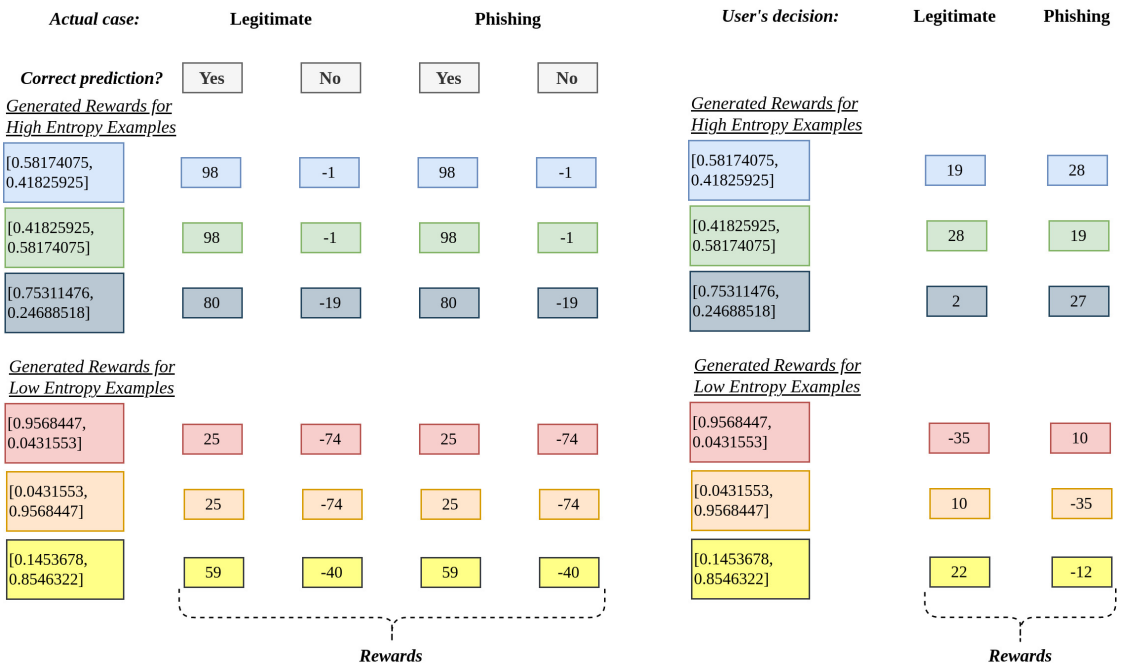
$$r = int((E_w - 1) * 100) \tag{6.5}$$

$$r = int(\frac{E_w - P(x)}{2} * 100) \tag{6.6}$$

In Equations 6.4, 6.5 and 6.6, the r is the reward; E_w is the entropy calculated for a web page using Equation 4.1. Equation 6.4 was used when the agent's action was correct, and if it was incorrect, Equation 6.5 would be used. If RL asked the user to select an action, Equation 6.6 would be executed. In Equation 6.6, the P(x) is the web page probability based on user feedback. For example, if the user sent the feedback as a trusted web page, then P(x) became the legitimate probability of the given web page. However, the relevant P(x) was picked from the DLM output. Equation 6.4 always gave a positive value, and Equation 6.5 was negative all the time. However, Equation 6.6 gives either a negative or positive value, depending on the requested web page and the user's feedback.

Equation 6.6 was formulated to generate rewards when the agent provided an action to get web user support in some instances. Since high rewards for such action promoted the agent to select it more often, Equation 6.6 was organised to minimise this action by giving a low reward. Therefore, Equation 6.6 includes a divide by two to decrease

the reward and gives half of the original reward to the agent since it is a collaborative work by the agent and the web user. However, Equation 6.6 gave a low negative value compared to Equation 6.5 for the same scenario, which was a strategic approach to teach the agent to get user support if any negative experience presented in the current state. Further, the reward was converted to a whole number in all the equations that range from -100 to 100 since it represents the common feedback approach in general learning systems. Figure 6.6 shows some example scenarios and the rewards an agent received in those examples (*see* Appendix B for more details).



(a) Rewards generated from Equation 6.4 and 6.5 (b) Rewards generated from Equation 6.6

Figure 6.6: Generated rewards by RLM in different scenarios

According to Figure 6.6(a), if an agent gets a high entropy task, the agent gets more rewards for collecting decisions and a low penalty for incorrect decisions in legitimate and phishing cases. It is because high entropy means a difficult task. If an agent is correctly decided in a problematic situation, the proposed architecture appreciates the agent more. However, if the agent fails, the reward keeps minimising since the agent failed in a difficult task, and it is not fair to give a high penalty. Although Figure 6.6(a) rewarding mechanism motivates an agent to take the risk since the penalty is low, 6.6(b) indicates that if the agent is not very sure, getting the support from the user

may keep the rewards positive, and it helps the agent to gather more rewards at the end. These three cases teach agents to take the correct decision and, if not, get user support.

However, in low entropy scenarios, since it indicates an easy task, incorrect decisions get a high penalty compared to the previous situations. Further, the proposed RLM will not appreciate an agent more if the agent makes a correct decision on an easy task. Therefore, rewards keep getting low on the positive side for correct decisions. Since getting user support for an easy task is useless, the proposed rewarding mechanism will not promote an agent to get user support in such tasks. However, as presented in 6.6(b), the rewarding process indicates that going to the user is better than making incorrect decisions when accumulating rewards by the agent.

6.3.6 Phishing detection framework

After adding all these elements, the RLM was constructed, as shown in Figure 6.3. The RLM can be considered a phishing detection framework since it could use more phishing detection criteria with minimum engineering effort. The RLM uses only three phishing detection criteria: phishing probability from DLM, community decision and Alexa ranking. However, more characteristics could also add to the RLM with minimum engineering effort. For example, if a phishing or legitimate decision based on the website's visual appearance can be produced via a separate solution, then that output could be integrated into the RLM state via a separate observation. Then, the RLM can reset its agent to the initial stage and give some time for an agent to train. Since the RL can learn from a small amount of data, the RLM agent will get the competency to filter websites using all of these four observations included in a state after a certain period. Therefore, the implemented RLM is a phishing detection framework configured to use different phishing detection criteria other than the initial criteria used in developing the proposed solution.

6.4 Phishing detection solution (RDLM)

Figure 6.7 shows that the study connected RLM with DLM and designed the final phishing detection solution. The solution was named RDLM for referencing sim-

plicity, and the name was selected since it was a reinforcement learning and deep learning-based model. The RDLM was the final solution produced after phase two of the implementation process, and it primarily answered the first research question of the study.

6.5 Data collection and preprocessing

The study has already constructed two classic and modern datasets in phase one of the implementation process. However, those were unsuitable for the RLM training and evaluation since the RLM required Alexa rank and community decision as two inputs other than the URL and HTML content. Further, the Alexa rank and community decision are frequently changing. Therefore, it is not correct to construct these values for past data. Thus, the study constructed two new datasets to train and evaluate the RLM by following the same procedure followed when constructing the modern dataset. However, when collecting phishing data, other than the PhishTank, the study also used OpenPhish. It was due to the community decision input used by the RLM. The RLM implementation also included PhishTank as a service when generating the community decision. Therefore, if the study collected data only from PhishTank, all the collected phishing data might have value under the community decision and result in a biased dataset. Therefore, OpenPhish was used to collect some phishing data that were not recorded in PhishTank via a self-generated script. Table 6.1 summarises the collected data to train and evaluate the RLM.

Table 6.1: Details of the dataset used in RLM training and evaluation

Dataset	Phishing		Legitimate	Total
	PhishTank	OpenPhish		
Training	1,011	3,351	4,338	8,700
Test	1,203	919	2,122	4,244

The training dataset was collected from July 27, 2021, to August 11, 2021. It had 8,700 data, including 4,362 phishing data and 4,338 legitimate data. The collection process used the Google search engine to collect legitimate data and followed Section 5.3.2 mentioned procedure to collect diverse data. The phishing data was collected

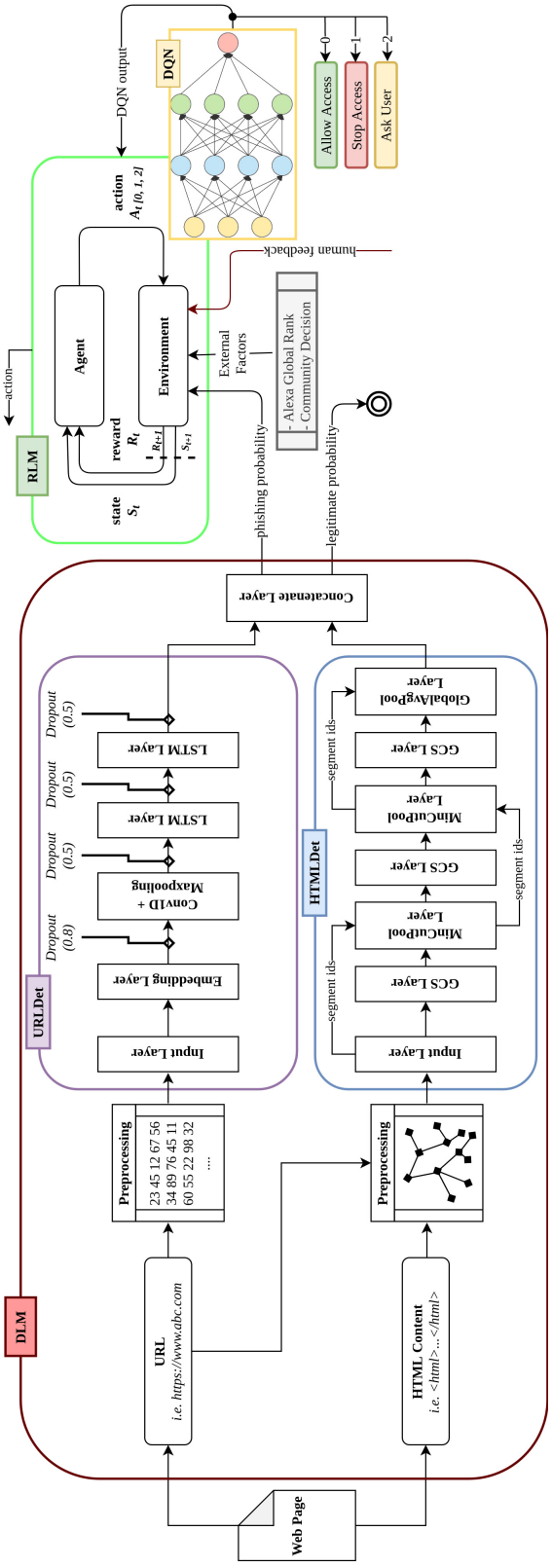


Figure 6.7: Overview of the proposed phishing detection solution

through PhishTank and OpenPhish with the support of self-generated scripts. The community decision and Alexa rank were collected simultaneously when the phishing and legitimate datasets were constructed. Similarly, the test dataset was collected following the same procedure from November 01, 2021, to November 12, 2021. It had 4,244 data in an equal amount of phishing and legitimate.

The RLM required three inputs to train and evaluate the model. Therefore, the data was preprocessed to meet the input requirements while constructing the datasets. First, the phishing probability was generated using a DLM retrained on July 21, 2021. The collected URLs and HTML contents were passed to that DLM, and the outputted phishing probability was collected. Then, the Alexa rank for the collected data was generated using the Equation 6.1 formula. Both legitimate and phishing URLs collected during the construction of the dataset were passed to PhishTank and GSB to get the community decision. The value was used if either mentioned a submitted request as a phishing record. Otherwise, the value zero was used to mention that a data item was legitimate or not recorded as phishing in PhishTank or GSB.

6.6 Model training

The phishing detection is a continuous task since the agent sees new states until the process stops. However, during the agent training, the study considered phishing detection an episodic task to measure the agent's learning ability via the total rewards achieved at the end of each episode. Since the agent trained with the training dataset, each episode had 8,700 steps. The total rewards earned were calculated and plotted at the end of each episode. Figure 6.8 shows the accumulated rewards in each episode during the agent training. After 55 episodes, the agent learning was converged. Therefore, the training was stopped in the 55th episode. The used agent during the training period is available in Appendix A.8. After several experiments with different hyperparameters, the used agent was selected as the effective agent in the current study.

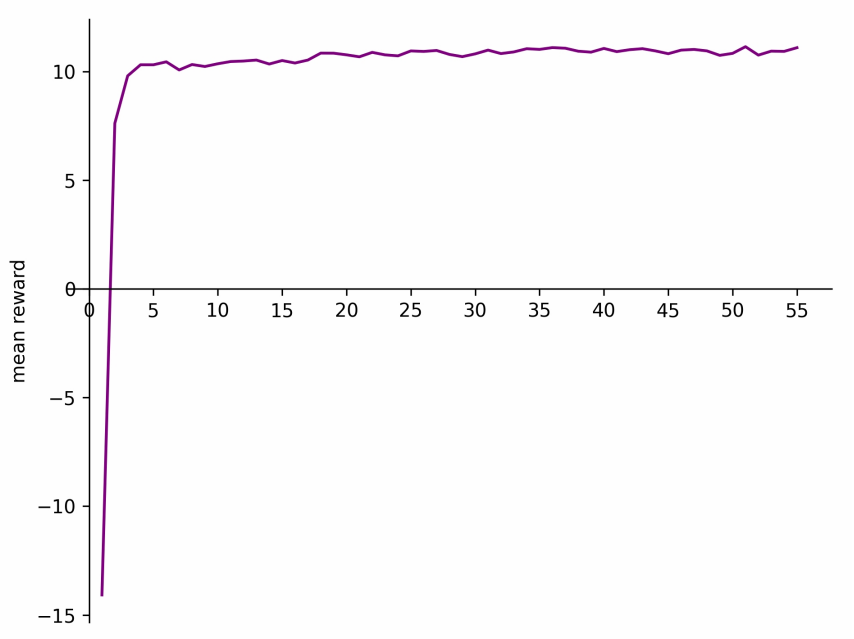


Figure 6.8: Accumulated mean reward in each episode

6.7 Model evaluation

After the successful training step, the RLM was evaluated under two criteria: overall performance and benchmarking.

6.7.1 Overall performance

The test dataset was used with the trained agent during the first experiment. Then, the agent's action and the actual status of the data item were analysed to generate the relevant confusion matrix, as shown in Figure 6.9. The performance matrices were then calculated based on the prediction results, and Table 6.2 presents those values.

6.7.2 Benchmarking

The RLM was benchmarked with the DLM to verify whether the enhancement done with the phase one solution significantly impacted phishing detection ability. Both models used the test dataset, and the final outputs were manually analysed to find the correctness of each decision. Table 6.2 presents the results achieved during this experiment.

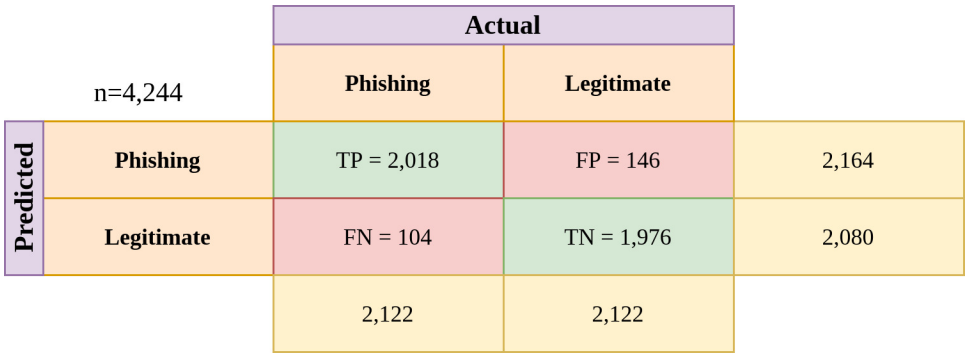


Figure 6.9: Summary of RLM's prediction results

Table 6.2: RLM and DLM performance with the test dataset

	Accuracy	Precision	Recall	*f1*-score	FNR
DLM	87.82%	98.31%	76.96%	86.33%	0.230
RLM	94.11%	93.25%	95.10%	94.17%	0.049

6.8 Results and discussion

According to Table 6.2, the accuracy and *f1*-score are higher in RLM than in DLM. Since the DLM was last retrained on July 2021, the DLM was nearly three months old when used with the experiment. As discussed previously, the main requirement of an RLM was to control this performance drop up to a certain extent. The experiment results indicate that if the DLM was involved, it could detect phishing attacks with 87.82% accuracy. However, the RLM could achieve 94.11% accuracy in the same environment. It shows that the improvement done to the phase one solution was greatly affected during the phishing detection.

Further, FNR is a significant metric in phishing detection since phishing as legitimate has a high impact on phishing. Table 6.2 shows that FNR is comparatively low in RLM. It implies that the RLM distinguishes phishing from legitimates better than DLM. The external information, especially Alexa rank related to a legitimate website seen by the RL agent, might be the reason for this low FNR.

However, the RLM cannot work along. It always depends on DLM since the DLM is responsible for providing the phishing probability of a website based on its URL and HTML content. It is one observation the RL agent sees when deciding on a website.

The other two are community decision and Alexa rank. Out of those two, one might argue that the high phishing detection rate is due to the community decision factor used by the RL agent when making decisions. It should greatly support the agent's decision, but the experiment shows that the RLM could detect phishing attacks with high accuracy when the community did not identify them before. During the experiment, 919 phishing attacks were not identified by the community before the RLM saw those since those are not listed either in PhishTank or GSB. However, out of those 919 instances, 817 were correctly detected by the RLM and proportionally, it is 88.9%. It indicates that the RL agent can detect phishing attacks with a high detection rate without the community's support.

Furthermore, 1,203 phishing instances were already known by the community during the experiment. The agent used the existing phishing detection knowledge during the detection, and 1,201 phishing instances out of 1,203 were correctly detected. As a percentage, it is 99.83%. It highlighted that the agent learned to effectively use the existing phishing detection knowledge to detect phishing attacks during the training period. Therefore, whether the community support was received or not, the RLM performed well during the experiment.

Since the RLM was introduced to improve the phase one solution, the phishing detection solution proposed through this study will be the RDLM. In there, the DLM is responsible for analysing the internal structure of a given website and deciding whether the website is phishing or not based on internal information like URL and HTML content. Then, the RLM looks at some of the external features alongside the DLM decision and takes the final decision. In the current study, those are community decision and Alexa rank. However, the RLM was implemented in a way that the number of external features can be increased with minimum engineering effort. Then, the agent could take the correct decision than the present.

Since the RLM agent is continuously learning by interacting with the natural environment, it can identify some slight changes that happen in the DLM-produced phishing probability with the trial-and-error experience. Then, it can dynamically adjust its policy to make a correct decision in future scenarios. With this support of this trial-and-

error concept, the RDLM can control the performance dropping problem in the phase one solution. Table 6.2 also highlighted that the final detection accuracy was improved by 6.29% with the support of RLM. However, as further discussed, the RLM agent may misdirect if the DLM knowledge is not updated after a certain period since phishing attacks are constantly changing. The literature highlighted that the performance of a phishing detection model drops to a reasonable number after two months. Therefore, as mentioned earlier, the current study planned to update the DLM knowledge about phishing detection features every three months.

Not only the DLM, the RLM also needs to learn about the present phishing environment to make better decisions continuously. In RLM, prior learning happens through the generated rewards, and human feedback is essential in that process. Therefore, a way to get human feedback continuously to the RLM is also vital for the success of the proposed phishing detection solution. Therefore, the proposed solution expects to integrate continuous learning support through a knowledge acquisition process into the RDLM solutions. It was a future work at this level in this study since the knowledge acquisition process was not yet discussed. However, Chapter 8 will complete the phishing detection solution proposed here by presenting how continuous learning was integrated into the RDLM with the support of the knowledge acquisition process.

6.9 Summary

By proposing a strategic approach to control the performance drop of the phase one solution, this chapter delivered a phishing detection framework named RDLM to detect phishing attacks effectively. It is an off-policy-based RL architecture that relies on phishing probability, community decision and Alexa global rank. The RDLM achieved 6.29% more detection accuracy than the phase one solution in the same environment. However, the study planned continuous learning support as a core part of the proposed methodology to achieve the study's aim. As the first step toward that, the next chapter discusses the proposed systematic way of collecting the latest phishing data to support the continuous learning of the RDLM delivered in this chapter.

7 PHISHING DATA COLLECTION PROCESS

7.1 Introduction

Chapter 6 introduced the RDLM, which is the core phishing detection component of the proposed solution. After implementing the RDLM, the next phase was the integration of the continuous learning support to update the existing phishing detection knowledge. As the first step towards that, a systematic approach was proposed to collect, verify, disseminate, and archive real-time phishing data. This chapter introduces the proposed phishing data collection approach. In addition, this chapter also discusses the diversity and effectiveness of the collected phishing data and the usefulness of the introduced approach when integrating continuous learning support into the proposed RDLM.

7.2 Overview of the proposed process

The current state of phishing data collection was discussed in Section 3.3. It revealed that there was no systematic approach to collecting the most recent phishing data. However, there is some evidence of such an approach called Phisherman in the literature, but it is not available online. As a result, systematic data collection for phishing was lacking in the current anti-phishing domain. As a result, this study adopted Phisherman's main concept and implemented a systematic approach to collecting, verifying, archiving, and disseminating phishing data. It was named as PhishRepo with the idea of a phishing data repository. PhishRepo is an online phishing data repository initially built to fill the phishing data gap in the anti-phishing domain. The primary motivation for this implementation was to have an Oracle to support the active learning process because machine learning models such as DLM and RLM require labelled data for learning tasks, and the study chose active learning as the technique to support the la-

belled data.

In most cases, the data collection in PhishRepo begins with a phishing URL submitted by an authentic user. Further, it has introduced five user roles to manage the access controls of users within the solution since PhishRepo was built with different intentions like collecting, verifying, and disseminating phishing data. Those are administrator, editor, reporter, beneficiary, and guest. The administrator is the root user with full access to all the services provided by PhishRepo. Then, the editor mainly contributes to labelling phishing data to have a quality output at the end. However, there are three levels of editors in PhishRepo. These are newbie, competent, and expert. The expert editor is the chief editor in the proposed solution. He is responsible for the final decision of incorrect submission, and if required, he can modify any of the available records in PhishRepo. The other two editors are mainly used in the labelling process. Except for expert editors, the other two editor levels are automatically updated based on the points earned by correctly marking submissions. The administrator decides the expert editors based on PhishRepo's recommendations. Next, the reporter is also in two types: individual and corporate. These two types were introduced by thinking about the future of the anti-phishing domain, where autonomous solutions are widely practised. The individual reporter type is for general users like humans in that design consideration. The corporate category supports active anti-phishing tools or humans willing to submit an automatic submission to PhishRepo.

Since the PhishRepo phishing data collection process is only available for authentic users, a beneficiary user role was added to PhishRepo to facilitate users who only need the most recent phishing data from PhishRepo. Therefore, this user can only access the proposed solution's data dissemination process described in Section 7.2.3. Consequently, the guest user can only access the public information available in the online system and reporting or downloading phishing websites are not allowed for that category. Figure 7.1 shows the landing page of the PhishRepo solution.

The proposed PhishRepo solution is only available to authenticated users. As a result, this study used two authentication methods: username-based and application key-based. Most software systems use the username and password combination to au-

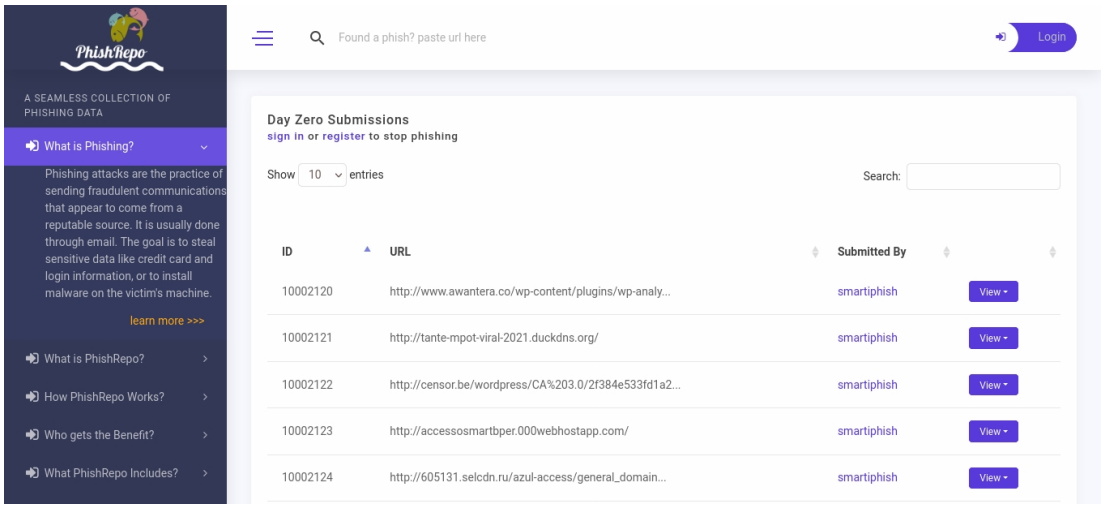


Figure 7.1: The landing page of PhishRepo

thenticate a user. It is commonly used as the first method in PhishRepo. Then, the second method only applies to corporate-type reporters and is only used when a corporate reporter sends an automatic submission to PhishRepo. However, that user must submit the request in the following format to become valid in the proposed solution.

```
key : ''4b6abb66jd21x4d1a6b00ed231fb377y4'',
        #String of relevant application key
url : ''https://phishing-example.com'',
        #URL of the detected phishing website
```

The primary goal of PhishRepo is to collect quality phishing data for retraining the RDLM proposed in Chapter 6. PhishRepo achieved this goal primarily through three processes: data collection, data labelling and data dissemination. The following sub-sections discuss these processes in detail.

7.2.1 Phishing data collection

Phishing data collection is the first sub-process of the proposed PhishRepo solution. It includes phishing URL collection, downloading the relevant information sources, and storing the downloaded sources for dissemination. Figure 7.2 summarises the data collection process of the proposed solution graphically.

As identified in Section 3.4, the URL, web page, and possible third-party infor-

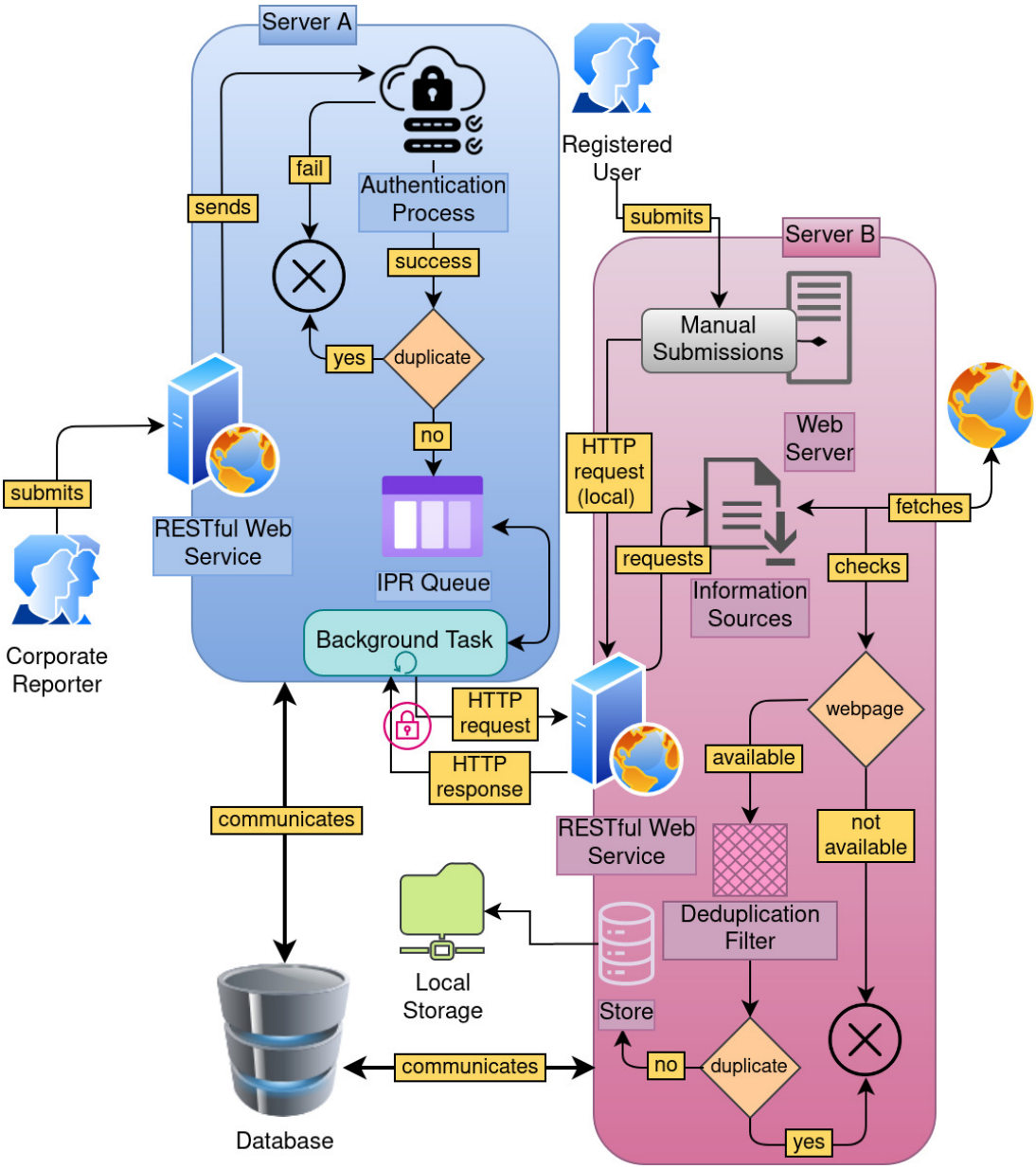


Figure 7.2: Workflow diagram of the data collection process

mation sources like Alexa statistics are essential in a phishing dataset to support a vast range of phishing detection solutions, including the RDLM. However, collecting such phishing data sources is challenging since 63% of the phishing campaigns last within the first two hours (Khonji et al., 2013). Therefore, fast collection of relevant resources by keeping the minimum difference in the detection and reporting time was crucial when designing the data collection process. Furthermore, automatic submission of phishing URLs was mainly considered in the solution design process to collect the

most active and online phishing URLs for the purpose of acquiring the required information sources effectively. PhishRepo's data collection process includes three steps: submission, accumulation, and deduplication. The followings discuss these steps in detail.

7.2.1.1 Submission

The first step of the data collection process is the submissions. PhishRepo accepts only URL submissions, which means a user can only submit a phishing website URL, and it will not allow the submission of any other information related to the website. It is one design consideration used by the study to eliminate the missing data from the collection process and to archive quality data downloaded from the sources. However, PhishRepo allows using two submission modes: manual and automatic. Although it has two, PhishRepo always promotes automatic submission since manual submission may have old phishing URLs, which may cause issues in the accumulation process that will be discussed in Section 7.2.1.2.

In PhishRepo, submission needs to come from an authentic source. Therefore, any user willing to submit phishing URLs needs to have an account in PhishRepo. However, administrator, editor and reporter-type users only can submit phishing URLs to PhishRepo. The reporter account is especially considered since that user was mainly linked with the submission step. Since the submission can be manual or automatic, previously mentioned authentication types are used accordingly to authenticate the submitter.

A user can use the Figure 7.3 interface to submit phishing URLs manually. The application key-based submission is only allowed for automatic submissions, and the format mentioned above is required in such submission. One request can submit only one phishing record in automatic submissions, and multiple requests are required for numerous submissions. However, as shown in Figure 7.2, manual and automatic submission are handled in two different ways in the proposed architecture. Since the manual submission are made by a login user, it directly passes to the accumulation step. However, automatic submissions are not like that. Those need to follow a rigid

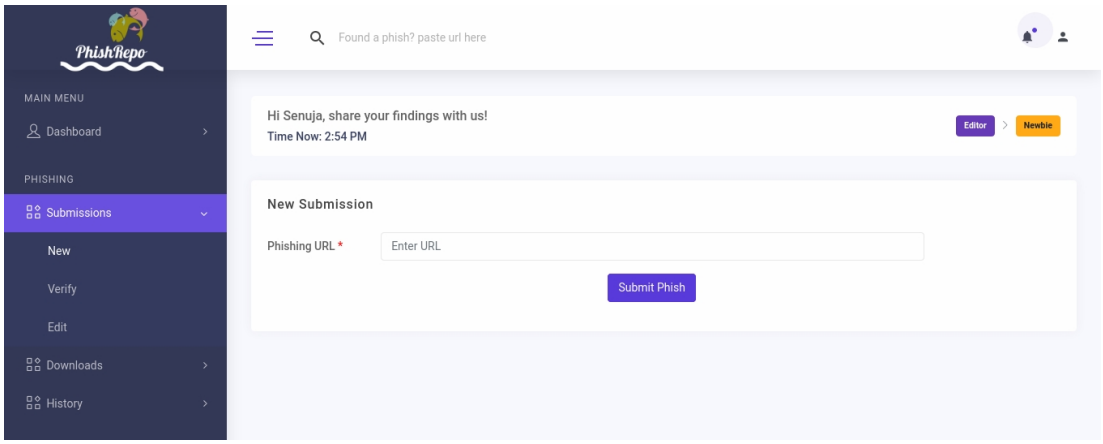


Figure 7.3: Manual submission interface

process before coming to the accumulation step. First, the submission source's authentication will be evaluated in the process. Once it has passed, it checks for duplication with already collected data through URL string matching. Then, if no duplication is found, the URL is added to the Initial Phishing Records (IPR) queue, which is a Double Ended Queue (Deque) that uses First In, First Out (FIFO) logic alongside the submitter information. Then, a background task is run to pick one record from this IPR and send it to the accumulation step for further actions. The task works like a loop, and once the accumulation step returns with a numeric one for the previous record, it will be removed from IPR and moved to the next IPR record.

7.2.1.2 Accumulation

Once a submission comes from the submission step, the accumulation step is responsible for the data collection. Since the proposed solution requires the collection of various information sources like a web page and third-party information for a submission, the accumulation step fetches that relevant information from the Internet. However, it has also been done in a predefined procedure. First, the complete web page of the submission URL will be fetched. If it could not be achieved, that submission will be discarded since the web page for a submitted URL is a vital information source in the proposed solution and since the DLM of the proposed solution requires a URL and web page in a retraining process. If the web page of the submitted URL is successfully downloaded, then the other information sources like a screenshot of the web page,

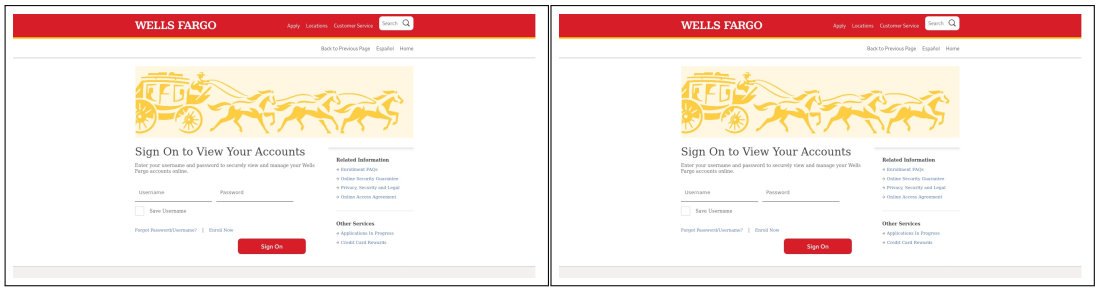
a complete view of the web page, and the offline web page, including images, style sheets and relevant scripts, will be downloaded alongside third-party information like Alexa statistics and response details of the request.

Although PhishRepo was designed to download all these additional information sources for a submission other than the web page, except the screenshot, other sources are not mandatory in PhishRepo since those are not required to retrain the DLM. However, the screenshot is considered essential alongside the web page in this step since the next step, the deduplication filter, mainly depends on this. Therefore, if a submission could not get the complete web page or the screenshot, the submission will be discarded in this step without further steps.

7.2.1.3 Deduplication

After the accumulation step, a submission enters a specific step that is used to filter redundant data that could be archived during the data collection process of PhishRepo. It is the deduplication filter. In phishing data collection, redundant data could be collected on two occasions. First, the same phishing attacks might come from several users. For example, *user A* submitted `https://phish-attack1.com` to the data collection process, and *user B* also submitted the same. Then, both submissions contain the same URL, resulting in some redundant data at the end. The second occasion is that the phishing attack may use different URLs for the same attack, as shown in Figure 7.4. It is possible since most phishing pages are created using phishing kits and released to the public (Chiew, Yong, & Tan, 2018). If that happens, it may create redundant data and cause data leakage in machine learning-based phishing detection processes. Both these occasions should be avoided to have a quality data collection process.

In PhishRepo, the first chance of collecting redundant data was handled in the submission step. As discussed in Section 7.2.1.1, newly submitted URLs were initially checked for URL duplication. Therefore, the initially collected URLs will not pass the submission step since those are discarded there. However, the second type of redundant data is not captured previously. Therefore, the deduplication filter was implemented in PhishRepo to stop such data collection.



(a) https://cbahospitalar.com.br/002WG/well-fargo-RD528-detail/

(b) https://mail.cbahospitalar.com.br/002WG/well-fargo-RD528-detail/

Figure 7.4: Example of a scenario of different URLs for the same phishing target

The primary responsibility of the deduplication filter of PhishRepo is to eliminate duplicate phishing attack submission that comes under different URLs. This was achieved using the Perceptual Hashing (pHash) technique (Zauner, 2010). The pHash has already been used by an earlier study in a different domain to handle a similar situation (D. T. Nguyen et al., 2017). Therefore, after analysing pHash functionality, this study successfully attached the pHash to the phishing data collection process. The deduplication filter removes the duplicate phishing submissions as an inline task. Figure 7.2 shows that this filter is applied before saving any data to the local storage. Therefore, none of the submissions is identified as duplicate records kept in PhishRepo's repository at any time.

The deduplication filter generates its output using the visual level screenshot downloaded in the accumulation step. It uses the pHash method to compare two phishing pages, and PhishRepo is responsible for keeping track of the hash values generated for previous submissions. As a result, once a new phishing page is captured, its hash value will be compared with the stored values to determine whether a newly captured one is a duplicate of a previously collected web page. This comparison primarily depends on the distance factor (d). If the d becomes zero, it implies that both pages are similar. Then, one of the records will be deleted from the PhishRepo to eliminate any redundant data. In that case, PhishRepo's 'Dedup Action' setting will determine which one needs to be removed. The Dedup Action has two values: 'new' and 'old'. If the value 'new' is enabled, the deduplication filter saves the new record and deletes the old one from the repository; otherwise, the old remains. However, it is up to the administrator

to decide what web page is to keep in the repository, and he has the freedom to change this configuration anytime through the Dedup Action setting.

Although the deduplication filter can detect duplicate web pages, the used logic could fail if the relevant web page is not loaded correctly during the screen capturing time. To avoid such cases, PhishRepo is configured to look for near-duplicates. However, near-duplicates are not checked for all collected web pages because this hash values comparison takes time. Therefore, this study considered only the last three days when checking the near-duplicates. It is mainly because most phishing attacks last three days or less (Gowtham & Krishnamurthi, 2014; Zeng et al., 2020).

However, the near-duplicates selection should have a systematic method. Therefore, this study introduced an optimal distance threshold (d_α) to have a meaningful near-duplicates selection. This threshold is selected based on 1,000 random samples collected after several months of the data collection. When selecting this threshold, first, pHash values of these 1,000 records were computed. Then, d values were calculated for each pair available in this sample. After that, a manual investigation was conducted to check the accuracy. According to this investigation, when d exceeded 10, the accuracy of a pair's similarity significantly decreased. Hence, d_α was selected as 10, as shown in Figure 7.5. Meanwhile, $0 < d < 10$ is considered a range of near-duplicates in PhishRepo. However, the near-duplicates removal process does not affect the Dedup Action setting discussed previously, and if a near duplicate is found, the new submission will be entirely removed from PhishRepo to maintain a diverse phishing data repository.

After the submission passes from the deduplication filter, it is saved to the local storage, as shown in Figure 7.2. Since it is unique data collected by the data collection process, then it should get a proper label. The labelling process is essential in phishing data collection because if incorrect data were submitted, it might affect the quality of the data produced by the process. As a result, all the submissions saved to the local storage initially get the 'submitted' flag. However, those are not treated as phishing data until correctly labelled.

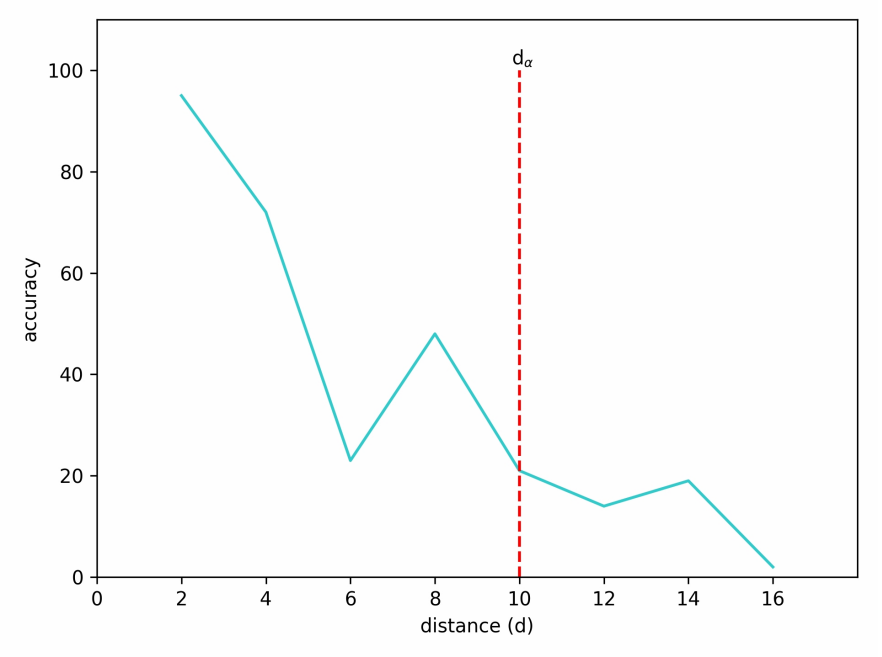


Figure 7.5: Estimation of distance threshold *d* for near-duplicate detection

7.2.2 Labelling process

Once the submission process was done, PhishRepo was designed to give a proper label to the collected phishing data. Since any authentic user, either a human or an autonomous anti-phishing tool, can submit phishing URLs to PhishRepo, incorrect phishing data may be collected during the data collection process. Therefore, a quality labelling process was proposed during the implementation of PhishRepo.

Section 3.3 discussed the available data labelling approaches, and it highlighted that crowdsourcing is the suitable approach to use when labelling phishing data. Therefore, PhishRepo was designed to use a crowdsourcing approach when labelling the collected data. However, Section 3.3 emphasised that it is difficult to get quality output from crowdsourcing due to several challenges. As a result, several strategies were used to get a quality label for a given data point during the solution implementation.

First, the proposed labelling process divided labelling into two steps. Those were then named Alpha and Beta labelling. Any collected data point first enters the Alpha step, and if it could not get a proper label, it then moves to Beta, as shown in Figure 7.6. The Beta labelling process depends on crowdsourcing architecture. However, the

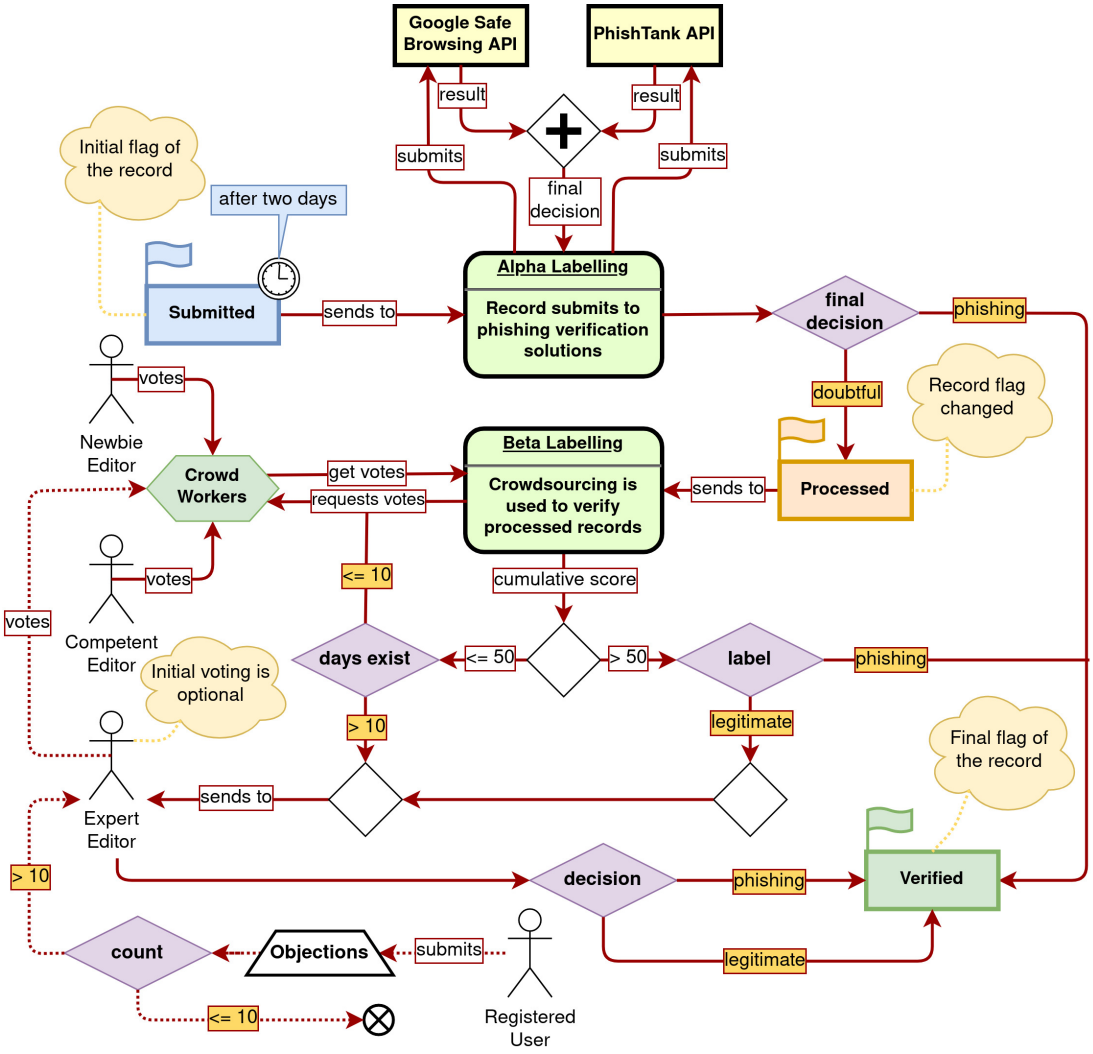


Figure 7.6: PhishRepo's submission labelling process

Alpha labelling step was designed to reduce the workload of the Beta. That design consideration was taken due to some per-mentioned challenges of crowdsourcing. The following sub-sections discuss these two steps in detail.

7.2.2.1 Alpha labelling

Alpha labelling was implemented using two popular phishing verification solutions: PhishTank and GSB. These solutions have free API support to access recorded phishing websites' details. According to the literature, 47%-83% of phishing URLs are added to the popular blacklists after 12 hours (Khonji et al., 2013). Therefore, once a submission is stored in the local storage with the submitted flag, it waits 24 hours before entering the Alpha step. Then, the submitted URL will be sent to PhishTank and GSB APIs and waits for their decision. If either one marks the submitted URL as phishing, the submitted flag of that submission will change to 'verified' and label it as a phishing data item. However, if none of the services marked the URL as phishing, it might be due to its legitimacy or not being recorded in the verification services. In that case, the flag is changed from 'submitted' to 'processed', as shown in Figure 7.6. It indicates that the Alpha step is already processed on that submission. If submission gets the 'processed' flag, next, it enters the Beta labelling step. Otherwise, the labelling process will be ended for that submission.

7.2.2.2 Beta labelling

The Beta labelling step was designed to get public support for the labelling process. Therefore, it is a crowdsourcing approach. The submission with the 'processed' flag enters this step, and the users who joined as editors can label these submissions. However, the final judgement on an incorrect submission depends entirely on the expert editor, and he is involved mainly with the labelling process if the majority of newbie or competent editors label the submission as legitimate or if the submission stays 'processed' state for more than ten days without getting a proper label.

In PhishRepo, the Beta labelling primarily depends on a voting scheme. Therefore, each 'processed' flagged submission appears to the editors to vote, as shown in Figure

7.6. Then, the editor can select either phishing or legitimate to award points using the Figure 7.7 voting interface during the labelling process. PhishRepo allows for labelling a submission multiple times since Zhao et al. (2011) highlighted that it is an effective technique to get a quality label in a crowdsourcing approach. In that case, Absolute Cumulative Majority Relabelling (ACMR) strategy is more effective since this strategy allows relabelling of the same data item multiple times (Zhao et al., 2011). Furthermore, ACMR is an interesting relabelling strategy which uses a voting mechanism to select majority voting. In this strategy, if a label achieves more than 50% voting, it sets that as the correct label for that data item. However, if none of the labels could earn more than 50%, the data item is discarded in the ACMR strategy. Since the ACMR strategy seems effective in labelling phishing data, this study inherited it to implement the Beta labelling step. Therefore, the submission must collect more than 50 points on the phishing label in the Beta labelling step to becoming a verified submission. However, in the ACMR strategy, if a data item could not earn more than 50%, it is discarded. It was not practical for this study. Therefore, it was slightly changed in PhishRepo. In PhishRepo, if a submission could not earn more than 50 points, it will be directed to an expert editor since he is the final decision maker for submission, as shown in Figure 7.6.

A submission gets points based on editor votes in the Beta labelling step. Since prior knowledge of the given task and novice workers affect the quality labelling process in crowdsourcing, as mentioned in Section 3.3, PhishRepo is designed to have different impact points in the voting process for expert, competent, and newbie editors. These levels are maintained based on prior experience in labelling. However, every time a fresh editor begins his journey with PhishRepo, he gets a newbie level. The newbie-type editor has a low impact on the labelling process since they do not have prior experience. PhishRepo controls these impact levels by giving different point scales for expert, competent, and newbie editors. For example, if a newbie selects one submission as phishing, then the submission gets 10 points under the phishing label. If a competent level user selects the same, the submission receives 25 points. Since the expert editor is the chief editor in PhishRepo, if he selects a submission as phishing,

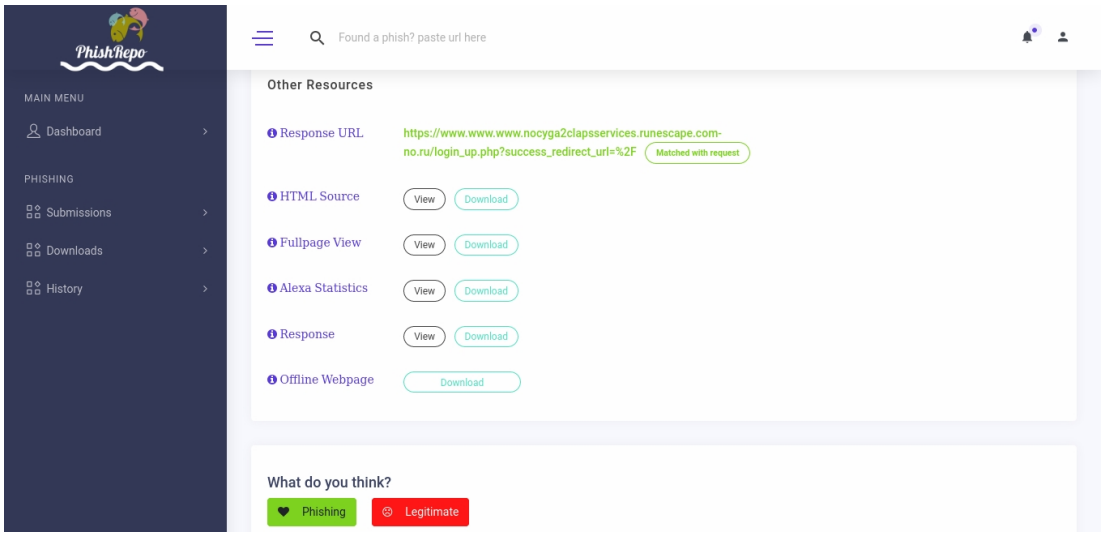


Figure 7.7: PhishRepo's voting interface

that submission gets 100 points.

The Beta step voting is both positive and negative. Suppose that a newbie marked a submission as phishing. Then, if a competent user legitimates the same, PhishRepo checks the voting trend in that submission, and since the voting trend is now on the phishing label, the submission gets a new mark of -25, and the final score becomes 15 on the legitimate side. However, as a general rule in PhishRepo, a submission containing less than 50 points either in phishing or legitimate remains as a 'processed' flagged submission. Any submission with more than 50 points on the phishing label automatically upgrades to the 'verified' flag state.

Further, if a submission achieves more than 50 points on the legitimate side, the submission is sent to an expert editor for review and is responsible for the final decision. However, after a submission comes to a verified state, PhishRepo welcomes objections through the objection reporting module in the proposed solution because Hansen et al. (2013) stated that peer review of the crowd worker work is essential to having a quality labelling process. Therefore, all the user accounts except guests could raise objections to a verified phishing submission, and if there are several objections, the expert editor reviews the submission again. The expert editor could disable future objections at the review time to avoid misuse of the objection process. However, if a submission exists in the local storage for more than ten days without being verified, it

appears in the expert editors' voting board to get their attention. Figure 7.8 shows the voting board interface of an editor.

PhishRepo hides the scoring history of submission from all the editor levels and displays only the final score through a progress bar, as shown in Figure 7.9. Then the editor does not get to know any past editors. That strategy was proposed to PhishRepo because Eickhoff (2018) showed that cognitive biases are the main problem in crowdsourcing. He further mentioned that the anchoring, bandwagon, and decoy effects occur in crowdsourcing, and anchoring cause 28% of accuracy losses. PhishRepo's hidden scoring history design consideration limits the cognitive biases since the present editors do not know who voted before for a submission. Although the history is hidden in PhishRepo, the expert editor gets an additional detail called impact, which describes how many negative (i.e., legitimate) and positive (i.e., phishing) votes were earned by submission when it comes to the current state. It was designed because the expert editor needs to make his final decision. Figure 7.9 shows how the impact is displayed to the editors in PhishRepo.

Further, PhishRepo implemented another extra attempt to maintain the quality of the labelling process. That is by asking for a brief explanation about the submitted label to avoid doubtful labels. It is done by asking a simple question from the editor. If the editor marks a submission as 'phishing', then PhishRepo asks, *can you find the targeted website?* as shown in Figure 7.10. In this case, the editor can submit a yes or no.

Furthermore, the Figure 7.10 interface contains an optional field to submit the targeted website URL. If the answer is *yes*, then the optional field becomes required to collect that information. Otherwise, the submitted label is stored. If the editor votes a submission as 'legitimate', then PhishRepo asks, *can you find this website in Google search engine?* as shown in Figure 7.11. If the answer is *yes*, then the additional text-area field remains optional. However, if the editor's answer is *no* to the question, the text-area field requires three things that most influenced the editor to mark the record as legitimate. However, the primary goal of these questions is to provide the editor with another opportunity to consider the decision before finally making a submission

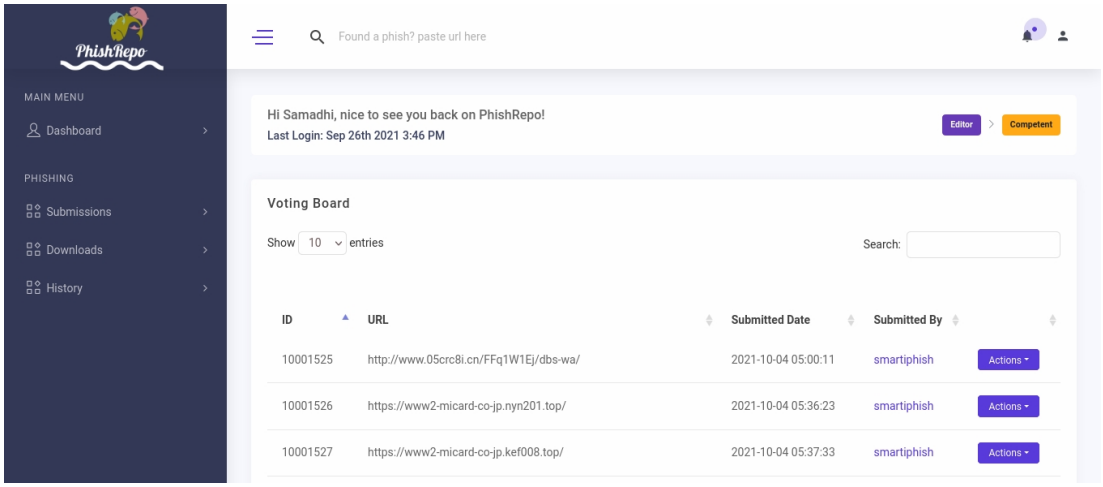


Figure 7.8: Editor's voting board

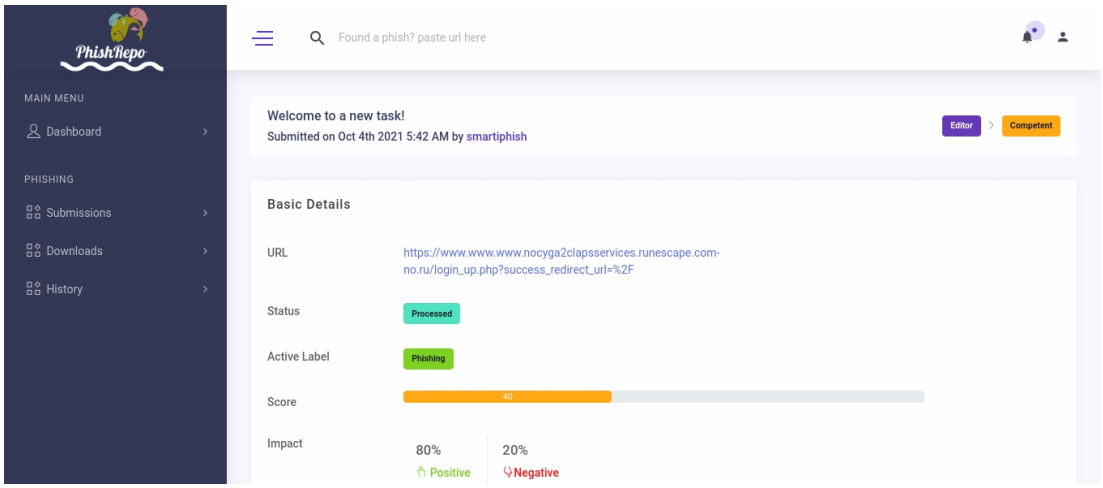


Figure 7.9: Basic details interface of a submission

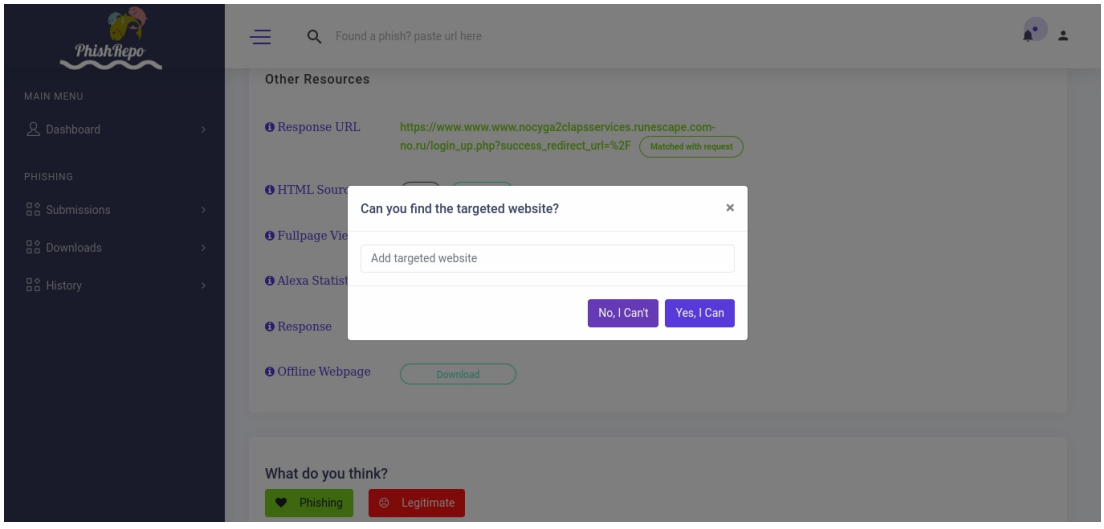


Figure 7.10: Commenting pop-up window for phishing votes

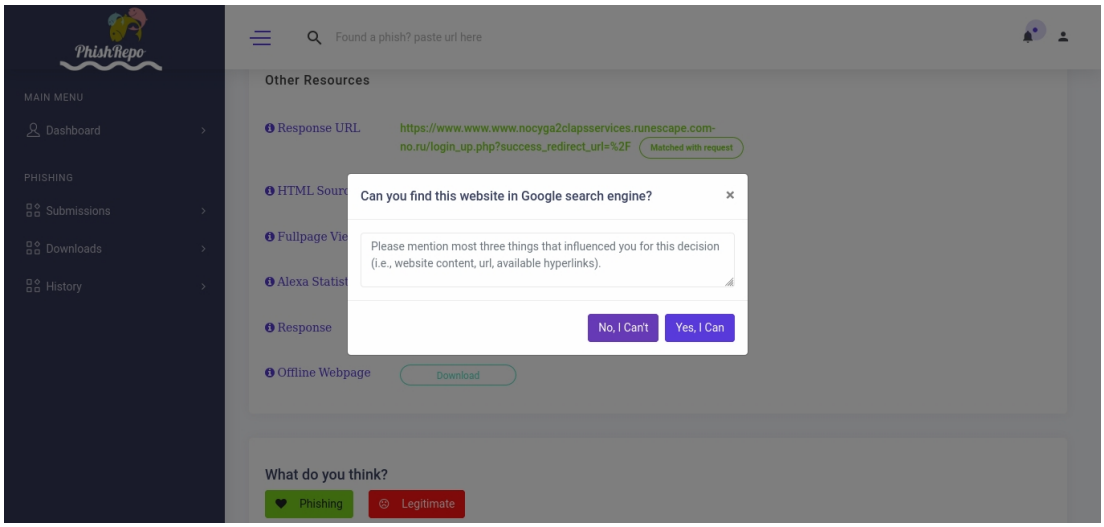


Figure 7.11: Commenting pop-up window for legitimate votes

to PhishRepo.

7.2.3 Data dissemination

Once the collected data are correctly labelled, they are ready for distribution. The data dissemination process mainly handles that distribution part. PhishRepo's main intention was to collect quality phishing data to support the knowledge acquisition process of this study. Therefore, PhishRepo was designed to distribute only phishing data.

In PhishRepo, the data dissemination process is only visible for registered accounts. The beneficiary type account was explicitly developed to aid the process of distribution. PhishRepo provides multimodal information in its raw form for individual download, as shown in Figure 7.12. These sources of information are the HTML page, visible level and full-page screenshots, response return for the made request, Alexa statistics and offline web page. Additionally, the process of disseminating data within PhishRepo can be accessed via two modules, which are the reporter subscription module and the user queries module. These methods were explicitly designed to support the proposed anti-phishing solution. The following discusses these two methods in detail.

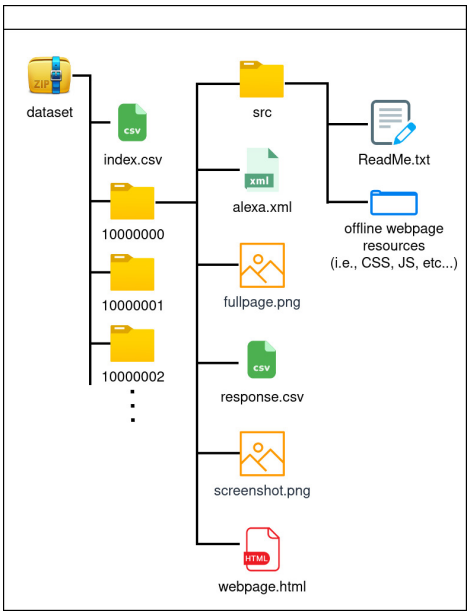


Figure 7.12: The hierarchical structure of the zip file

7.2.3.1 User queries

The users who have registered can sign in to PhishRepo and request phishing data via Figure 7.13 interface. The interface seeks the sources of information required by the user as well as the data duration. The separate menu items contain the complete downloadable dataset, and this comprises phishing records that have been verified. The user queries module can be used for selected or total data downloads. Once the user requests from PhishRepo have been finalised, the dissemination process will lead to the release of a final output (i.e., zip file), which includes an index file to make it easier to navigate. Figure 7.12 depicts the hierarchical structure of the zip file that has been downloaded. Nevertheless, special cases such as sources of information missing in some folders could arise. This could be a result of exceptions during the process of accumulation in the course of data collection. In this scenario, the file tagged 'index.csv' is essential in locating the missing items since it has eight-digit columns which track 'index.csv' file and the dataset folders, the request URL, the response URL, the data collection date, and attributes indicating the presence of the visible level screenshot, full-page view, Alexa statistic file, the offline web page, and response header file.

The user queries module is responsible for providing necessary phishing data to the

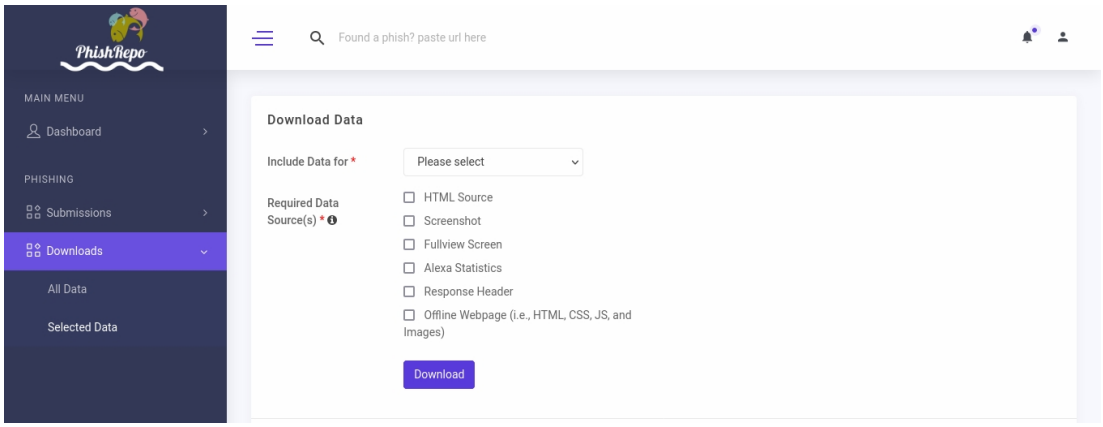


Figure 7.13: User query interface

DLM of the proposed solution. After a certain period (i.e., three months), the DLM can ask PhishRepo to produce the latest phishing records for a particular period. Since this module is configured to set required resources only, the DLM can only get the URLs and the relevant web pages from PhishRepo. Then, the DLM can retrain with new phishing data, and this process will be further discussed in Chapter 8.

7.2.3.2 Reporter subscription

The reporter subscription module primarily aims to provide a special benefit to corporate reporters to appreciate their essential contributions since they are the critical users who run the proposed solution for extended periods. As a result, PhishRepo is designed to release feedback automatically on their accounts based on the details sent to the repository. This is intentionally done to offer a source of encouragement and assistance to the corporate reporter. In PhishRepo, corporate reporters have a distinct field referred to as 'return address'. PhishRepo utilises this field value and delivers daily feedback to the corporate reporter for their submissions to make things easy for them.

Nevertheless, PhishRepo did not return the feedback the same day the submission happened. It first waits until the submission is labelled correctly, and then only the feedback is sent to the corporate reporter. It is also crucial for PhishRepo to follow up on the feedback sent by itself to prevent a situation where the feedback is duplicated. Therefore, PhishRepo usually keeps track of sent records to avoid duplicate feedback.

In PhishRepo, the feedback report takes the format of a CSV file, as shown below.

```
record_id, url, label
18333415, http://phishing-example-1.com, 1
18733122, https://phishing-example-2.com, 1
18432149, https://legitimate-example.com, 2
```

The reporter subscription module was designed mainly to support the RLM of the proposed solution. The RLM becomes a corporate reporter and submits the records to PhishRepo automatically using an application key. Since the reporter subscription module can send feedback automatically, the RLM can use the feedback given by PhishRepo to update its knowledge. Chapter 8 will discuss this more.

7.3 Target attack prevention (TAP)

PhishRepo was initially implemented to collect phishing data required for the knowledge acquisition process. Since it mainly supports the proposed anti-phishing solution, the quality of data it produces and the seamless collection of the latest phishing data should be protected from external attacks. In actual execution, denial-of-service (DoS) attacks and false data injections are potential threats to PhishRepo. Nevertheless, the network architecture depicted in Figure 7.2 improves network-level security to a certain extent in order to defend against these threats, which cannot be dealt with solely at the network level. As a result, the TAP component that has been implemented provides PhishRepo with application-level protection. In addition to the standard security practices, the TAP component employs four distinct procedures to improve application security.

1. *Application Key-based Authentication:* Users who are privileged to have an application key are enabled to participate in the automatic records submission process.

2. *High-volume Restriction Strategy:* This strategy restricts the number of submissions from a corporate account.

3. *Maximum IPR Queue Length Strategy:* This is to help limit the volume of requests by the accumulation step.

4. *False Ban Strategy:* The reporters who submit incorrect records are banned appropriately via this strategy.

The submission step, as described in Section 7.2.1.1, further illustrates that an application key is owned by the corporate reporters for the purpose of submitting a phishing URL to PhishRepo automatically. This key-based authentication limits the trend of an attack because an attacker must obtain a valid application key to gain access to the system. If an attacker has the correct application key, the countermeasures that follow try to reduce the extent of attacks.

When the number of legitimate URLs is compared to phishing URLs, the likelihood of encountering a phishing URL is relatively low (Aassal et al., 2020). This means that the chances of submitting many records in a short period are limited because PhishRepo requires real-time submission and does not encourage batch processing. As previously stated, the submissions take the form of requests, usually one per submission. This implies that the number of submissions equals the number of requests received by a reporter. Therefore, a limited number of submissions are made possible in PhishRepo for a specified period via the high-volume restriction strategy. If the reporter exceeds the defined requests per minute (i.e., ten requests) limit, his account will be banned for a few minutes (i.e., five minutes), and if this happens frequently, the account will be blocked by the TAP. TAP also reports abnormal behaviour to the administrator to take appropriate action.

On a rare occasion, if an attacker avoids the first two countermeasures, the maximum IPR queue length will be used to maintain a fixed-length queue to prevent memory overload. Then, there may be no performance hits, and PhishRepo's functionality may continue uninterrupted. Meanwhile, the accumulation step pulls URLs from the IPR queue. This pulling process implies that some submitted records may be erased without being processed in any significant attack. Although it seems wasted, PhishRepo does not intend to collect all of the submitted phishing records and will

instead work only with possible submissions when expanding the available phishing records.

Furthermore, the false ban strategy is adopted in PhishRepo to eliminate the possibility of false data injections. False data can alter the proposed solution's credibility and lead to the wastage of many resources. Hence, PhishRepo determines the validity of the phishing records weekly and evaluates the accuracy percentage based on the submission's label after the labelling process. If the rate is less than the value of a defined threshold for an account (i.e. 80%), a suspension will be placed automatically, and a report will be sent to the administrator for necessary actions.

7.4 Diversity of the collected phishing data

The primary goal of PhishRepo is to provide diverse phishing data to support the knowledge acquisition process of the proposed solution. As a result, several perspectives were used to determine PhishRepo output to know if the proposed solution achieved a wide data distribution. Section 5.3.4 explained that the varying domains, varying TLDs, the use of HTTPS, and URL character length distribution are essential when checking a diversity of a phishing dataset. Although the four criteria could be a determinant in the assessment of diversity, previous studies have yet to consider the possibility of data leakage in a specific data set, as discussed in Section 7.4.3. However, it is considered the fifth criterion because data leakage affects the proposed model's final accuracy when checking the diversity of the dataset for the PhishRepo. However, the PhishRepo dataset's diversity analysis was compared to two public phishing datasets recently used to implement high-accurate anti-phishing solutions.

Further, the diversity analysis used two versions of the PhishRepo dataset, PhishRepo and Ex-PhishRepo, alongside those public datasets to show the effect of the fifth criterion in phishing datasets. Table 7.1 shows the details of those datasets. The Ex-PhishRepo and PhishRepo datasets were collected using the proposed phishing data collection process. However, the data in Ex-PhishRepo were not filtered via the deduplication filter since the primary purpose of these two datasets was to show the effect of the deduplication filter. The PhishRepo data were collected by activating the dedu-

plication filter. Therefore, the effectiveness of the filter should be visible in those data. Furthermore, the PhishRepo and Ex-PhishRepo datasets downloaded initial phishing URLs from PhishTank and OpenPhish communities. As a result, the phishing data in these two datasets were authentic phishing instances. Moreover, the PhishRepo system archived the Ex-PhishRepo data from September 29, 2021 to October 17, 2021, while the data for PhishRepo were gathered from October 23, 2021 to February 02, 2022.

Table 7.1: Used phishing datasets' details

Name of Dataset	Number of Data	
	Phishing	Legitimate
PhishRepo	5,275	0
Ex-PhishRepo	2,029	0
Web2Vec (Feng et al., 2020)	21,296	24,800
PhishPedia (Lin et al., 2021)	29,048	22,252

When constructing an anti-phishing solution, Feng et al. (2020) utilised the Web2Vec dataset, which is an online phishing dataset. Web2Vec dataset had 21,303 phishing instances obtained from PhishTank for a period of three months, from September to November 2019 (Feng et al., 2020). However, this work could not use all phishing instances because of issues relating to data extraction. Hence, it used only 21,296 phishing instances. Similarly, the PhishPedia dataset was also used recently by Lin et al. (2021). This dataset comprises phishing web pages of about 29,496, and the premium account of OpenPhish was utilised in the course of downloading the data. The dataset is available for download by anyone; this was made possible by the authors who decided to share it with the public. Due to some issues during the data extraction, the study used only 29,048 PhishPedia phishing items out of the total phishing instances.

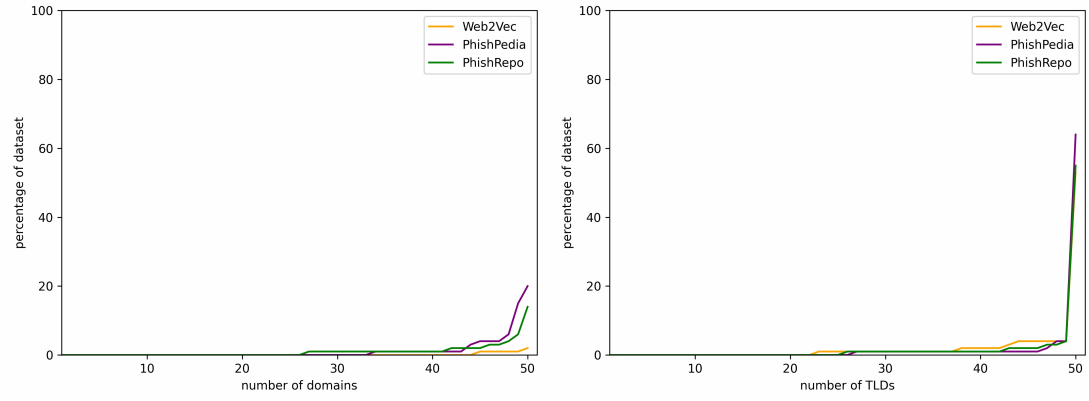
Owning to the fact that the PhishRepo solution distributes only data connected to the phishing attack, the Ex-PhishRepo datasets and those of the PhishRepo lack legitimate data, as revealed in Table 7.1. Nevertheless, Web2Vec and PhishPedia datasets have been used in anti-phishing studies in recent times. Therefore, these datasets were linked to the legitimate data utilised by those studies. Interestingly, the legitimate data source in both these cases was Alexa.

After collecting these datasets, the study examined the diversity of PhishRepo com-

pared to these datasets. As mentioned above, the Ex-PhishRepo dataset was used only to show the effect of the deduplication filter under the fifth criterion, and in all other criteria, the PhishRepo dataset was examined with Web2Vec and PhishPedia. The following sections describe each of these criteria in detail.

7.4.1 Domains distribution and TLDs

A dataset's domain and TLD distribution depend on the phishing page's URL. Hence, the study extracted these domains and TLDs from the individual dataset. Afterwards, their frequencies were evaluated individually. Then, the top fifty TLDs and domains were chosen from each dataset. Lastly, the percentage of selected domains related to dataset size was determined, and those values were plotted in ascending order to get a meaningful distribution. Figure 7.14 shows the distribution of domains (Figure 7.14(a)) and TLDs (Figure 7.14(b)) in each dataset.



(a) Domain distribution

(b) TLDs distribution

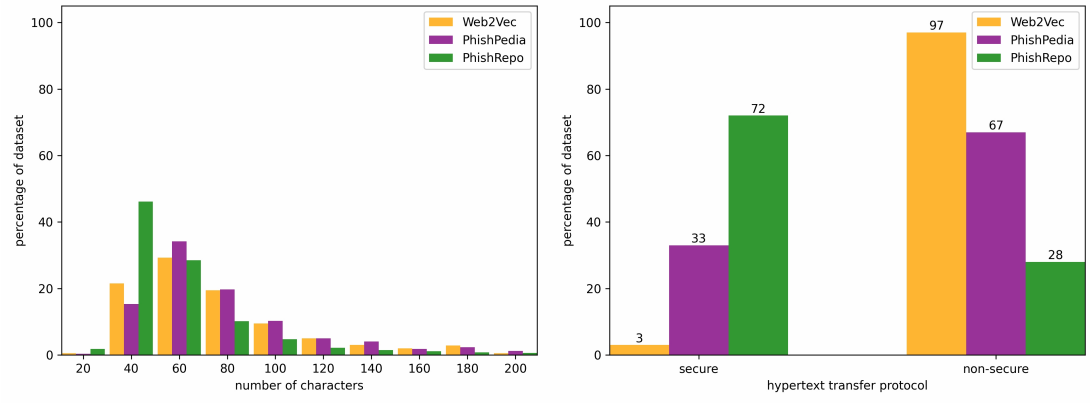
Figure 7.14: Distributions of domains and TLDs in the selected datasets

Figure 7.14 shows that the datasets have utilised over fifty different domains and TLDs. Among these datasets, a high proportion of an equal domain belongs to PhishPedia, and in relation to the whole dataset, this was 20%. 14% of the same domain belongs to the PhishRepo dataset, while the lowest number of exact domains was recorded by Web2Vec. However, the occurrence varies differently in regards to the distribution of TLD, as shown in Figure 7.14. According to Aassal et al. (2020), famous TLDs such as '.com' are often commonly used for phishing purposes. This

experiment also reveals that all the whole three datasets have utilised over 50% of the '.com' TLD. Even though '.com' ranked high in all three datasets, more than 50 diverse TLDs have been added to PhishRepo, Web2Vec, and PhishPedia datasets. Such domains and TLDs' distribution indicate a dataset that is diverse (Aassal et al., 2020). Hence, the perspective of domains and TLDs' distribution in relation to PhishRepo's dataset is diverse.

7.4.2 Distribution of URL character length and HTTPS

When detecting a phishing attack, if a dataset for phishing has no URL character length that is distributed in its standard form, as showcased in the work of Li et al. (2019), it may lead to models that are not adequate for real situations (Verma et al., 2019). In addition, the HTTPS label is usually present in over 80% of phishing attacks, as reported in the APWG (2021a) report, depicting that a large proportion of HTTPS in a phishing dataset is also essential to have a situation that is realistic during a practical model training. Hence, characters in a URL and the secure phishing URL percentage corresponding to the dataset size were evaluated, as shown in Figure 7.15.



(a) Distribution of character length

(b) Secure phishing URLs' percentage

Figure 7.15: Phishing URL character length and HTTPS distributions

Figure 7.15(a) shows that the three datasets' URL character lengths showed a standard distribution. The dataset with the character length category of about 20 to 40 had the largest number of PhishRepo URLs. The other two datasets had a high proportion of 40 to 60-character length URLs. However, the three datasets had URLs under

distinct categories. This distribution indicates that these three datasets are diverse in relation to the character length of URLs.

On the other hand, the PhishPedia and Web2Vec datasets did not have enough secure URLs. Figure 7.15(b) showed that the PhishPedia dataset had 33% secure URLs while the Web2Vec dataset had only 3% secure URLs. Nevertheless, recent statistics indicated that about 80% of phishing URLs utilised HTTPS in the trending context of phishing (APWG, 2021a). If such context is ignored in a phishing dataset, it may eventually bring about inadequate models for detecting the latest phishing attacks. As shown in Figure 7.15(b), this secure URL context does not reflect in the PhishPedia and Web2Vec datasets, so these datasets could not train fair models to detect the latest attacks. However, PhishRepo has more than 70% of the used dataset shown and has displayed a high proportion of secure phishing instances. This distribution indicates that the phishing nature currently in use is adequately absorbed by the proposed solution, and the dataset of the PhishRepo solution is updated to the present phishing context.

7.4.3 The tendency of data leakage

Among machine learning errors, data leakage has a leading role. This data leakage often occurs whenever the model's train data shows up during the testing time. It brings about poor prediction outcomes in the end. This data leakage can occur in two ways within the scope of phishing data collection. The first is that the same set of data is engaged several times. For instance, the phishing website https://xyz.com is seen in the dataset on several occasions. Secondly, it could also occur due to different URLs for equal phishing websites, as depicted in Figure 7.4. In both cases, data leakage could occur if the duplication pairs are high. As a result, the present study's data leakage tendency is measured according to the available duplication pairs in the datasets that have been used. Nonetheless, previous studies have not used that kind of test to determine the possibility of data leakage, which makes this the first of its kind.

Screenshots captured from phishing web pages were used in the course of the experiment since the phishers are constantly trying to get a fake page that is quite similar

to the target page. Hence, the present study has the assumption that web pages having similar visual features also possess the same HTML structure. In addition, the experiment used pixolution Flow[19], a commercial tool and an AI-powered search engine to search and manage visual data when trying to locate near-duplicates and duplicates. The pixolution Flow possesses a docker image which has the ability to index up to 5,000 images. This docker is used in the course of the experiment. Hence, about 5,000 random samples were thus selected from each of the datasets prior to the commencement of the experiment.

Based on the threshold recommended by the tool, a 1.0 threshold was used in the experiment when looking out for duplicates, while the search for the near-duplicates was configured to use a threshold of 0.9. Figure 7.16 presents the percentages of the duplicates and near-duplicate of the individual dataset. Nevertheless, during the initial indexing step of the pixolution Flow docker, the dataset of Web2Vec was unable to index all the 5,000 selected screenshots because twenty-two images had some problems. Hence, the percentages of the Web2Vec dataset presented are evaluated from 4,978 data items.

The results of the experiment have shown that the deduplication filter of PhishRepo was well supported to ensure that the duplicate images were reduced to zero while those of the near-duplicate kept around 20%, as depicted in Figure 7.16. In addition, the experiment reveals that the remaining datasets, such as Ex-PhishRepo, PhishPedia, as well as Web2Vec, have higher degrees of duplication than PhishRepo. Hence, this experiment attests that deduplication is vital when eliminating data leakage in the phishing data collection process, which is effectively used in PhishRepo solution. However, phishing data like in Figure 7.17 is possible in PhishRepo since the deduplication filter cannot eliminate all the near-duplicates, as discussed in Section 7.2.1.3. Furthermore, from the aspect of data leakage, the experiment reveals that the PhishRepo solution collected dataset is well suited for the machine learning-based anti-phishing tasks.

All these five experiments carried out under the five main criteria mentioned earlier

[19] `https://pixolution.org/`

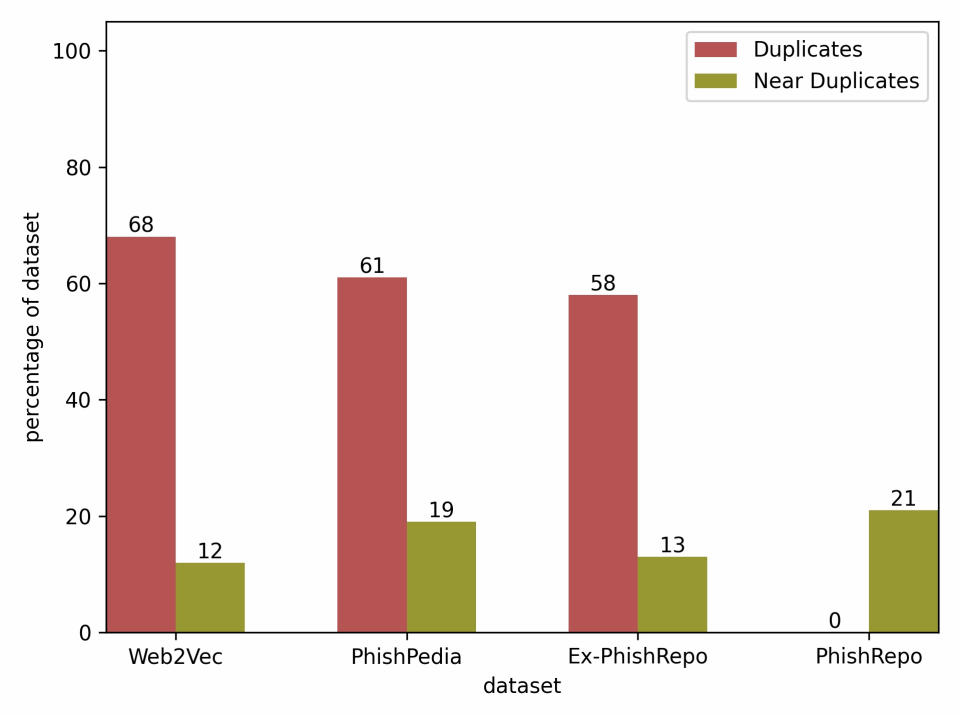
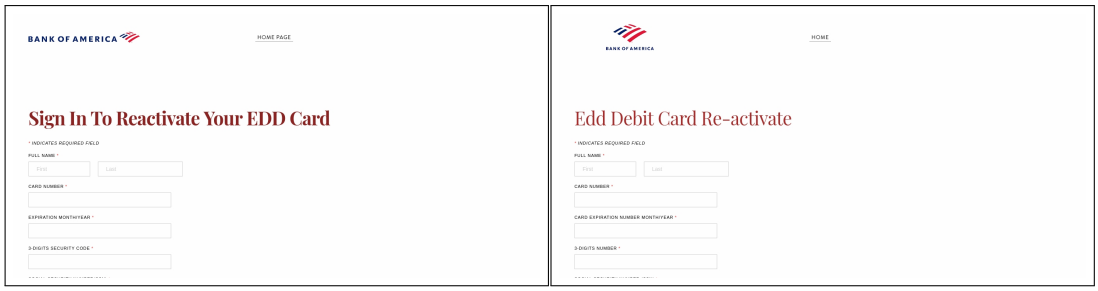


Figure 7.16: Percentage Distribution of duplicates and near-duplicates

show that the proposed phishing data collection process produces diverse data to support the knowledge acquisition process of the proposed solution. This has shown that none of the recent phishing datasets was considered a data leakage issue that highly impacts machine learning-based anti-phishing tools. However, the proposed data collection process has successfully achieved 0 duplication pairs by eliminating the critical issue of data leakage.

7.5 Effectiveness of the collected data in machine learning

PhishRepo was mainly implemented to collect diverse phishing data to support the continuous learning process of the proposed anti-phishing solution via a knowledge acquisition process. The proposed phishing detection solution was a machine learning-based solution. Therefore, the collected data through PhishRepo was evaluated with several machine learning solutions introduced in the literature to show the effectiveness of the collected data in the machine learning process. Table 7.2 shows the existing anti-phishing solutions used by the experiment, and when selecting these, the use of



(a) https://eddservicesupport.weebly.com/ [Submitted on 26 October 2021]

(b) https://edddebitcardre-activate.weebly.com/ [Submitted on 30 October 2021]

Figure 7.17: Example of a near-duplicate found in the PhishRepo dataset

machine learning and the online availability of the model implementations were highly considered.

Table 7.2: Details of the used machine learning-based solutions

Solution	Description
Desai et al. (2017) *Name*: RFAlgo*	Random Forest-based classifier that uses URL and HTML content features.
H. Le et al. (2018) *Name: URLNet*	A deep learning approach to detect malicious URLs directly from the URL.
Li et al. (2019) *Name: StackModel*	Detect phishing attacks with the support of URL and HTML content features.
Ariyadasa et al. (2020) *Name: HybridDLM*	A deep learning model uses direct URLs with manually extracted HTML content features.

*The names are used only to reference easiness, and some were self-invented.

The experiment was planned with Table 7.1 datasets. However, the Ex-PhishRepo dataset was a different version of the PhishRepo dataset. Therefore, it was not considered during this experiment. Web2Vec, PhishPedia, and PhishRepo datasets presented in Table 7.1 were considered throughout this experiment.

7.5.1 Constructing the datasets

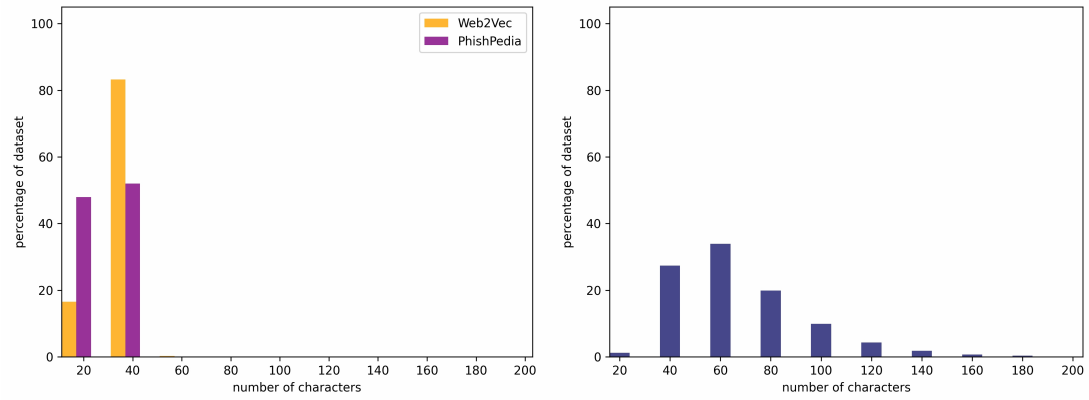
Generally, a machine learning model needs a training dataset and a test dataset during the learning process. Therefore, first, those datasets were constructed by the study. This experiment aims to show the effect of the latest phishing data when retraining machine learning models. In that case, the PhishRepo dataset had the latest phishing

data since it collected phishing attacks until February 02, 2022. Therefore, first, it was used to separate phishing samples for the test dataset. As a result, the last ten days of phishing attacks were separated from the PhishRepo dataset added to the test dataset. After that, the test dataset had 518 records. Then, the remaining data until January 21, 2022 were selected as PhishRepo's training dataset. It had 4,757 data. However, that training sample was reasonable to train a machine learning-based anti-phishing solution since Opara, Chen, and Wei (2020) also did a successful anti-phishing study with 4,700 total phishing instances.

In machine learning, the number of training samples always matters. Therefore, when the experiment was planned, the study wanted to have the same training samples from the Web2Vec and PhishPedia datasets. However, these two datasets had more than 20,000 phishing data items. Then, the study randomly selected 4,757 data from both these datasets and constructed Web2Vec and PhishPedia training datasets.

After phishing data were added to the test and training datasets, the study collected legitimate data. Since the experiment's main intention was to check the effectiveness of the existing anti-phishing models with PhishRepo collected data, the experiment used the same legitimate data in training and testing. Further, it used balanced phishing and legitimate data samples. Because of that, the test and training dataset collected 518 and 4,757 legitimate data, respectively. However, Web2Vec and PhishPedia were initially considered when collecting legitimate data since those datasets had legitimate instances. Although it was considered, the study identified that both of these datasets collected legitimate data from Alexa. In a previous study, Verma et al. (2019) mentioned that if Alexa data samples were mixed with PhishTank data samples, there might be an issue with URL length since Alexa produces the top domains in most cases by removing the sub-domains and URL path (Aassal et al., 2020). Therefore, the study wanted to confirm that such a problem might happen if legitimate data were collected from Web2Vec or PhishPedia. As a result, the URL lengths of those two datasets' legitimate parts were plotted. Figure 7.18(a) shows how those lengths were distributed. Then, it was compared with Figure 7.15(a), where the phishing URL lengths were plotted. Then, it visualised that the Verma et al. (2019) finding existed, and if these

legitimate data were used, the URL length might play a significant role when detecting phishing attacks. Hence, those two legitimate data were neglected by the study. Then, the study plotted the modern dataset's legitimate part URL's character length. It is shown in Figure 7.18(b). When compared Figure 7.18(b) with Figure 7.15(a), according to Li et al. (2019), it had a reasonable URL character length distribution. Therefore, the modern dataset's legitimate part was selected to collect 518 and 4,757 legitimate data for test and train datasets.



(a) Web2Vec and PhishPedia datasets (b) Modern dataset

Figure 7.18: Distributions of legitimate URL character length in selected datasets

After the required dataset was constructed, the training of the selected solution was then carried out. The following section describes how the training was done during the experiment.

7.5.2 Training the solutions

The experiment selected four anti-phishing models, as shown in Table 7.2. Out of those four, the HybridDLM was an output of this study discussed in Section 5.2.4. Several researchers in the past introduced the other three solutions.

The RFAlgo model is implemented initially as a chrome web browser extension to detect real-time phishing attacks. The original implementation of the solution is available online[20], and the experiment used that source. First, it required extracting

[20]https://github.com/philomathic-guy/Malicious-Web-Content-Detection-Using-Machine-Learning

features from URL and HTML content, and the downloaded resources had a script to extract those relevant features. However, some of the Alexa statistics-based features are not valid for offline training since Alexa produces data based on the last three months, and most of the phishing data used in training are comparatively old. Further, some features required some third-party services like the WHOIS service. However, these services have certain limitations when retrieving data. Therefore, the experiment did not use five features out of twenty-two features used by the original work. Those five features are domain registration length, abnormal URL, domain age, web traffic, and google index. After extracting the remaining features, three datasets trained the model and constructed three separate models for evaluation.

Next, the URLNet is a deep learning model, and it does not need any feature extraction step. The original implementation is online[21], and the experiment has downloaded it from the appropriate location. The URLs were directly inputted to URLNet in the original implementation. However, when passing the URLs to the model for training, it has a defined format to use. The models were trained after constructing the URLs in a suitable format. The experiment used the provided execution script, and it trained the model in different embedding values. Therefore, each dataset had several models based on the used embedding values. However, all the models were used for the evaluation task since there is no standard way to filter the most effective model.

After that, the experiment used the StackModel. It is the first stack model-based phishing detection solution introduced in the literature. However, the original implementation of the solution could not be found, and a separate implementation[22] of the solution was shared by Lin et al. (2021). That implementation was used by Lin et al. to benchmark their solution with StackModel. Since it is an accepted work in the literature, the experiment used Lin et al. StackModel implementation for this experiment. The StackModel also required a feature extraction as the first step. The experiment used the feature extraction script to extract training and test dataset features. Then, the provided training scripts were used and constructed three separate models based on Web2Vec, PhishPedia and PhishRepo datasets.

[21] `https://github.com/Antimalweb/URLNet`

[22] `https://drive.google.com/drive/folders/1T4uHRxbUk5kXcJrq68mZ-ezWSQgse`

Finally, HybridDLM was selected. It is one of the outputs of this study. Therefore, all the required scripts to extract relevant features were available. However, HybridDLM did not require URL feature extraction. As a result, HTML features were only extracted. After the feature extraction was done with all three datasets, three separate models were trained using three datasets.

Since all the three datasets had phishing data in raw formats, the experiment easily constructed the required features with the support of the provided feature extraction scripts. After the models were trained, the evaluation was carried out with the test dataset. The following section discusses the results achieved by the trained models with the test dataset.

7.5.3 Performance evaluation

Table 7.3 shows the results achieved by every model with three datasets used during the experiment. As mentioned in Section 7.5.2, the URLNet experiment had a different set of models based on the used embedding values. After the evaluation was done with all those models, the best-performed model during the evaluation process was selected. It is presented in Table 7.3.

Table 7.3: Performance of the trained models with the selected datasets

Solution	Dataset	Accuracy	*f1*-Score	FNR
RFAlgo	Web2Vec	57.34%	27.30%	0.840
	PhishPedia	56.95%	26.16%	0.847
	PhishRepo	**78.19%**	**76.80%**	**0.278**
URLNet	Web2Vec	72.20%	70.79%	0.326
	PhishPedia	78.09%	77.72%	0.236
	PhishRepo	**82.24%**	**83.45%**	**0.104**
StackModel	Web2Vec	58.11%	36.55%	0.759
	PhishPedia	75.87%	69.96%	0.438
	PhishRepo	**89.00%**	**88.97%**	**0.112**
HybridDLM	Web2Vec	67.18%	51.98%	0.645
	PhishPedia	88.22%	87.07%	0.207
	PhishRepo	**93.92%**	**93.89%**	**0.066**

As shown in Figure 7.19, the PhishRepo dataset has shown high accuracies and *f1*-score with all four existing models compared to the other two datasets. The lowest

accuracy was recorded with RFAlgo, and the highest was recorded with HybridDLM. The removal of five significant features from RFAlgo's original work during the training might be a reason to have low accuracy of that model.

Further, FNR is a significant metric in phishing detection since phishing as legitimate has a high impact on phishing. Figure 7.19 shows that FNR is comparatively low in all four cases where the PhishRepo dataset was used. It implies that the models effectively learned most phishing scenarios during training. It also indicates that the PhishRepo dataset is more effective when presenting phishing examples during the training than the other selected datasets. Furthermore, the datasets size, legitimate examples, and test dataset were constant in all cases during the experiment. Therefore, it is clear that the phishing examples made a performance difference in each case. It indicates that the PhishRepo data are well-suited for machine learning-based anti-phishing studies since it presents more diverse phishing examples during the training time to have a high detection rate. It guaranteed that the proposed phishing data collection approach succeeded in collecting phishing data to support the knowledge acquisition process of the proposed solution.

7.6 Discussion

The main idea behind this phishing data collection process was to support the proposed knowledge acquisition process. Since the study primarily identified phishing data collection as a challenging task, a systematic approach was designed. According to the literature, there was no suitable solution to support systematic phishing data collection. However, Phisherman was one such attempt that did not succeed in the past. The study analysed the Phisherman project and implemented a systematic process called PhishRepo by going beyond the Phisherman. Therefore, PhishRepo is conceptually better than Phisherman in many design considerations, as summarised in Table 7.4.

PhishRepo is especially getting the support of automated submission architecture, and it produces diverse information sources in raw format due to its design which is essential to the proposed anti-phishing solution. Further, the deduplication filter that guarantees diverse data collection and the elegant labelling process used in PhishRepo

(a) RFAlgo Accuracies (b) RFAlgo f1-score (c) RFAlgo FNR

(d) URLNet Accuracies (e) URLNet f1-score (f) URLNet FNR

(g) StackModel Accuracies (h) StackModel f1-score (i) StackModel FNR

(j) HybridDLM Accuracies (k) HybridDLM f1-score (l) HybridDLM FNR

Figure 7.19: Distribution of Accuracy, f1-score, and FNR for the selected solutions under different datasets

The graphs display the performance metrics (Accuracy, f1-Score, and FNR) for multiple solutions (RFAlgo, URLNet, StackModel, and HybridDLM) on various datasets (Web2Vec, PhishPedia, and PhishRepo). Each graph represents the corresponding metric's values for a specific solution across different datasets. The x-axis shows the dataset names, while the y-axis represents the metric values. The color-coded bars help differentiate the solutions for easy comparison. The graphs provide insights into the relative performance of each solution on different datasets for the selected metrics.

Table 7.4: Comparison of Phisherman and PhishRepo

Criteria	Phisherman	PhishRepo	Remarks
Initial data submission	Manual + Automatic	Manual + Automatic	Phisherman's data submission is primarily anonymous and multi-format, but PhishRepo has a standard format for automatic submissions, and the submitter is easily traceable due to application key authentication.
Initial Input	Email	URL	Phisherman has different data workflows due to the multi-format emails it gets, but PhishRepo uses URLs at the initial level, so the data workflow is standard for all submissions.
Data format	Preprocessed	Raw	Phisherman converts every submission into IODEF report format, but PhishRepo archives in raw format.
Missing features	High	Low	Phisherman saves inactive phishing website data to the database, but PhishRepo stops processing such submissions.
Deduplication filter	Not available	Available	Deduplication filtering is an important component of eliminating redundant data collection, and PhishRepo has integrated it effectively into the data collection process.
Verification	Two-step	Two-step	Phisherman's first verification is for selected submissions, and the second depends on a set of heuristic rules. PhishRepo uses two well-known verification services first and crowdsourcing second.
Objection report	Not available	Available	Objection report is essential to report incorrect labels since both use automatic verification.
TAP	Depends on network architecture	Network-level + application-level	PhishRepo introduces a layered architecture at the application level to prevent targeted attacks.
Malicious submission detection	Not available	Available	PhishRepo tries to avoid users who frequently submit erroneous submissions, but Phisherman does not mention such a constraint in their architecture.
Data distribution	Web view and XML file	ZIP file and CSV file	Phisherman data could be viewed on a web interface or downloaded as an XML file. PhishRepo distributes data via a ZIP file. It includes all raw data for regular users, and corporate users get additional CSV for their submitted records.

produce quality data at the end. The objection reporting supports maintaining the quality further. Moreover, the innovative data distribution structure is purposely designed to attract users, primarily autonomous anti-phishing tools, to live the solution long-term. Further, from the security perspective, the proposed network architecture and TAP strategies are vital for the smooth running of the solution.

PhishRepo provides multi-modal information sources in raw format. It was designed like that since the proposed solution used representation learning. Therefore, raw data are essential during the knowledge acquisition process. The experiment in Section 7.5 also highlighted the benefit of raw data when PhishRepo data was used with different existing solutions. One goal of PhishRepo was to collect diverse phishing data to use in the knowledge acquisition process. The deduplication filter introduced in the data collection process is vital since the Section 7.4.3 experiment highlighted that the PhishRepo data did not contain any duplication pairs.

Further, Section 7.5 experiment also highlighted the importance of the deduplication filter from a different perspective. According to that experiment, the amount of learning a model can gain through the training set becomes lower if high duplicate phishing instances are available since several data items produce the same knowledge during the training. Therefore, the PhishRepo data produce more knowledge than the other two datasets used during the experiment, and it might be a reason for such accuracy recorded by all the existing models with PhishRepo since all the other datasets reported more than 50% duplication pair rate in a similar amount of training samples.

Although the used models got the deduplication filter support, one might argue that high accuracy was due to the latest phishing examples the PhishRepo data contained. In that perspective, PhishRepo is aligned with its goals because the main objective of PhishRepo was to collect the latest phishing data to support the proposed knowledge acquisition process since such an approach was lacking in the literature. The experiment results showcase that if a model is trained with old phishing data, that model may not perform well with the latest phishing attacks. The main reason behind this conclusion is the constantly changing nature of phishing attacks that change significant phishing detection features over time. That was what exactly happened during

the performed experiment in Section 7.5. Since the test dataset contained the latest phishing attacks and both Web2Vec and PhishPedia had old phishing examples, current significant phishing detection features might not be captured during the training. However, PhishRepo is constantly collecting these phishing examples. Therefore, it contained the latest phishing examples, and more significant characteristics existed in PhishRepo examples to detect the latest phishing attacks. Thus, the models trained with the PhishRepo dataset captured these new characteristics to perform well during the experiment.

Although PhishRepo inherited some of the concepts from the Phisherman project, PhishRepo is a unique solution, and it has introduced innovative design concepts throughout the process of achieving its goal:

1. The unique auto submission system is introduced by thinking about the future of anti-phishing tools since anti-phishing is more towards the autonomous side, and the data requirement is high in those solutions to work independently. However, it directly assists the RLM in getting feedback from human users' about its actions.

2. The design considerations used by the data collection process, especially the deduplication filter, which was introduced the first time in the phishing domain when collecting data, support PhishRepo to produce a diverse dataset with zero tendencies of data leakage. Since the proposed solution was a machine learning approach, zero tendencies of data leakage are essential to visualise the reality of the solution.

3. The labelling process used in PhishRepo uses the crowdsourcing technique effectively by addressing some of the limitations in crowdsourcing, like cognitive biases in personal impression collection. It ultimately produces quality data for the knowledge acquisition process.

4. The two-step labelling process reduces human users' workload and requests expert involvement only once required. It is essential in the phishing domain since experts are scarce.

5. The interactive data distribution module introduced by PhishRepo helps disseminate the collected data to support continuous learning of both DLM and RLM.

Consequently, according to Section 7.4 experiments, the PhishRepo data are more diverse and absorb the current phishing nature. Further, PhishRepo proposed a high-quality labelling process that guaranteed the quality of phishing data distribution. Moreover, PhishRepo data are more effective in machine learning-based studies, as shown in Section 7.5 experiment. Therefore, it is clear that the proposed PhishRepo architecture could produce quality diverse phishing data for the planned knowledge acquisition process to support continuous learning of the proposed anti-phishing solution.

However, the reliability of PhishRepo mostly depends on the submissions it gets. Therefore, the reporters are vital in the proposed architecture, and corporate reporters are essential since PhishRepo promotes real-time submissions rather than manual ones. The editor, especially the crowd user, plays another crucial role in PhishRepo and is always essential to the success of the beta labelling process. However, Alpha reduces the need for a beta. Therefore, PhishRepo assumes that a few editors can manage the beta labelling if Alpha works as expected. However, the reporters' and editors' contributions are vital in PhishRepo to continue the phishing data collection process.

7.7 Summary

As the first step towards a systematic knowledge acquisition process, this chapter introduced a phishing data collection process named PhishRepo. Generally, legitimate data are common on the Internet, but collecting phishing data is challenging due to the lift-time of these attacks. Therefore, PhishRepo uses a systematic approach to collect, verify, archive and disseminate phishing data to support the planned knowledge acquisition process. PhishRepo was only the first step of the proposed knowledge acquisition process, which was mainly implemented to provide the data needs. At the same time, the next chapter discusses the other essentials for a successful knowledge acquisition process. In addition, the next chapter also introduces the outcome of this study by integrating the knowledge acquisition process with RDLM for effective phishing detection.

8 PROPOSED ANTI-PHISHING SOLUTION

8.1 Introduction

This study aimed to implement an autonomous anti-phishing solution that automatically updates existing phishing detection knowledge through a systematic knowledge acquisition process. As a first step towards that, Chapter 6 introduced the RDLM, the core component used to detect phishing attacks in the proposed architecture. Following that, Chapter 7 introduced PhishRepo to generate the data required for the knowledge acquisition process, which is the next step in achieving the study's aim. However, PhishRepo is only one step in this knowledge acquisition process. Therefore, this chapter first introduces the complete knowledge acquisition process proposed in this study. Following the introduction of that process, this chapter presents the study's outcome: the integration of RDLM and the knowledge acquisition process. The chapter then discusses some of the proposed solution's challenges in real-world execution and the enhancements made to address those challenges.

8.2 Knowledge acquisition process

This study identifies knowledge acquisition as an integral part of a successful phishing detection approach since phishing attacks continuously change. Therefore, the study proposed a five-step process to acquire the relevant knowledge to update the existing phishing detection features used by the proposed phishing detection approach. The following sub-sections introduce these steps in detail.

8.2.1 Data production

Data is the main ingredient for successful knowledge acquisition. Therefore, the proposed knowledge acquisition process also starts from a data production step. Since

this process intends to support the proposed anti-phishing solution, the study wanted to produce phishing and legitimate data in this step. The study identified that the RLM is the key component to producing this required data since it interacts with the natural web environment. Therefore, the RLM is given the key responsibility of producing phishing and legitimate data.

According to the RLM architecture, it always gets a website URL from outside. It can be either legitimate or phishing. Once a new URL comes to the RLM, it sees additional observations of this URL: phishing probability, community decision, and Alexa rank. Then, it produces a new state for the RL agent, as discussed in Section 6.3. Once the new state appears, the RL agent acts on it with the existing knowledge and takes some action. After that step, the RLM is configured to save both the state and the URL into local storage for future use. The data production for the proposed knowledge acquisition process happened at this level. There are two database tables named 'data' and 'states' in the current implementation, as shown in Figure 8.1. The RLM saves the URL and metadata like the active label of the URL and submission date to the 'data' table. It saves the state and RL agent's action to the 'states' table. Since the agent's action is not yet verified, the active label of the 'data' table always fills with the community decision value in the initial saving step. Figure 8.1 shows that these two tables keep the data required to train the DLM and RLM of the proposed solution.

However, the data saved in this step are not correctly labelled by an external party. Since those are not reviewed externally, and wrong labelled data might affect the learning process, those data cannot be directly used for the future steps of the knowledge acquisition process. Therefore, once data is produced in this step, it is submitted to two external services to get a proper label. The data submission step discusses these services in detail.

8.2.2 Data submission

The proposed process labels the first step data through PhishRepo and Google Search. PhishRepo is the phishing data collection process introduced in Chapter 7, and it handles the main labelling process of the proposed solution. Google Search is a simple

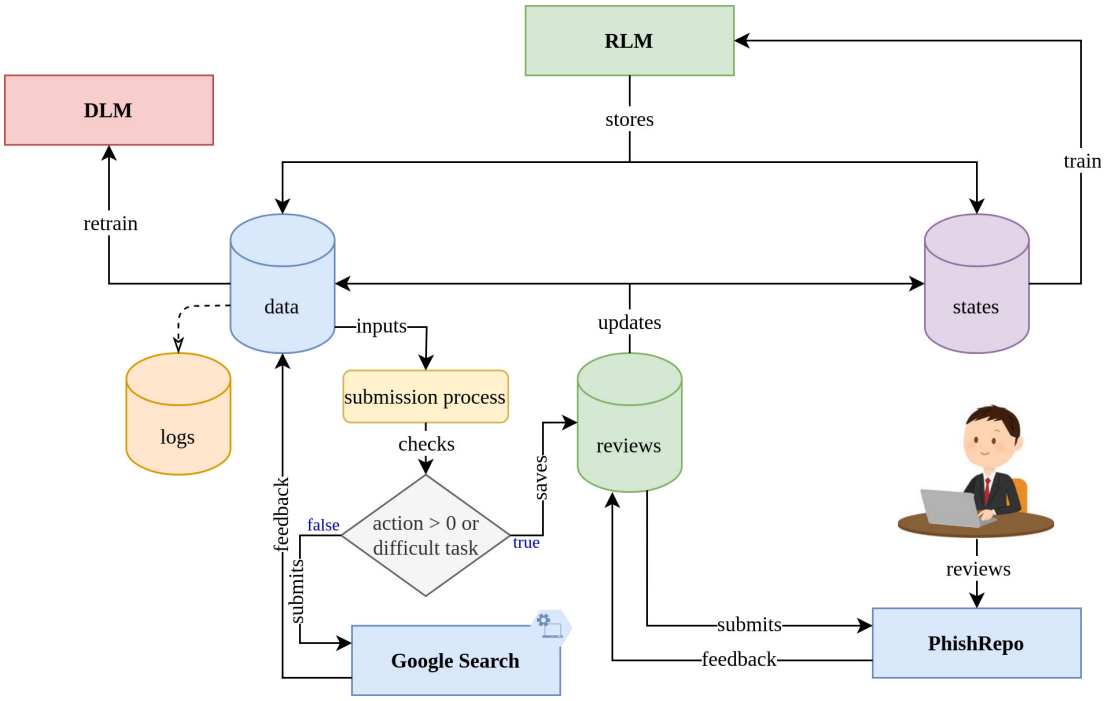


Figure 8.1: Proposed knowledge acquisition process

pythonic script (*see* Appendix B - Google Search script) that executes a simple Google search on a provided URL. The main intention of this script is to verify whether a given URL is legitimate. Therefore, it is configured to return whether the given URL is available in the first five search results it received from 'google.com'. This approach was proposed to label legitimate data because some of the previous studies (Zhang et al., 2007; Dunlop et al., 2010) have used a similar approach when finding legitimate websites. Although it was practised previously, the literature has shown that the website's popularity highly affects the final result of this approach (Jain & Gupta, 2017). However, that limitation will not affect the proposed solution since PhishRepo feeds such samples if the RL agent misclassifies those.

Once the RLM saves data in the previous step, the current step checks two conditions: the agent's action and entropy value. If the agent decides the saved URL is a phishing one or if the entropy value is high (i.e., > 0.8), the submission step submits the URL to PhishRepo since it has a high possibility of becoming a phishing attack. If both mentioned conditions are not fulfilled, the URL will be submitted to Google Search. As mentioned in Chapter 7, PhishRepo submission takes more than 24 hours to get a

proper label. Therefore, once a URL is submitted to PhishRepo, the proposed knowledge acquisition process needs to wait for a minimum of two days to get a response. Therefore, the proposed process introduced another database table named 'review' to keep track of PhishRepo submissions. The 'review' table is responsible for keeping track of data items submitted to the PhishRepo.

However, Google Search does not need such storage since it can send instant responses for the submitted URLs. Therefore, once the proposed process responds, it directly updates the active label of the 'data' table. After the data submission step, the data enters into the labelling step.

8.2.3 Labelling

The labelling step was introduced to visualise the labelling process of the submitted URL. However, the labelling is done by either PhishRepo or Google Search. Chapter 7 explained PhishRepo's labelling procedure for a submitted item. Google Search is labelling the submissions using the Google search results as explained earlier. Once these defined procedures are followed, the submitted submission is labelled. However, if unlabelled data exists in the legitimate category and is later found as a phishing web page via community decision, then such data is moved to a separate database table called 'logs' during the knowledge acquisition process. This scenario is possible in the proposed approach since active learning will not label all the produced data. In such a scenario, the SmartiPhish administrators can investigate and label these records manually.

8.2.4 Data construction

Once the labelling is done, the relevant label for the submitted data will return to the knowledge acquisition process. It can be instant or delayed, as discussed previously. Once the label is received, the data construction step updates the 'data' table's active label field. However, that is not the main functionality of the data construction step. The data construction step is to collect required data sources for retraining DLM and RLM. As discussed in Chapter 6, the DLM requires the URLs and the relevant web

pages. The RLM requires a state and feedback, which means the correct label since the reward function depends on the agent's correctness.

The data construction step maintains a specific flag called 'retrain' with 0 as the default value to filter the eligible data items for the learning process. This flag is available in the 'data' and 'states' tables and has three values: 0, 1, and 2. Once a record is saved in local storage in step one, this flag sets to 0. Then once the data item is properly labelled, the step updates the flag value to 1 in the 'state' table since the value one denotes the item's eligibility for the next learning cycle. However, the 'data' table retrain flag updates only if the item is appropriately labelled, and required information sources for DLM learning are fully collected since the web page is required when training the DLM. After the data item's knowledge is acquired within the process, the flag is automatically updated to 2.

Once a response is received by a service, if it is from Google Search, the data construction step downloads the relevant web page and updates the 'retrain' flag accordingly. However, if it is from PhishRepo, the process is slightly changed. Since the knowledge acquisition process access PhishRepo as a corporate user, the response is received from PhishRepo's reporter subscription module (Section 7.2.3.2). Then, the process gets a CSV file with several past submissions. Generally, the CSV file contains PhishRepo's record identification number, submit URL, and the given label. The PhishRepo's record identification number is saved under the 'review' table for future use, and both the 'data' and 'states' tables update the 'retrain' flag based on the label. Since PhishRepo is configured to collect multi-modal information sources, including the web page, the data construction step collects the relevant web pages from PhishRepo using the 'review' table support. PhishRepo's user query module (*see* Section 7.2.3.1) is used in this task.

In the proposed knowledge acquisition process, steps one to four are repeated every time a new submission enters the RLM. Therefore, these four steps come under an interactive process. However, the final step of the proposed process is not an interactive one. It is a batch processing step. It waits for a particular period until the first four steps collect a certain amount of data and then process those at once.

8.2.5 Automatic knowledge acquisition

The primary responsibility of the automatic knowledge acquisition step is to help both the DLM and the RLM to acquire new knowledge from newly collected data. As shown in Figure 8.1, the 'data' table provides data required to train the DLM, and the 'states' table is responsible for training the RLM. In the proposed process, both these training happens in batches. Therefore, a set of data collected through the first four steps of this process will be used to train the DLM and the RLM in the fifth step. Therefore, this step is mainly associated with how the RLM and the DLM continuously learn from the latest data to have an up-to-date phishing detection solution. The following sections discuss the learning of DLM and RLM in detail.

8.2.5.1 DLM learning process

According to Chapter 5, the DLM uses representation learning when extracting the significant features during the learning process. In that situation, the responsibility of the knowledge acquisition process is to provide data. As mentioned in the previous section, the 'data' table proposed in this study is directly linked with the DLM's learning process. Therefore, once the DLM learning cycle is planned, the 'data' table produces the required data.

The DLM learning is not frequent, and the study was decided nearly three months between two learning cycles. However, that interval was decided based on the DLM's initial retrain experiment done during the study. That experiment affected several factors. First, the literature has highlighted that the performance of a phishing detection model is degraded after two months (W. Chen et al., 2018; Opara, Chen, & Wei, 2020). However, the DLM is a deep learning model. Therefore, it needs a considerable amount of data to execute a learning cycle. The study faced some difficulties when collecting new data within two months.

Furthermore, the DLM requires high computational power due to the used architecture, and it runs for several days once the learning cycle starts. Therefore, two months is not practical for the study to achieve such high-end resources due to the resource constraints of the study. However, after three months, the study executed a learning

cycle on DLM. After the experiment, it was discovered in the study that the DLM performance was decreased. Further, that experiment has shown that the new learning cycle has improved the model performance. Since the three-month interval seems practical, the same frequency was followed several times by the study. Then, similar behaviour, decreasing and increasing performance was noticed, as shown in Figure 8.2. The October retrain[23] has shown that the performance has dropped in percentage compared to previous terms. However, after the learning process was executed, the DLM retained the normal accuracy level, as shown in Figure 8.2. Since the three-month interval was succeeded several times during the study, it was defined as a feasible interval to execute the DLM's learning cycle.

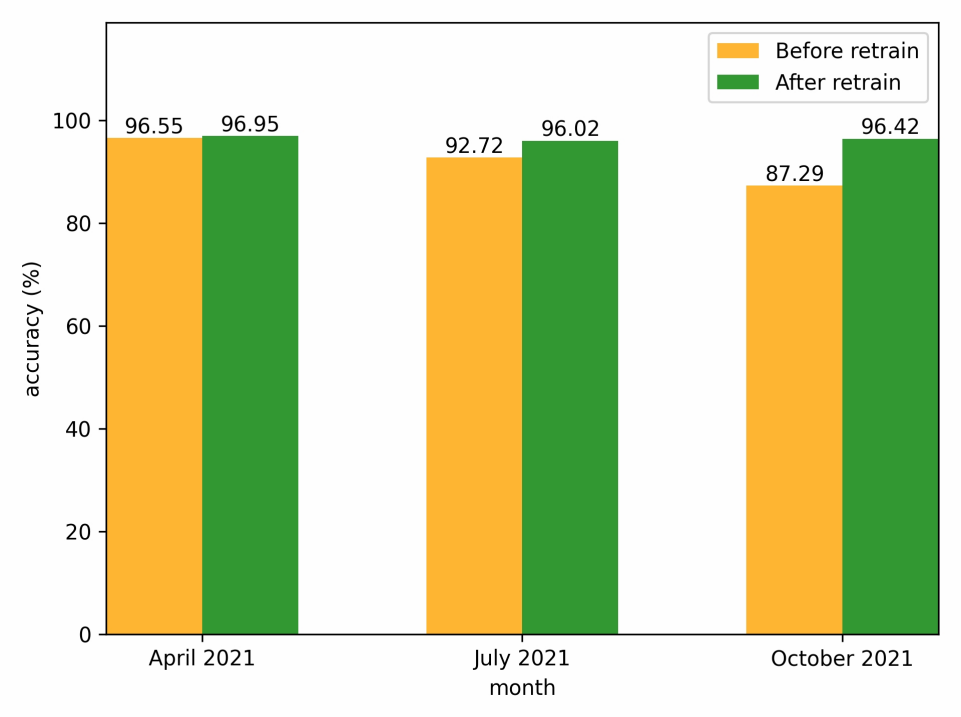


Figure 8.2: DLM's performance during the continuous learning process

After every three months, the performance of the DLM was reduced by a few percentages. However, after retraining, the model regained its performance.

The DLM learning happens as a batch processing task. First, the eligible data is selected from the 'data' table to construct the batch. That could include only the new data. However, the DLM learning process always requires new and old data to

[23]The October scheduled retrain was executed on November. However, it used phishing data collected until October 31, 2021.

avoid catastrophic forgetting phenomena associated with deep learning environments. Therefore, once the newly collected data is selected, several old data are mixed with those representing the batch's present and past knowledge. However, the learning process always maintains balanced phishing and legitimate data since Aassal et al. (2020) mentioned that imbalanced phishing datasets might affect the final performance of the model.

Once the dataset for the learning cycle is finalised, first, the data is preprocessed by the process. Then, the preprocessed data will be forwarded to a new learning cycle. Once the new learning is done, the process evaluates whether the new model is better than the past. If the new DLM is improved, the old DLM will be replaced by the new one. Otherwise, the old model continues its service until the next learning cycle. Table 8.1 summarises the proposed DLM's learning process.

8.2.5.2 RLM learning process

RLM learning is not occasional like DLM. It happens every midnight. As mentioned earlier, the learning process is linked with the 'states' table that stored previously seen states and the correct label acquired through the knowledge acquisition process. Since the RLM learning happens as batches, a batch of eligible data from the 'states' table is selected every night. However, the number of phishing data that could be collected within a day is lower compared to the number of legitimate data (Aassal et al., 2020). Therefore, a batch might contain many legitimate data compared to phishing data. Then, it results in many legitimate instances compared to phishing in the buffer maintained by the RLM. Since the study uses a limited buffer due to resource constraints, it might affect the ultimate goal of the experience replay technique. Therefore, the study proposed a 1:1 legitimate to phishing ratio when constructing the batch. It was achieved using the number of phishing data collected within a day.

When constructing a batch for the RLM learning, first, the process calculates the number of eligible phishing data. Then, the batch is constructed by adding the phishing data with the same amount of eligible legitimate data. Once the batch is constructed, the RLM processes the batch to train the DQN with new data. Finally, the newly

Table 8.1: DLM's learning process

Step 1: Construction of the dataset for the next learning cycle • Filter eligible data for the current learning cycle from the 'data' table • Randomly collect a set of old data from the 'data' table • Balance the legitimate and phishing instances using the 'data' table • Split the dataset randomly into two sub-datasets as training and testing in 4:1 ratio, and 10% of training data are again randomly split as the validation dataset
Step 2: Preprocess the datasets to have DLM compatible data format • Convert URLs to have the required numeric format • Convert HTML pages to graphs
Step 3: Training the DLM • The training and validation dataset is used to retrain the DLM • The early stopping technique is configured to avoid overfits during the retraining
Step 4: Performance evaluation • The test dataset is used with old DLM and new DLM • Compare the performance of the old DLM and new DLM
Step 5: Deployment • Backup old DLM into a backup location • If the new DLM has improved the performance, a new DLM is deployed to continue the service • Otherwise, the old DLM continues the service • Restart RLM to integrate the DLM changes

trained DQN is saved to be used in real-time. Once the learning cycle is completed, the 'retrain' flag of the data used during the training is updated to the next level. Then, the same data will not be available for the next learning cycle. However, those data are now in the buffer. Therefore, those will be used several times randomly in RLM's future learning cycles. Table 8.2 summarises the proposed RLM's learning process.

8.3 Autonomous anti-phishing solution

To challenge the identified research problem, the study aimed to implement a continuously learning phishing detection solution to detect phishing attacks with high accuracy. It was achieved mainly in two steps. First, a new phishing detection approach was implemented. Then, a knowledge acquisition process was integrated with the proposed solution to have continuous learning support. Finally, it became an autonomous phishing detection solution that can automatically acquire the newer phishing detection knowledge to adjust the model accordingly when detecting the latest phishing attacks. However, for referencing ease, the final output of the study was named SmartiPhish.

8.3.1 SmartiPhish

Figure 8.3 shows the overview of the SmartiPhish solution. It combines the DLM, the RLM and the knowledge acquisition process. The primary input to the SmartiPhish is a URL. Once the SmartiPhish receives a URL, it starts its process to make a final decision on it. Since the RLM is the decision-maker in SmartiPhish's process, the SmartiPhish produces three outputs: 'Allow Access', 'Stop Access', and 'Ask User'.

When a URL reach SmartiPhish, it executes three parallel processes: DLM, Community Decision (ComD), and Alexa. The DLM produces the phishing probability for the given URL and sends it to the RLM. The ComD accesses PhishTank and GSB APIs to check the status of the submitted URL and then processes these results to return the final decision about the URL. The Alexa service generally sends an XML file with available statistics, and the process itself extracts the website's global rank from the file and converts it into a standard format using Equation 6.1 formula and sends the value to the RLM. After all these processes return the relevant values, the SmartiPhish

Table 8.2: RLM's learning process

Step 1: Construction of the dataset for the next learning cycle • Filter eligible data for the current learning cycle from the 'states' table • Count the number of phishing instances available in the dataset • Balance the legitimate and phishing instances using the 'states' table
Step 2: Train the RLM • Acts on the available states in the dataset • Generate rewards according to the agent's actions • Train the DQN based on the agent's correctness
Step 3: Deployment • Save final DQN to the local storage once the agent acts on all the states available in the dataset • New DQN continues the service

constructs the state as defined in Chapter 6. Then, the state appears to the RLM, and the agent acts on it and produces action. The action returns to the SmartiPhish, and it then passes to the appropriate location as SmartiPhish's decision for the submitted URL. At the same time, the SmartiPhish invokes the proposed knowledge acquisition process to construct data for the next learning task.

Further, the RLM learning process is executed by SmartiPhish every midnight, and the DLM learning process runs after three months. Figure 8.4 summarises SmartiPhish's information flow.

8.3.2 Defense against adversarial attacks

As highlighted in Chapter 3, adversarial attacks are a key challenge for an anti-phishing solution since such attacks can break the trustworthiness of a solution. Therefore, the study extended the SmartiPhish solution to defend against adversarial attacks. The study proposed a GAN-based approach in this case to minimise the effect of adver-

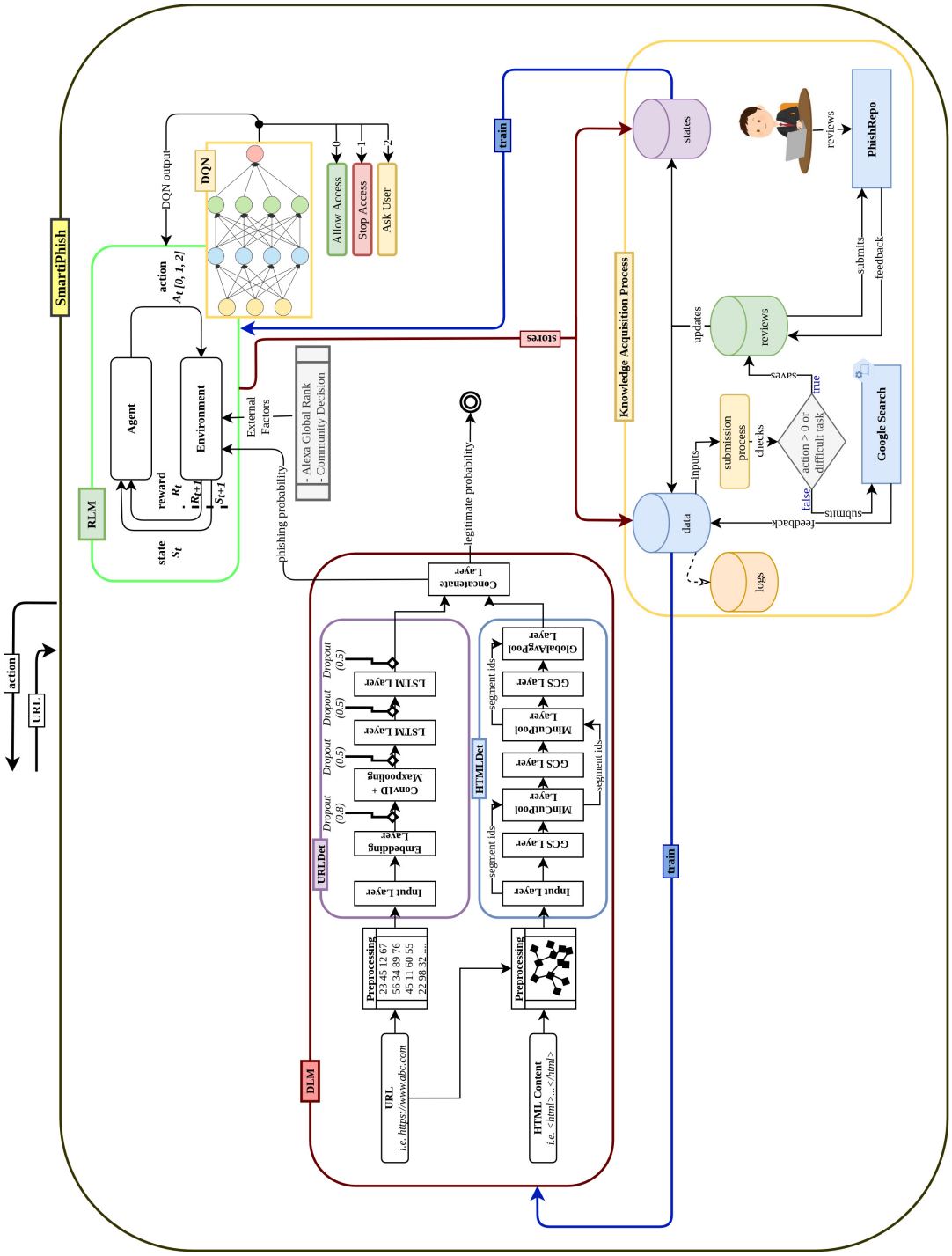


Figure 8.3: SmartiPhish solution

sarial attacks. However, the proposed approach could not defend against all types of adversarial attacks mentioned in Chapter 3. It was trained to detect specific inputs designed to get wrong results from SmartiPhish. However, it did base on certain assumptions. Mainly, the study assumed that the adversarial attacks are generated from

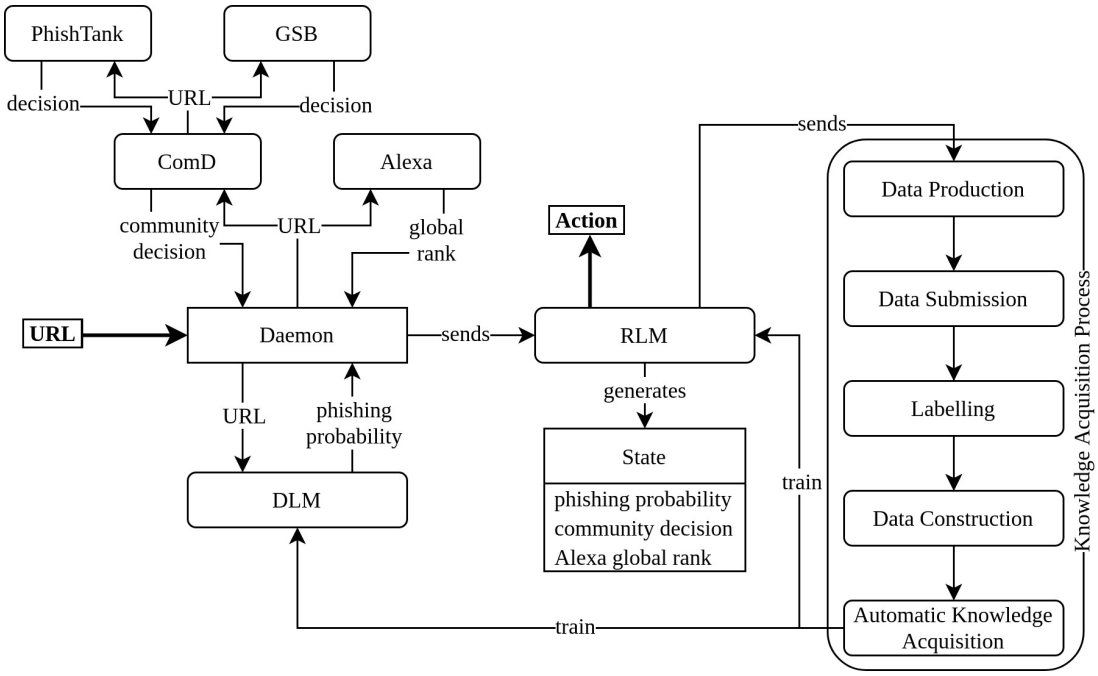


Figure 8.4: SmartiPhish's information flow diagram

non-popular sources and challengeable to DLM to produce a fair decision.

Based on these assumptions, the study proposed a GN to produce adversarial attacks to the SmartiPhish to train the solution against such attacks. The GN was built using the GAN theory (Creswell et al., 2017). In this approach, first, a GAN was implemented. As shown in Figure 8.5, generally, a GAN has a discriminator and generator (Creswell et al., 2017). Therefore, the study first implemented discriminator and generator networks. The discriminator was a feedforward network. It had three hidden layers with a ReLU activation function. It used binary cross-entropy as the loss function and adam optimiser. The discriminator was configured to accept 64-size inputs, and the output layer used a sigmoid activation function.

Similarly, the generator is also a feedforward network. It had four hidden layers with a ReLU activation function. Figure 8.5 shows that a latent space produced the generator input. The space was used to generate a random input with 128 sizes. Therefore, the generator's input size was 128. Since the study planned to plug the GN network into the concatenation layer of the DLM, the generator's output layer was configured as a 64-size dense layer. It also used the ReLU activation function. After both discriminator and generator were finalised, the study implemented the GAN network.

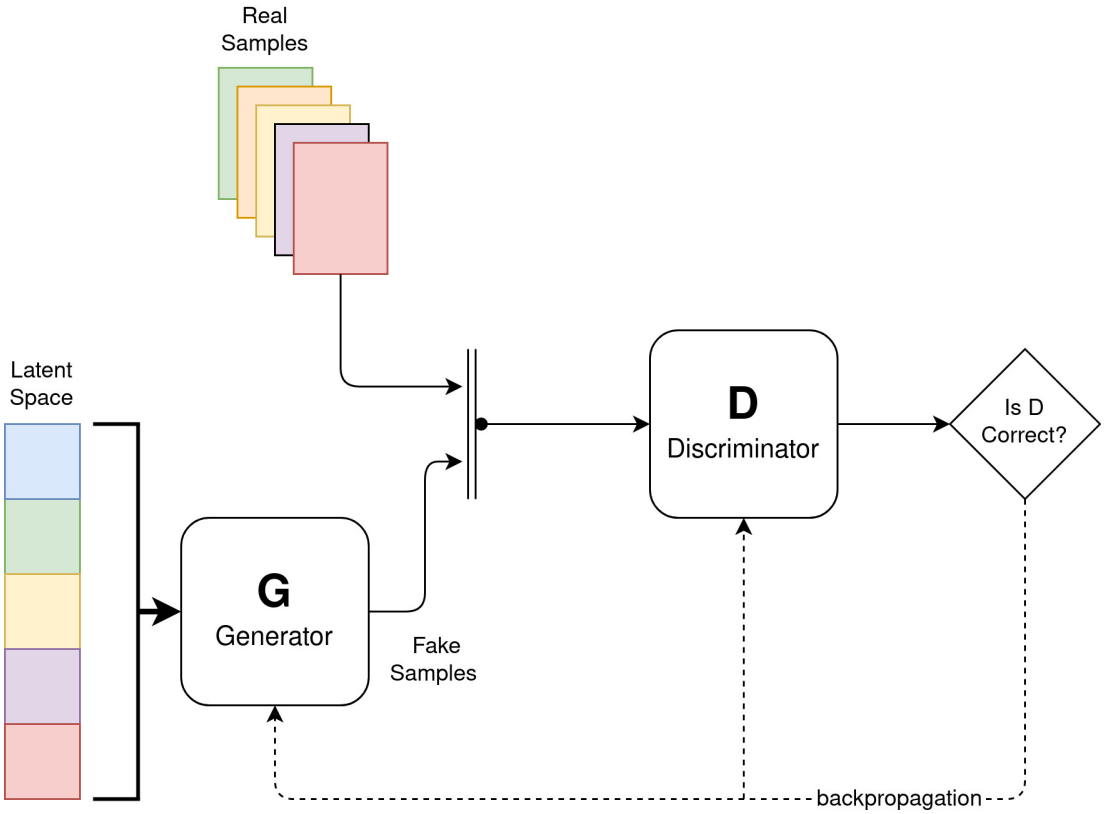


Figure 8.5: Overview of a GAN network

Appendix A.9 presents the implemented GAN network[24]. Then, the GAN network was trained for 10,000 epochs. Then, the generator was unplugged from the original GAN network and plugged into the SmartiPhish solution.

Then, GN is configured to pass some adversarial attack inputs to the DLM concatenation layer to produce a challenging phishing probability. Then, the SmartiPhish collects that probability and randomly produces a zero or lower Alexa rank value to generate a state. The zero or lower Alexa rank value is configured because, as mentioned earlier, the study assumed that the adversarial attacks are generated from non-popular sources. Further, the study assumed that this attack is not reported in any of the phishing verification systems used by the ComD component. Therefore, the community decision for all adversarial attacks is generally zero. After the state is generated, it passes to the RLM to decide.

The study expected the RLM to learn this behaviour through the trial-and-error

[24]The GANInputGenerator class required to execute GAN network is available in SmartiPhish codebase (*see* Appendix B)

concept. Therefore, there was no particular experiment to train the RLM. Every day, the GN is configured to generate several adversarial attacks. Then, the RLM acts on those and learns about adversarial attacks from these examples. After this procedure was executed for several months, the study verified the progress of the proposed solution by checking the rewards generated by the adversarial attack examples. Figure 8.6 shows the mean reward values accumulated by the agent for a week from February 22, 2022, to March 01, 2022, by defending the generated adversarial attack examples. The positive mean reward was reported during the evaluated period, and it emphasises that the RLM could detect adversarial attacks generated by the GN most of the time. Therefore, the study assumed that the agent learned the appropriate behaviour against pre-planned adversarial attacks to minimise the adversarial attack impact of the proposed SmartiPhish solution. However, due to the time and resource limitations, the study did not get a chance to validate the reliability of the proposed adversarial attack defence mechanism in a natural environment.

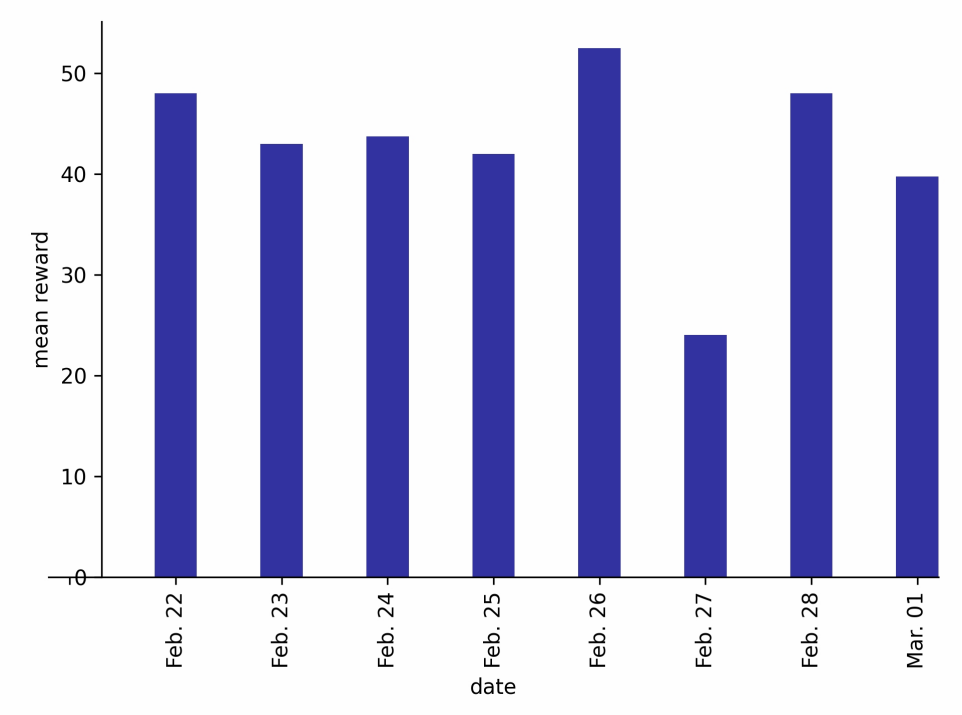


Figure 8.6: SmartiPhish's daily performance against adversarial attacks

8.3.3 Real-time phishing detection

After the SmartiPhish solution was proposed, the study wanted to demonstrate its functionality at the Internet user level since it aimed to minimise the impact of phishing threats on Internet users. Therefore, the study planned to integrate SmartiPhish into a web browser since the web browser is the standard software a user uses to browse the Internet. As a result, the study analysed a web browser's functionality and architecture.

Generally, a user requests a page from a web browser by entering a URL into the address bar, clicking a link, or submitting a form. Whenever a user makes such an action, the browser calls the navigator function to invoke a web page loading task[25]. This navigator function is mainly implemented in the browser engine, and changing its behaviour requires complex engineering effort. A specific processing pipeline is used once the browser downloads the requested web page, as shown in Figure 8.7, to display the HTML content to the user. The study found that the browser must send a request to SmartiPhish after constructing the rendering tree to get an effectual output. It is because then only the effects of JavaScript is visible on the HTML content page. However, this processing happens inside the browser engine. The study found it difficult to break this pipeline to call SmartiPhish solution due to the time and resource constraints associated with such work because of the engineering complexity. Therefore, the study decided to get SmartiPhish's decision before invoking the navigator, and if SmartiPhish sends the 'Allow Access' action to the browser, it asks the navigator to load the web page. Otherwise, it is configured to display a pre-defined message to the user.

However, the new design requirement slightly changed SmartiPhish's initial architecture. The study was planned to accept web pages and URLs both at the beginning to avoid downloading the relevant web page within SmartiPhish since it causes several problems. However, due to the problem that occurred on the browser end, SmartiPhish was configured to accept URLs only, and the responsibility of downloading the web page was handed over to the DLM. Although it could manage the browser side issue,

[25] `https://developer.mozilla.org/en-US/docs/Web/Performance/How\_browsers\_work`

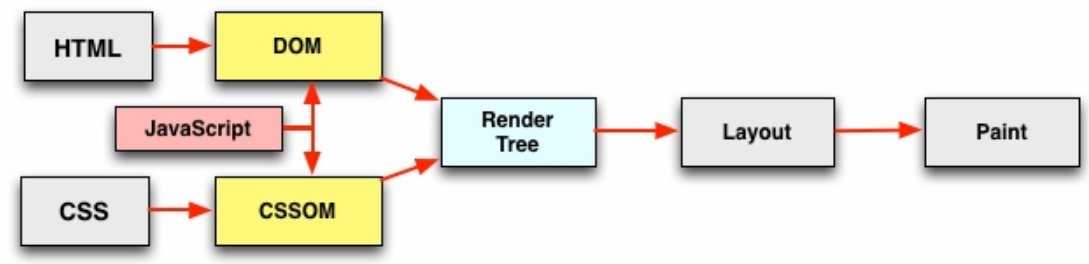


Figure 8.7: Browser processing pipeline
Source: https://hpbn.co/primer-on-web-performance/

the new change added two limitations to the study's final solution. The first one is that SmartiPhish will not respond to dynamic changes in the web page content after the web page is loaded. For example, if a scripting technique (i.e., JavaScript) modifies the content dynamically, it will not affect the decision since SmartiPhish decided based on the initial web page structure. The next one is that if a web page contained the geographical location-based content, it might not be correctly reflected in the web page during the decision-making since the request was made from a different location.

Generally, a web page can be downloaded using the request function available in the Python environment. If such an approach is used, the web page is not going through Figure 8.7 presented pipeline since the browser is not involved in this downloading process. Then, some script-based modifications like JavaScript might not be reflected on the web page. It becomes a problem when making the final decision since real content may be hidden from the DLM. Therefore, the study used an innovative web page downloading functionality to download a requested web page. It was implemented using geckodriver[26], Selenium[27], and Firefox browser's headless mode[28]. The geckodriver was used to link Selenium tests and the Firefox browser to have an automated web page loading facility. Once the page is completely loaded in the Selenium environment, SmartiPhish uses Selenium architecture to download the web page. Although it was not direct like the request function, the study could compile all scripting technologies before downloading the web page. Therefore, it was adapted to the SmartiPhish environment. Appendix A.10 shows SmartiPhish's web page download-

[26] `https://firefox-source-docs.mozilla.org/testing/geckodriver/`
[27] `https://www.selenium.dev/`
[28] `https://hacks.mozilla.org/2017/12/using-headless-mode-in-firefox/`

ing functionality.

However, the SmartiPhish solution required a high computational power since it runs the DLM, the RLM and the knowledge acquisition process. Therefore, deploying it on the client-side was not feasible. Hence, the study planned to deploy the SmartiPhish as a RESTful web service (*see* Figure 8.8). Then a web browser could pass a URL to the SmartiPhish service and get its decision. Appendix A.11 and 12 show the implemented RESTful service to access the SmartiPhish solution.

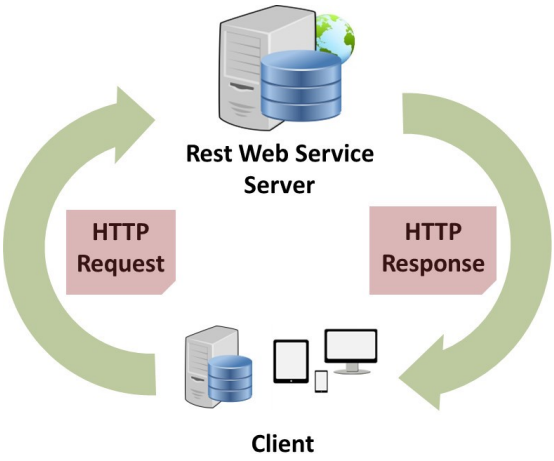


Figure 8.8: A REST API-based architecture
Source: https://ahmetozlu.medium.com/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc

After the SmartiPhish solution was deployed, the study wanted to have an interface to access that solution. As mentioned earlier, the web browser is the appropriate software to use in this case. However, integrating the SmartiPhish service into a browser was difficult since modifying a browser engine-defined procedure is not easy, as it takes more time and resources. However, for the demonstration purpose, the study wanted to have an application that works with SmartiPhish to check the real-time phishing detection ability of the proposed solution. Therefore, the study planned to implement a lightweight web browser with SmartiPhish support. As a result, the study introduced a MORA browser.

MORA browser is a lightweight browser implemented using the electron framework[29]. Electron framework is an open-source software framework developed and

[29]`https://www.electronjs.org/`

maintained by GitHub. It is top of the Chromium rendering engine and the Node.js runtime. Building a browser with the electron is challenging, but it could be done quickly. The study's main intention was to implement a browser for demonstration purposes. Therefore, the MORA browser was implemented with minimum features. Figure 8.9 shows the home page of the MORA browser.

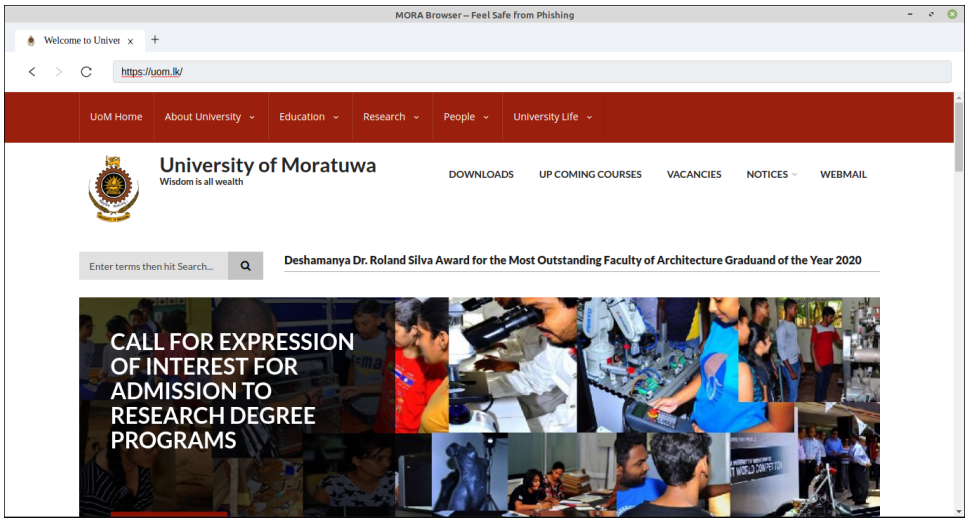


Figure 8.9: MORA browser interface

MORA browser was configured to work with the SmartiPhish service. If the service is not online, the browser will not work correctly. When a user opens the MORA browser, he can enter a URL into the address bar or click a link to start browsing. Any action first calls to the SmartiPhish service, and if the service sends the 'Allow Access' action, the browser calls to the navigator to load the relevant web page. If the service sends 'Stop Access', Figure 8.10 interface will be displayed to the user. However, if SmartiPhish responds with 'Ask User', Figure 8.11 interface will be visible to the user and ask for the legitimacy of the requested web page. Then, the user can submit his feedback and the feedback ultimately helps RLM improve its learning process.

The study used the MORA browser only for demonstration purposes, and it was not tested with the end-users due to resource constraints associated with concurrent usage of the browser. Therefore, the implemented browser was not promoted during the study. However, the MORA browser is available for Windows and Linux platforms, and anyone can install it to get the experience of the SmartiPhish solution. This study

shared an online video[30] to demonstrate how the browser functions in Windows and Linux platforms to safeguard Internet users from phishing attacks. This demonstration used actual phishing attacks that were active while recording the video.

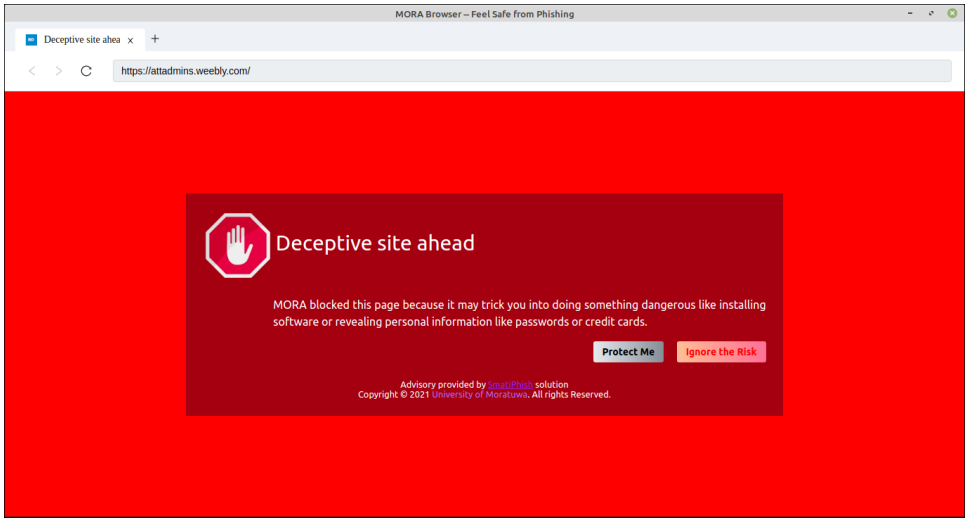


Figure 8.10: MORA browser's 'Stop Access' interface

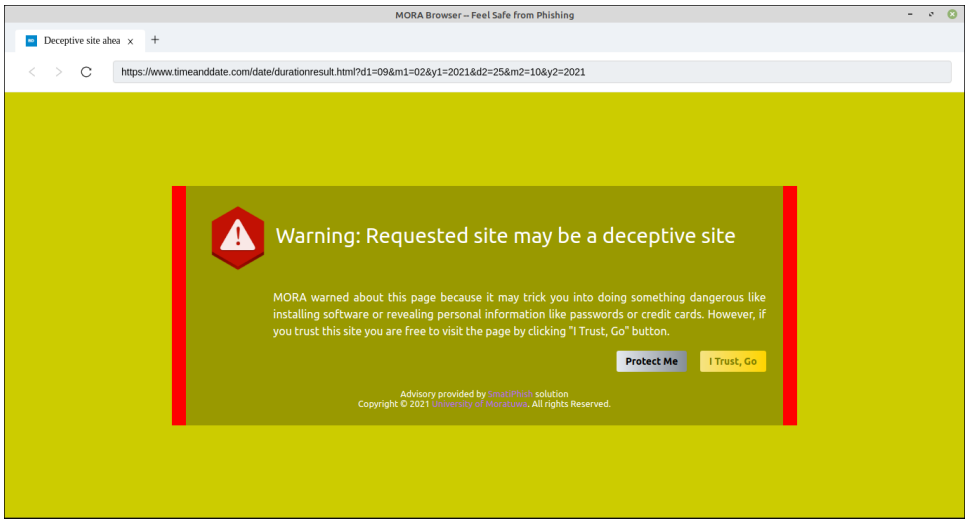


Figure 8.11: MORA browser's 'Ask User' interface

After integrating the MORA browser and adversarial attack prevention into the solution proposed in Section 8.3.1, SmartiPhish architecture was slightly changed. Figure 8.12 shows the final architecture of the proposed solution implemented by the study to achieve the study's aim. After finalising the implementation, SmartiPhish was deployed to the university-owned server for real-time phishing detection.

[30] `https://youtu.be/_MddiKIFvXM`

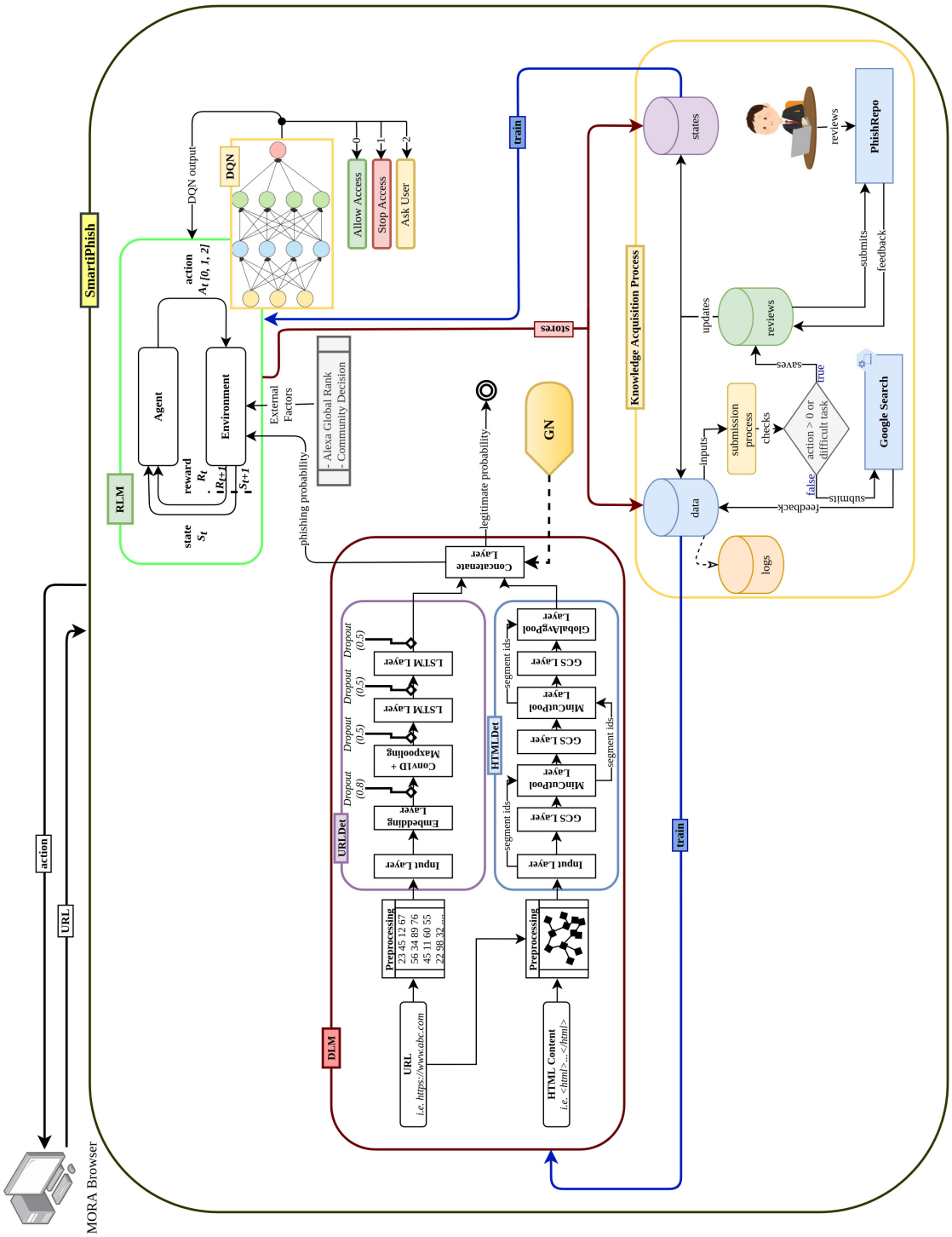


Figure 8.12: Proposed phishing detection solution

8.4 Summary

A systematic knowledge acquisition process was introduced at the beginning of this chapter. Then it was integrated with RDLM proposed in Chapter 6 to have the outcome

of this study named SmartiPhish. However, the solution has opened up several challenges when executed in the real environment. Therefore, SmartiPhish architecture was improved from different perspectives to address those. Furthermore, a browser called MORA Browser has been introduced here, which was implemented to demonstrate an example usage of SmartiPhish. After finalising the proposed solution, this SmartiPhish solution underwent different experiments to evaluate its performance. The next chapter presents these experiments by discussing their results.

9 EXPERIMENTS AND RESULTS

9.1 Introduction

Chapter 9 introduced the proposed SmartiPhish solution. It is an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process. However, the effectiveness of this solution when detecting phishing attacks and the impact of the knowledge acquisition process is yet to be evaluated. This chapter focuses on that aspect. Therefore, this chapter first introduces different types of experiments conducted when evaluating the performance of the SmartiPhish. Then, the results achieved by these experiments will be analysed to show that the SmartiPhish is able to solve the identified research problem.

9.2 Experiments

SmartiPhish mainly initiated its duty at the university server on November 12, 2021. After that, SmartiPhish was trained with real-world phishing and legitimate websites until December 31, 2021. Since SmartiPhish did not interact with the public, a simulated web environment was implemented using three scripts[31] (i.e., Legitimate data extractor, PhishTank data extractor and OpenPhish data extractor) to send phishing and legitimate samples. These scripts were invoked via a cronjob[32] in every hour, and the number of samples varied based on the script.

Moreover, during the training period, the RLM directly interacted with the real phishing and legitimate websites, and it learned through the knowledge acquisition process every midnight. Since the RLM learning process should be visualised to justify the effect of the knowledge acquisition process, three RLM instances were down-

[31] The implementations of these scripts can be obtained through Appendix B - Specific code samples.
[32] A cron job is a Linux command used to schedule a job that is executed periodically.

loaded in different time frames. Table 9.1 shows the details of those RLM instances. Similarly, the DLM was also retrained three times during the study, as shown in Figure 8.2. After the training period ended, SmartiPhish performance was evaluated under different criteria described in the following sections.

Table 9.1: RLM instances

Downloaded date	Instance name	Remarks
November 11, 2021	RLM1	This is the initial RLM model constructed via the Chapter 6 training process.
November 27, 2021	RLM2	Once the RLM1 trained for two weeks via the knowledge acquisition process, RLM2 was downloaded.
December 28, 2021	RLM3	After a month from the RLM2, RLM3 was downloaded.

9.2.1 Overall performance

After the initial training, SmartiPhish's overall performance was evaluated using the RLM3 instance. In this experiment, the phishing data was mainly collected from PhishTank and OpenPhish. Since the SmartiPhish training period ended on December 31, 2021, 2,595 phishing data that PhishTank or OpenPhish verified from January 1, 2022, to February 28, 2022, were collected to evaluate SmartiPhish's overall performance. Since legitimate data is also essential to evaluate the solution effectively, 2,595 legitimate data were collected using Section 5.3.2 legitimate data collection procedure. However, the legitimate data collection was carefully done and skipped Section 5.3.2 selected data to have unseen legitimate data sample during the experiment. Table 9.2 shows the results of the experiment.

Table 9.2: SmartiPhish overall performance

Precision	Recall	*f1*-score	Accuracy	FNR
95.71%	97.15%	96.42%	96.40%	0.029

9.2.2 Continuous learning ability

The primary aim of this study is to implement a phishing detection solution that can learn newer phishing detection features over time via a knowledge acquisition process to minimise the performance drop that exercises in similar solutions. Therefore, it is essential to evaluate the learning ability of the DLM and RLM to verify whether the proposed solution has these characteristics.

The DLM mainly controls the selection of significant phishing detection features, and it is a black-box environment. Therefore, the DLM selected phishing detection features cannot be visualised after a training process. However, it could be present via new and old DLM performance. Since phishing attacks are constantly changing, theoretically, the old model could not perform well with newer phishing attacks. However, once the model is trained with newer phishing attacks, it should perform better than the old model if it acquires the newer phishing detection features. Therefore, in this experiment, the DLM's learning ability was evaluated in different time frames by comparing the old and new DLM performances. Figure 8.2 shows the results of this evaluation.

The RLM is different compared to the DLM. It is the leading decision agent in SmartiPhish, and the learning ability of RLM mainly depends on the decisions it makes. The RLM learns every midnight via the knowledge acquisition process, and if it is learned as planned, the performance of the RLM should be increased over time. Therefore, in this experiment, the Section 9.2.1 dataset was evaluated with RLM1, RLM2, and RLM3 instances to show its performance changes over time. Figure 9.1 shows the results achieved by the experiment.

The RDLM performance fluctuation with a new DLM was also evaluated separately under continuous learning ability criteria to check the effectiveness of the new DLM deployment. In there, July and October trained DLMs (*see* Section 8.2.5.1) were used. This experiment was conducted from January 25, 2022, to February 08, 2022. Initially, this experiment used July trained DLM, and on February 01, 2022, the October trained DLM was deployed to the SmartiPhish environment, and this DLM was used after that. Then, the RDLM accuracies for January 25-31, 2022 and February

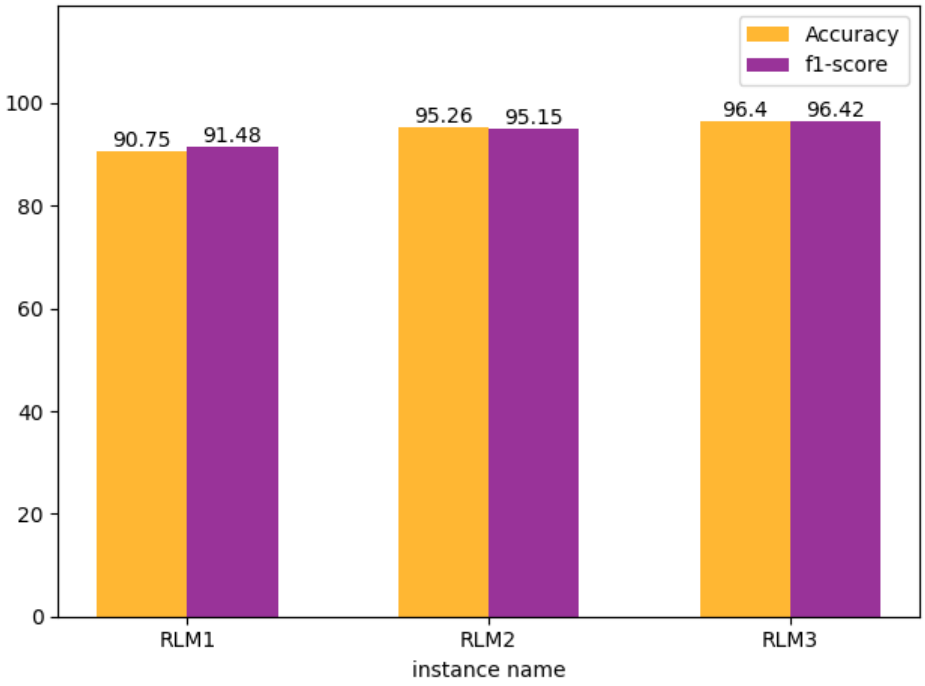


Figure 9.1: SmartiPhish performance change overtime

RLM's F1-score and detection accuracy significantly improved after integrating the automated knowledge acquisition process.

01-07, 2022 were separately evaluated, and these results are presented in Table 9.3. Furthermore, the performance of the July DLM during February 01-07, 2022, was also evaluated, as shown in Table 9.3, to check the RDLM performance in the absence of the new DLM.

Table 9.3: Performance fluctuation during a new DLM deployment

According to the results available in this table, it is clear that if the DLM is absent, the accuracy of the solution will be below 90%. However, once a newly trained DLM is used, the accuracy improves significantly, demonstrating the impact of retraining the DLM.

Duration		Number of Data		DLM Instance	
From	To	Legitimate	Phishing	July	October
January 25	January 31	279	279	86.20%	-
February 01	February 07	240	240	87.29%	95.83%

9.2.3 Zero-day protection

Zero-day detection is essential in any anti-phishing solution. Therefore, SmartiPhish's zero-day protection ability was evaluated using the phishing attacks collected by Section 9.2.1. The zero-day attacks were collected based on the SmartiPhish community decision that mainly depended on PhishTank or GSB phishing verification systems. If SmartiPhish receives a zero value for community decision, it means the community (i.e., PhishTank and GSB) has not seen this attack before. Such attacks were considered zero-day attacks during this experiment. Then, the number of zero-day attacks was filtered from the dataset and SmartiPhish's decision on those attacks was evaluated. Table 9.4 shows the results obtained by this experiment.

Table 9.4: SmartiPhish's zero-day detection results

Total attacks	Detected attacks	Accuracy
1,545	1,473	95.34%

9.2.4 Benchmarking

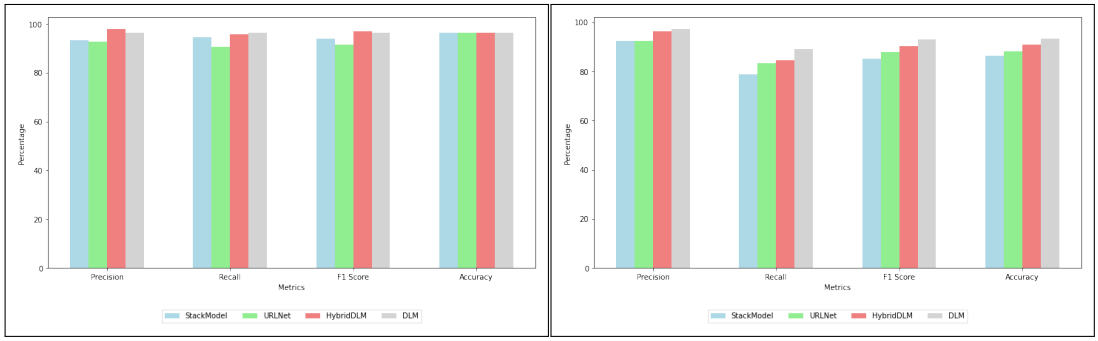
SmartiPhish was benchmarked with DLM, HybridDLM, StackModel, and URLNet. All the benchmark solutions were initially trained with the modern dataset. Table 9.5 shows the performance of these solutions after the initial training. Then, the performance was evaluated using the Section 9.2.1 collected dataset. Table 9.6 presents the performance achieved by these solutions during the experiment. As illustrated in Figure 9.2, the performance of all solutions declined after three months' time.

Table 9.5: Initial performances of the benchmark solutions

Solution	Precision	Recall	*f1*-score	Accuracy	FNR
StackModel	93.18%	94.58%	93.87%	93.83%	0.054
URLNet	92.62%	90.64%	91.62%	91.71%	0.094
HybridDLM	97.96%	95.82%	96.88%	96.91%	0.042
DLM	96.4%	96.44%	96.42%	96.42%	0.036

Table 9.6: SmartiPhish comparison with selected phishing detection solutions

Solution	Precision	Recall	*f1*-score	Accuracy	FNR
StackModel	92.50%	78.88%	85.15%	86.24%	0.211
URLNet	92.50%	83.43%	87.73%	88.25%	0.166
HybridDLM	96.44%	84.66%	90.17%	90.77%	0.153
DLM	97.31%	89.13%	93.04%	93.33%	0.109
SmartiPhish	**95.71%**	**97.15%**	**96.42%**	**96.40%**	**0.029**



(a) Starting performance

(b) Performance after three months time

Figure 9.2: Performance trends of benchmark solutions over 3 months

The bar chart illustrates the performance trends of benchmark solutions over a period of three months. The metrics considered are Precision, Recall, F1 Score, and Accuracy, measured in percentage values. The solutions examined include StackModel, URLNet, HybridDLM, and DLM. It is evident from the chart that all benchmark solutions experienced a decline in performance in accuracy and f1-score during the three-month period. This decline is observed in Precision, Recall, F1 Score, and Accuracy, indicating that the solutions' performance diminished over time.

9.2.5 Detection time

The detection time experiment was performed on an Intel Core i5-7200U machine with 8 GB of memory. It used 5,000 randomly selected websites from the Section 9.2.1 used dataset, including an equal amount of phishing and legitimate samples. Figure 9.3 shows SmartiPhish detection time over each website, and it was 4.34 seconds mean value per request. However, during this experiment, the detection time was calculated by omitting the network transfer times to better understand the SmartiPhish detection time.

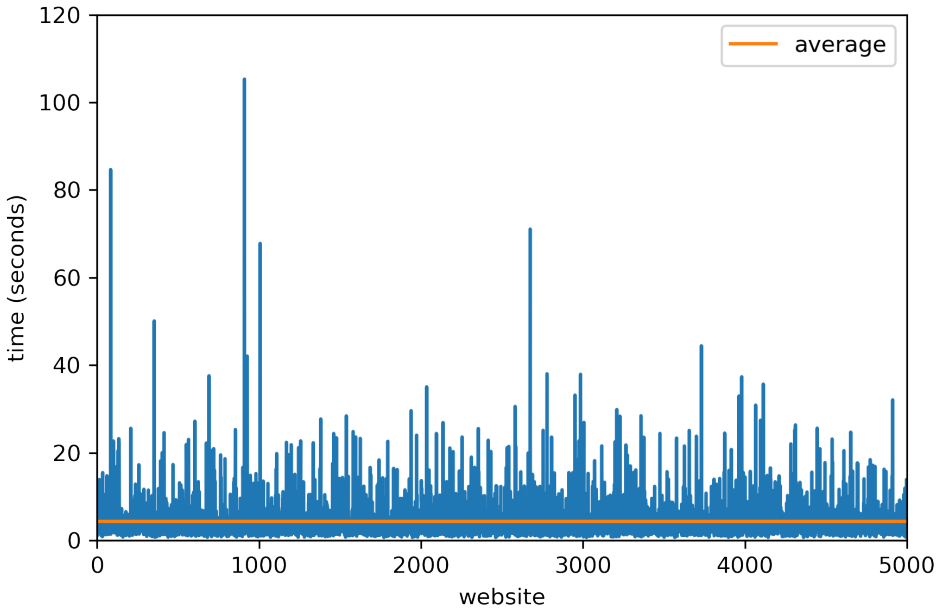


Figure 9.3: SmartiPhish detection time curve

9.2.6 Imbalanced test

On the natural Internet, phishing websites are much less than legitimate websites. Therefore, an imbalanced dataset is the most successful approach a study can use to conduct a more realistic evaluation of a proposed anti-phishing solution (Aassal et al., 2020). However, the literature lacks a well-defined legitimate-to-phishing ratio when developing an imbalanced environment (Aassal et al., 2020). Therefore, this study used different legitimate to phishing ratios during the experiment, and the study maintained the ratio from 1:1 to 1:10 by taking the idea of Aassal et al. (2020). In this experiment, the phishing counts were always constant, and the mentioned ratios were maintained using the legitimate data count.

The experiment used 1,000 random phishing data from the Section 9.2.1 dataset. Further, it selected 11,000 legitimate data by following the Section 5.3.2 procedure. However, that 11,000 included 2,500 data that was used in Section 9.2.1. The imbalanced experiment was planned not only for SmartiPhish. It used Section 9.2.4 experimented benchmark solutions to compare SmartiPhish's performance with similar solutions in an imbalanced environment. Figure 9.4 demonstrates the performance of these solutions alongside SmartiPhish. The f1-score was used to measure the per-

formance during this experiment instead of accuracy since it is the most appropriate metric for an imbalanced environment.

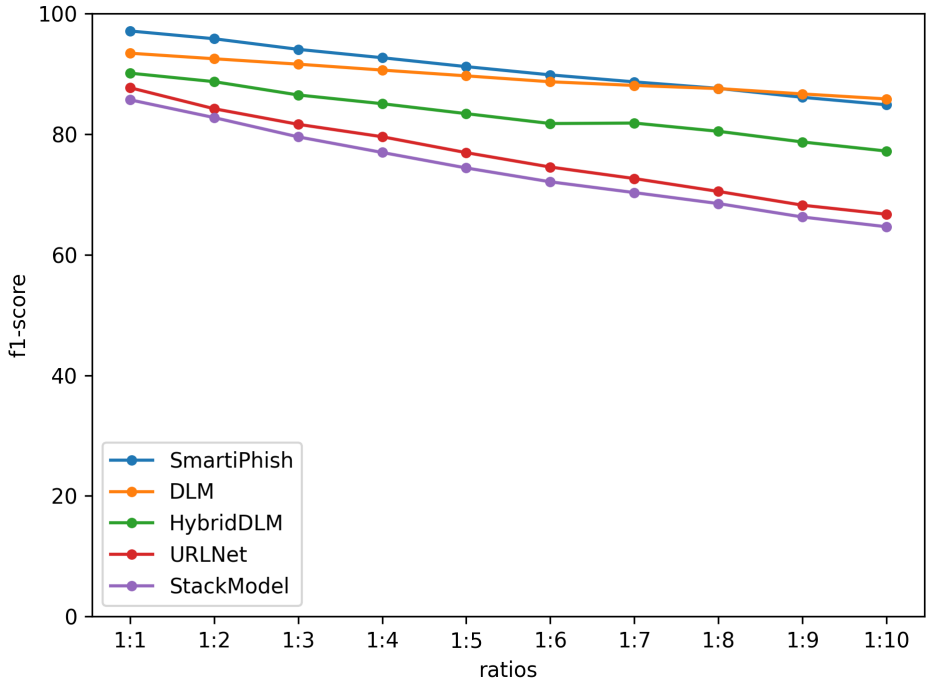


Figure 9.4: Performance comparison with different legitimate to phishing ratios

9.2.7 Real-time phishing detection

SmartiPhish is a real-time phishing detection solution. Therefore, it is essential to evaluate how it behaves in a natural web environment since all the above experiments were done with offline datasets. Since the study simulated a natural environment for SmartiPhish, an experiment was conducted from November 12, 2021 to November 30, 2021, to collect the daily performance of the SmartiPhish. Meanwhile, the detection time of each of these requests was also recorded to arrive at the correct conclusion about SmartiPhish's real-world detection time. Since the legitimate to phishing ratio is not equal in the natural environment, the f1-score was used to measure the daily performance of the proposed anti-phishing solution. Figure 9.5 illustrates the daily performance of SmartiPhish, while Figure 9.6 shows the detection times of 5,000 randomly selected web requests received by SmartiPhish during the experiment period, and it was 16.65 seconds mean value per request. However, this experiment's detection time is differ-

ent from Section 9.2.5 detection time because this experiment included the network transfer times in the calculation to illustrate the real-world scenario.

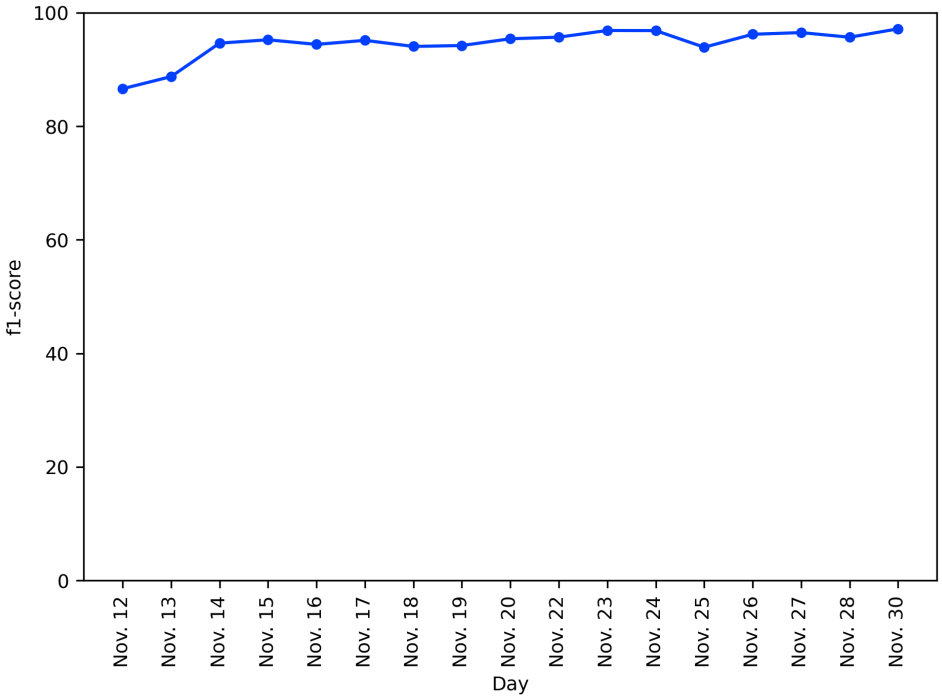


Figure 9.5: SmartiPhish's daily performance

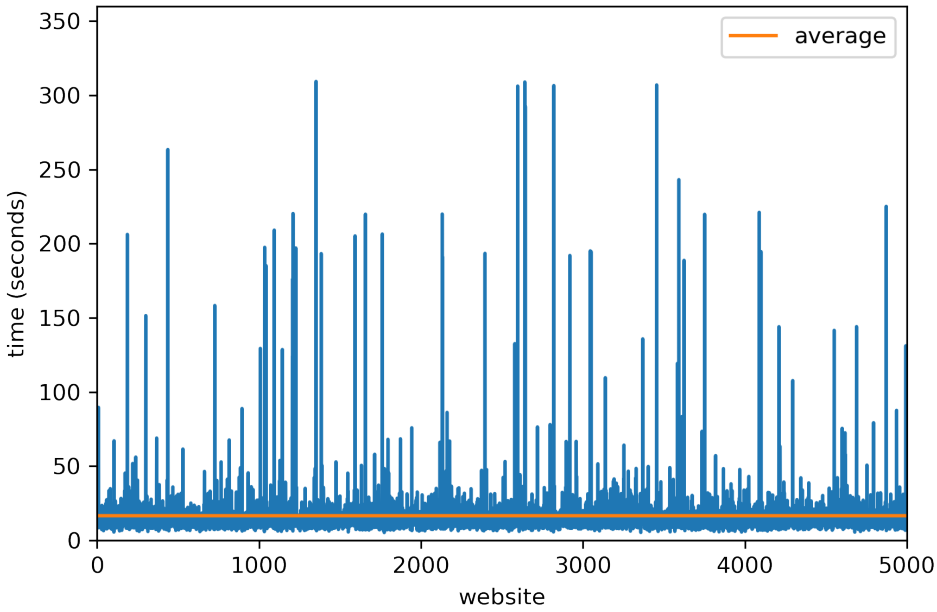


Figure 9.6: SmartiPhish's real-world detection time curve

9.3 Results analysis

Present anti-phishing solutions are faced with performance decreases over time due to constantly changing phishing detection features. The literature also highlighted that machine learning-based anti-phishing solutions recorded significant performance drops two months after the initial training. SmartiPhish is proposed in such a background as a differentiated anti-phishing solution. It has an integrated knowledge acquisition process that enables the solution to learn newer phishing detection features over time. Therefore, in the previous section, SmartiPhish performance was evaluated using different criteria. Although SmartiPhish is a new approach to the anti-phishing domain, it has shown some favourable results during the experiments.

As tabulated in Table 9.2, SmartiPhish achieved 95.71% precision, 97.15% recall, 96.42% f1-score, 96.40% accuracy and 0.029 FNR. More importantly, SmartiPhish's accuracy and f1-score were increased over time, and Figure 9.1 shows that SmartiPhish improved its accuracy from 90.75% to 96.40% during six weeks. This is mainly due to the continuous learning process integrated into the solution, and both DLM and RLM updated the existing phishing detection knowledge via this implemented learning process.

Furthermore, Table 9.3 also presented that the latest DLM deployment increased the SmartiPhish performance by 8.54%. Since the old DLM was trained in July and was nearly six months old, it was ineffective during January and February. Therefore, the overall performance of the July DLM was around 87%. However, SmartiPhish's performance was noticeably changed once the October trained DLM was deployed to the solution. This indicates that the October or latest DLM updated its phishing detection knowledge through the knowledge acquisition process to handle many latest phishing attacks than July DLM. This experiment further illustrates the importance of continuous learning when detecting the latest phishing attacks.

In Section 9.2.4, SmartiPhish was benchmarked with several anti-phishing solutions introduced through this study and state of the art. Table 9.6 clearly shows that SmartiPhish's recall value is well over other solutions, and it indicates that SmartiPhish can identify phishing instances correctly during the detection process. It is an essential

factor when implementing an anti-phishing solution since incorrect marking of phishing instances may have a high impact on potential users.

The experiment further illustrates that SmartiPhish performed impressively in the used dataset than the existing anti-phishing solutions. However, according to Table 9.5, the benchmark solutions have shown good performance after the initial training, but those performances significantly decreased once they came to the experiment dataset. This may be due to the newer phishing attacks in the Section 9.2.1 dataset since these benchmark solutions were trained a few months ago with a modern dataset. However, SmartiPhish, which also had the initial training with the benchmark solutions, has not decreased its performance over time and has shown its best two months after the initial deployment. This proves that SmartiPhish controlled the performance dropping issue that exists in similar solutions due to the constantly changing phishing attacks.

Furthermore, according to the Section 9.2.3 experiment, SmartiPhish's zero-day phishing detection ability is also high. This shows that out of 1,545 zero-day phishing attacks, SmartiPhish correctly detected 1,473 attacks, and this is proportional to 95.34%. Since SmartiPhish uses community decisions as an observer for RLM, one might argue that SmartiPhish's high phishing detection ability is due to this existing knowledge. However, SmartiPhish's zero-day attack detection results falsify this argument because this experiment only considered phishing attacks that were not known by the community (i.e., PhishTank and GSB) before. This indicates that SmartiPhish can detect phishing attacks at a high rate without community support.

Detection time plays a significant role in phishing detection since delayed response makes unsatisfied users. Generally, SmartiPhish first gets a request from outside, and then fetches relevant resources like a web page, Alexa rank, PhishTank decision, and GSB decision to process the request. Once the required resources are downloaded from suitable locations, the final decision is processed. After that, the decision needs to be transferred to the client-side. In each of these tasks, except decision processing, significant network time is used by SmartiPhish. However, that time depends entirely on the Internet connectivity, the solution and the user use. Therefore, SmartiPhish's detection time was evaluated in two ways to have a correct conclusion. Section 9.2.5

calculates the detection time only considering the decision processing time. Figure 9.3 shows the output of that experiment. The average detection time recorded in Section 9.2.5 experiment is 4.34 seconds. This means that under the mentioned machine configurations, SmartiPhish needs 4.34 average seconds to make one decision. This looks like a reasonable value since the Yang et al. (2019) solution also took 3.5 seconds when detecting phishing attacks.

However, Section 9.2.7 considered the real scenario and calculated the average detection time, including the network transfer times. In that experiment, a user to SmartiPhish and SmartiPhish to a user network transfer times were not captured because the requests were made on the same machine where the SmartiPhish was deployed. In there, the average detection time was recorded as 16.65 seconds. This indicates that in the actual execution, a user needs to wait a 16.65 average seconds to get a decision from SmartiPhish. However, that time will increase or decrease based on the solution's Internet connectivity. Although it shows some high detection time, Figure 9.5 shows that SmartiPhish's real-time phishing detection was high, and it visualised that SmartiPhish's f1-score was increased over time via the knowledge acquisition process. This was a significant achievement of SmartiPhish since it proves that SmartiPhish can maintain a high detection rate for an extended period.

Even though SmartiPhish's performance is good in a balanced dataset, Figure 9.4 illustrates its performance declines in an imbalanced dataset. Further, the f1-score was downgraded from 97.15% to 84.90% when the phishing to legitimate ratio changed from 1:1 to 1:10, and this is proportional to a 12.25% decline. However, that pattern is not only for SmartiPhish. All the benchmark solutions also show similar behaviour. However, in most cases, SmartiPhish shows the highest f1-score among other solutions, and it is a considerable achievement of SmartiPhish since the literature also mentioned that the classifier performance declines when the dataset becomes more and more imbalanced (Aassal et al., 2020). Although that factor is mentioned, the literature lacks a well-accepted legitimate phishing ratio to have a realistic imbalanced experiment. However, SmartiPhish averagely maintained a 90.81% of f1-score with the experimented dataset.

9.4 Summary

SmartiPhish performance was evaluated with different experiments in this chapter. The result of these experiments was then analysed to arrive at a conclusion about its real-time phishing detection ability. According to these experiments, SmartiPhish achieved 96.40% detection accuracy and outperformed all the benchmark solutions exercised during the experiments. It has recorded a realistic detection time and an interesting zero-day performance. Furthermore, the knowledge acquisition process used in this proposed solution has shown the impact of continuous learning when detecting phishing attacks. Therefore, the proposed solution has experimentally shown that it achieved the study's aim. However, the achievement should be critically evaluated before making the final remarks of this study. Therefore, the next chapter evaluates the proposed solution aligned to the defined objectives of this study to justify that this study has achieved its aim with this SmartiPhish solution.

10 EVALUATION

10.1 Introduction

This study aimed to find a better way to detect phishing attacks by addressing limitations in the anti-phishing domain. As a result, Chapter 8 introduced the SmartiPhish solution as the outcome of this study and the previous chapter experimentally evaluated its performance in a simulated web environment. This chapter focuses on a critical evaluation of the conducted study when it comes to achieving the study's aim. As a result, this chapter first provides an overview of the study and then discusses how the defined aim was successfully achieved by attaining the study's objectives. Following that, the resolved research problem, the novelty, and contributions to the current anti-phishing domain are discussed. The limitations of the proposed solution are reviewed in the latter part of this chapter to highlight potential improvements.

10.2 Research overview

This research was mainly motivated by the increasing number of phishing threats in present cyberspace. After a successful background study, this research identified that the existing anti-phishing solutions are ineffective against the changing nature of phishing attacks since they were not planned for frequent knowledge acquisition to update existing phishing detection features. As a result, when phishing attacks change, many solutions suffer performance degradation. It is a problem in the current anti-phishing domain since the newly introduced solutions are becoming useless after a few months, and a similar effort is required again to retain the performance of these solutions. As a solution to this problem, an autonomous anti-phishing solution with a systematic knowledge acquisition process was proposed to reduce the impact of phishing attacks on Internet users.

According to the research nature, positivism research philosophy was initially selected. Therefore, the existing knowledge of phishing detection was analysed. As a result, machine learning was selected as the phishing detection technique based on the previous evidence to construct the implementation process. Since machine learning was categorised under a quantitative research approach, the collected data was primarily used when finding patterns, making predictions, and generalising results during the study. However, the data collection was not a one-time task since the followed methodology required time-series data to achieve the study's aim. As a result, several datasets, such as the classic and modern datasets, were utilised.

According to the proposed methodology, the solution implementation was not straightforward. It included three separate phases. The first two belonged to the implementation of the phishing detection solution. The next was developing the knowledge acquisition process, which integrated the phishing detection solution to facilitate continuous learning support. After completing these three phases, the proposed solution named SmartiPhish was implemented. Then, a web browser named MORA Browser was implemented to demonstrate the actual usage of the proposed solution. After these implementations, SmartiPhish was evaluated from different perspectives, and the results were thoroughly analysed to show that the mentioned aim was successfully achieved.

10.3 Achieving the aim and objectives of the study

This study aims to develop an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process to reduce the impact of phishing attacks on Internet users. This aim was succeeded by following three research objectives defined at the beginning. The following explains how these objectives were achieved.

RO1 – To identify the phishing detection features and detection techniques utilised by the anti-phishing domain in effective phishing detection

RO1 is primarily linked to the literature on phishing attack detection. Therefore, the existing anti-phishing studies were first analysed to collect phishing detection features

already used. After this analysis, these features were classified as URL-based, HTML-based, and third-party features, as shown in Table 3.1. Out of these, the third-party features can increase the detection time of a solution because most of them rely on external services. However, external features such as Alexa and Google ranking have significantly contributed to phishing detection in the past. Besides detection time, service limitations and costs associated with some of these services have also caused issues when using third-party features in phishing detection. Furthermore, earlier solutions that only used URL-based features had some limitations, and phishing detection features gathered from all these categories were mentioned as beneficial in a successful phishing detection strategy. Despite these feature categories, the literature has mentioned that the features that take longer to mine are not ideal for real-time phishing detection because they contribute to a higher detection time and create dissatisfied users.

After analysing the phishing detection features, feature extraction techniques were also analysed under this objective since feature extraction was also essential when achieving the study's aim. Therefore, the feature extraction techniques were examined during the literature analysis and mainly identified manual and representation learning techniques. However, the manual feature extraction techniques had several limitations, while representation learning techniques automatically extracted significant phishing detection features by traversing the raw data. Further, representation learning techniques like deep learning are black-box techniques where the selected phishing detection features were not visible to the outside. This is also another advantage for phishing detection solutions because attackers cannot use the selected attributes to circumvent these solutions. Additionally, representation learning provides better support when updating the present significant features that could be used to detect phishing attacks. It is essential in phishing detection because phishing attacks are constantly changing. Therefore, representation learning techniques are more suitable when detecting phishing attacks than manual techniques.

Moreover, phishing detection techniques have mainly been classified into user education and software-based detection techniques in Chapter 3. Among these, software-

based techniques seem to be practical since phishing attacks are constantly changing, and user education is identified as an impractical approach due to the cost incurred and the readiness of the trainees. Therefore, machine learning techniques that come under software-based techniques have shown significant achievement in the past decade due to their ability to manage frequent data changes and unique learning processes. As a result, many contemporary phishing detection interests are toward machine learning. Deep learning is the most popular because it is a representation learning technique with many advantages in phishing detection, as previously described. Further, Table 3.3 also visualises the use of deep learning techniques in recent phishing detection and their achievements.

After a successful literature analysis, this study made the following conclusions to construct the foundation for the RO2.

- URL-based and HTML-based phishing detection features are considered internal features, and phishers have more control when creating new phishing attacks.
- Even though third-party phishing detection features have some limitations, these are effective when detecting phishing attacks since the phishers' impact on these features is less than the internal features.
- Representation learning is the most appropriate technique to be followed when extracting phishing detection features because these significant features need to be updated frequently due to constantly changing phishing attacks.
- Representation learning is responsible for extracting the relevant phishing detection features from the raw data during the learning process, which eliminates the separate feature extraction process required in many other techniques.
- Deep learning is a representation learning technique that is effective in phishing detection due to its ability to manage frequent data changes, unique learning processes and previous success in phishing detection.

With these critical findings, RO1 was successfully achieved. As a result, the main consequence of RO1 was the knowledge gained about current phishing detection fea-

tures and techniques. This knowledge was then applied to achieve RO2 and RO3 in the following stages.

RO2 – To implement an effective anti-phishing solution by overcoming the identified phishing detection challenges

RO2 is primarily linked with the research question RQ1, which is one main implication of this study. It mainly targeted the implementation of an effective anti-phishing solution. Therefore, when solving RQ1, this study first found an anti-phishing solution overcoming the identified challenges and then the performance of this solution was experimentally evaluated to demonstrate its effectiveness. The following section presents the implementation of the proposed anti-phishing solution and its performance evaluation conducted in a simulated web environment.

RQ1: How can an effective anti-phishing solution be implemented while overcoming the identified challenges?

A comprehensive literature analysis of present phishing detection challenges was conducted as the first step in answering this question. It was mainly to design a better solution by overcoming the present phishing detection challenges. This analysis identified six main challenges that should be addressed when achieving the study's aim. They were mainly discussed in Section 3.7. After finding some positive approaches to overcoming these challenges, an effective anti-phishing solution was proposed.

Proposed solution: This solution had two main phases, as mentioned in Chapter 4. The objective of phase one was to implement a phishing detection solution that could extract phishing features from internal features because these features could frequently change with the phishers' impact, as identified under RO1. Therefore, these internal features need to be retrained occasionally to detect the latest phishing attacks so as to overcome the one and fourth challenges mentioned in Section 3.7. Therefore, as identified under RO1, a deep learning-based phishing detection solution was initially planned with the support of URL and HTML content features.

As a result, DLM was implemented to oversee the URL and HTML content of the website.

According to the explored literature, DLM could be considered the third phishing detection approach, which uses representation learning to detect phishing attacks using URL and HTML content features simultaneously. However, this would be the first phishing detection solution which employs GNN in phishing detection. Even though two similar existing solutions are available, they used LSTM and CNN to build their deep networks. However, the literature has mentioned that GNN is an effective technique for analysing HTML content to detect phishing attacks due to the graph representation of HTML content.

Also, no one in the available literature has used GNN efficiently to analyse HTML content. This may be due to the dynamic content of HTML pages, which generate different structures and lengths from page to page. This dynamic behaviour causes problems when generating a GNN-compliant graph in an automatic process like DLM. As a solution for this problem, a new HTML page traverser was developed during this study to go through HTML pages level by level to produce a GNN-compliant graph. Therefore, the DLM could use the powerful GNN-based architecture to analyse the HTML contents, and LSTM/CNN-based architecture was only used with URL analysis.

DLM achieved 96.42% detection accuracy and 0.036 FN rate. However, compared to Table 3.3, this detection accuracy is slightly lower. Therefore, a benchmarking experiment was used to re-evaluate the DLM performance. According to the benchmarking experiment, the DLM detection accuracy was recorded as 99.57%, which is the best among the other accuracies in Table 3.3. When comparing these two results, the impact of the diversity of the used dataset was visualised. According to Section 5.3.4, the benchmark dataset is not diverse, and 99.57% accuracy might be due to that cause. However, the modern dataset exercised in the first experiment was diverse. Therefore, this study has decided to consider 96.42% as the detection accuracy of the proposed DLM.

Even though there was no solution like DLM at the beginning of this study, two

similar solutions were introduced later, as discussed in Section 3.5. Both solutions used the same strategies when downloading data for these studies. The Web2Vec showed the highest performance, and the benchmark dataset was employed there. Therefore, this study also evaluated the DLM with the benchmark dataset as discussed previously. This measured accuracy is 0.57% higher than the Web2Vec recorded accuracy. It indicates that the DLM has outperformed similar solutions by proving the effectiveness of the novel architecture presented in the proposed DLM.

Further, the DLM was highly effective against zero-day attacks. It recorded 96.43% detection accuracy with zero-day attacks, and the overall accuracy was 90.12%. It is well above the GSB, which recorded 49.88% detection accuracy in the same experiment. Like zero-day attacks, the DLM detection time is also interesting. It took only 1.8 seconds to give a decision for a given webpage. This reported detection time is quick compared to the literature because, in a similar solution, Yang et al. (2019) completed one webpage within 3.5 seconds, which is twice the DLM detection time.

Although the DLM has shown a reasonable detection accuracy, time, and zero-day attack detection, it could not properly overcome the main research problem identified in this study. The performance degradation of an anti-phishing solution occurs due to constantly changing phishing attacks. The impact of this has been experimentally demonstrated in Section 5.5.2.1. According to that, the DLM had a 9.35% low performance after one year. However, after following a successful retraining step, the DLM regained its typical behaviour. This indicates that retraining is a successful strategy for retaining the performance of anti-phishing solutions.

Even though this strategy could maintain the performance of the DLM, it could not be frequently performed due to the data challenges mentioned as the second and third in Section 3.7. In general, a deep learning solution like DLM requires large data to perform a successful learning step. However, due to the previously mentioned data collection challenges, collecting a large volume of data in a shorter period is not easy in the phishing domain. As a result, phase two implementation

was proposed to control this performance drop until a retraining step was met.

Phase two output was the RDLM, which is a phishing detection framework. It primarily depended on RLM, with the DLM as an observation of the RLM. Since the RO1 identified that the phishers' impact is less in third-party features, the DLM output with third-party features was used to control the performance drop of the phase one solution. As a result, the community decisions and Alexa ranking were selected by considering their feasibility. The Alexa ranking provides information about a website's reputation, which highly supported decreasing FP rate and the community decision, which supported decreasing FN rate.

The RLM implementation initially faced two challenges identified in Section 3.7. One was the data collection problem, and the next was the third-party feature limitations. Generally, RL does not require much data. It can work with zero or less data because it is a trial-and-error concept learned through experience. Therefore, a small amount of data was enough for RLM to train the model. As presented in Table 6.1, the study could collect 8,700 data during the RLM implementation to overcome this data challenge. The next challenge encountered in this implementation was the detection time because the other identified third-party feature limitations like cost and service constraints do not apply to the selected external services (i.e., PhishTank, GSB and Alexa).

As described in Chapter 6, the DLM implementation was designed as a service for the RLM. Therefore, the RLM calls DLM and other external services simultaneously. As a result, both Alexa and ComD features could construct their values during the DLM decision processing time. This strategy did not consume additional time for the third-party feature construction because, in most cases, the DLM took a longer time (*see* Figure 5.23), and the external features could be constructed within that period. Otherwise, if the RDLM is built as a sequential process solution, as Yang et al. (2019) proposed, the RLM must wait until the DLM has completed its duty before calling external services. Then, additional time will be calculated, and it directly affects the detection time of the final solution. However, the detection time of the RDLM always depends on the network transfer time because of the external

services.

The ultimate solution found for RQ1 was the RDLM. It was introduced as a phishing detection framework. This indicates that RDLM is a structure which can be changed to have differentiated solutions. Therefore, other researchers can use the RDLM concept to try out different observations when finding effective solutions against phishing attacks. Even though this work used three observations in the RDLM, the RLM can incorporate many different observations with little engineering effort. For example, suppose a visual similarity-based value could be calculated for phishing web pages using some visual component. In that case, this visual similarity value can be added as a separate observation to the RLM. Then, with a little engineering work, this new observation might be configured with the RDLM for effective phishing detection.

Section 6.7.2 evaluated the performance of RDLM over DLM via a specific experiment. This experiment shows that the main objective of RDLM was accomplished because the DLM achieved only 87.82% detection accuracy, while the RDLM achieved 94.11%. This observation is due to the declining performance of the DLM because the used DLM was retrained in July 2021, and it was more than three months old when conducting this experiment. Therefore, as mentioned in the literature, the DLM should expect a performance drop. In this case, it was recorded as 8.2% since the DLM had a detection accuracy of 96.02% at the beginning. It indicates that if the DLM were the only solution offered against phishing, it would detect phishing attacks with an accuracy of 87.82% after three months. However, the level two implementation maintained overall solution accuracy at 94.11% while reducing performance drop to 1.91% compared to the initial accuracy of the DLM.

In conclusion, the experiment demonstrated that the level two implementation delivered a reasonable solution to the identified problem, reducing performance drop from 8.2% to 1.91% during the experiment. However, these numbers only belong to this experiment because RDLM expects to improve its detection ability once it is exposed to more examples since RDLM learns through experience. This was visible in Section 9.2.4, where RDLM achieved 96.4% accuracy after six weeks of

exposure to the real world. Furthermore, Section 9.2.4 empirically demonstrated that, after several months of initial training, while all of the benchmark solutions recorded a dropping performance, the RDLM recorded a rising performance. This observation indicates that the RDLM is capable of controlling this decreasing performance. It is also a practical phishing detection approach that has addressed the relevant challenges identified by the initial literature analysis. Therefore, with the RDLM, this study found a successful answer to RQ1 to detect phishing attacks that support the study's aim. Next, the effectiveness of this solution is evaluated to justify that the proposed solution is an effective anti-phishing solution.

Solution evaluation: The RDLM performance was critically evaluated in Chapter 9 under SmartiPhish experiments. Since the RDLM is the main phishing detection component of the SmartiPhish solution, these experiments demonstrated critical aspects of RDLM to uplift it as an effective anti-phishing solution. These aspects can be presented below to justify the effectiveness of the RDLM in phishing detection.

1. **High-accurate phishing detection**

   The RDLM achieved a detection accuracy of 96.40%, an f1-score of 96.42%, and an FNR of 0.029, as shown in Section 9.2.4. It outperformed all the benchmark solutions in this experiment, and the DLM recorded the closest accuracy to the RDLM, which was 93.33%. These results indicate that the accuracy recorded by the RDLM is much better than other solutions employed in this experiment. Additionally, the recall value of SmartiPhish in this benchmark experiment was comparatively high. It shows that PhishRepo can detect phishing attacks very accurately than the other solutions used in the experiment. It is an advantage of the proposed solution because incorrect marking of phishing instances may significantly impact potential users.

   Further, when comparing the RDLM accuracy with recent anti-phishing solutions available in Table 3.3, the RDLM recorded accuracy is slightly lower. However, the accuracy of an anti-phishing solution always depends on the diversity of the evaluation dataset, as discussed in Section 5.3.4. This study performed a benchmarking experiment in Section 5.5.2.2 with the Web2Vec

solution, which holds the best accuracy among the recent anti-phishing solutions in Table 3.3. According to this experiment, the DLM outperformed the Web2Vec solution. In contrast, RDLM outperformed DLM in Section 9.2.4. These observations inferred that the RDLM is a more accurate anti-phishing solution than the explored solutions.

2. **Protection against zero-day phishing attacks**

   As identified in the literature analysis, zero-day protection is an essential aspect of an anti-phishing solution. Therefore, in Section 9.2.3, RDLM performance was evaluated against zero-day attacks. According to this experiment, RDLM can detect zero-day attacks with 95.34% detection accuracy. This accuracy was an outstanding achievement of RDLM since the overall detection accuracy was also 96.40%.

   However, this study was unable to find any anti-phishing solution that had performed a specific experiment to measure the accuracy of zero-day attack detection. Therefore, this study could not compare the RDLM's zero-day performance to that of prior solutions.

3. **Better performance in the natural web environment**

   Users visit more legitimate sites than phishing sites in a typical web environment. As a result, the natural web environment is always unbalanced, with legitimate websites outnumbering phishing websites. Section 9.2.6 simulated this imbalanced environment using different ratios because no practical ratio for an imbalanced experiment has been presented in the literature. According to that experiment, increasing the number of legitimates reduced the f1-score of the RDLM solution to a 90.81% mean value. It is nearly a 6% reduction from the recorded f1-score in the balanced environment.

   However, the literature has already observed this performance drop in an imbalanced environment. Aassal et al. (2020) mentioned that almost all the benchmark solutions they used in their imbalanced experiment showed a decreasing trend when the ratios were changed. This was a noticeable point because the same observation was also seen in the imbalanced experiment

performed in this study. Although RDLM has not performed in its imbalanced environment like in the balanced environment, it outperformed all the benchmark solutions in most cases when the phishing-to-legitimate ratio was changed from 1:1 to 1:10, as illustrated in Figure 9.4. This inferred that even though the proposed anti-phishing solution shows a decreased performance in the natural environment, the RDLM is better than the other solutions.

4. **Effective against real-time phishing attacks**

   This study's ultimate goal was to reduce the impact of phishing on Internet users. As a result, real-time phishing detection is a critical feature of the proposed RDLM solution. Section 9.2.7 used a simulated web browser experiment to evaluate this characteristic of the RDLM. In this experiment, most phishing attacks were collected by reducing reporting and collection time to create a real-time environment. According to this experiment, RDLM has consistently maintained its phishing detection ability above 95%. Furthermore, as shown in Figure 9.5, the f1-score reached above 96.5% in many cases in the latter part. This shows that the RDLM is effective at detecting phishing attacks in real-time.

According to these interesting aspects of the RDLM solution, the RQ1 solution can be considered an effective anti-phishing solution. Therefore, with RDLM implementation, this study has successfully solved the RQ1 of this study.

Since RQ1 was directly linked to the RO2 of this study, a successful solution to RQ1 supported RO2 to achieve its goal. Therefore, the primary outcome of RO2 was the RDLM solution implemented through the RQ1. However, the success of RDLM depends on the continuous learning process. Section 6.7.2 and 9.2.2 experiments demonstrate this situation precisely. According to these experiments, the RDLM's detection accuracy and f1-score dropped by 3.36% and 2.69%, respectively, over a span of nearly four months. It was mainly due to the lack of support that the RDLM got from the continuous learning process. Therefore, RO3 was critical when achieving this study's aim.

RO3 – To incorporate an autonomous knowledge acquisition process to update the existing knowledge of the phishing detection features to minimise the performance loss over time

The continuous learning process is an essential aspect of the RDLM since RLM and DLM both need to learn over time to maintain their performance against the latest phishing attacks. RDLM primarily depends on RLM, an RL environment that is expected to improve its detection ability by interacting with real websites. On the other hand, the DLM also need to be retrained occasionally. Otherwise, its performance loss indirectly affects the RLM since the DLM is one critical observation of the RDLM architecture. Therefore, if the DLM deviates from the expected behaviour, it indirectly affects the performance of the RLM.

However, the initial idea of implementing the RDLM was to control the phishing detection ability until DLM could find a retraining point. This indicates that once a considerable amount of data is collected, the DLM could retrain to regain its normal behaviour to detect the latest phishing attacks. Until that, the RLM is responsible for managing the performance of RDLM with the support of external features alongside the DLM output. However, when achieving this study's aim, RDLM required a continuous learning process to update its knowledge to fight against the latest phishing attacks. In that step, finding answers to the second implication of this study, RQ2, was important. RQ2 mainly targeted a systematic knowledge acquisition process which supports RDLM to update its present phishing detection knowledge, and this is to minimise its performance loss. Therefore, when solving RQ2, this study first designed the knowledge acquisition process and then demonstrated how this process helped the RDLM to minimise its performance loss. The following section presents the proposed knowledge acquisition process and its impact on the RDLM when detecting phishing attacks over time.

RQ2 – How can existing knowledge of phishing detection features be automatically updated to minimise performance loss over time?

According to the literature, collecting phishing data is more challenging than legiti-

mate data because phishing websites are removed from the Internet within a shorter period. Therefore, collecting phishing data take more time than legitimate data in anti-phishing studies. However, these data cannot be constructed from a single night, and a continuous process is required. This problem is mainly associated with the solutions like RDLM, which takes multi-modal data. If a solution used only URLs, then phishing URLs could be downloaded from different phishing verification systems discussed in Section 3.3. Therefore, finding the latest labelled phishing data when acquiring new knowledge was a challenge when solving RQ2, as mentioned in the second and third challenges listed in Section 3.7. Hence, a practical approach for collecting and labelling training data was required first when solving RQ2. After finding a successful data collection and labelling process to overcome these challenges, a systematic knowledge acquisition process was proposed to update the existing knowledge of the RDLM.

Proposed solution: In Chapter 7, this study implemented an online phishing data repository called PhishRepo as the first step toward answering this RQ2. PhishRepo is an online phishing data repository which collects, verifies, disseminates and archives phishing data. Although it is considered a phishing data repository, it is designed to oversee both legitimate and phishing data during its process. However, the legitimate data collection was not promoted through PhishRepo and was used only with legitimate websites with high uncertain value. It is primarily due to the crowdsourcing approach used by PhishRepo. If more legitimate websites were passed to PhishRepo, the crowds' effort would become useless because online services like Google Search and Alexa can provide the same output without much effort. As a result, PhishRepo's main intention was to collect and label phishing websites and high uncertainty legitimate websites, as described in Chapter 8. This was a limitation in the proposed PhishRepo solution. However, this limitation was addressed by using the Google Search service. This service verified the legitimate websites that were not handled by PhishRepo to balance the number of legitimate and phishing websites collected by the knowledge acquisition process.

The primary goal of the proposed knowledge acquisition process was to col-

lect the most recent data and send it to the RDLM to acquire new phishing detection knowledge. It was mainly to identify the latest phishing attacking strategies to maintain the performance of the proposed anti-phishing solution. This process was a five-step process which has data production, data submission, labelling, data construction and automatic knowledge acquisition, as discussed in Chapter 8.

In the knowledge acquisition process, the relevant knowledge is primarily extracted by the RDLM. Therefore, the existing knowledge of phishing detection features will be updated automatically due to the architecture of the RDLM. Further, this knowledge acquisition will be effective only if the training data collected through this proposed process is diverse and up-to-date. However, when updating the phishing detection knowledge, the main influencer for the changed knowledge is the phishing websites because those are constantly changing. Therefore, the diversity of the collected phishing websites and the effectiveness of those in the learning process when identifying the latest phishing attacks were evaluated in Chapter 7.

Section 7.4.3 demonstrates that the PhishRepo dataset met all of the diversity requirements tested during the review, and the results achieved with regard to data leakage were exceptional. This tendency of data leakage was identified as a critical factor, but it has not been exercised before in the anti-phishing domain. However, PhishRepo solution integrated it into the data collection process as a deduplication filter to produce zero-duplicate data to eliminate the data leakage issue that could occur during the RDLM learning process. This deduplication filter is useful because the final phishing detection solution is autonomous and requires correct data to make accurate decisions. Figure 7.16 shows that the objective of this filter was successfully achieved by producing zero-duplicate data for the RDLM learning process.

Further, the effectiveness of the anti-phishing solutions trained by the PhishRepo dataset was evaluated with the latest phishing attacks in Section 7.5. In this experiment, PhishRepo's dataset outperformed both the benchmarking datasets by improving almost all the anti-phishing solutions' accuracies when detecting the latest phishing attacks. These results show that the phishing data collection approach pro-

vided in this work ensures quality data collection when updating existing phishing detection knowledge in anti-phishing solutions. According to these experiments and diversity analyses performed, the proposed knowledge acquisition process is well-suited to support the continuous learning process of the RDLM solution.

With this knowledge acquisition process, this study successfully found a solution to RQ2. Next, the effectiveness of this solution was evaluated to verify whether the performance loss of the RDLM was minimised after integrating the RQ2 solution with the RDLM.

Solution evaluation: The primary goal of the RQ2 was to update the RDLM's existing phishing detection knowledge to minimise future performance losses. However, due to the RDLM's black-boxing nature, updating existing knowledge cannot be visualised. As a result, when assessing this RQ2 solution, the RDLM's performance loss after a certain period and regained performance once the knowledge acquisition process integrated with it was considered. Based on that consideration, the proposed RQ2 solution was evaluated in three cases, as discussed below.

1. **RDLM improved its detection accuracy by 6% when integrated with the knowledge acquisition process.**

   According to Section 6.7.2, the phishing detection accuracy of the RDLM was 94.11%. However, in Section 9.2.1 RDLM (RLM1 instance), detection accuracy dropped by nearly 4% and was recorded as 90.75%. This indicates that within four months, the accuracy of the RDLM was decreased due to the absence of the knowledge acquisition process. However, the RLM3 instance, which received the knowledge acquisition process support for nearly six weeks, achieved 96.40% detection accuracy in the same experiment. This indicates that the RDLM increased its detection accuracy by 6% by getting the knowledge acquisition process support. The conclusion is that the proposed knowledge acquisition process supports RDLM in updating its existing phishing detection knowledge to have better phishing detection.

2. **RDLM recorded an increasing performance during the experiment**

   In Section 9.2.1 experiment, RDLM had 90.75% initial detection accuracy, but within two weeks, this accuracy was increased to 95.26%. After another four weeks, the accuracy of the RDLM reached 96.40% with the same test data, as shown in Figure 9.1. This implies that the RDLM has an increasing performance trend when exposed to more websites. It is mainly due to the continuous learning process integrated with the RDLM. It implies that the RDLM is updating its existing knowledge with the help of the knowledge acquisition process.

3. **DLM regained its typical behaviour through the knowledge acquisition process**

   As identified by Section 8.2.5.1, DLM noted a considerable phishing detection accuracy loss every three months. However, as shown in Figure 8.2, the DLM achieved 96.95%, 96.02%, and 96.42% detection accuracies during three re-training cycles performed during this study. Furthermore, Table 9.3 results showed that 8.54% of detection accuracy improvement of the RDLM was reported once the old DLM was replaced with the latest DLM. This indicates that the knowledge acquisition process supported the DLM to regain its typical behaviour over time by updating the existing phishing detection knowledge.

These cases indicate that the RDLM updated its phishing detection knowledge automatically with the support of the proposed knowledge acquisition process. Further, this knowledge acquisition process helped the RDLM to maintain an increasing performance trend by minimising the performance loss, as shown in Figures 9.1 and 9.5. Therefore, the proposed knowledge acquisition process can update the existing knowledge of phishing detection features automatically to minimise performance loss over time.

The goal of RO3 was to integrate a knowledge acquisition process with RDLM to update the existing knowledge of the phishing detection features to minimise its decreasing performance. Since this goal was linked to the RQ2 of this study, the success-

ful knowledge acquisition process found in RQ2 supported RO2 in achieving its goal. Therefore, the primary outcome of RO2 was the knowledge acquisition process integrated RDLM solution named SmartiPhish. Since this knowledge acquisition process proposed through RO3 did not involve direct human interaction, it was considered an autonomous knowledge acquisition process. Therefore, SmartiPhish is an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process.

SmartiPhish is a highly accurate real-time phishing detection solution that is effective against zero-day attacks. It can work effectively in balanced and imbalanced phishing environments with an average detection time of 4.34 seconds without network transfer time. Furthermore, a real-world SmartiPhish execution has taken 16.65 seconds of average time to detect a given request. It is comparatively high (Yang et al., 2019), but it would again depend on several conditions like hosting machine performance and network bandwidth. Once the SmartiPhish was implemented, this study found that the adversarial inputs at runtime threatened this proposed solution. As a result, a GN was proposed in Chapter 8 to control the runtime adversarial inputs effect on SmartiPhish. The proposed GN was reasonable because it recorded positive mean rewards for self-generated adversarial attacks during the evaluation, as shown in Figure 8.6. It indicates that most of these attacks were correctly identified by the SmartiPhish during the evaluation period.

As mentioned in the study's aim, the outcome of this work should minimise the impact of phishing attacks on Internet users. This indicates that a successful application of the implemented solution should exist to use with real users. As a result, the MORA Browser was proposed in Chapter 8. It was implemented to demonstrate how Internet users will protect themselves against phishing attacks with the help of SmartiPhish. The video shared publicly in Section 8.3.3 demonstrates how a user can access a legitimate website and be protected from phishing websites with SmartiPhish support. It further demonstrates the practical usage of SmartiPhish. However, this application is only one application of SmartiPhish, and it can be integrated with other relevant sources like smartphones since SmartiPhish was initially implemented as a

web service.

As previously mentioned, SmartiPhish is an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process. The performance presented in different experiments showed that SmartiPhish detected phishing attacks effectively for an extended period without having a noticeable performance drop. This infers that SmartiPhish can reduce the impact of phishing attacks via accurate detection. Then, the proposed MORA Browser is the application of SmartiPhish, which supports Internet users to benefit from the SmartiPhish solution. Therefore, this study finally proposed an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process to reduce the impact of phishing attacks on Internet users. This proposed solution was the aim of this study. It indicates that this study has achieved its aim successfully.

10.4 Resolving the research problem

Even though numerous anti-phishing solutions are available at present, this study has identified that these solutions have not considered the constantly changing phishing attacks which impact the significant phishing detection features. As a result, the performance of these anti-phishing solutions is declining over time, and it has become a significant problem in the current anti-phishing domain, as identified in Section 3.8. As a solution to this identified problem, this study proposed to develop an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process to reduce the impact of phishing attacks on Internet users. As a result, the SmartiPhish solution was developed through integration with MORA Browser.

As mentioned earlier, the main problem identified by this study was the performance drop of the existing anti-phishing solutions, which increases the phishing impact on Internet users. According to the literature, this performance drop is visible on machine-learning solutions after two months from the initial training. Therefore, a benchmark experiment was conducted using four machine learning solutions ini-

tially trained in November 2021. Subsequently, these solutions were again evaluated after three months with a separate dataset that included the latest phishing attacks, as described in Section 9.2.4. Once this experiment was completed, the results were evaluated to check the performance of these solutions, including the SmartiPhish. According to Table 9.5, these solutions showed good accuracy in the initial stage, but the accuracies were reduced between 3% to 8% after three months, as shown in Table 9.6. However, SmartiPhish had shown a 94.11% detection accuracy before starting this experiment in Section 6.7.2. Once this benchmarking experiment was performed with SmartiPhish, it showed that SmartiPhish achieved 96.40% detection accuracy. This indicates that SmartiPhish showed an increasing performance while others recorded a decreasing trend in this experiment.

With this benchmarking experiment results, this study has experimentally shown that the proposed SmartiPhish solution has not recorded a performance drop after a certain period. However, as presented in the literature, other solutions employed in this benchmarking experiment recorded a performance drop. This is mainly due to the continuous learning facility SmartiPhish has proposed to update the existing phishing detection knowledge. Therefore, this study concludes that the proposed SmartiPhish is a possible solution for the identified research problem.

10.5 Research novelty and contributions

According to the literature review conducted in Chapter 3, there is no phishing detection solution that can update its phishing detection knowledge automatically to adapt itself to the constantly changing phishing attacks. According to this study, this type of solution is a current demand for the anti-phishing domain to lessen the phishing impact on Internet users.

10.5.1 Research novelty

This study proposed an anti-phishing approach that can automatically update its phishing detection knowledge to be effective in the most recent phishing attacks. This approach is being used for the first time in the anti-phishing domain, and the results show

that it is a reliable solution to the identified problem. Furthermore, in this study, a GCN-based HTML content analysis approach was introduced to detect phishing attacks to overcome a challenge encountered during phase one of the proposed implementation process. It was a novel approach, and it became the first GNN application in the anti-phishing domain. Furthermore, to convert an HTML page into a GCN-based input, this GCN-based approach initially required an effective HTML page traverser to generate a GNN-compatible graph. According to the literature, no such traverser could perform this task effectively. As a result, this study has developed a novel HTML page traverser that can traverse any number of nested levels in a hierarchical order on a given web page.

Furthermore, this study filled a gap in phishing data collection by introducing an online phishing data repository that collects, verifies, distributes, and archives real-world phishing examples. With the support of this data repository, a reinforcement learning-based phishing detection framework was introduced as a novel concept. These new concepts and techniques contributed a large amount of knowledge to the anti-phishing domain, allowing for better phishing detection in the future.

10.5.2 Main contribution

The main contribution of this study was the SmartiPhish solution. It is an autonomous anti-phishing solution that can update the existing phishing detection knowledge via a systematic knowledge acquisition process. This contribution directly addresses the performance decreasing problem identified with the existing anti-phishing solutions. As shown in Table 9.2.1, SmartiPhish has increased its performance over time, while other solutions have recorded a decreased performance compared to their initial performance. This indicates that SmartiPhish is effective when detecting the latest phishing attacks. Therefore, SmartiPhish architecture opens a new direction for anti-phishing researchers to use a systematic knowledge acquisition process to update their phishing detection knowledge to detect the latest phishing attacks effectively.

10.5.3 Value-added contributions

In addition to this main contribution, this study introduced several other contributions to the scientific community. The following points discuss these specific contributions in detail.

- **A novel anti-phishing solution to detect phishing attacks using URL and HTML content features.**

  This study proposed the DLM, a representation learning-based phishing detection approach that uses URL and HTML features simultaneously. According to the accessed literature, this is the third phishing detection approach that uses representation learning to detect phishing attacks from URL and HTML content. Even though there are two similar solutions, this is the first anti-phishing solution using LRCN and GCN architecture in the anti-phishing domain. This solution outperforms the best similar solution reported in the literature. Therefore, the proposed DLM is a differentiated phishing detection approach which has recorded a high success in the anti-phishing domain. Furthermore, this contribution will benefit other anti-phishing researchers because they can explore this novel approach to have a better phishing detection in future.

- **A novel approach to detect phishing attacks using HTML content analysis.**

  In phishing detection, the HTML content analysis was conducted through manual feature extraction in the past. There was no representation learning-based strategy for analysing HTML content to develop an efficient phishing detection approach at the start of this work. Therefore, this study implemented a novel representation learning approach to analyse HTML contents when detecting phishing attacks. It used GCN to extract HTML content features for effective phishing detection. However, some other researchers introduced two representation learning-based HTML content analysis solutions. These solutions used LSTM and CNN to build their deep networks. Therefore, the proposed GCN approach will be the first GNN-based phishing detection deep network introduced in the anti-phishing domain. This GNN approach will open many research directions

in the anti-phishing field to have better solutions against this prevalent Internet threat.

- **A novel HTML page traverser which generates a GNN compatible graph from a given HTML page.**

  GNN is a graph-based neural network that requires graphs as inputs. Therefore, this study introduced a novel HTML page traverser to generate GNN-compatible graphs. At the initial stage of the GNN approach, this study faced several challenges, such as the dynamic content of web pages which generate different tree structures, length of the web page, and a different number of children elements under a parent element. Therefore, a functional programming approach was used to build an effective HTML page traverser using several python libraries. This HTML traverser was evaluated more than 100,000 times, and it generated accurate results every time. It can work with any page length and any number of nested elements when generating GNN-compatible graphs. Therefore, this novel HTML traverser will be a useful resource for the scientific community who likes to work with GNN. Furthermore, the logic used when developing this traverser could be used for different tasks by modifying specific code segments to extract information from the web page.

- **A reinforcement learning-based collaborative phishing detection framework.**

  The RDLM solution introduced in this research is a phishing detection framework. It is a reinforcement learning approach that uses several observations to decide whether a web page is phishing or legitimate. This architecture can be used in future research by changing the observations provided to the RL agent. However, it needs little engineering effort, but this RDLM can be used as a framework when displaying different observations like visual similarity score, Google page ranking and Alexa ranking to have differentiated phishing detection solutions in future.

- **An online phishing data repository uses to collect, validate, disseminate, and archive real-time phishing data.**

As identified in this study, a specific location to download trustable latest phishing data is a research gap in the anti-phishing domain. PhishRepo is the solution proposed to fill this identified research gap. PhishRepo, as mentioned earlier, is an online phishing data repository used to collect, validate, disseminate, and archive real-time phishing data. It used some innovative design artefacts to support better phishing data collection. Since PhishRepo is proposed as an online solution, other researchers also can use this to download the latest phishing data for their needs. Further, PhishRepo also gives support to RL environments by providing interactive feedback. Other researchers can also use this system to get the required data and labelling requirements. Therefore, this proposed PhishRepo will benefit anti-phishing researchers by eliminating their phishing data hassle.

- **A large-scale, multi-modal phishing data in raw format could be used for future research.**

  Since this research collected data at different time points for a successful evaluation process, a large amount of data has been collected during the study time. These data are in both legitimate and phishing categories and include several information sources like URLs, HTML pages, screenshots, Alexa information and request header details. These datasets were uploaded to the Mendeley online data repository for others' use. Therefore, other researchers also can use these datasets to develop effective phishing detection solutions.

- **A phishing-free web browser for safe browsing.**

  SmartiPhish was integrated with a web browser application named MORA browser to provide a phishing-free environment for Internet users. Any interested party can download this browser and use it for web surfing tasks. This browser included all the required facilities to perform a web search. However, it does not have some fancy stuff a modern browser has today. The shared video in Chapter 8 demonstrates its UI and how this browser works. Any Internet user can use this browser to have a phishing-free environment during web browsing.

10.6 Research limitations

Phishing attacks are mainly categorised under deceptive and technical subterfuge. These categories are again divided into several sub-categories, as mentioned in Chapter 3. Out of these, this study only focused on spoofed website attacks, a sub-category of deceptive phishing attacks. Therefore, the other types of phishing attacks are not considered when developing the proposed solution, and those attacks could not be detected via this solution. Even though the proposed solution detects only spoofed websites, the website needs to have a text/HTML web page to be employed in the detection process. Otherwise, the graphs generation process which comes under the HTMLDet model will fail, and the solution will not make any decision on that request. Therefore, any web page input must have text/HTML content type to have a successful phishing detection with the proposed solution.

In the proposed solution, the internal phishing detection features were extracted using a deep learning architecture. Generally, deep learning techniques are considered a black boxing technique that hides the visibility of the extracted features. Therefore, in the proposed solution, the extracted internal phishing detection features are not visible to the outside. As a result, the constantly changing significant phishing detection feature could not be shown to the outside. According to the literature, these phishing detection features must be updated after two months. However, the proposed solution updates the internal phishing detection features every three months due to the resource limitations such as phishing examples and high-performance computer costs associated with the retraining process.

The RL environment was updated with daily feedback every midnight. However, that feedback process is delayed feedback due to the proposed architecture. It was mainly due to the lack of experts in the labelling process. Therefore, the proposed labelling architecture uses online verification systems first and then moves to humans. In that process, once a submission enters the labelling process, it waits a minimum of two days before entering this process due to the limitations that exist with these verification systems. Therefore, the RL feedback process has a minimum of two days delay. Even though these verification services are used, these services are free services

which do not have any agreement with the proposed solution. Therefore, any shortages of these services will interrupt the proposed solution's execution, which is another limitation of this solution.

The proposed solution is a web service, and a browser can call this service to make a decision. However, this entire process takes 16.65 seconds, and frequently calling this service will make the user uncomfortable since it decreases the browser performance. Therefore, the proposed web browser calls to the proposed solution only at the beginning of a new website. As a result, the proposed architecture is vulnerable to dynamic attacks such as tab napping, which could be loaded after the web page is loaded to the users. It may be eliminated by making requests for every content change, but it will badly affect the browser's performance.

According to the architecture of the proposed solution, the requested web page is downloaded from the server-side. This adds another limitation to the proposed solution. That is, the unavailability of geographical location-based content. Since the requested location of the web page and downloaded location could be different, the location-based content might not be correctly reflected in the web page during the decision-making. Even though these detection time-related experiments were carried out in this study, these experiments were done in a controlled environment. It is mainly due to the time constraints and the sensitivity of phishing attacks. According to the nature of phishing attacks, asking real users to load phishing content is very challenging because if the proposed solution fails on some content, it may affect the user. Therefore, none of the real users was used in the presented experiments. Hence, real-world experiments could not be performed during the study.

Even though this study identified several adversarial attacks, it only addressed one type of attack due to the less impact of others, as described in Chapter 4. However, these justifications are based on literature, and there might be some impact on the other types of adversarial attacks. The experimented adversarial attacks were tested only in a controlled environment based on certain assumptions described in Chapter 8. However, failure of these assumptions might impact the solution execution, which is again a limitation of the proposed solution. Furthermore, the other type of adversarial

attacks were not tested during the study. Therefore, those attacks might be effective on the solution in actual execution. Further, the limitations of the execution environment are critical in this study. Since the proposed GCN-based architecture requires high-computational power and more time for a retraining cycle, time and resource limitations were added as limitations to this study. Therefore, the conducted experiments were limited and could not perform for a more extended time.

10.7 Summary

This chapter focused on a critical assessment of the objectives attained in this study to achieve its goal. The first objective, RO1, is directly related to the literature and achieved through a thorough literature review. The research questions RQ1 and RQ2 were then directly linked to the RO2 and RO3 objectives. This study proposed and evaluated an RDLM and knowledge acquisition process as solutions to these two questions. Following the successful implementation of these two solutions, RO2 and RO3 were achieved, which ultimately accomplished the aim of this study. The proposed solution was a novel anti-phishing solution that has made several contributions to the field. However, like many other solutions, this one has some limitations, which were explicitly discussed here. Since the research goal has now been accomplished, the following chapter will introduce the concluding remarks by summarising the key points highlighted throughout this study.

11 CONCLUSION AND RECOMMENDATIONS

11.1 Introduction

Chapter 10 provided a critical evaluation of the study's aim and objectives. It discussed the proposed solution's contributions, emphasising its novelty while highlighting its limitations. This chapter focuses on the final remarks of the conducted study because the key objectives of the study have been achieved at this level. As a result, this chapter will summarise the key research findings and discuss how the knowledge developed from this study has been disseminated to the scientific community. This chapter also suggests specific research directions that may be undertaken in the near future.

11.2 Overall findings

This research aimed to develop an autonomous anti-phishing solution that can update existing phishing detection knowledge through a systematic knowledge acquisition process to reduce the impact of phishing attacks on Internet users. The results show that the anti-phishing solution developed in this study can detect phishing attacks with high accuracy and reduce the impact of phishing attacks on Internet users. This solution achieved 96.40% detection accuracy and 96.42% f1-score in the most recent testing period and may improve over time due to the continuous learning support provided by this solution. Further experiments reveal that this study effectively answered the major research problem, which was the performance degradation of previous anti-phishing solutions. According to these experiments, the proposed solution improves accuracy by 6% over time, whereas other benchmark solutions degrade over time.

Additionally, two research questions are posed, RQ1: How can an effective anti-phishing solution be implemented while overcoming the identified challenges? and RQ2: How can existing knowledge of phishing detection features be automatically up-

dated to minimise performance loss over time? And these questions should be resolved at the outset. The first question was answered using a phishing detection framework based on reinforcement learning, and the second was solved by integrating a systematic knowledge acquisition process with the RQ1 output. After answering these two questions, this study accomplished its goal by introducing SmartiPhish, a real-time phishing detection solution that continuously learns to adapt to constantly changing phishing attacks.

SmartiPhish is the first anti-phishing solution, employing an innovative phishing detection approach that includes powerful GNN and RL architectures. SmartiPhish demonstrated significant performance during the experiments, particularly against zero-day attacks, where it maintained 95.34% detection accuracy against the experimented zero-day attacks. SmartiPhish took 4.34 seconds to decide on a specific website, and the overall detection time was 16.65 seconds. However, the 16.65 seconds includes the network transfer time, which may vary depending on SmartiPhish's operating environment. Furthermore, in most cases, SmartiPhish's f1-score was at the top among benchmark solutions in the natural web environment, and it was 90.81% for SmartiPhish.

In addition to these findings, this study filled an existing research gap identified in the anti-phishing domain during the implementation of the knowledge acquisition process. This study proposed PhishRepo, an innovative systematic phishing data collection approach which collects, labels, disseminates, and archives phishing websites. PhishRepo is an online phishing data repository enriched with innovative design considerations such as automated submission, deduplication filtering, automated verification, crowdsourcing-based labelling, an objection reporting window, and target attack prevention techniques. Furthermore, when training an anti-phishing solution, this PhishRepo solution collects the most recent and zero-duplicate data. Therefore, the PhishRepo collected data outperforms when training the existing machine learning models over two similar benchmark datasets. The PhishRepo data increased the phishing attack detection ability of all of the selected machine learning-based solutions, according to the presented experiments.

Moreover, this study also proposed a practical application for the developed SmartiPhish solution. It is a web browser named MORA Browser. It has used a modern browser environment with the SmartiPhish solution, and it supports a phishing-free browsing experience to reduce the impact of phishing attacks on Internet users. These findings indicate that an effective anti-phishing solution that can automatically update the existing phishing detection knowledge via continuous learning can maintain its performance for a more extended time.

11.3 Dissemination of the knowledge

After a successful problem-solving stage, this study developed a set of knowledge that can be transferred to the scientific community for future anti-phishing-related studies. This knowledge was in different forms, such as theories, datasets, and practical implementations. In the knowledge dissemination step, this study transferred the developed knowledge via publications[33] and online repositories such as GitHub and Mendeley.

The theoretical knowledge of this research was mainly transferred via a written medium that includes a dissertation and four journal articles. These articles contained the implementation processes followed in SmartiPhish, DLM, PhishRepo and Hybrid DLM solutions. These articles further presented extensive experiments to validate these solutions and included descriptive discussion to transfer the developed knowledge to the scientific community. By accessing these written media, the anti-phishing domain can better understand how these novel concepts were introduced and the implications of these solutions. It will be crucial for future anti-phishing studies to eliminate repeated work and develop differentiated solutions.

The phishing URLs collected during this study mainly depended on online phishing verification systems such as PhishTank and OpenPhish. After that, these URLs were used to collect publicly available data to construct relevant phishing datasets for this study. These verification systems provided free data access permissions for academic researchers and have no restrictions on sharing these data. Further, the legitimate data were mainly collected via Google search and various public resources. Therefore, the

[33]The journal publications relevant to this study are listed in Appendix B.

data collected during this study was not associated with legal compliance when shared with others. Thus, this study disseminated the constructed phishing and legitimate datasets via the Mendeley cloud-based repository (*see* Appendix B) for others' use.

This study was primarily implementation-oriented. Therefore, the planned solutions were practically implemented to evaluate their actual execution performance. Hence, this study proposed different novel implementations for the scientific community. These implementations may support other researchers when doing similar works in any domain. Therefore, the study's main codebase was shared via GitHub (*see* Appendix B) to increase the reusability of the developed knowledge.

11.4 Recommendations for future research

The proposed solution for the identified research problem was SmartiPhish. However, this solution was primarily restricted to spoofed website attacks, which are only a subset of deceptive phishing attacks described in the literature. As a result, this study's findings can be applied to other phishing attack categories to check whether any interesting findings can be extracted. Furthermore, the proposed solution only considers text/html content type web pages to resolve graph generation issues that occur with other types of web pages such as application/pdf. A separate graph generation process that can be used with any web page will be an added benefit for the proposed architecture to become more robust than it is now.

As stated in the limitations, the proposed solution is vulnerable to dynamic attacks that can be carried out by client-side technologies such as Javascript. In the future, some solutions to reduce these attacks may be required to improve the reliability of the solution. Furthermore, this study only tested the research outcome for a limited period due to resource and time constraints. However, in the future, the solution should be monitored for an extended period to address any potential issues that may arise during real-world implementation. Additionally, evaluating the model's accuracy through several sets of experiments will enable the conduction of statistical significance tests. These tests will help to compare the proposed solution with other benchmark solutions and determine its ranking among the other proposed solution.

The proposed RDLM architecture can be extended to include more observations as a separate work to enhance its detection capabilities in the future. In that case, it is worth exploring the potential of using certificates for phishing detection. While this aspect was not studied in the current research, certificate-based techniques have the potential to improve phishing detection methods and enhance overall security measures. Such investigations can lead to advancements in the field of cybersecurity and the development of stronger tools to combat phishing attacks. Despite not currently utilizing certificate-related information, the RLM part of SmartiPhish can still adapt to receive new input, such as certificate-related data, and train itself to be an effective anti-phishing solution.

Furthermore, the RL agent and deep network structures employed in the proposed solution can be adjusted in the future to have better combinations in phishing detection. Such experiments may be helpful to find lesser detection time and lower the hardware requirements when detecting phishing attacks. In addition to that, the behaviour of the proposed solution with GN may require a more extended execution than experimented here to see its actual impacts. These findings will help fine-tune the proposed solution to work as expected. From the PhishRepo side, a few more anti-phishing communities can be integrated under the alpha labelling process to strengthen its labelling capacity and reduce the human workload in the verification. Furthermore, archiving some incorrect pages (e.g., 403 pages, 404 pages, and content not found pages) impacts the PhishRepo data quality. As a result, future work may include automatically detecting erroneous or unwanted pages via web page screenshots and removing such data points from the repository. Then, the quality of the data collected through PhishRepo could be improved even further.

Moreover, to gain deeper insights into the performance of the proposed approach and the specific contributions of each module, conducting a comprehensive ablation study is crucial. This study would systematically analyse the impact of individual components, shedding light on their effectiveness within the overall solution. The findings from this ablation study would enable further optimisation and refinement of SmartiPhish, enhancing its capabilities and effectiveness.

11.5 Summary

The primary outcome of this study was SmartiPhish. It is a self-contained anti-phishing solution that continuously updates the phishing detection features. Furthermore, this primary outcome was recorded with a detection accuracy of 96.40%, and it was a feasible solution for the identified research problem because it improved its detection ability by 6% over time. This SmartiPhish development process yielded a wealth of valuable knowledge to anti-phishing researchers. As a result, the developed knowledge was transferred to the scientific community via standard media. Finally, this chapter concludes this dissertation's main content by recommending future directions for the conducted research. The references and appendices sections follow to provide additional references for the presented content.

REFERENCES

Aassal, A. E., Baki, S., Das, A., & Verma, R. M. (2020). An in-depth benchmarking and evaluation of phishing detection research for security needs. *IEEE Access*, *8*, 22170–22192. Retrieved from https://doi.org/10.1109/access.2020.2969780 doi: 10.1109/access.2020.2969780

Aburrous, M., Hossain, M., Dahal, K., & Thabtah, F. (2010, December). Intelligent phishing detection system for e-banking using fuzzy data mining. *Expert Systems with Applications*, *37*(12), 7913–7921. Retrieved from https://doi.org/10.1016/j.eswa.2010.04.044 doi: 10.1016/j.eswa.2010.04.044

Afroz, S., & Greenstadt, R. (2011, September). PhishZoo: Detecting phishing websites by looking at them. In *2011 IEEE fifth international conference on semantic computing*. IEEE. Retrieved from https://doi.org/10.1109/icsc.2011.52 doi: 10.1109/icsc.2011.52

Alabdan, R. (2020, September). Phishing attacks survey: Types, vectors, and technical approaches. *Future Internet*, *12*(10), 168. Retrieved from https://doi.org/10.3390/fi12100168 doi: 10.3390/fi12100168

AlEroud, A., & Karabatis, G. (2020, March). Bypassing detection of URL-based phishing attacks using generative adversarial deep neural networks. ACM. Retrieved from https://doi.org/10.1145/3375708.3380315 doi: 10.1145/3375708.3380315

Alkhalil, Z., Hewage, C., Nawaf, L., & Khan, I. (2021, March). Phishing attacks: A recent comprehensive study and a new anatomy. , *3*. Retrieved from https://doi.org/10.3389/fcomp.2021.563060 doi: 10.3389/fcomp.2021.563060

APWG. (2021a). Phishing activity trends report: 2nd quarter 2021. *Anti-Phishing Working Group. Retrieved September, 22*, 12.

APWG. (2021b). Phishing activity trends report: 4th quarter 2020. *Anti-Phishing*

Working Group. Retrieved February, 09, 13.

Ariyadasa, S., Fernando, S., & Fernando, S. (2020, July). Detecting phishing attacks using a combined model of LSTM and CNN. *International Journal of ADVANCED AND APPLIED SCIENCES*, *7*(7), 56–67. Retrieved from https://doi.org/10.21833/ijaas.2020.07.007 doi: 10.21833/ijaas.2020.07.007

Bahnsen, A. C., Bohorquez, E. C., Villegas, S., Vargas, J., & Gonzalez, F. A. (2017, April). Classifying phishing URLs using recurrent neural networks. In *2017 APWG symposium on electronic crime research (eCrime)*. IEEE. Retrieved from https://doi.org/10.1109/ecrime.2017.7945048 doi: 10.1109/ecrime.2017.7945048

Bahnsen, A. C., Torroledo, I., Camacho, L. D., & Villegas, S. (2018). Deepphish : Simulating malicious ai..

Baslyman, M., & Chiasson, S. (2016, June). "smells phishy?": An educational game about online phishing scams. In *2016 APWG symposium on electronic crime research (eCrime)*. IEEE. Retrieved from https://doi.org/10.1109/ecrime.2016.7487946 doi: 10.1109/ecrime.2016.7487946

Beck, K., & Zhan, J. (2010, August). Phishing using a modified bayesian technique. In *2010 IEEE second international conference on social computing*. IEEE. Retrieved from https://doi.org/10.1109/socialcom.2010.100 doi: 10.1109/socialcom.2010.100

Bell, S., & Komisarczuk, P. (2020, January). An analysis of phishing blacklists: Google safe browsing, OpenPhish, and PhishTank. In *Proceedings of the australasian computer science week multiconference*. ACM. Retrieved from https://doi.org/10.1145/3373017.3373020 doi: 10.1145/3373017.3373020

Bianchi, F. M., Grattarola, D., Livi, L., & Alippi, C. (2021). Graph neural networks with convolutional ARMA filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1. Retrieved from https://doi.org/10.1109/tpami.2021.3054830 doi: 10.1109/tpami.2021.3054830

Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning

methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18*(2), 1153–1176. Retrieved from https://doi.org/10.1109/comst.2015.2494502 doi: 10.1109/comst.2015.2494502

Butnaru, A., Mylonas, A., & Pitropakis, N. (2021, June). Towards lightweight URL-based phishing detection. *Future Internet*, *13*(6), 154. Retrieved from https://doi.org/10.3390/fi13060154 doi: 10.3390/fi13060154

Canova, G., Volkamer, M., Bergmann, C., & Borza, R. (2014). NoPhish: An anti-phishing education app. In *Security and trust management* (pp. 188–192). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-319-11851-2_14 doi: 10.1007/978-3-319-11851-2_14

Cao, Y., Han, W., & Le, Y. (2008). Anti-phishing based on automated individual white-list. In *Proceedings of the 4th ACM workshop on digital identity management - dim' 08*. ACM Press. Retrieved from https://doi.org/10.1145/1456424.1456434 doi: 10.1145/1456424.1456434

Chang, J. C., Amershi, S., & Kamar, E. (2017, May). Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *Proceedings of the 2017 CHI conference on human factors in computing systems*. ACM. Retrieved from https://doi.org/10.1145/3025453.3026044 doi: 10.1145/3025453.3026044

Chatterjee, M., & Namin, A. S. (2019). Deep reinforcement learning for detecting malicious websites. *CoRR*, *abs/1905.09207*. Retrieved from http://arxiv.org/abs/1905.09207

Chauhan, N. K., & Singh, K. (2018, September). A review on conventional machine learning vs deep learning. In *2018 international conference on computing, power and communication technologies (GUCON)*. IEEE. Retrieved from https://doi.org/10.1109/gucon.2018.8675097 doi: 10.1109/gucon.2018.8675097

Chen, J.-L., Ma, Y.-W., & Huang, K.-L. (2020, October). Intelligent visual similarity-based phishing websites detection. *Symmetry*, *12*(10), 1681. Retrieved from https://doi.org/10.3390/sym12101681 doi: 10.3390/sym12101681

Chen, W., Zhang, W., & Su, Y. (2018). Phishing detection research based on LSTM recurrent neural network. In *Communications in computer and information science* (pp. 638–645). Springer Singapore. Retrieved from https://doi.org/10.1007/978-981-13-2203-7_52 doi: 10.1007/978-981-13-2203-7_52

Chiew, K. L., Chang, E. H., Tan, C. L., Abdullah, J., & Yong, K. S. C. (2018). Building standard offline anti-phishing dataset for benchmarking. *International Journal of Engineering & Technology*, *7*(4.31), 7–14.

Chiew, K. L., Tan, C. L., Wong, K., Yong, K. S., & Tiong, W. K. (2019, May). A new hybrid ensemble feature selection framework for machine learning-based phishing detection system. *Information Sciences*, *484*, 153–166. Retrieved from https://doi.org/10.1016/j.ins.2019.01.064 doi: 10.1016/j.ins.2019.01.064

Chiew, K. L., Yong, K. S. C., & Tan, C. L. (2018, September). A survey of phishing attacks: Their types, vectors and technical approaches. *Expert Systems with Applications*, *106*, 1–20. Retrieved from https://doi.org/10.1016/j.eswa.2018.03.050 doi: 10.1016/j.eswa.2018.03.050

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2017). Generative adversarial networks: An overview. *CoRR*, *abs/1710.07035*. Retrieved from http://arxiv.org/abs/1710.07035

Denning, T., Lerner, A., Shostack, A., & Kohno, T. (2013). Control-alt-hack. In *Proceedings of the 2013 ACM SIGSAC conference on computer & communications security - CCS '13*. ACM Press. Retrieved from https://doi.org/10.1145/2508859.2516753 doi: 10.1145/2508859.2516753

Desai, A., Jatakia, J., Naik, R., & Raul, N. (2017, May). Malicious web content detection using machine leaning. In *2017 2nd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE. Retrieved from https://doi.org/10.1109/rteict.2017.8256834 doi: 10.1109/rteict.2017.8256834

Dixon, M., Arachchilage, N. A. G., & Nicholson, J. (2019, May). Engaging users with educational games. In *Extended abstracts of the 2019 CHI conference on*

human factors in computing systems. ACM. Retrieved from https://doi.org/10.1145/3290607.3313026 doi: 10.1145/3290607.3313026

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015, June). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr).*

Dong, X., Clark, J. A., & Jacob, J. (2008, May). Modelling user-phishing interaction. In *2008 conference on human system interactions.* IEEE. Retrieved from https://doi.org/10.1109/hsi.2008.4581513 doi: 10.1109/hsi.2008.4581513

Dong, Z., Kane, K., & Camp, L. J. (2016). Detection of rogue certificates from trusted certificate authorities using deep neural networks. *ACM Transactions on Privacy and Security (TOPS)*, *19*(2), 1–31.

Dong, Z., Kapadia, A., Blythe, J., & Camp, L. J. (2015). Beyond the lock icon: real-time detection of phishing websites using public key certificates. In *2015 apwg symposium on electronic crime research (ecrime)* (pp. 1–12).

Dou, Z., Khalil, I., Khreishah, A., Al-Fuqaha, A., & Guizani, M. (2017). Systematization of knowledge (SoK): A systematic review of software-based web phishing detection. *IEEE Communications Surveys & Tutorials*, *19*(4), 2797–2819. Retrieved from https://doi.org/10.1109/comst.2017.2752087 doi: 10.1109/comst.2017.2752087

Drury, V., & Meyer, U. (2019). Certified phishing: taking a look at public key certificates of phishing websites. In *Fifteenth symposium on usable privacy and security (soups 2019)* (pp. 211–223).

Drutsa, A., Farafonova, V., Fedorova, V., Megorskaya, O., Zerminova, E., & Zhilinskaya, O. (2019). Practice of efficient data collection via crowdsourcing at large-scale. *CoRR*, *abs/1912.04444*. Retrieved from http://arxiv.org/abs/1912.04444

Dunlop, M., Groat, S., & Shelly, D. (2010). GoldPhish: Using images for content-based phishing analysis. In *2010 fifth international conference on internet mon-*

itoring and protection. IEEE. Retrieved from https://doi.org/10.1109/icimp.2010.24 doi: 10.1109/icimp.2010.24

Eickhoff, C. (2018, February). Cognitive biases in crowdsourcing. In *Proceedings of the eleventh ACM international conference on web search and data mining.* ACM. Retrieved from https://doi.org/10.1145/3159652.3159654 doi: 10.1145/3159652.3159654

El-Alfy, E.-S. M. (2017, April). Detection of phishing websites based on probabilistic neural networks and k-medoids clustering. *The Computer Journal*, *60*(12), 1745–1759. Retrieved from https://doi.org/10.1093/comjnl/bxx035 doi: 10.1093/comjnl/bxx035

ENISA. (2020). *Enisa threat landscape - phishing.* Publications Office. Retrieved from https://data.europa.eu/doi/10.2824/552242 doi: 10.2824/552242

ENISA. (2021). *Enisa threat landscape 2021: April 2020 to mid july 2021.* Publications Office. Retrieved from https://data.europa.eu/doi/10.2824/324797 doi: 10.2824/324797

Feng, J., Zou, L., Ye, O., & Han, J. (2020). Web2vec: Phishing webpage detection method based on multidimensional features driven by deep learning. , *8*, 221214–221224. Retrieved from https://doi.org/10.1109/access.2020.3043188 doi: 10.1109/access.2020.3043188

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, *11*(3-4), 219–354. Retrieved from https://doi.org/10.1561/2200000071 doi: 10.1561/2200000071

Fu, A. Y., Wenyin, L., & Deng, X. (2006, October). Detecting phishing web pages with visual similarity assessment based on earth mover's distance (EMD). *IEEE Transactions on Dependable and Secure Computing*, *3*(4), 301–311. Retrieved from https://doi.org/10.1109/tdsc.2006.50 doi: 10.1109/tdsc.2006.50

Goodfellow, I., Bengio, Y., & Courville, A. (2017). Deep learning (adaptive computation and machine learning series). *Cambridge Massachusetts*.

Gowtham, R., & Krishnamurthi, I. (2014). A comprehensive and efficacious architec-

ture for detecting phishing webpages. *Computers & Security*, *40*, 23–37.

Gu, X., Wang, H., & Ni, T. (2013). An efficient approach to detecting phishing web. *Journal of Computational Information Systems*, *9*(14), 5553–5560.

Gupta, B. B., Arachchilage, N. A. G., & Psannis, K. E. (2017, May). Defending against phishing attacks: taxonomy of methods, current issues and future directions. *Telecommunication Systems*, *67*(2), 247–267. Retrieved from https://doi.org/10.1007/s11235-017-0334-z doi: 10.1007/s11235-017-0334-z

Gupta, B. B., Tewari, A., Jain, A. K., & Agrawal, D. P. (2016, March). Fighting against phishing attacks: state of the art and future challenges. *Neural Computing and Applications*, *28*(12), 3629–3654. Retrieved from https://doi.org/10.1007/s00521-016-2275-y doi: 10.1007/s00521-016-2275-y

Hansen, D. L., Schone, P. J., Corey, D., Reid, M., & Gehring, J. (2013). Quality control mechanisms for crowdsourcing. In *Proceedings of the 2013 conference on computer supported cooperative work - CSCW '13*. ACM Press. Retrieved from https://doi.org/10.1145/2441776.2441848 doi: 10.1145/2441776.2441848

Huang, C.-Y., Ma, S.-P., Yeh, W.-L., Lin, C.-Y., & Liu, C.-T. (2010, November). Mitigate web phishing using site signatures. In *TENCON 2010 - 2010 IEEE region 10 conference*. IEEE. Retrieved from https://doi.org/10.1109/tencon.2010.5686582 doi: 10.1109/tencon.2010.5686582

Huang, H., Zhong, S., & Tan, J. (2009). Browser-side countermeasures for deceptive phishing attack. In *2009 fifth international conference on information assurance and security*. IEEE. Retrieved from https://doi.org/10.1109/ias.2009.12 doi: 10.1109/ias.2009.12

Jain, A. K., & Gupta, B. B. (2016, May). A novel approach to protect against phishing attacks at client side using auto-updated white-list. *EURASIP Journal on Information Security*, *2016*(1). Retrieved from https://doi.org/10.1186/s13635-016-0034-3 doi: 10.1186/s13635-016-0034-3

Jain, A. K., & Gupta, B. B. (2017). Phishing detection: Analysis of visual similarity based approaches. *Security and Communication Networks*, *2017*, 1–20. Re-

trieved from https://doi.org/10.1155/2017/5421046 doi: 10.1155/2017/5421046

Jain, A. K., & Gupta, B. B. (2018a, April). A machine learning based approach for phishing detection using hyperlinks information. *Journal of Ambient Intelligence and Humanized Computing*, *10*(5), 2015–2028. Retrieved from https://doi.org/10.1007/s12652-018-0798-z doi: 10.1007/s12652-018-0798-z

Jain, A. K., & Gupta, B. B. (2018b). PHISH-SAFE: URL features-based phishing detection system using machine learning. In *Advances in intelligent systems and computing* (pp. 467–474). Springer Singapore. Retrieved from https://doi.org/10.1007/978-981-10-8536-9_44 doi: 10.1007/978-981-10-8536-9_44

Jeeva, S. C., & Rajsingh, E. B. (2016, July). Intelligent phishing url detection using association rule mining. *Human-centric Computing and Information Sciences*, *6*(1). Retrieved from https://doi.org/10.1186/s13673-016-0064-3 doi: 10.1186/s13673-016-0064-3

Joshi, Y., Saklikar, S., Das, D., & Saha, S. (2008, December). PhishGuard: A browser plug-in for protection from phishing. In *2008 2nd international conference on internet multimedia services architecture and applications.* IEEE. Retrieved from https://doi.org/10.1109/imsaa.2008.4753929 doi: 10.1109/imsaa.2008.4753929

Kamiri, J., & Mariga, G. (2021). Research methods in machine learning: A content analysis. *International Journal of Computer and Information Technology (2279-0764)*, *10*(2).

Kashyap, R. (2020, Jan). *Council post: Are you ready for the age of adversarial ai? attackers can leverage artificial intelligence too.* Forbes Magazine. Retrieved from https://www.forbes.com/sites/forbestechcouncil/2020/01/09/are-you-ready-for-the-age-of-adversarial-ai-attackers-can-leverage-artificial-intelligence-too/?sh=76e9150d4703

Khonji, M., Iraqi, Y., & Jones, A. (2013). Phishing detection: A literature survey. *IEEE Communications Surveys & Tutorials*, *15*(4), 2091–2121. Retrieved from

https://doi.org/10.1109/surv.2013.032213.00009 doi: 10.1109/surv.2013.032213.00009

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, *abs/1609.02907*. Retrieved from http://arxiv.org/abs/1609.02907

Knickerbocker, P., Yu, D., & Li, J. (2009). Humboldt: A distributed phishing disruption system. In *2009 ecrime researchers summit* (pp. 1–12).

Le, H., Pham, Q., Sahoo, D., & Hoi, S. C. H. (2018). Urlnet: Learning a URL representation with deep learning for malicious URL detection. *CoRR*, *abs/1802.03162*. Retrieved from http://arxiv.org/abs/1802.03162

Le, Q., & Mikolov, T. (2014, 22–24 Jun). Distributed representations of sentences and documents. In E. P. Xing & T. Jebara (Eds.), *Proceedings of the 31st international conference on machine learning* (Vol. 32, pp. 1188–1196). Bejing, China: PMLR. Retrieved from https://proceedings.mlr.press/v32/le14.html

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May). Deep learning. *Nature*, *521*(7553), 436–444. Retrieved from https://doi.org/10.1038/nature14539 doi: 10.1038/nature14539

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, *abs/2005.01643*. Retrieved from https://arxiv.org/abs/2005.01643

Li, Y., Yang, Z., Chen, X., Yuan, H., & Liu, W. (2019, May). A stacking model using URL and HTML features for phishing webpage detection. *Future Generation Computer Systems*, *94*, 27–39. Retrieved from https://doi.org/10.1016/j.future.2018.11.004 doi: 10.1016/j.future.2018.11.004

Lin, Y., Liu, R., Divakaran, D. M., Ng, J. Y., Chan, Q. Z., Lu, Y., ... Dong, J. S. (2021, August). Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages. In *30th usenix security symposium (usenix security 21)* (pp. 3793–3810). USENIX Association. Retrieved from https://www.usenix.org/conference/usenixsecurity21/presentation/lin

Mao, J., Tian, W., Li, P., Wei, T., & Liang, Z. (2017). Phishing-alarm: Robust and effi-

cient phishing detection via page component similarity. *IEEE Access*, *5*, 17020–17030. Retrieved from https://doi.org/10.1109/access.2017.2743528 doi: 10.1109/access.2017.2743528

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015, February). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. Retrieved from https://doi.org/10.1038/nature14236 doi: 10.1038/nature14236

Mohammad, A. H., & Al Saiyd, N. A. M. (2010). A framework for expert knowledge acquisition. *IJCSNS*, *10*(11), 145.

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2012). An assessment of features related to phishing websites using an automated technique. In *2012 international conference for internet technology and secured transactions* (pp. 492–497).

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2013, November). Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications*, *25*(2), 443–458. Retrieved from https://doi.org/10.1007/s00521-013-1490-z doi: 10.1007/s00521-013-1490-z

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2014). Intelligent rule-based phishing websites classification. *IET Information Security*, *8*(3), 153–160.

Mohammad, R. M., Thabtah, F., & McCluskey, L. (2015, August). Tutorial and critical analysis of phishing websites methods. *Computer Science Review*, *17*, 1–24. Retrieved from https://doi.org/10.1016/j.cosrev.2015.04.001 doi: 10.1016/j.cosrev.2015.04.001

Mousavi, S. S., Schukat, M., & Howley, E. (2017, August). Deep reinforcement learning: An overview. In *Proceedings of SAI intelligent systems conference (IntelliSys) 2016* (pp. 426–440). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-319-56991-8_32 doi: 10.1007/978-3-319-56991-8_32

Netcraft. (2021, Dec). *Web server survey.* Retrieved from https://news.netcraft.com/archives/category/web-server-survey/

Nguyen, D. T., Alam, F., Ofli, F., & Imran, M. (2017). *Automatic image filtering on*

social networks using deep learning and perceptual hashing during crises.

Nguyen, L. A. T., To, B. L., Nguyen, H. K., & Nguyen, M. H. (2014, October). An efficient approach for phishing detection using single-layer neural network. In *2014 international conference on advanced technologies for communications (ATC 2014).* IEEE. Retrieved from https://doi.org/10.1109/atc.2014.7043427 doi: 10.1109/atc.2014.7043427

Nguyen, L. D., Le, D.-N., & Vinh, L. T. (2014). Detecting phishing web pages based on DOM-tree structure and graph matching algorithm. In *Proceedings of the fifth symposium on information and communication technology - SoICT '14.* ACM Press. Retrieved from https://doi.org/10.1145/2676585.2676596 doi: 10.1145/2676585.2676596

Odeh, A., Keshta, I., & Abdelfattah, E. (2021). Phiboost-a novel phishing detection model using adaptive boosting approach. *Jordanian Journal of Computers and Information Technology (JJCIT)*, *7*(01).

Opara, C., Chen, Y., & Wei, B. (2020). Look before you leap: Detecting phishing web pages by exploiting raw URL and HTML characteristics. *CoRR*, *abs/2011.04412*. Retrieved from https://arxiv.org/abs/2011.04412

Opara, C., Wei, B., & Chen, Y. (2020, July). HTMLPhish: Enabling phishing web page detection by applying deep learning techniques on HTML analysis. In *2020 international joint conference on neural networks (IJCNN).* IEEE. Retrieved from https://doi.org/10.1109/ijcnn48605.2020.9207707 doi: 10.1109/ijcnn48605.2020.9207707

Orunsolu, A., Sodiya, A., & Akinwale, A. (2019, December). A predictive model for phishing detection. *Journal of King Saud University - Computer and Information Sciences*. Retrieved from https://doi.org/10.1016/j.jksuci.2019.12.005 doi: 10.1016/j.jksuci.2019.12.005

Pham, T. T. T., Hoang, V. N., & Ha, T. N. (2018). Exploring efficiency of character-level convolution neuron network and long short term memory on malicious URL detection. In *Proceedings of the 2018 VII international conference on network, communication and computing - ICNCC 2018.* ACM Press. Retrieved

from https://doi.org/10.1145/3301326.3301336 doi: 10.1145/3301326.3301336

Prakash, P., Kumar, M., Kompella, R. R., & Gupta, M. (2010, March). PhishNet: Predictive blacklisting to detect phishing attacks. In *2010 proceedings IEEE INFOCOM*. IEEE. Retrieved from https://doi.org/10.1109/infcom.2010.5462216 doi: 10.1109/infcom.2010.5462216

Pratiwi, M. E., Lorosae, T. A., & Wibowo, F. W. (2018, December). Phishing site detection analysis using artificial neural network. *Journal of Physics: Conference Series*, *1140*, 012048. Retrieved from https://doi.org/10.1088/1742-6596/1140/1/012048 doi: 10.1088/1742-6596/1140/1/012048

Ramasubramanian, K., & Singh, A. (2018, December). Deep learning using keras and TensorFlow. In *Machine learning using r* (pp. 667–688). Apress. Retrieved from https://doi.org/10.1007/978-1-4842-4215-5_11 doi: 10.1007/978-1-4842-4215-5_11

Rosiello, A. P. E., Kirda, E., Kruegel, C., & Ferrandi, F. (2007). A layout-similarity-based approach for detecting phishing pages. In *2007 third international conference on security and privacy in communications networks and the workshops - SecureComm 2007*. IEEE. Retrieved from https://doi.org/10.1109/seccom.2007.4550367 doi: 10.1109/seccom.2007.4550367

Sahingoz, O. K., Buber, E., Demir, O., & Diri, B. (2019, March). Machine learning based phishing detection from URLs. *Expert Systems with Applications*, *117*, 345–357. Retrieved from https://doi.org/10.1016/j.eswa.2018.09.029 doi: 10.1016/j.eswa.2018.09.029

Sahoo, D., Liu, C., & Hoi, S. C. H. (2017). Malicious URL detection using machine learning: A survey. *CoRR*, *abs/1701.07179*. Retrieved from http://arxiv.org/abs/1701.07179

Sameen, M., Han, K., & Hwang, S. O. (2020). PhishHaven—an efficient real-time AI phishing URLs detection system. , *8*, 83425–83443. Retrieved from https://doi.org/10.1109/access.2020.2991403 doi: 10.1109/access.2020.2991403

Sánchez-Paniagua, M., Fidalgo, E., González-Castro, V., & Alegre, E. (2020, August). Impact of current phishing strategies in machine learning models for phishing detection. In *13th international conference on computational intelligence in security for information systems (CISIS 2020)* (pp. 87–96). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-57805-3_9 doi: 10.1007/978-3-030-57805-3_9

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009, January). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80. Retrieved from https://doi.org/10.1109/tnn.2008.2005605 doi: 10.1109/tnn.2008.2005605

Settles, B. (2009). *Active learning literature survey* (Computer Sciences Technical Report No. 1648). University of Wisconsin–Madison.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, *27*(3), 379–423.

Sheng, S., Magnien, B., Kumaraguru, P., Acquisti, A., Cranor, L. F., Hong, J., & Nunge, E. (2007). Anti-phishing phil. In *Proceedings of the 3rd symposium on usable privacy and security - soups' 07*. ACM Press. Retrieved from https://doi.org/10.1145/1280680.1280692 doi: 10.1145/1280680.1280692

Shirazi, H., Bezawada, B., Ray, I., & Anderson, C. (2019). Adversarial sampling attacks against phishing detection. In *Data and applications security and privacy XXXIII* (pp. 83–101). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-22479-0_5 doi: 10.1007/978-3-030-22479-0_5

SLCERT. (2020). Annual activity report 2020. *Sri Lanka CERT Annual Reports. Retrieved November, 04*, 15.

Smadi, S. (2017). *Detection of online phishing email using dynamic evolving neural network based on reinforcement learning* (Unpublished doctoral dissertation). Northumbria University.

Subasi, A., Molah, E., Almkallawi, F., & Chaudhery, T. J. (2017, November). Intelligent phishing website detection using random forest classifier. In *2017 inter-*

national conference on electrical and computing technologies and applications (ICECTA). IEEE. Retrieved from https://doi.org/10.1109/icecta.2017.8252051 doi: 10.1109/icecta.2017.8252051

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tally, G. (2009, March). Phisherman: A phishing data repository. In *2009 cybersecurity applications & technology conference for homeland security*. IEEE. Retrieved from https://doi.org/10.1109/catch.2009.24 doi: 10.1109/catch.2009.24

Tally, G., Sames, D., Chen, T., Colleran, C., Jevans, D., Omiliak, K., & Rasmussen, R. (2006, July). The phisherman project: Creating a comprehensive data collection to combat phishing attacks. *Journal of Digital Forensic Practice*, *1*(2), 115–129. Retrieved from https://doi.org/10.1080/15567280601015564 doi: 10.1080/15567280601015564

Tchakounté, F., Wabo, L. K., & Atemkeng, M. (2020). A review of gamification applied to phishing.

Teraguchi, N. C. R. L. Y., & Mitchell, J. C. (2004). Client-side defense against web-based identity theft. *Computer Science Department, Stanford University. Available: http://crypto. stanford. edu/SpoofGuard/webspoof. pdf*.

Thakur, T., & Verma, R. (2014). Catching classical and hijack-based phishing attacks. In (pp. 318–337). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-319-13841-1_18 doi: 10.1007/978-3-319-13841-1_18

Tizhoosh, H. R. (2005). Reinforcement learning based on actions and opposite actions. In *International conference on artificial intelligence and machine learning* (Vol. 414).

Verma, R. M., Zeng, V., & Faridi, H. (2019, November). Data quality for security challenges. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. ACM. Retrieved from https://doi.org/10.1145/3319535.3363267 doi: 10.1145/3319535.3363267

Villiers, M. R. R. D. (2012). Models for interpretive information systems research, part 1. In *Research methodologies, innovations and philosophies in software systems engineering and information systems* (pp. 222–237). IGI Global. Retrieved from https://doi.org/10.4018/978-1-4666-0179-6.ch011 doi: 10.4018/978-1-4666-0179-6.ch011

Wang, W., Zhang, F., Luo, X., & Zhang, S. (2019, October). PDRCNN: Precise phishing detection with recurrent convolutional neural networks. *Security and Communication Networks*, *2019*, 1–15. Retrieved from https://doi.org/10.1155/2019/2595794 doi: 10.1155/2019/2595794

Wen, Z. A., Lin, Z., Chen, R., & Andersen, E. (2019, May). What.hack. In *Proceedings of the 2019 CHI conference on human factors in computing systems.* ACM. Retrieved from https://doi.org/10.1145/3290605.3300338 doi: 10.1145/3290605.3300338

Wu, C.-Y., Kuo, C.-C., & Yang, C.-S. (2019, August). A phishing detection system based on machine learning. In *2019 international conference on intelligent computing and its emerging applications (ICEA).* IEEE. Retrieved from https://doi.org/10.1109/icea.2019.8858325 doi: 10.1109/icea.2019.8858325

Yang, P., Zhao, G., & Zeng, P. (2019). Phishing website detection based on multidimensional features driven by deep learning. *IEEE Access*, *7*, 15196–15209. Retrieved from https://doi.org/10.1109/access.2019.2892066 doi: 10.1109/access.2019.2892066

Younis, Y. A., & Musbah, M. (2020, August). A framework to protect against phishing attacks. In *Proceedings of the 6th international conference on engineering & MIS 2020.* ACM. Retrieved from https://doi.org/10.1145/3410352.3410825 doi: 10.1145/3410352.3410825

Yu, W. D., Nargundkar, S., & Tiruthani, N. (2008, July). A phishing vulnerability analysis of web based systems. In *2008 IEEE symposium on computers and communications.* IEEE. Retrieved from https://doi.org/10.1109/iscc.2008.4625681 doi: 10.1109/iscc.2008.4625681

Yue, C., & Wang, H. (2008). Anti-phishing in offense and defense. In *2008 annual*

computer security applications conference (acsac) (pp. 345–354).

Zauner, C. (2010). Implementation and benchmarking of perceptual image hash functions.

Zeng, V., Baki, S., Aassal, A. E., Verma, R., Moraes, L. F. T. D., & Das, A. (2020, March). Diverse datasets and a customizable benchmarking framework for phishing. In *Proceedings of the sixth international workshop on security and privacy analytics.* ACM. Retrieved from https://doi.org/10.1145/3375708.3380313 doi: 10.1145/3375708.3380313

Zhang, Y., Hong, J. I., & Cranor, L. F. (2007). Cantina. In *Proceedings of the 16th international conference on world wide web - www' 07.* ACM Press. Retrieved from https://doi.org/10.1145/1242572.1242659 doi: 10.1145/1242572.1242659

Zhao, L., Sukthankar, G., & Sukthankar, R. (2011, October). Incremental relabeling for active learning with noisy crowdsourced annotations. In *2011 IEEE third int'l conference on privacy, security, risk and trust and 2011 IEEE third int'l conference on social computing.* IEEE. Retrieved from https://doi.org/10.1109/passat/socialcom.2011.193 doi: 10.1109/passat/socialcom.2011.193

APPENDIX A. SAMPLE SOURCE CODES

Code Listing 1: URLDet implementation

```
import re, pickle
import numpy as np
import pandas as pd
from numpy import load
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model,
                                    load_model
from tensorflow.keras import regularizers
from tensorflow.keras import initializers
from tensorflow.keras.layers import *
from tensorflow.keras import optimizers
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import EarlyStopping

# Parameters
epochs = 500              # Number of training epochs
es_patience = 50          # Patience for early stopping
batches = 64              # Batch size
learning_rate = 1e-4      # Learning rate
reg_l2 = 1e-5             # l2 regularisation

# Load data
X_train = load("tr.npy", allow_pickle=True)
X_test = load("te.npy", allow_pickle=True)
X_val = load("val.npy", allow_pickle=True)

y_train = load("tr_class.npy", allow_pickle=True)
y_test = load("te_class.npy", allow_pickle=True)
```

```
y_val = load("val_class.npy", allow_pickle=True)

def lstm_conv(max_len=150, emb_dim=256, max_vocab_len=100,
                                lstm_output_size=32):
# Input
main_input = Input(shape=(max_len,), dtype="int32", name="
                                main_input")
# Embedding layer
emb = Embedding(input_dim=max_vocab_len, output_dim=emb_dim,
                                input_length=max_len,
embeddings_regularizer=regularizers.l2(reg_l2))(main_input)
emb = Dropout(0.8)(emb)

# Conv layer
conv = Conv1D(kernel_size=3, filters=256, padding="same",
                                activation="relu",
kernel_initializer="he_uniform", bias_initializer="zeros",
kernel_regularizer=regularizers.l2(reg_l2))(emb)

conv = MaxPooling1D(pool_size=4)(conv)
conv = Dropout(0.5)(conv)

# LSTM layer 1
lstm1 = LSTM(units=lstm_output_size,kernel_regularizer=
                                regularizers.l2(reg_l2),
                                return_sequences=True)(conv)
lstm1 = Dropout(0.5)(lstm1)

# LSTM layer 2
lstm2 = LSTM(units=lstm_output_size,kernel_regularizer=
                                regularizers.l2(reg_l2))(lstm1)
lstm2 = Dropout(0.5)(lstm2)

# Output layer
output = Dense(1, activation="sigmoid", name="output")(lstm2)
```

```
  # Compile model and define optimizer
  model = Model(inputs=[main_input], outputs=[output])
  adam = optimizers.Adam(lr=learning_rate)
  model.compile(optimizer=adam, loss="binary_crossentropy",
                                  metrics=["accuracy"])
  return model

  model = lstm_conv()

  es = EarlyStopping(patience=es_patience, restore_best_weights=
                                  True, verbose=1)

  history = model.fit(X_train, y_train, validation_data=(X_val,
                                  y_val), epochs=epochs,
  batch_size=batches, callbacks=[es], verbose=1)

  # Evaluate the model
  loss, accuracy = model.evaluate(X_test, y_test, verbose=1)

  # Save model
  model.save("URLDet.h5")
```

Code Listing 2: HTMLDet implementation

```
import pickle, requests
import numpy as np
from numpy import load
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.metrics import categorical_accuracy
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
from tensorflow.keras import initializers
from spektral.layers import GraphConvSkip, GlobalAvgPool
from spektral.layers.ops import sp_matrix_to_sp_tensor
from spektral.layers.pooling import MinCutPool
```

```
from spektral.utils import batch_iterator
from spektral.utils.convolution import normalized_adjacency
from spektral.utils.data import Batch

import warnings
warnings.filterwarnings("ignore")

# Parameters
n_channels = 32             # Channels per layer
activ = "relu"              # Activation in GNN and mincut
GNN_l2 = 1e-3               # l2 regularisation of GNN
pool_l2 = 1e-3              # l2 regularisation for mincut
epochs = 500                # Number of training epochs
es_patience = 100           # Patience for early stopping
learning_rate = 1e-3        # Learning rate
batch_size = 1              # Batch size. NOTE: it MUST be 1 when
                                  using MinCutPool and DiffPool

# Load data
X_train, A_train, y_train = load("tr_features.npy", allow_pickle=
                                 True), list(load("tr_adjacency.
                                 npy", allow_pickle=True)), load(
                                 "tr_class.npy", allow_pickle=
                                 True)
X_test, A_test, y_test = load("te_features.npy", allow_pickle=
                                 True), list(load("te_adjacency.
                                 npy", allow_pickle=True)), load(
                                 "te_class.npy", allow_pickle=
                                 True)
X_val, A_val, y_val = load("val_features.npy", allow_pickle=True)
                                 , list(load("val_adjacency.npy",
                                  allow_pickle=True)), load("
                                 val_class.npy", allow_pickle=
                                 True)

# Preprocessing adjacency matrices for convolution
```

```
A_train = [normalized_adjacency(a) for a in A_train]
A_val = [normalized_adjacency(a) for a in A_val]
A_test = [normalized_adjacency(a) for a in A_test]

# Parameters
F = X_train[0].shape[-1]  # Dimension of node features
n_out = y_train[0].shape[-1]  # Dimension of the target
average_N = np.ceil(np.mean([a.shape[-1] for a in A_train]))  #
                                    Average number of nodes in
                                    dataset

# Build model
X_in = Input(shape=(F, ), name="X_in", dtype=tf.float64)
A_in = Input(shape=(None,), sparse=True, dtype=tf.float64)
I_in = Input(shape=(), name="segment_ids_in", dtype=tf.int32)

X_1 = GraphConvSkip(n_channels,
activation=activ,
kernel_regularizer=l2(GNN_l2),
kernel_initializer="he_uniform",
bias_initializer="zeros")([X_in, A_in])
X_1, A_1, I_1 = MinCutPool(k=int(average_N // 2),
activation=activ,
kernel_regularizer=l2(pool_l2),
kernel_initializer="he_uniform",
bias_initializer="zeros")([X_1, A_in, I_in])
X_2 = GraphConvSkip(n_channels,
activation=activ,
kernel_regularizer=l2(GNN_l2),
kernel_initializer="he_uniform",
bias_initializer="zeros")([X_1, A_1])
X_2, A_2, I_2 = MinCutPool(k=int(average_N // 4),
activation=activ,
kernel_regularizer=l2(pool_l2),
kernel_initializer="he_uniform",
bias_initializer="zeros")([X_2, A_1, I_1])
```

```
X_3 = GraphConvSkip(n_channels,
activation=activ,
kernel_regularizer=l2(GNN_l2),
kernel_initializer="he_uniform",
bias_initializer="zeros")([X_2, A_2])

# Output block
avgpool = GlobalAvgPool()([X_3, I_2])
output = Dense(n_out, activation="softmax")(avgpool)

# Build model
model = Model([X_in, A_in, I_in], output)
model.compile(optimizer="adam",  # Doesn't matter, won't be used
loss="categorical_crossentropy")

# Training setup
opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)
loss_fn = model.loss_functions[0]
acc_fn = lambda x, y: K.mean(categorical_accuracy(x, y))

@tf.function(experimental_relax_shapes=True)
def train_step(inputs, targets):
with tf.GradientTape() as tape:
predictions = model(inputs, training=True)
loss = loss_fn(targets, predictions)
gradients = tape.gradient(loss, model.trainable_variables)
opt.apply_gradients(zip(gradients, model.trainable_variables))
return loss, acc_fn(targets, predictions)

# Fit model
current_batch = 0
model_loss = 0
model_loss_values = []
model_val_loss_values = []
model_acc = 0
model_acc_values = []
```

```
model_val_acc_values = []
best_val_loss = np.inf
best_weights = None
patience = es_patience
batches_in_epoch = np.ceil(y_train.shape[0] / batch_size)

batches = batch_iterator([A_train, X_train, y_train], batch_size=
                                    batch_size, epochs=epochs)
for b in batches:
X_, A_, I_ = Batch(b[0], b[1]).get("XAI")
A_ = sp_matrix_to_sp_tensor(A_)
y_ = b[2]
outs = train_step([X_, A_, I_], y_)

model_loss += outs[0]
model_acc += outs[1]
current_batch += 1
if current_batch % batches_in_epoch == 0:
model_loss /= batches_in_epoch
model_acc /= batches_in_epoch

# Compute validation loss and accuracy
val_loss, val_acc = evaluate(A_val, X_val, y_val, [loss_fn,
                                    acc_fn], batch_size=batch_size)
logging.warning("Ep. {} - Loss: {:.2f} - Acc: {:.2f} - Val loss:
                                    {:.2f} - Val acc: {:.2f}"
.format(current_batch // batches_in_epoch, model_loss, model_acc,
                                     val_loss, val_acc))

# Check if loss improved for early stopping
if val_loss < best_val_loss:
best_val_loss = val_loss
patience = es_patience
print("New best val_loss {:.3f}".format(val_loss))
best_weights = model.get_weights()
else:
```

```
patience -= 1
if patience == 0:
print("Early stopping (best val_loss: {})".format(best_val_loss))
break
model_loss_values.append(model_loss)
model_acc_values.append(model_acc)
model_val_loss_values.append(val_loss)
model_val_acc_values.append(val_acc)
model_loss = 0
model_acc = 0

# Load best model
model.set_weights(best_weights)

# Evaluate model
def evaluate(A_list, X_list, y_list, ops, batch_size):
batches = batch_iterator([A_list, X_list, y_list], batch_size=
                                    batch_size)
output = []
for b in batches:
X, A, I = Batch(b[0], b[1]).get("XAI")
A = sp_matrix_to_sp_tensor(A)
y = b[2]
pred = model([X, A, I], training=False)
outs = [o(y, pred) for o in ops]
output.append(outs)
return np.mean(output, 0)

test_loss, test_acc = evaluate(A_test, X_test, y_test, [loss_fn,
                                    acc_fn], batch_size=batch_size)

# Save model
model.save("HTMLDet.h5")
```

Code Listing 3: DLM implementation

```
import pickle
```

```
import numpy as np
from numpy import load
from sklearn import metrics
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import *
from spektral.layers import GraphConvSkip, GlobalAvgPool
from spektral.layers.ops import sp_matrix_to_sp_tensor
from spektral.layers.pooling import MinCutPool
from spektral.utils.convolution import normalized_adjacency
from spektral.utils import batch_iterator
from spektral.utils.data import Batch

import warnings
warnings.filterwarnings("ignore")

# Load data
X_train_A = load("tr.npy", allow_pickle=True)
X_test_A = load("te.npy", allow_pickle=True)
X_val_A = load("val.npy", allow_pickle=True)

X_train, A_train, y_train = load("tr_features.npy", allow_pickle=
                                 True), list(load("tr_adjacency.
                                 npy", allow_pickle=True)), load(
                                 "tr_class.npy", allow_pickle=
                                 True)
X_test, A_test, y_test = load("te_features.npy", allow_pickle=
                                 True), list(load("te_adjacency.
                                 npy", allow_pickle=True)), load(
                                 "te_class.npy", allow_pickle=
                                 True)
X_val, A_val, y_val = load("val_features.npy", allow_pickle=True)
                                 , list(load("val_adjacency.npy",
                                  allow_pickle=True)), load("
```

```
                                    val_class.npy", allow_pickle=
                                    True)

# Preprocessing adjacency matrices for convolution
A_train_B = [normalized_adjacency(a) for a in A_train_B]
A_val_B = [normalized_adjacency(a) for a in A_val_B]
A_test_B = [normalized_adjacency(a) for a in A_test_B]

# Load pre-trained models
URLDet = load_model("URLDet.h5")
HTMLDet = load_model("HTMLDet.h5", custom_objects={"GraphConvSkip
                                    ": GraphConvSkip, "MinCutPool":
                                    MinCutPool, "GlobalAvgPool":
                                    GlobalAvgPool})

# Get the output before last layer of both models
URLDet = Model(inputs=URLDet.inputs, outputs=URLDet.layers[-2].
                                    output)
HTMLDet = Model(inputs=HTMLDet.inputs, outputs=HTMLDet.layers[-2]
                                    .output)

# Parameters
epochs = 500                # Number of training epochs
es_patience = 20            # Patience for early stopping
learning_rate = 1e-5        # Learning rate
batch_size = 1              # Batch size. NOTE: it MUST be 1 when
                                    using MinCutPool and DiffPool

# Constructing the model
concat = Concatenate(-1)([model_A.output, model_B.output]) #
                                    merge outputs
concat = Dense(2, activation="softmax", name="output")(concat)

model = Model(inputs=[URLDet.inputs, HTMLDet.inputs], outputs=
                                    concat)
model.compile(optimizer="adam",
```

```
loss="categorical_crossentropy")

# Training setup
opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)
loss_fn = model.loss_functions[0]
acc_fn = lambda x, y: K.mean(tf.keras.metrics.
                                    categorical_accuracy(x, y))

# Training function
@tf.function(experimental_relax_shapes=True)
def train_step(inputs, targets):
with tf.GradientTape() as tape:
predictions = model(inputs, training=True)
loss = loss_fn(targets, predictions)
gradients = tape.gradient(loss, model.trainable_variables)
opt.apply_gradients(zip(gradients, model.trainable_variables))
return loss, acc_fn(targets, predictions)

current_batch = 0
model_loss = 0
model_loss_values = []
model_val_loss_values = []
model_acc = 0
model_acc_values = []
model_val_acc_values = []
best_val_loss = np.inf
best_weights = None
patience = es_patience
batches_in_epoch = np.ceil(y_train_B.shape[0] / batch_size)

# Fitting model
batches = batch_iterator([A_train_B, X_train_B, y_train_B,
                                    X_train_A], batch_size=
                                    batch_size, epochs=epochs)
for b in batches:
X_, A_, I_ = Batch(b[0], b[1]).get("XAI")
```

```
A_ = sp_matrix_to_sp_tensor(A_)
y_ = b[2]
X_A_ = b[3]
outs = train_step([X_A_, X_, A_, I_], y_)

model_loss += outs[0]
model_acc += outs[1]
current_batch += 1
if current_batch % batches_in_epoch == 0:
model_loss /= batches_in_epoch
model_acc /= batches_in_epoch

# Compute validation loss and accuracy
val_loss, val_acc = evaluate(A_val_B, X_val_B, y_val_B, X_val_A,
                             [loss_fn, acc_fn], batch_size=
                             batch_size)
logging.warning("Ep. {} - Loss: {:.2f} - Acc: {:.2f} - Val loss:
                             {:.2f} - Val acc: {:.2f}"
.format(current_batch // batches_in_epoch, model_loss, model_acc,
                              val_loss, val_acc))

# Check if loss improved for early stopping
if val_loss < best_val_loss:
best_val_loss = val_loss
patience = es_patience
logging.warning("New best val_loss {:.3f}".format(val_loss))
best_weights = model.get_weights()
else:
patience -= 1
if patience == 0:
logging.warning("Early stopping (best val_loss: {})".format(
                             best_val_loss))
break
model_loss_values.append(model_loss)
model_acc_values.append(model_acc)
model_val_loss_values.append(val_loss)
```

```
model_val_acc_values.append(val_acc)
model_loss = 0
model_acc = 0

# Model evaluation function
def evaluate(A_list, X_list, y_list, X_A_list, ops, batch_size):
batches = batch_iterator([A_list, X_list, y_list, X_A_list],
                                    batch_size=batch_size)
output = []
for b in batches:
X, A, I = Batch(b[0], b[1]).get("XAI")
A = sp_matrix_to_sp_tensor(A)
y = b[2]
X_A = b[3]
pred = model([X_A, X, A, I], training=False)
outs = [o(y, pred) for o in ops]
output.append(outs)
return np.mean(output, 0)

# Load best model
model.set_weights(best_weights)

# Evaluate model
test_loss, test_acc = evaluate(A_test_B, X_test_B, y_test_B,
                                    X_test_A, [loss_fn, acc_fn],
                                    batch_size=batch_size)

# Save model
model.save("DLM.h5")
```

Code Listing 4: Hybrid DLM's URLDet implementation

```
import re, os
from string import printable
from sklearn import model_selection
import tensorflow as tf
from keras.models import Sequential, Model, load_model
```

```
from keras import regularizers
from keras.layers import *
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers.core import Dense, Dropout, Activation, Lambda,
                                Flatten
from keras.preprocessing import sequence
from keras import optimizers
from keras.utils import np_utils
from keras import backend as K
from tensorflow.contrib import rnn

#Parameters
epochs = 100
batches = 64
learning_rate = 0.001

#Fix random seed for reproducibility
seed = 7

#Load data
df = pd.read_csv("data.csv")
X_train, X_test, y_train, y_test = model_selection.
                                train_test_split(df.drop(columns
                                =["Result"]), df.Result,
                                test_size=0.1, random_state=seed
                                )

#Convert to numeric format
url_int_tokens_train = [[printable.index(x) + 1 for x in url if x
                                 in printable] for url in
                                X_train.url]
url_int_tokens_test = [[printable.index(x) + 1 for x in url if x
                                in printable] for url in X_test.
                                url]

#Step 2: Cut URL string at max_len or pad with zeros if shorter
```

```
max_len=150
X = sequence.pad_sequences(url_int_tokens_train, maxlen=max_len)
X_t = sequence.pad_sequences(url_int_tokens_test, maxlen=max_len)

def lstm_conv(max_len=max_len, emb_dim=256, max_vocab_len=100,
                                    lstm_output_size=32, W_reg=
                                    regularizers.l2(1e-4)):
#Input
main_input = Input(shape=(max_len,), dtype="int32", name="
                                    main_input")
#Embedding layer
emb = Embedding(input_dim=max_vocab_len, output_dim=emb_dim,
                                    input_length=max_len,
W_regularizer=W_reg)(main_input)
emb = Dropout(0.25)(emb)
#Conv layer
conv = Conv1D(kernel_size=5, filters=256, border_mode="same",
                                    activation="relu")(emb)
conv = MaxPooling1D(pool_size=4)(conv)
conv = Dropout(0.5)(conv)
#LSTM layer
lstm = LSTM(lstm_output_size)(conv)
lstm = Dropout(0.5)(lstm)
#Output layer
output = Dense(1, activation="sigmoid", name="output")(lstm)
# Compile model and define optimizer
model = Model(input=[main_input], output=[output])
adam = optimizers.Adam(lr=learning_rate, beta_1=0.9, beta_2=0.999
                                    , epsilon=1e-08, decay=0.0)
model.compile(optimizer=adam, loss="binary_crossentropy", metrics
                                    =["accuracy"])
return model

#Model fit
model = lstm_conv()
history = model.fit(X, y_train, validation_split=0.2, epochs=
```

```
                                    epochs, batch_size=batches,
                                    verbose=0)

#Evaluate the model
loss, accuracy = model.evaluate(X_t, y_test, verbose=0)

#Save model
model.save("Hybrid_URLDet.h5")
```

Code Listing 5: Hybrid DLM's HTMLDet implementation

```
import numpy as np
import re, os
import time
from string import printable
from sklearn import model_selection
import tensorflow as tf
from keras.models import Sequential, Model, load_model
from keras import regularizers
from keras.layers import *
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers.core import Dense, Dropout, Activation, Lambda,
                                Flatten
from keras.preprocessing import sequence
from sklearn.preprocessing import StandardScaler
from keras.optimizers import Adam
from keras.utils import np_utils
from keras import backend as K
from tensorflow.contrib import rnn
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline

#Parameters
epochs = 100
batches = 64
```

```
learning_rate = 0.001

#Fix random seed for reproducibility
seed = 7
df = pd.read_csv("data.csv")

#Apply standard scaler
html_len = df[["html_length"]].values.astype(float)
n_hyperlinks = df[["n_hyperlinks"]].values.astype(float)
n_script_tag = df[["n_script_tag"]].values.astype(float)
n_link_tag = df[["n_link_tag"]].values.astype(float)
n_comment_tag = df[["n_comment_tag"]].values.astype(float)

scaler = StandardScaler()
html_len_scaled = scaler.fit_transform(html_len)
n_hyperlinks_scaled = scaler.fit_transform(n_hyperlinks)
n_script_tag_scaled = scaler.fit_transform(n_script_tag)
n_link_tag_scaled = scaler.fit_transform(n_link_tag)
n_comment_tag_scaled = scaler.fit_transform(n_comment_tag)

#Remove column and add to data frame
df = pd.concat([df.drop(columns=["html_length","n_hyperlinks","
                                 n_script_tag","n_link_tag","
                                 n_comment_tag"]),
pd.DataFrame(html_len_scaled, columns=["html_length_std"]),
pd.DataFrame(n_hyperlinks_scaled, columns=["n_hyperlinks_std"]),
pd.DataFrame(n_script_tag_scaled, columns=["n_script_tag_std"]),
pd.DataFrame(n_link_tag_scaled, columns=["n_link_tag_std"]),
pd.DataFrame(n_comment_tag_scaled, columns=["n_comment_tag_std"])
                                 ], axis=1, join="inner")

X_train, X_test, y_train, y_test = model_selection.
                                 train_test_split(df.drop(columns
                                 =["Result"]), df.Result,
                                 test_size=0.1, random_state=seed
                                 )
```

```
#Input (x) variables
x = X_train.drop(columns=["url"]).values.astype(float)
x_test = X_test.drop(columns=["url"]).values.astype(float)

#Reshape input (x)
X_2 = x.reshape(x.shape[0], x.shape[1], 1)
X_2t = x_test.reshape(x_test.shape[0], x_test.shape[1], 1)

def conv_1d():
#Input
main_input_2 = Input(shape=X_2.shape[1:3], name="main_input_2")

conv1D = Conv1D(filters=256, kernel_size=5, activation="relu",
                                    name="conv1D_1_layer")(
                                    main_input_2)
conv1D = Dropout(0.5, name="conv1D_dropout_layer_1")(conv1D)
conv1D = Conv1D(filters=128, kernel_size=5, activation="relu",
                                    name="conv1D_2_layer")(conv1D)
conv1D = MaxPooling1D(pool_size=2, name="conv1D_maxpooling_layer"
                                    )(conv1D)
conv1D = Dropout(0.5, name="conv1D_dropout_layer_2")(conv1D)
conv1D = Flatten()(conv1D)
conv1D = Dense(32, activation="relu", name="conv1D_dense_layer")(
                                    conv1D)

#Output
output = Dense(1, activation="sigmoid", name="
                                    conv1D_sigmoid_layer")(conv1D)

#Model compile
model = Model(input=[main_input_2], output=[output])
adam = Adam(lr=learning_rate , beta_1=0.9, beta_2=0.999, epsilon=
                                    1e-08, decay=0.0, amsgrad=False)
model.compile(loss="binary_crossentropy", optimizer=adam, metrics
                                    =["accuracy"])
```

```
return model

#Model fit
model = conv_1d()
history = model.fit(X_2, y_train, validation_split=0.2, epochs=
                                    epochs, batch_size=batches,
                                    verbose=0)

#Evaluate the model
loss, accuracy = model.evaluate(X_2t, y_test, verbose=0)

#Save model
model.save("Hybrid_HTMLDet.h5")
```

Code Listing 6: Hybrid DLM implementation

```
import numpy as np
import re, os
import time
from string import printable
from sklearn import model_selection
import tensorflow as tf
from keras.models import Sequential, Model, load_model
from keras import regularizers
from keras.layers import *
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers.core import Dense, Dropout, Activation, Lambda,
                                    Flatten
from keras.preprocessing import sequence
from sklearn.preprocessing import StandardScaler
from keras.optimizers import Adam
from keras.utils import np_utils
from keras import backend as K
from tensorflow.contrib import rnn
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.pipeline import Pipeline

#Parameters
epochs = 50
batches = 64
learning_rate = 0.001

#Fix random seed for reproducibility
seed = 7
df = pd.read_csv("data.csv")

#Apply standard scaler
html_len = df[["html_length"]].values.astype(float)
n_hyperlinks = df[["n_hyperlinks"]].values.astype(float)
n_script_tag = df[["n_script_tag"]].values.astype(float)
n_link_tag = df[["n_link_tag"]].values.astype(float)
n_comment_tag = df[["n_comment_tag"]].values.astype(float)

scaler = StandardScaler()
html_len_scaled = scaler.fit_transform(html_len)
n_hyperlinks_scaled = scaler.fit_transform(n_hyperlinks)
n_script_tag_scaled = scaler.fit_transform(n_script_tag)
n_link_tag_scaled = scaler.fit_transform(n_link_tag)
n_comment_tag_scaled = scaler.fit_transform(n_comment_tag)

#Remove column and add to data frame
df = pd.concat([df.drop(columns=["html_length","n_hyperlinks","
                                 n_script_tag","n_link_tag","
                                 n_comment_tag"]),
pd.DataFrame(html_len_scaled, columns=["html_length_std"]),
pd.DataFrame(n_hyperlinks_scaled, columns=["n_hyperlinks_std"]),
pd.DataFrame(n_script_tag_scaled, columns=["n_script_tag_std"]),
pd.DataFrame(n_link_tag_scaled, columns=["n_link_tag_std"]),
pd.DataFrame(n_comment_tag_scaled, columns=["n_comment_tag_std"])
                                 ], axis=1, join="inner")
```

```
X_train, X_test, y_train, y_test = model_selection.
                                   train_test_split(df.drop(columns
                                   =["Result"]), df.Result,
                                   test_size=0.25, random_state=
                                   seed)

def create_X_1(temp_X_1):
url_int_tokens = [[printable.index(x) + 1 for x in url if x in
                                   printable] for url in temp_X_1.
                                   url]
max_len=150
X_new_1 = sequence.pad_sequences(url_int_tokens, maxlen=max_len)
return X_new_1

def create_X_2(temp_X_2):
#Input (x) variables
x = temp_X_2.drop(columns=["url"]).values.astype(float)

#Reshape input (x)
X_new_2 = x.reshape(x.shape[0], x.shape[1], 1)
return X_new_2

model_A = load_model("model_A.h5")
model_A.layers.pop()
model_A = Model(inputs=model_A.inputs, outputs=model_A.layers[-1]
                                   .output)

model_B = load_model("model_B.h5")
model_B.layers.pop()
model_B = Model(inputs=model_B.inputs, outputs=model_B.layers[-1]
                                   .output)

def final_model():
mergedOut = Add()([model_A.output,model_B.output])

#Output layer
```

```
mergedOut = Dense(1, activation="sigmoid")(mergedOut)

model = Model([model_A.input,model_B.input], mergedOut)
adam = Adam(lr=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=
                                    1e-08, decay=0.0)
model.compile(optimizer=adam, loss="binary_crossentropy", metrics
                                    =["acc"])
return model

model = final_model()
history = model.fit([create_X_1(X_train),create_X_2(X_train)],
                                    y_train, validation_split=0.2,
                                    epochs=epochs, batch_size=
                                    batches, verbose=0)

#Evaluate the model
loss, accuracy = model.evaluate([create_X_1(X_test),create_X_2(
                                    X_test)], y_test, verbose=0)

#Save model
model.save("Hybrid_DLM.h5")
```

Code Listing 7: RL environment

```
import gym
import numpy as np
from gym import spaces

class Cyberspace(gym.Env):
    metadata = {"render.modes": ["human"]}

    def __init__(self):
        self.index = 0
        self.done = False
        self.actions = ["ALLOW ACCESS","STOP ACCESS","ASK USER",]
        # Action space
        self.action_space = spaces.Discrete(len(self.actions))
```

```
        # Observation space
        self.observation_space = spaces.Box(low=np.array([0.0, 0.
                                            0, 0.0]), high=np.array(
                                            [1.0, 1.0, 1.0]), dtype=
                                            np.float32)

    def init_dataset(self, data=None, url=None, external_factors=
                                        None):
        self.data = data
        self.url = url
        self.external_factors = external_factors
        self.states, self.community_feedback = self.get_states_cf
                                            (self.data, self.url)

    def get_states_cf(self, data, url):
        l = len(data)
        processed_data = []
        cf_data = []
        res = []
        res.append(data[0][1]) #add phishing probability
        res.append(self.external_factors[0]) #add community
                                            feedback
        res.append(float(self.external_factors[1])) #add global
                                            alexa rank
        state, cf = np.array([res]), self.external_factors[0]
        for t in range(2):
            processed_data.append(state)
            cf_data.append(cf)
        return processed_data, cf_data

    def _next_observation(self):
        obs = self.states[self.index]
        self.index = self.index + 1
        return obs, self.index

    def get_rewards(self, action, state):
```

```
        reward = None
        url = self.url[self.index - 1]
        DL_output = [(1 - state[0][0]), state[0][0]]
        community_decision = self.community_feedback[self.index -
                                                     1]
        record_id, user_feedback = self.get_user_feedback(url,
                                                           community_decision)
        entropy = 0 if (DL_output[0] == 0 or DL_output[0] == 1)
                                                           else -np.sum(DL_output *
                                                            np.log2(DL_output))
        true_reward = int(entropy * 100)
        if user_feedback == 2:
            reward = int((entropy - (1 - DL_output[action]))  *
                                                               100) if (action == 0
                                                                or action == 1)
                                                               else 0
        else:
            if user_feedback == action:
                reward = int(entropy * 100)
            elif action == 2:
                reward = int((entropy - DL_output[
                                                                    community_decision
                                                                    ])  * 100)
            else:
                reward = int((entropy - 1) * 100)
        return reward, record_id, true_reward

    def step(self, action, state):
        # Execute one time step within the environment
        reward, record_id, true_reward = self.get_rewards(action,
                                                           state)
        if self.index >= (len(self.states) - 1):
            self.done = True
        obs, _ = self._next_observation()
        return obs, reward, self.done, record_id, true_reward, {}
```

```
    def reset(self):
        self.index = 0
        self.done = False
        return self._next_observation()

    def render(self, mode="human", close=False):
        print("render")

    def get_user_feedback(self, url, community_decision):
        feedback = community_decision
        record_id = 0
        ## If the human feedback is available for the passed URL,
                                        it should be fetched
                                       here.
        ## Otherwise, this function will pass the community
                                       decision as the feedback
                                       .
        return record_id, feedback
```

Code Listing 8: RL Agent

```
import random
import numpy as np
from collections import deque
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

class Agent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.batch_size = 128
        self.memory = deque(maxlen=10000)
        self.gamma = 0.8
        self.epsilon = 1.0
        self.epsilon_min = 0.001
```

```
        self.epsilon_decay = 0.99975
        self.learning_rate = 0.001

        # Main model
        self.model = self._build_model()
        # Target network
        self.target_model = self._build_model()
        self.load("phishing-dqn.h5")

        # Used to count when to update target network with main
                                            network's weights
        self.target_update_counter = 0
        # Used to update target network after this no of episodes
                                             ends
        self.target_update_frequency = 1

    def _build_model(self):
        # Neural Net for Deep-Q learning Model
        model = Sequential()
        model.add(Dense(units=128, input_dim=self.state_size,
                                            activation="relu"))
        model.add(Dense(units=128, activation="relu"))
        model.add(Dense(self.action_size, activation="linear"))
        model.compile(loss="mse", optimizer=Adam(lr=self.
                                            learning_rate), metrics=
                                            ["accuracy"])
        return model

    def memorize(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state,
                                            done))

    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return int(random.randrange(self.action_size))
        act_values = self.model.predict(state)
```

```
        return int(np.argmax(act_values[0]))  # returns action

    def replay(self, batch_size=64):
        minibatch = random.sample(self.memory, batch_size)
        for state, action, reward, next_state, done in minibatch:
            target = reward
            if not done:
                target = (reward + self.gamma *
                    np.amax(self.target_model.predict(next_state)
                                                        [0]))
            target_f = self.model.predict(state)
            target_f[0][action] = target
            self.model.fit(state, target_f, epochs=1, verbose=0)

        if self.epsilon > self.epsilon_min:
            self.epsilon -= self.epsilon_decay

    def load(self, name):
        self.model.load_weights(name)
        self.target_model.load_weights(name)

    def save(self, name):
        self.model.save_weights(name)
```

Code Listing 9: GAN implementation

```
import numpy as np
import pandas as pd
import seaborn as sns
from pathlib import Path

from numpy.random import randn
from tensorflow.keras.layers import Dense
from sklearn.manifold import TSNE
from tensorflow.keras.models import Sequential

from gan_inputs import GANInputGenerator
```

```
class GAN:
    def __init__(self):
        # initializing GANInputGenerator
        n_training_example, n_eval_example, n_pred_example = 3000
                                            , 2000, 1000
        gan_inputs = GANInputGenerator(ex=n_training_example,
                                            eval_ex=n_eval_example,
                                            pred_ex=n_pred_example)

        # generate relevent data
        self.train_data = np.array(gan_inputs.get_train_data())
        self.eval_data = np.array(gan_inputs.get_eval_data())
        pred_l_data, pred_p_data = gan_inputs.get_pred_data()
        self.pred_l_data = np.array(pred_l_data)
        self.pred_p_data = np.array(pred_p_data)

        # network parameters
        self.n_inputs = 64
        self.latent_dim = 128
        self.batch_size = self.train_data.shape[0]
        self.epochs = 10000
        self.eval_frequency = 50

        # save accuracy and loss
        self.acc_list = []
        self.loss_list = []

        # create the discriminator
        self.discriminator = self.define_discriminator()
        # create the generator
        self.generator = self.define_generator()
        # create the gan
        self.gan = self.define_gan()

    # define the standalone discriminator model
```

```
def define_discriminator(self):
    model = Sequential()
    model.add(Dense(64, activation="relu", kernel_initializer
                                    ="he_uniform", input_dim
                                    =self.n_inputs))
    model.add(Dense(64, activation="relu", kernel_initializer
                                    ="he_uniform"))
    model.add(Dense(64, activation="relu", kernel_initializer
                                    ="he_uniform"))
    model.add(Dense(1, activation="sigmoid"))
    # compile model
    model.compile(loss="binary_crossentropy", optimizer="adam
                                    ", metrics=["accuracy"])
    return model

# define the standalone generator model
def define_generator(self):
    model = Sequential()
    model.add(Dense(512, activation="relu",
                                    kernel_initializer="
                                    he_uniform", input_dim=
                                    self.latent_dim))
    model.add(Dense(256, activation="relu",
                                    kernel_initializer="
                                    he_uniform"))
    model.add(Dense(256, activation="relu",
                                    kernel_initializer="
                                    he_uniform"))
    model.add(Dense(128, activation="relu",
                                    kernel_initializer="
                                    he_uniform"))
    model.add(Dense(self.n_inputs, activation="relu"))
    return model

# define the combined generator and discriminator model, for
                                updating the generator
```

```
def define_gan(self):
  # connect them
  model = Sequential()
  # add generator
  model.add(self.generator)
  # add the discriminator
  model.add(self.discriminator)
  # compile model
  model.compile(loss="binary_crossentropy", optimizer="adam")
  return model

# generate n real samples with class labels
def generate_data(self, task="train"):
    x_real = self.train_data if (task == "train") else self.
                                        eval_data
    # generate class labels
    y_real = np.ones((x_real.shape[0], 1))
    # generate points in latent space
    x_input = self.generate_latent_points(x_real.shape[0])
    # predict outputs
    x_fake = self.generator.predict(x_input)
    # create class labels
    y_fake = np.zeros((x_real.shape[0], 1))
    return x_real, y_real, x_fake, y_fake

# generate points in latent space as input for the generator
def generate_latent_points(self, n):
  # generate points in the latent space
  x_input = randn(self.latent_dim * n)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n, self.latent_dim)
  return x_input

# evaluate the discriminator
def summarize_performance(self, epoch, folder):
    # prepare real and fake samples
```

```
        x_real, y_real, x_fake, y_fake = self.generate_data(task=
                                            "eval")
        # merge real and fake sample
        data = np.concatenate((x_real, x_fake), axis=0)
        labels = np.concatenate((y_real, y_fake), axis=0)
        # evaluate discriminator
        loss, acc = self.discriminator.evaluate(data, labels,
                                            verbose=0)
        # summarize discriminator performance
        self.acc_list.append(acc)
        self.loss_list.append(loss)
        print("-----")
        print("Epoch: " + str(epoch))
        print("   Dis. acc.: " + str(acc))

    # train the generator and discriminator
    def train(self):
        for i in range(self.epochs):
            # prepare real and fake samples
            x_real, y_real, x_fake, y_fake = self.generate_data()
            # update discriminator
            self.discriminator.trainable = True # make weights in
                                                the discriminator
                                               not trainable
            self.discriminator.train_on_batch(x_real, y_real)
            self.discriminator.train_on_batch(x_fake, y_fake)
            # prepare points in latent space as input for the
                                               generator
            x_gan = self.generate_latent_points(self.batch_size)
            # create inverted labels for the fake samples
            y_gan = np.ones((self.batch_size, 1))
            # update the generator via the discriminator's error
            self.discriminator.trainable = False # make weights
                                               in the discriminator
                                                not trainable
            self.gan.train_on_batch(x_gan, y_gan)
```

```
            # evaluate the model every n_eval epochs
            if i == 0 or (i+1) % self.eval_frequency == 0:
                r_folder = "resources/gan/epoch_" + str(i+1)
                Path(r_folder + "/models").mkdir(parents=True,
                                                 exist_ok=True)
                self.summarize_performance((i+1), r_folder)
                self.discriminator.save(r_folder + "/models/
                                                 discriminator-
                                                 gan.h5")
                self.generator.save(r_folder + "/models/generator
                                                 -gan.h5")
                self.gan.save(r_folder + "/models/gan.h5")
```

Code Listing 10: Web page downloading function

```
from selenium import webdriver
from selenium.webdriver.firefox.options import Options

def get_webpage(url):
    opts = Options()
    opts.headless = True
    brower = webdriver.Firefox(options=opts)
    brower.set_page_load_timeout(300)
    brower.get(url)
    webContent = brower.page_source
    p = open("webpage.html", "w")
    p.write(webContent)
    p.close
    brower.close()
```

Code Listing 11: SmartiPhish service - part I

```
import gym, csv, json, pickle, asyncio, requests
import numpy as np
from flask import Flask, request, jsonify
from flask_cors import CORS

import env, gym_register
```

```
from agent import Agent
from connection import DBConnection

app = Flask(__name__)
cors = CORS(app, resources={r"/moraphishdet": {"origins": "*"}, r
                                    "/moraphishup": {"origins": "*"}
                                    })
env = gym.make("gym_register:MORAPhishDet-v0") # Make Cyberspace
                                    gym environment

def moraphishdet(data, url, cf, global_rank):
    env.init_dataset(data, url, external_factors=[cf, global_rank
                                        ]) # input data to the
                                        environment
    state_size = env.observation_space.shape[0]
    action_size = env.action_space.n
    agent = Agent(state_size, action_size, is_eval=True)
    done = False

    state, index = env.reset()
    state = np.reshape(state, [1, state_size])

    for time in range(len(data)):
        action = agent.act(state)
        next_state, reward, done, record_id, true_reward, _ = env
                                            .step(action, state)
        next_state = np.reshape(next_state, [1, state_size])
        agent.memorize(state, action, reward, next_state, done)
        state = next_state
        if done:
            logging.warning("Rewards: {0}".format(reward))
            break
    return action, record_id

@app.route("/moraphishdet", methods=["GET", "POST"])
def get_action():
```

```
    if not request.json or "url" not in request.json:
        abort(400)

    content = request.json
    requested_url = content["url"]
    res = requests.post("http://127.0.0.1:5001/moraphishinputs",
                                    json={"url":requested_url})
    if res.ok:
        if res.json()["status"] == 1:
            dict = json.loads(res.json()["data"])
            data = [np.array(x) for x in dict]
            url = res.json()["url"]
            cf = json.loads(res.json()["cf"])
            global_rank = json.loads(res.json()["global_rank"])
            action, record_id = moraphishdet(data, url, cf,
                                            global_rank)
        else:
            action = res.json()["status"]
            url = [requested_url]
            record_id = 0
    else:
        action = 5
        url = [requested_url]
        record_id = 0
    return jsonify({"action":int(action), "id":int(record_id), "
                                    url":str(url[0])})

@app.route("/moraphishup", methods=["GET", "POST"])
def update():
    if not request.json or "id" not in request.json:
        abort(400)
    content = request.json
    record_2_update = content["id"]
    connection = DBConnection("smartiphish").get_connection()
    cursor = connection.cursor(buffered=True)
    sql = "SELECT * FROM reviews WHERE rec_id = (SELECT
```

```
                                    review_table_id FROM data
                                    WHERE rec_id = %s LIMIT 1)
                                    AND result = %s AND updated
                                    = %s"
    val = (record_2_update, 2, 0)
    cursor.execute(sql, val)
    if (cursor.rowcount > 0):
        sql = "UPDATE reviews SET status = %s WHERE rec_id = (
                                        SELECT review_table_id
                                        FROM data WHERE rec_id =
                                         %s LIMIT 1)"
        val = (0, record_2_update)
        cursor.execute(sql, val)
        connection.commit()
    cursor.close()
    return jsonify({"status":"success"})
app.run()
```

Code Listing 12: SmartiPhish service - part II

```
import gym, json, pickle, asyncio
import numpy as np
from flask import Flask, request, jsonify
from flask_cors import CORS

import env, gym_register
from connection import DBConnection
from generate_data import GenerateData
from knowledge_model import KnowledgeModel

app = Flask(__name__)
cors = CORS(app, resources={r"/moraphishinputs": {"origins": "*"}
                                    })
env = gym.make("gym_register:MORAPhishDet-v0") # Make Cyberspace
                                    gym environment
km = KnowledgeModel() # Initialize the Knowledge Model
```

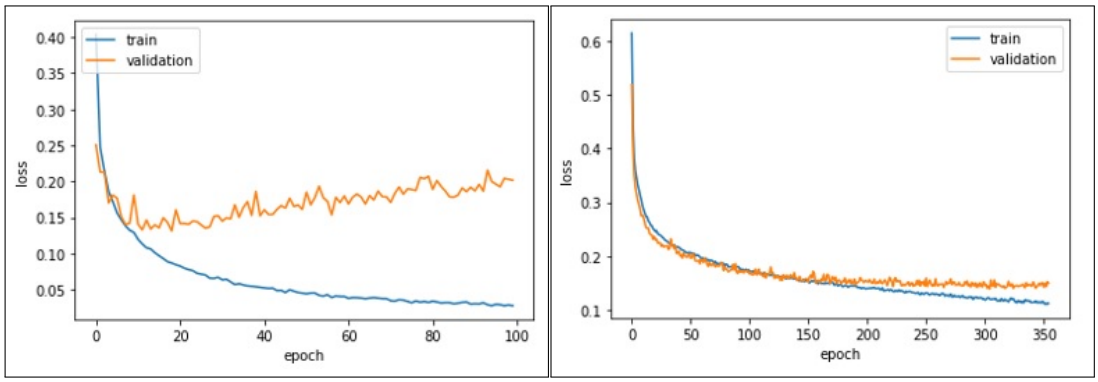
```
class NumpyEncoder(json.JSONEncoder):
    """ Special json encoder for numpy types """
    def default(self, obj):
        if isinstance(obj, np.integer):
            return int(obj)
        elif isinstance(obj, np.floating):
            return float(obj)
        elif isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# loading tokenizer
with open("tokenizer.pkl", "rb") as handle:
    tokenizer = pickle.load(handle)

async def get_inputs(A, X, X_A, url, env, km):
    # Create task to do so:
    task1 = asyncio.ensure_future(km.get_km_prediction(A, X, X_A)
                                  ) #get DL decision
    task2 = asyncio.ensure_future(env.get_google_decision(url)) #
                                  get google feedback
    task3 = asyncio.ensure_future(env.get_phishtank_decision(url)
                                  ) #get phishtank feedback
    task4 = asyncio.ensure_future(env.AlexaRank(url)) #get Alexa
                                  Ranking
    await task1, task2, task3, task4 # wait until all task
                                  finished
    google_decision = task2.result()
    phishtank_decision = task3.result()
    final_decision = 1 if (google_decision or phishtank_decision)
                                   else 0
    return(task1.result(), final_decision, task4.result())

@app.route("/moraphishinputs", methods=["GET", "POST"])
def index():
    content = request.json
```

```
    requested_url = content["url"]
    gen_data = GenerateData(tokenizer)
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    X, A, X_A, url = loop.run_until_complete(gen_data.
                                    generate_requested_data(
                                    requested_url))
    loop.close()
    if X == "URL GET ERROR":
        action = 3
        return jsonify({"status":int(action)})
    elif X == "DIFF URL ERROR":
        action = 4
        return jsonify({"status":int(action)})
    elif X == "NOT AN HTML ERROR":
        action = -1
        return jsonify({"status":int(action)})
    else:
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)
        data, cf, global_rank = loop.run_until_complete(
                                        get_inputs(A, X, X_A,
                                        url[0], env, km))
        loop.close()
        logging.warning(str(url[0]) + " [successfully generated]"
                                        )
        dumped_data = json.dumps(data, cls=NumpyEncoder)
        dumped_cf = json.dumps(cf, cls=NumpyEncoder)
        dumped_global_rank = json.dumps(global_rank, cls=
                                        NumpyEncoder)
        return jsonify({"status":int(1), "data":dumped_data, "url
                                        ": url, "cf":dumped_cf,
                                        "global_rank":
                                        dumped_global_rank})
app.run(port="5001")
```

APPENDIX B. ADDITIONAL EXPERIMENTS

Hyperparameter Optimisation in DLM

When optimizing the DLM architecture, the study conducted several experiments by altering specific hyperparameters. The following experiments illustrate how these modifications impacted the original model once implemented. These hyperparameter optimizations were performed individually for the URLDet, the HTMLDet, and the DLM.

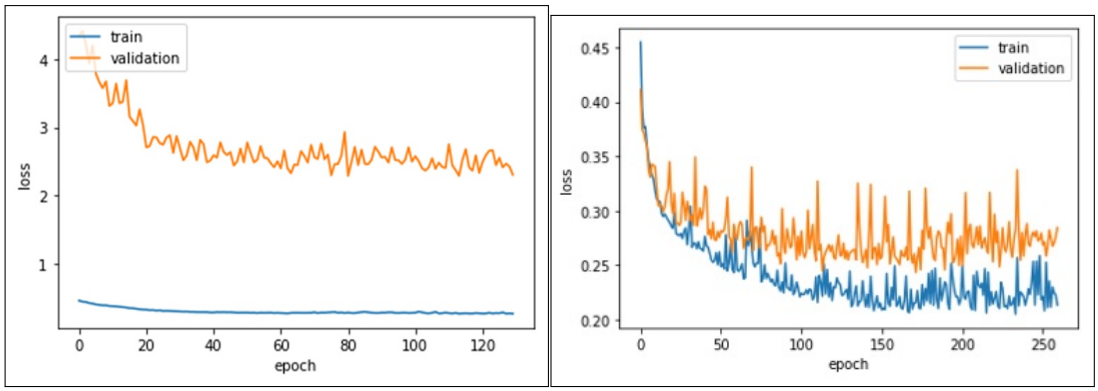


(a) Old Model

(b) New Model

Hyperparameter tuning for URLDet

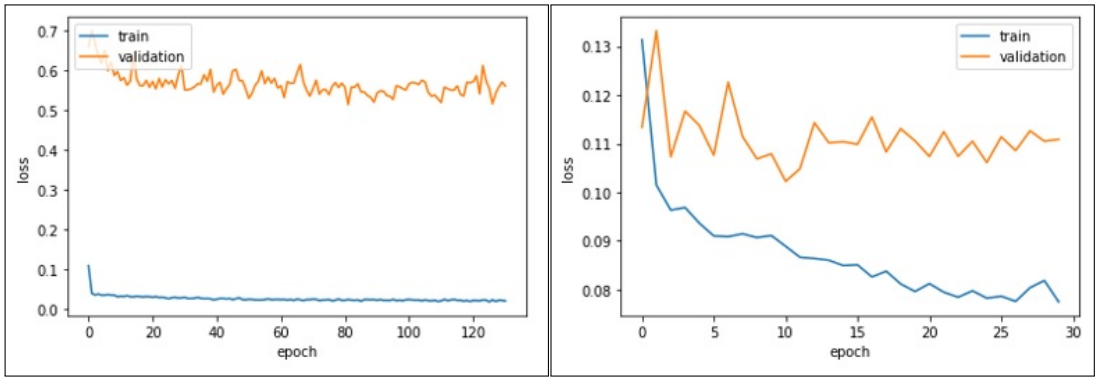
The old model was set up with general configurations that included only small dropout and a single LSTM layer. These configurations might have been simple and less complex, potentially limiting the model's ability to capture intricate patterns and relationships in the data. In contrast, the new model, several changes were made to improve its performance. Dropout was increased, which helps prevent overfitting by randomly deactivating certain neurons during training. The kernel_initializer was set to 'he_uniform', and the bias_initializer to 'zeros', which determine how the weights and biases in the model are initialized. These initializations can impact the model's ability to learn effectively from the data. Furthermore, a kernel_regularizer was applied with a strength of 1e-4, which adds a penalty to the model's loss function to encourage simpler weight values, thus preventing overfitting. Additionally, the model was augmented with a Stack LSTM, which means two LSTM layers were stacked on top of each other. This stacking enables the model to capture more complex patterns and dependencies in the data, potentially leading to improved performance. Overall, these changes were implemented in the new model to enhance its learning capacity, improve generalization, and achieve better results compared to the previous version.



(a) Old Model (b) New Model

Hyperparameter tuning for HTMLDet

The old model was configured with general settings and achieved an accuracy of 89.67%, precision of 92.34%, recall of 87.07%, and f1-score of 89.63% on the test data. The corresponding loss was 2.2474. However, a new model was introduced with specific changes, including a kernel_initializer set to 'he_uniform', bias_initializer set to 'zeros', and kernel_regularizer with a strength of 1e-3. The activation function was updated to softmax, and the loss function was changed to categorical_cross_entropy. With these modifications, the new model demonstrated significant improvement, achieving an accuracy of 91.14%, precision of 92.98%, recall of 89.46%, and f1-score of 91.18%. The corresponding loss substantially decreased to 0.2605. These performance gains in the new model suggest that the adjustments made to the architecture and loss function have enhanced its ability to make more accurate predictions and better fit the given data.



(a) Old Model (b) New Model

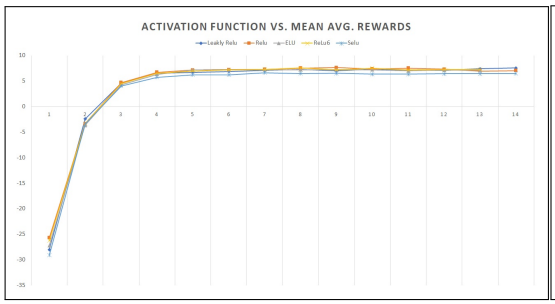
Overall Performance of DLM

The DLM model demonstrated a minimized loss with the new architecture, where the softmax activation function and categorical_cross_entropy loss function were utilized. These changes in the model's architecture and loss function contributed to improved convergence during training, resulting in a significantly lower loss value. The utilization of softmax and categorical_cross_entropy in combination allowed the model to better capture the relationships between classes and produce more accurate predictions, leading to the observed reduction in the loss metric.

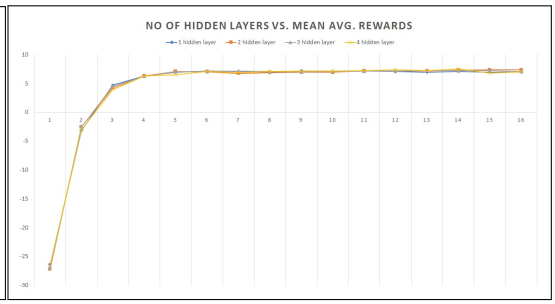
Selecting the Optimal Network Architecture for DQN

When selecting the optimum DQN architecture, this study has conducted several experiments by changing hyperparameters such as the activation function, number of

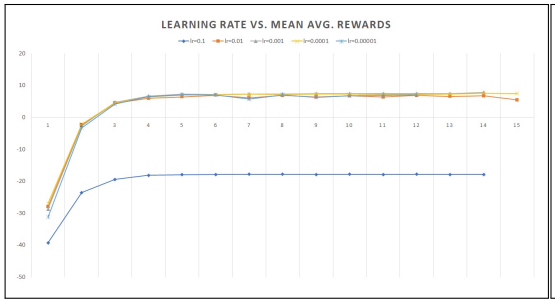
hidden layers, hidden neurons, learning rate, target network update frequency, and loss function. The objective of these experiments was to explore the effects of different hyperparameter configurations on the performance of the DQN model. By systematically varying these hyperparameters, the study aimed to identify the combination that yields the best results for the specific task at hand.



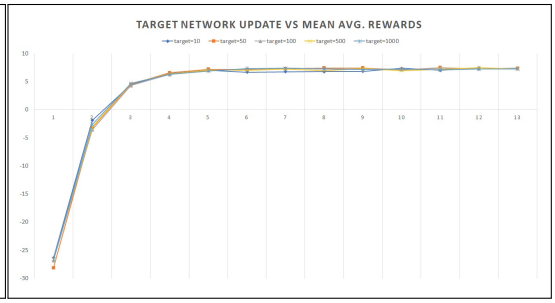
(a) Activation Function



(b) Number of Hidden Layers

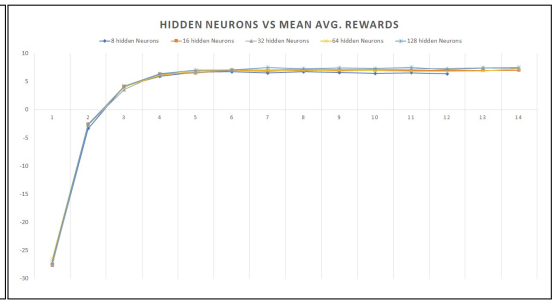


(a) Learning Rate



(b) Target Network Update

(a) Loss Function



(b) Number of Hidden Neurons

Mean average rewards with different hyperparameters

In these experiments, the mean average rewards collected at the end of each episode were considered as the primary evaluation metric. The mean average reward served as a crucial performance indicator, representing the agent's ability to achieve its objec-

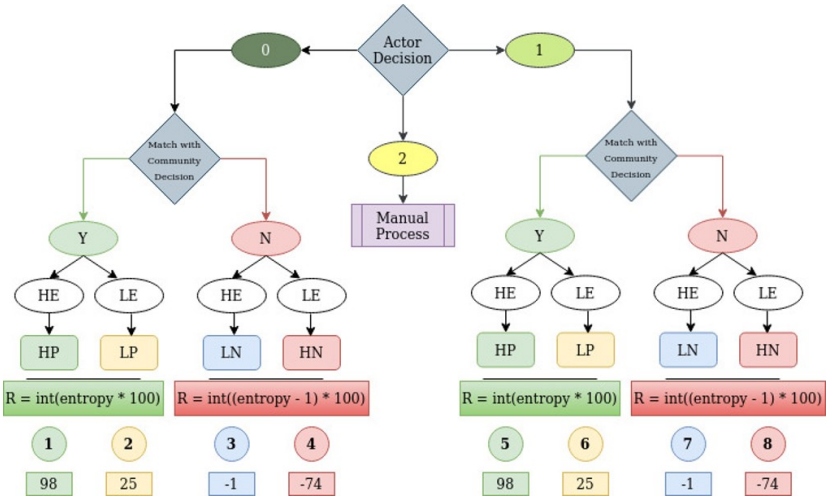
tives and indicating how well it navigated the environment and learned the optimal policy.

The study analyzed the results obtained from each experiment, comparing the mean average rewards for different hyperparameter settings. This analysis provided insights into the impact of each hyperparameter on the agent's performance. It allowed the researchers to identify the configurations that led to higher rewards and more efficient learning.

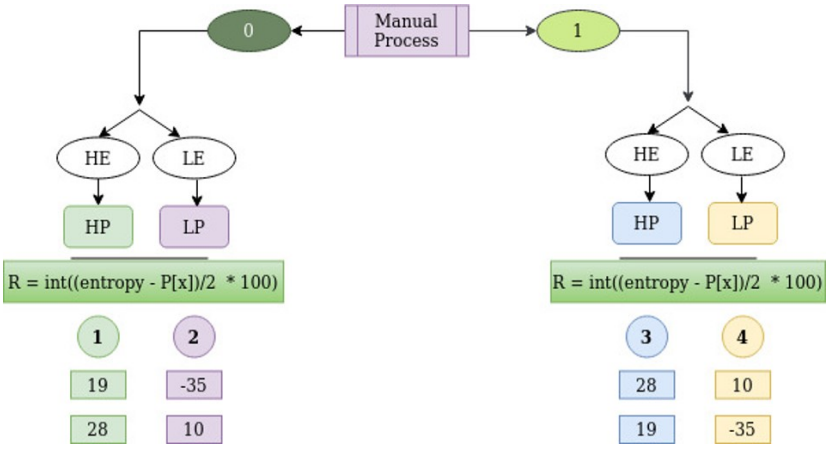
Experimenting with RL Reward Mechanisms

The RL agent's reward functions were subjected to experimentation under various scenarios. The following illustrates how three different formulas discussed in Chapter 6 generate distinct reward values in different scenarios. This analysis serves to validate the appropriateness of each selected reward function for specific situations.

By exploring the reward values generated by these formulas, we can gain insights into how the agent responds to different environmental conditions and task objectives. This investigation allows us to assess the effectiveness of each reward function in guiding the agent's behavior towards achieving desired outcomes. Ultimately, the justification of the selected reward functions for each scenario enhances our understanding of how to design effective RL reward systems tailored to specific problem domains.



Automated reward generation process



Manual process in which the agent collaborates with real users

APPENDIX C. SUPPLEMENTARY INFORMATION

Publications

1. "Detecting phishing attacks using a combined model of LSTM and CNN", *International Journal of Advanced and Applied Sciences*, vol. 7, no. 7, pp. 56-67, 2020. Available: 10.21833/ijaas.2020.07.007.

2. "PhishRepo: A Seamless Collection of Phishing Data to Fill a Research Gap in the Phishing Domain", *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 5, pp. 852-865, 2022. Available: 10.14569/ijacsa.2022.0130597.

3. "Combining Long-term Recurrent Convolutional and Graph Convolutional Networks to Detect Phishing Sites using URL and HTML", *IEEE Access*, vol. 10, pp. 82355-82375, 2022. Available: 10.1109/ACCESS.2022.3196018.

4. "SmartiPhish: A Reinforcement Learning-based Intelligent Anti-Phishing Solution to Detect Spoofed Website Attacks". *Manuscript submitted for publication*.

Project Sources

Main codebases

- Hybrid DLM implementation: https://github.com/sna-hm/HybridDLM
- DLM implementation: https://github.com/sna-hm/DLM
- SmartiPhish implementation: https://github.com/sna-hm/SmartiPhish
- MORA Browser implementation: https://github.com/sna-hm/MORABrowser

Specific code samples

- HTML feature extractor: https://github.com/sna-hm/HybridDLM/blob/main/extractor.py
- Noisy data remover: https://github.com/sna-hm/Miscellaneous/blob/main/preprocess-phishing-pages.py
- PhishTank data extractor: https://github.com/sna-hm/Miscellaneous/blob/main/PhishTankDataExtractor.py
- Legitimate data extractor: https://github.com/sna-hm/Miscellaneous/blob/main/LegitimateDataExtractor.py
- OpenPhish data extractor: https://github.com/sna-hm/Miscellaneous/blob/main/OpenPhishDataExtractor.py
- Google Search script: https://github.com/sna-hm/Miscellaneous/blob/main/GoogleSearchIndexed.py

Datasets

- Phishing Websites Dataset: https://data.mendeley.com/datasets/n96ncsr5g4
- PhishRepo Dataset: https://data.mendeley.com/datasets/ttmmtsgbs8
- Google Search Keywords: https://github.com/sna-hm/Miscellaneous/blob/main/GoogleSearchKeywords

Videos

- MORA Browser: https://youtu.be/_MddiKIFvXM

Everything is theoretically impossible, until it is done.

—Robert A. Heinlein

[The End]