Imperial College London

Department of Computing

Deep Natural Language Processing Model for Phishing Detection and Target Identification

Author:
Congyun Guo

Supervisor:
Dr Sergio Maffeis

Submitted in partial fulfillment of the requirements for the MSc degree in MSc of Imperial College London

September 2023

Abstract

Phishing attacks have alarmingly increased in recent years, leveraging hand-crafted emails to deceive victims and obtain confidential information. Reports from the Anti-Phishing Work Group underscore a concerning 150% annual rise in such incidents since 2019. Furthermore, the Verizon Data Breach Investigation Report reveals that phishing-driven attacks accounted for over 19% of 2022's breaches; the urgency for efficient countermeasures has never been more palpable. Notably, the problem is not limited to detecting phishing attacks; it is equally important to identify the target of zero-day attacks for immediate preventive actions.

This project aims to develop a deep learning pipeline that achieves accurate phishing email detection and target identification for both existing (known) and new (unknown) phishing attacks using email content.

The final pipeline successfully combines classification, extractive, and generative models, achieves high performance, and exhibits strong generalization to unknown phishing attacks. The integration of the novel dynamic loss and K nearest neighbor ensemble techniques further enhances the pipeline's overall performance.

Acknowledgments

I wish to express my profound gratitude to Dr. Sergio Maffeis for his invaluable guidance and support throughout this project. I am very thankful for his understanding and encouragement since the project started. My gratitude also goes to Professor Giuliano Casale for his co-supervision and for clarifying many questions I had at the beginning of the project.

Additionally, I would like to thank Zlatanov Vasil. His generous support – from understanding data structure to discussing existing pipeline architecture - enriched the quality of this work.

Contents

Chapter 1

Introduction

1.1 Motivations

Phishing is a fraudulent attempt to obtain targets' personal or confidential information [1]. The attacker often impersonates legitimate and credible companies and sends victims hand-crafted emails with embedded URLs, baiting the victim into inadvertently submitting confidential information. Over the last few years, phishing emails have significantly increased. According to Anti-Phishing Work Group's (APWG) Q4 report of 2022, "the number of phishing attacks has grown by more than 150% per year since 2019", with over 4.7 million attacks in 2022 [2]. Phishing has increased exponentially, and so has the harm caused by it. According to Verizon Data Breach Investigation Report (DBIR), phishing attacks caused over 19% of breaches in 2022 [3], resulting in billions of ransomware damage. While phishing attack can be conducted through various medium, this works focus on email phishing attack, as it is the most common channel [4].

Detecting phishing emails is a complex task and has been in active research in the past decade. As phishing attacks evolve, earlier methods, such as heuristic and blacklisting, become less effective [5]. Consequently, researchers attempt to use Machine Learning (ML) methods with hand-crafted features, from hyperlinks to detect phishing emails. However, recent phishing attacks, such as shortened URL or redirection attacks, have found ways of circumventing traditional URL-based methods. This encourages the study of phishing detection using solely email content. Currently, deep learning (DL) methodology with word embeddings achieves state-of-the-art performance in many phishing detection datasets [6, 7, 8].

The pressing challenge, however, extends beyond phishing detection. In the ever-evolving landscape of phishing attacks, it is equally crucial to detect and identify the target of zero-day (unknown) attacks to take preventive measures. Despite this, research on identifying the target of phishing is limited. While [9, 10] designed ML methodologies for detecting zero-day phishing websites, zero-day phishing email target detection remains unexplored.

1.2 Objectives

This study aims to develop a pipeline that can accurately detect phishing emails and identify the target of the attack using email body content. Specifically, we strive to construct a robust pipeline that excels at existing (known) targets while generalizes well for new (unknown) targets. Our objectives include:

- Adapt and compare DL, ML, and transformer-based models for phishing detection and target identification.
- Develop an accurate and robust pipeline using state-of-the-art phishing detection and target identification models. The pipeline should be carefully designed, and different combining strategies should be explored.
- Evaluate and improve the pipeline performance for known and unknown target identification.

1.3 Contributions

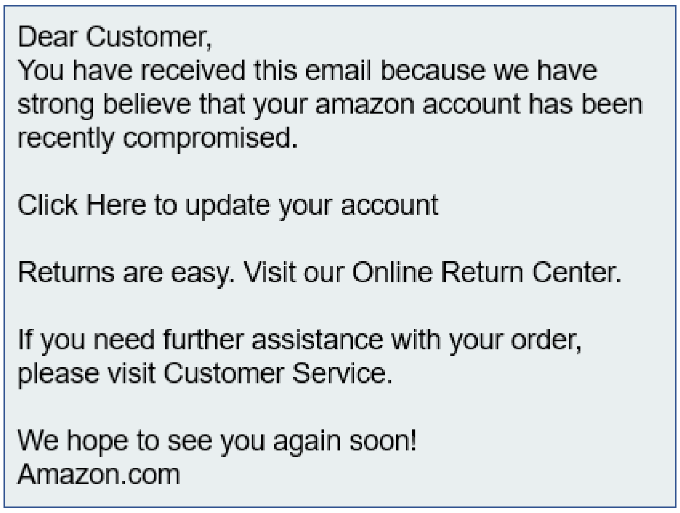
The main contributions of this project can be summarized as follows:

- Compared, adapted and fine-tuned various DL, ML, and transformer-based models for phishing detection (Section 3.2) and target identification (Section 3.3)
- Designed a four-component pipeline combining classification, extractive, and generative models with good performance (Section 3.4).
- Explored various techniques to improve the model's performance further. Proposed dynamic loss and KNN ensemble techniques for hard example mining and improving pipeline's generalization ability. (Section 3.5)
- Evaluated pipeline phishing detection (Section 4.2) and target identification (Section 4.3) performance for both known and unknown targets.

Chapter 2

Background Knowledge

2.1 Phishing Detection


Dear Customer,
You have received this email because we have strong believe that your amazon account has been recently compromised.

Click Here to update your account

Returns are easy. Visit our Online Return Center.

If you need further assistance with your order, please visit Customer Service.

We hope to see you again soon!
Amazon.com

Figure 2.1: Example of phishing email from phishing corpus [11]

Phishing detection is the identification of malicious attacks to acquire sensitive information by masquerading as a trustworthy entity in electronic communication. While phishing attacks can be conducted via phone calls (Voice phishing) or text messages (SMS phishing), email phishing is the most common. This method involves luring the recipient into clicking on a link in the email that redirects them to a fraudulent website, often designed to mimic a legitimate service. The victim is then prompted to enter personal information, as illustrated in Figure 2.1.

Cybercriminals utilize many phishing techniques to deceive their targets. Among these is spear phishing, in which the attackers gather detailed information about the victim to craft an email that appears authentic, thereby increasing the chances of success. Another method is clone phishing, where the attackers replicate a legitimate email that the victim has previously received and modify it to include malicious links or attachments. Social engineering is also often employed, where attackers expertly manipulate human psychology to trick individuals. This may involve using urgent language or leveraging current events to make the request seem legitimate. These sophisticated strategies highlight phishing attempts' nuanced and complex nature and emphasize the need for robust phishing detection methods to keep pace with evolving cyber threats.

2.2 Target Identification

In addition to detecting phishing attempts, a critical aspect of this research is identifying the specific company being impersonated by the attacker. Recognizing the targeted entity enables immediate protective actions, such as customer and employee alerts and enhanced security measures. Different from phishing detection, for the email in Figure 2.1, target identification aims to pinpoint the specific target "Amazon".

Besides conventional multiclass classification, this study employs several advanced methods for target identification. This section provides a concise overview of these tasks: named entity recognition, token classification, extractive question answering, abstractive summarization, and text generation.

Named Entity Recognition (NER) Named Entity Recognition (NER) is the task of identifying and classifying named entities (such as persons, organizations, locations, and dates) in a given text into predefined categories.

Token Classification (TC) Token Classification (TC) categorizes individual words into specific classes based on their context. Typical applications include part-of-speech tagging, where tokens are classified as nouns, verbs, and adjectives. For target identification

Extractive Question Answering (EQA) Extractive question answering (EQA) involves pinpointing and extracting specific segments or spans from a given text that directly answers a posed question. Instead of generating new content or providing a synthesized response, the system identifies the most relevant portion of the original text, ensuring the answer is factual, contextually accurate, and directly sourced from the provided material.

Text Generation (TG) Text generation's (TG) objective is to create coherent and contextually relevant textual content based on specific input or prompts. Utilizing various algorithms, including DL models like T5, this task spans myriad applications from story writing and poetry creation to generating conversational responses in chatbots.

Abstractive Summarization (AS) Abstractive summarization (AS) aims to generate a concise and coherent summary of a given text, capturing its core meaning and presenting it in new, synthesized sentences. Unlike extractive summarization, which merely selects sentences from the source text, abstractive methods reformulate and condense the content, producing more natural-sounding summaries.

2.3 Machine Learning (ML) Models

Machine Learning (ML) is a branch of computer science that focuses on building machines to recognize patterns from data and make decisions without explicit programming. ML algorithms are broadly classified into supervised, unsupervised, and reinforcement learning. In particular, supervised learning uses a training set that includes both the input data and the label. For this study, we utilize supervised learning approaches to categorize phishing emails and identify their targets. The following sections detail the specific ML models employed in this research.

2.3.1 Random Forest (RF)

Random Forest (RF) is an ensemble learning method that combines multiple decision trees to generate a more accurate and stable prediction [12]. Each tree in the forest is grown using a bootstrap sample from the training data, and during the construction of trees, only a random subset of features is considered for splitting. This randomness introduced both in data and features ensures diversity among trees. For classification tasks, the mode of the classes predicted by individual trees is taken as the final prediction. Owing to its versatility and robustness against overfitting, RF has found applications across diverse domains, including phishing detection.

2.3.2 Logistic Regression (LR)

Logistic Regression (LR) is a statistical method that models the probability of a binary outcome using a logistic function. Unlike linear regression, which predicts continuous values, LR predicts the likelihood that a given instance belongs to a particular category. It calculates the weighted sum of input features and passes the result through a logistic (sigmoid) function to squeeze the output between 0 and 1. The weights are learned using methods like maximum likelihood estimation. Due to its simplicity and efficiency, LR is widely used in phishing detection.

2.4 Deep Learning (DL) Models

Deep Learning (DL), a subset of ML, employs neural networks with multiple layers (deep architectures) to analyze various data. The strength of DL lies in its ability to process vast amounts of data, automatically extracting features that traditional ML models might overlook. Subsequent sections delve into two prominent DL architectures.

2.4.1 Convolution Neural Network (CNN)

Convolution Neural Network (CNN) is a type of neural network that contains a convolutional layer on top of other common layers, such as fully connected layers [13]. In a CNN, each neuron only sees a small input region and extracts useful features from the local region. The local information is then combined to allow the network to form a global understanding of the input. In NLP, a typical convolutional operation involves sliding a kernel across text embedding. The output value is obtained by element-wise multiplication of the weights associated with the kernel and the embedding [13]. Pooling layer is often used with convolutional layers to reduce the dimension of feature representations.

2.4.2 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a type of neural network containing a memory unit, making it very useful and applicable to tasks requiring sequential data processing. As RNN considers the outputs from previous timestamps when computing the current output, it stores the previously seen information and uses it to make future predictions [14]. This makes RNN suitable for phishing email detection as RNN excels at modeling long-distance dependencies across paragraphs.

2.5 Transformer-based Models

Transformer-based models, building upon deep learning foundations, signify a paradigm shift in the design and training of models. Originating from the paper “Attention Is All You Need” by Vaswani et al. in 2017 [15], the Transformer architecture introduced the self-attention mechanism, allowing the model to weigh input tokens differently, making it particularly adept at handling varying contexts in language. Subsequent sections explore a selection of state-of-the-art transformer-based models in detail.

2.5.1 Transformers

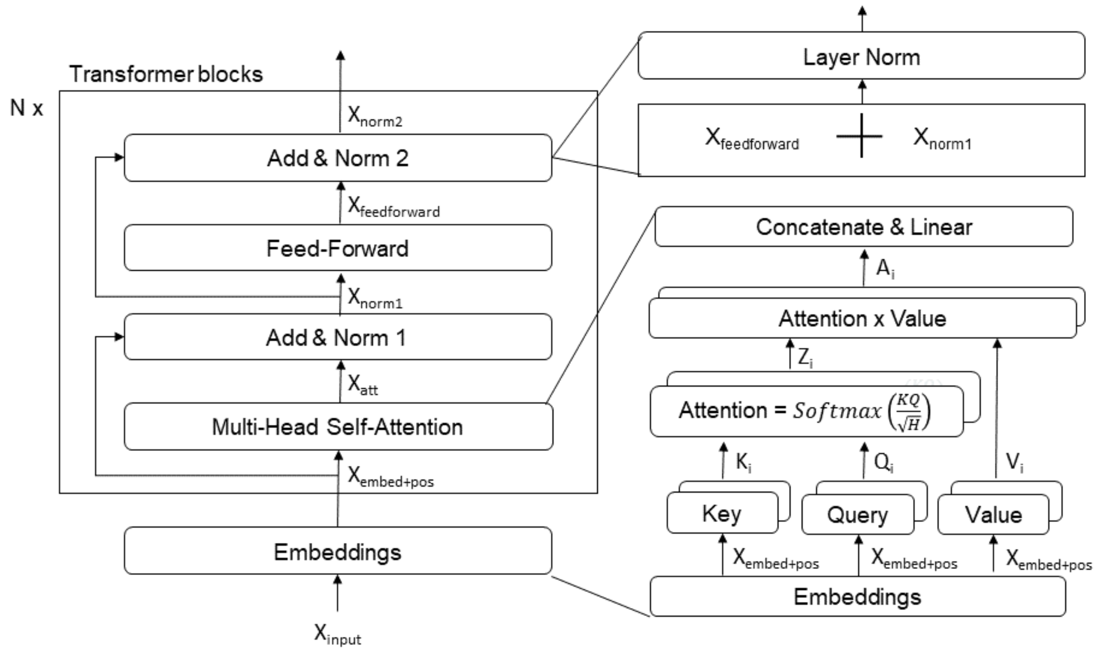


Figure 2.2: Illustration of transformer block

In [15] Vaswani et al. introduced transformers encoder and decoder. This work focuses primarily on the transformer encoder, a multilayer architecture containing six transformer blocks and one embedding layer. Each transformer block comprises a multi-head attention module, a feedforward network, normalization modules, and residual connectors.

Embedding layer For a vector of input tokens of length N $X_{input} \in R^N$, it first passes through an embedding layer that converts each token into an H dimensional embedding, resulting in $X_{embed} \in R^{N \times H}$. Element-wise addition is then performed between X_{embed} and X_{pos}, a sinusoidal positional embedding of shape $R^{N \times H}$. The X_{pos} is defined as follows:

$$X_{pos(j,2i)} = \sin\left(\frac{j}{10000^{2i/H}}\right), X_{pos(j,2i+1)} = \cos\left(\frac{j}{10000^{2i/H}}\right) \tag{2.1}$$

Transformer block The added embedding, say $X_{embed+pos} \in R^{N \times H}$, is fed to the multi-headed attention module, and we have $X_{att} \in R^{N \times H}$ as the output. The input and output of the attention module, $X_{embed+pos}, X_{att}$, is connected residually and layer-wise normalised as follows:

$$X_{norm1} = LayerNorm(X_{embed+pos} + X_{att}) \tag{2.2}$$

Let $X_{residual} = X_{embed+pos} + X_{att} \in R^{N \times H}$, layer normalisation normalises the $X_{residual}$ across the H dimension [16]. For position $j \in 0 \ldots N-1$ and dimension $i \in 0 \ldots H-1$, the normalised matrix $X_{norm1} \in R^{N \times H}$ is calculated as:

$$X_{norm1(j,i)} = \frac{X_{residual(j,i)} - \mu_j}{\sigma_j} \tag{2.3}$$

where mean μ_j and variance σ_j are calculated as follows:

$$\mu_j = \frac{1}{H} \sum_{i=0}^{N-1} X_{residual(j,i)}, \sigma_j = \sqrt{\frac{1}{H} \sum_{i=0}^{N-1} (X_{residual(j,i)} - \mu_j)^2} \tag{2.4}$$

The normalized hidden state, X_{norm1}, is then fed to a two-layered feedforward network, outputting $X_{feedforward} \in R^{N \times H}$. Again, the X_{norm1} and $X_{feedforward}$ are residually connected and normalized as follows:

$$X_{norm2} = LayerNorm(X_{norm1} + X_{feedforward}) \tag{2.5}$$

Where $X_{norm2}, \in R^{N \times H}$, is the transformer block's output.

Multi-head attention Multi-head attention is a mechanism that attends to various positions in the sequence, enabling the transformer model to capture complex token-wise relationships. Consider as M the total number of heads, the i_{th} attention head calculate the attention matrix $A_i \in R^{N \times \frac{H}{z}}$ as follows:

$$A_i = Softmax(\frac{Q_i \times K_i}{\sqrt{H}})V_i \tag{2.6}$$

where Q_i, K_i, V_i, are linear transformations of $X_{embed+pos}$ with weight matrix $W_{q,i}, W_{k,i}, W_{v,i} \in R^{Hx\frac{H}{Z}}$ respectively.

$$Z_{(k,j)} = \frac{\exp U_{(k,j)}}{\sum_{j=1}^{K} \exp U_{(k,j)}} = Softmax(U_k)_j \tag{2.7}$$

Multiplication between $Softmax(\frac{Q_i K_i}{\sqrt{H}})$ and V_i allows each token to attend to other token's values. All attention heads, $A_1 \ldots A_M$, are then concatenated and projected with $W_o \in R^{N \times N}$ as follows:

$$Y = W_o[A_1 \ldots A_M] \tag{2.8}$$

where $Y \in R^{N \times H}$, is the multi-head attention layer output.

Position-wise Feedforward Layers The transformer's feedforward network consists of two fully connected layers. The feed-forward network calculates $P \in R^{N \times H}$ using the output Y from the multi-head attention module as follows:

$$P = ReLU(PW_1 + b_1)W_2 + b_2 \tag{2.9}$$

where W_i, b_i is i_{th} layer's weights and biases.

2.5.2 Bidirectional Encoder Representations from Transformers (BERT)

Devlin et al. developed BERT, a bidirectional transformer encoder based on the original transformer model. BERT uses trainable positional embedding instead of the fixed sinusoidal vector. BERT uses the word piece tokenizer, splitting tokens into subwords to handle out-of-vocabulary (OOV) words [17]. BERT was pre-trained on mask language modelling (MLM) and next-sentence prediction tasks (NSP) to achieve high transferability in downstream tasks [17]. For the MLM task, BERT aims to predict the correct masked token given its context. Whereas for the NSP task, given two sequences, BERT aims to determine whether the first sentence precedes the second in the original text This work uses BERT as the baseline model due to its near state-of-the-art performance in text classification and its wide usage in NLP [17, 18].

2.5.3 A Lite BERT (ALBERT)

ALBERT designed to address the hardware limitations associated with scaling traditional BERT architectures, introduces two critical parameter-reduction techniques: factorized embedding parameterization and cross-layer parameter sharing. These innovations significantly reduce the model's parameters without significantly impacting performance, resulting in an 18x reduction compared to BERT-large and a 1.7x faster training time. ALBERT's design improves parameter efficiency, stabilizes training, and aids generalization. Furthermore, the introduction of a self-supervised loss for sentence-order prediction (SOP) enhances inter-sentence coherence, setting new state-of-the-art results on benchmarks such as GLUE, RACE, and SQuAD [19].

2.5.4 Pre-training with Extracted Gap-sentences for Abstractive Summarization Sequence-to-sequence models (PEGASUS)

PEGASUS is a model designed explicitly for abstractive text summarization, extending the success of Transformer-based models to this domain [20]. Unlike traditional extractive summarization, PEGASUS generates the summarization based on the original document. Its pre-training objective, gap sentence generation (GSG), focuses on masking whole sentences from a document and generating these gap sentences from the rest of the document. This approach encourages whole-document comprehension and summary-like generation. PEGASUS has shown the ability to equal or exceed state-of-the-art performance across various summarization tasks, such as news, science, instructions, and emails. Its design allows for high adaptability even in low-resource summarization scenarios, quickly fine-tuning to achieve top results with minimal supervised pairs.

2.5.5 Text-to-Text Transfer Transformer (T5)

T5 is a model developed by Raffel et al. that treats every text processing problem as a text-to-text task [21]. This unified approach enables the model to take any text as input and generate new text as output, covering a wide range of English-based NLP tasks such as translation, question-answering summarization, and sentiment classification. The text-to-text framework of T5 allows the application of the same model, objective, training procedure, and decoding process across various tasks. T5's methodology explores the limits of transfer learning by scaling up models and data sets beyond what has previously been considered, introducing the "Colossal Clean Crawled Corpus" (C4) and obtaining state-of-the-art results in numerous tasks.

2.6 Related Work

Traditionally, filtering techniques, such as blacklist and heuristic, are often used for fast phishing detection [22, 23, 24]. The author of [23] proposed a blacklist-based approach in which the URLs of phishing websites are recorded. When the user receives a suspicious URL, the URL is classified as phishing if it is similar to the ones on blacklist websites. List-based approaches are often ineffective when encountering a webpage not included in those pre-established lists. In addition, these lists require frequent updates, which can be computationally expensive. In [24], Moghimi et al. uses CANTINA+, a heuristic-based approach, to classify phishing URLs. The algorithm extracts the most frequent words on the webpage for any potentially malicious website and searches them in a search engine. The webpage is classified as legitimate if it appears in the first research results. With that being said, the attacker can manipulate these entries and make the phishing website the top result.

Researchers also proposed various ML models to solve the problem. Examples of frequently used ML techniques are LR, RF, and support vector machine (SVM) [25, 26, 27, 28, 29]. In [27], Ho et al. trained an RF classifier to discover unseen phishing attacks using "phishy" keyword features and URL reputation information. While their method requires a large dataset of URL reputation information, this historical data may not always be available. In [28], Kumar et al. introduced a hybrid methodology that uses SVM for feature extraction and Probabilistic Neural System (PNN) for further phishing detection. While this strategy demonstrates good performance, the system's performance is limited by the quality of SVM-extracted features. Despite this, SVM's large computation cost makes it less efficient on input data. Thus, our work uses LR and RF instead.

Some recent work has employed various DL models to extract features and detect phishing attacks automatically [30, 31, 7, 8]. In [30], Halgas et al. proposed a Long Short Term Memory (LSTM) based model to detect phishing attacks using word sequences. The author of [31] introduced a Convolutional Neural Network (CNN) that only extracts email headers to detect potential spam messages. This work also uses RNN and CNN models with text embedding for phishing detection. However, our experiment demonstrates that the transformer-based models outperform the RNN and CNN models.

There are relatively few studies that utilize a transformer-based approach. Maneriker et al. [32] proposed URLTran, a transformer-based phishing URL classification model, and achieved high performance compared to other DL methods. Extending from [32] Wang et al. [33] proposed TCURL, which uses a CNN model to capture local correlations and combines it with a transformer model. This work distinguishes itself from the studies presented in [32, 33] by focusing on phishing detection within the email content. This approach requires the model to capture morphological and semantic information embedded within the phishing email content. Additionally, [34] blends Topic Modelling, Named Entity Recognition (NER), and Structural features with image processing, utilizing RF, SVM, and LogitBoost classifiers to detect phishing emails. While this work also uses NER and other topic modelling models, we use the extracted entities for target identification instead of as features for phishing detection. Furthermore, in [35], Lee et al. introduced Context-Aware Tiny Bert, which detects phishing emails based on email content information. While this work also uses BERT base models, we emphasize improving phishing detection performance instead of inference time. Additionally, different from [32, 33, 35], this work also focuses on target identification.

Many studies explore methodologies for target identification. However, most of them focus on webpages-based methodologies. For instance, in [36], a DBSCAN clustering-based approach was proposed to discern phishing targets, utilizing relationships such as link, ranking, text similarity, and layout similarity among web pages. However, given their complexity and inconsistent performance, these clustering techniques often prove ineffective for email data. Similarly, [37], the authors present a graph-based approach that leverages a suspicious webpage's associated relationships to identify its phishing target automatically. Despite this, due to the distinct characteristics of phishing emails, graph-based approaches are not always applicable, and this work uses supervised methods instead. Similar to this work, [38] also uses a supervised phishing detection system with a gradient-boosting target identification component. However, this work also uses DL and transformer-based methods, which demonstrated improved performance over gradient-boosting methods. Moreover, unlike webpage-based methodologies, this work primarily focuses on email text data.

Few studies have explored target identification for text messages and emails. [39] offers a content-focused, multi-stage methodology employing Conditional Random Field (CRF) and Latent Dirichlet Allocation (LDA) to both detect phishing content and discern the impersonated entities. [40] details PhishWHO, a tri-phase mechanism leveraging a weighted N-gram model for keyword extraction, a search engine integration for domain targeting, and a three-tiered system for identity verification. This work primarily uses the transformer base models for target identification, which is different from [39, 40]. Our uniquely designed pipeline integrates classification, extraction, and generative components. Notably, it excels in recognizing known targets and also exhibits commendable generalization capabilities for unknown target identification.

Chapter 3

Methodology

This chapter presents the process and techniques to arrive at the final pipeline. We start with introducing the dataset and how data is preprocessed and split. Next, we discuss the binary classification models used for phishing detection. After that, we present the multiclass, generative, and extractive models used for target identification. The integration method used to combine those models will also be explained. Then, we present an overview of the final pipeline architecture and discuss additional techniques used to improve pipeline performance. Finally, we describe the hyperparameter tuning procedure for each model.

3.1 Dataset

3.1.1 Dataset Overview

This work evaluates models using three datasets: merged phishing corpus dataset ($dataset1$), repartitioned Netcraft dataset ($dataset2$), and original Netcraft dataset. ($dataset3$).

This work employs Netcraft's dataset ($dataset2$ and $dataset3$) for target identification and phishing detection tasks. The Netcraft dataset consists of 458,683 (57.8%) phishing emails and 334,434 (42.2%) benign emails. Each email in the dataset is labelled into one of 445 distinct categories with class distribution illustrated in Figure 3.1. Each email within this dataset adheres to the MIME format and includes details such as the email body, sender information, subject line, and embedded URLs.

In the absence of previous research utilizing NetCraft's dataset, additional phishing datasets were employed to identify suitable models and enable comparison with existing literature. Due to a scarcity of multiclass datasets, this work leveraged binary classification datasets, specifically the SpamAssassin dataset [41] and the Nazario phishing corpus [11] as they are frequently used in previous research [42]. The phishing corpus dataset, assembled and hand-classified by cybersecurity researcher José Nazario [11], consists of 7,315 phishing emails. These encompass attributes such as sender's information, subject line, body content, and embedded URLs, offering a comprehensive insight into various phishing techniques, such as spear phishing and brand impersonation. The SpamAssassin dataset, part of the

Apache SpamAssassin project, includes 6,950 benign and 5,932 non-benign emails. The emails contained in the dataset have been collected and hand-classified by contributors. It features key attributes like email content, subject, headers, and other metadata for analyzing spam characteristics.

The phishing corpus and SpamAssassin dataset are highly unbalanced regarding phishing and benign email distribution, making them unsuitable for evaluating performance. As a solution, and in alignment with previous literature [43, 8, 7, 29], a more balanced dataset combined from sampling the two datasets. The combined dataset ($dataset1$) contains 4,000 emails, including 2,000 benign and 2,000 phishing emails. The benign emails were sourced from the SpamAssassin project [41], while the phishing emails were from Nazario [11]. This approach, with closer distribution in the Netcraft dataset, thus facilitates the underlying model to better learn and discriminate between phishing and benign email.

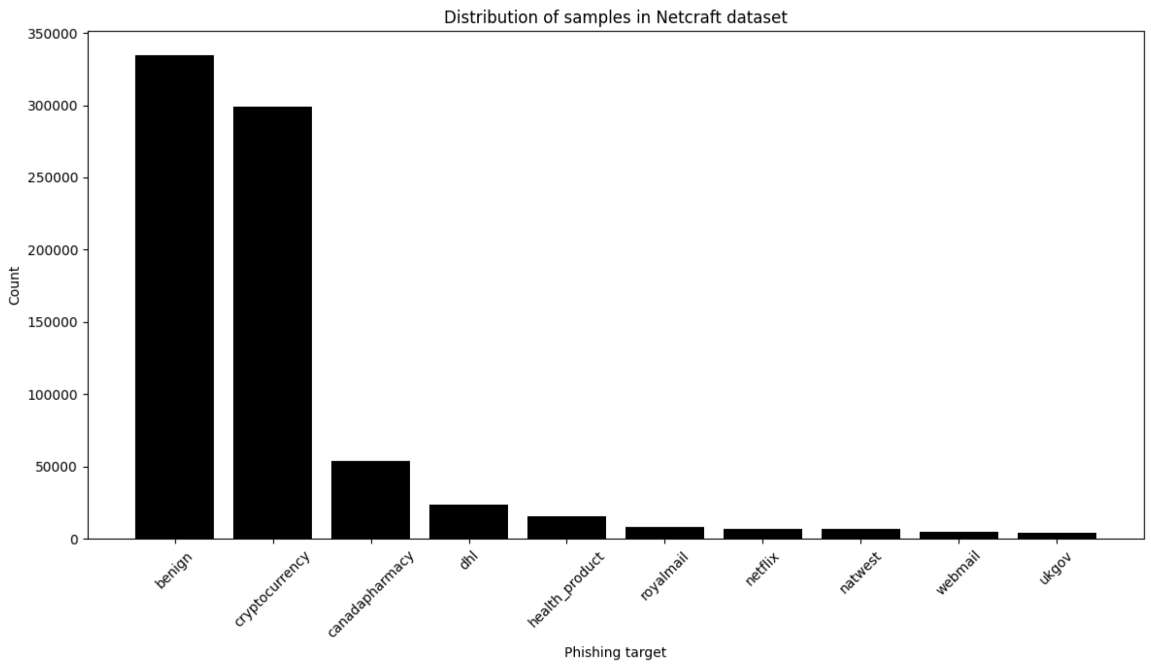


Figure 3.1: Distribution of samples per top ten phishing class in Netcraft dataset

3.1.2 Dataset Preprocess

This work preprocesses the three datasets through several key steps, as illustrated in Figure 3.2. The process begins by extracting only the email body from each dataset. Then, we clean the text by removing unnecessary text, HTML tags, and excessive line breaks and symbols. Moreover, all embedded URL was eliminated to ensure the analysis of only semantic text. Emails containing fewer than ten tokens were excluded since they usually comprise only a link or a phrase, which is unbeneficial for training. In line with the work's focus on English text, non-English content was removed. Finally, additional anonymizing preprocessing scripts were applied to the

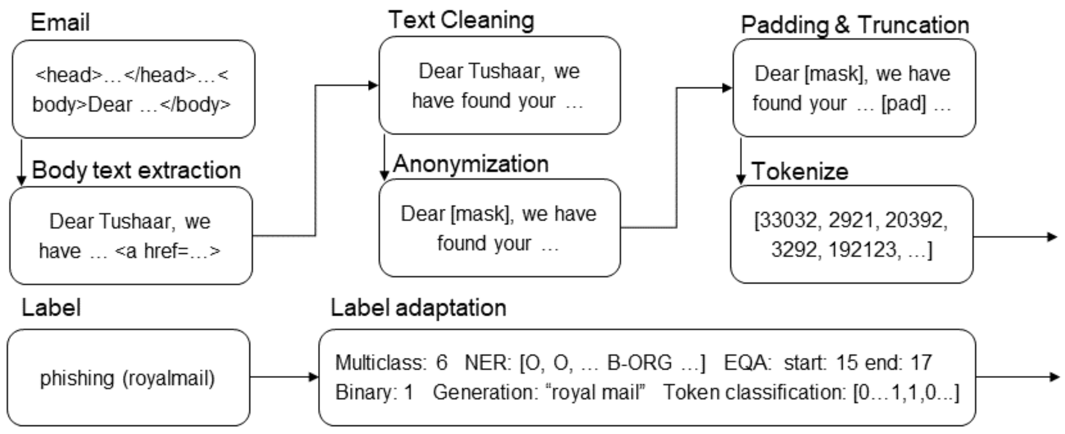

Email
<head>…</head>…<body>Dear …</body>
Body text extraction
Dear Tushaar, we have … <a href=…>
Text Cleaning
Dear Tushaar, we have found your …
Anonymization
Dear [mask], we have found your …
Padding & Truncation
Dear [mask], we have found your … [pad] …
Tokenize
[33032, 2921, 20392, 3292, 192123, …]
Label
phishing (royalmail)
Label adaptation
Multiclass: 6 NER: [O, O, … B-ORG …] EQA: start: 15 end: 17
Binary: 1 Generation: "royal mail" Token classification: [0…1,1,0…]

Figure 3.2: Illustration of dataset preprocessing pipeline

datasets, further eradicating any remaining sensitive information and strengthening privacy protection and ethical compliance.

Distinct tokenization methods are applied based on the specific model to optimize training time. For transformer-based models such as BERT, the corresponding BERT tokenizer is employed, and the text is truncated and padded to a fixed length of 512 tokens. In contrast, the word_tokenize method from the NLTK library is utilized for non-transformer-based models.

Furthermore, the label for each email is also preprocessed for tasks such as binary classification, multiclass classification, and token classification. Particularly, we prepare the binary classification label by mapping all phishing examples into one and benign examples into zero. The labels for each task are preprocessed into the appropriate form for each task and stored in the dataset, with details of the preprocessing step listed in the appropriate sections.

3.1.3 Dataset Splitting

	Train	Valid	Test	Test-unk	Description
dataset1	80	10	10	-	Merged phishing corpus dataset
dataset2	70	10	10	10	Repartitioned Netcraft dataset
dataset3	90	-	10	-	Original Netcraft dataset

Table 3.1: Ratio (%) of train, validation, test, test-unk subsets

The original Netcraft dataset ($dataset3$) is already partitioned into training ($dataset3_{train}$) and testing ($dataset3_{test}$), as shown in Table 3.1. We repartitioned $dataset3$ to get $dataset2$, with two distinct test sets: $dataset2_{test}$ and $dataset2_{test-unk}$. $dataset2_{test}$ is used to assess models' efficacy on known classes, whereas $dataset2_{test-unk}$ evaluates model performance on unknown classes. In detail, the 445 classes in $dataset2$ are pseudo-randomly split into known (95%) and unknown (5%) subsets with disjointed classes. The known subset is sampled into

	$dataset2_{train}$		$dataset2_{valid/test}$		$dataset2_{test-unk}$	
	Count	Ratio %	Count	Ratio %	Count	Ratio %
Benign	246425	44.4	35203	44.4	17603	22.2
Crypto Currency	220110	42.2	31444	39.6	15723	19.8
Canada Phramacy	39704	7.2	5672	7.2	2836	3.6
DHL	-	-	-	-	23568	29.7
Health product	11602	2.1	1657	2.1	830	1.05
Royal Mail	5872	1.1	838	1.1	422	0.5
Netflix	5335	1.0	762	1.0	382	0.5
Natwest	-	-	-	-	7023	8.9
Webmail	3508	0.6	501	0.6	252	0.3
UKGOV	2947	0.5	421	0.5	211	0.3
...	-	-	-	-	-	-
Total	555000	-	79310	-	79310	-

Table 3.2: Distribution of samples for each partition of $dataset2$

training ($dataset2_{train}$ 70%), validation ($dataset2_{valid}$ 10%), test ($dataset2_{test}$ 10%), and test-unknown ($dataset2_{test-unk}$ 5%) subset. The unknown subset is merged with $dataset2_{test-unk}$, resulting in 10% $dataset2_{test-unk}$ subset, as shown in Table 3.1 and 3.2. Since half of the data in $dataset2_{test-unk}$ have unknown classes, it is used to assess the model's robustness against unknown targets.

For $dataset1$, given its primary role in facilitating comparisons with existing literature, we adopted a more straightforward approach. The dataset was stratified sampled into training ($dataset1_{train}$ 80%), validation ($dataset1_{valid}$ 10%), and testing ($dataset1_{test}$ 10%) subsets. These were designated for training, hyperparameter tuning, and evaluating, respectively.

3.2 Phishing Detection

3.2.1 Binary Classification

Phishing detection can be viewed as a binary classification task, where given an email text, the classification models output 1 if it is phishing and 0 otherwise. To compare models' performance against previous literature performance, this work trains transformer-based, DL, and ML models on $dataset1_{train}$, $dataset2_{train}$, and evaluates $F1_{micro}$ on $dataset1_{test}$, $dataset2_{test}$, $dataset2_{test-unk}$.

For transformer base models, we use "bert-base-uncased" (BERT) and "albert-base-v2" (ALBERT) due to their state-of-the-art performance in many natural language tasks [19, 17]. To utilize these models, $dataset1$ and $dataset2$ are tokenized from the Hugging Face library using corresponding tokenizers - BERTtokenizer, ALBERTtokenizer. These tokenizers convert the string texts into input indexes, which are then fed into the pre-trained models from the Hugging Face repository to fine-tune them on the datasets.

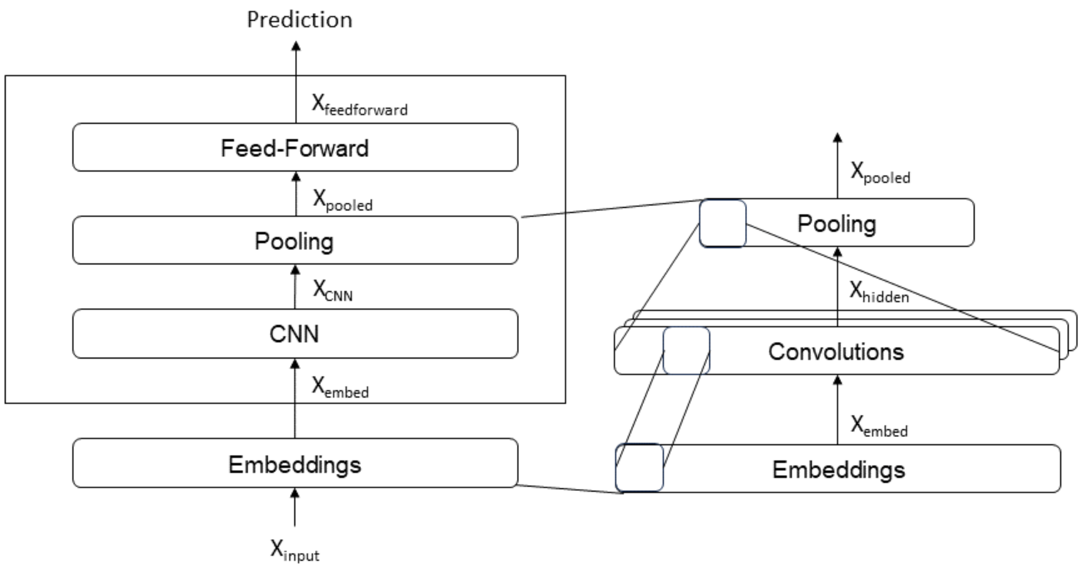


Figure 3.3: Illustration of CNN model architecture

For DL models, we choose RNN and CNN due to their proven state-of-the-art performance in phishing detection [8]. Again, we tokenize $dataset1$ and $dataset2$ text into indexes using the NLTK library's tokenizer. Then, the preprocessed indexes are converted into numerical vectors of 300 dimensions using word2vec from Gensim's library. Taking the word embeddings as input, we fine-tuned the CNN and RNN on the binary classification task.

For ML models, we employ RF and LR due to their frequent application in phishing detection [29]. Though SVM also yields strong performance, its memory and space complexity render it unsuitable for $dataset2$. Like the DL model's preprocessing steps, the text is tokenized using NLTK and converted into 300-dimensional vectors with word2vec.

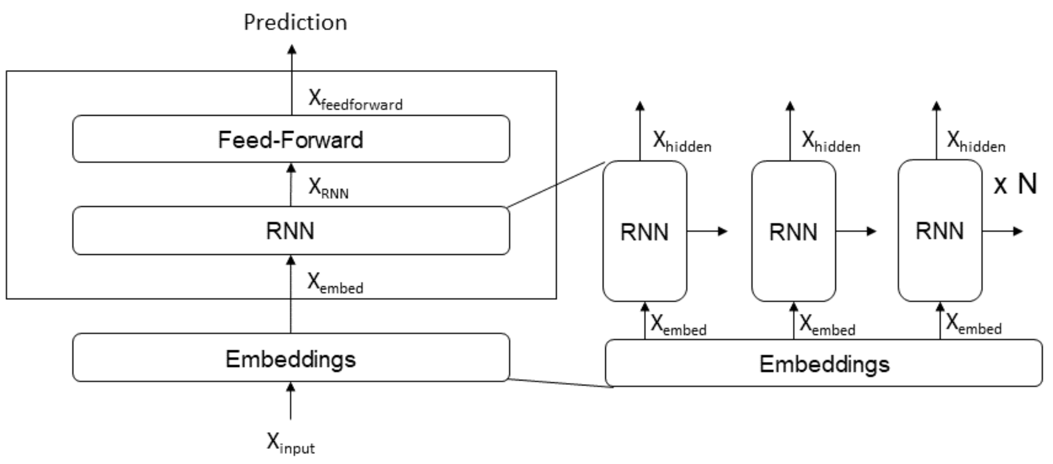


Figure 3.4: Illustration of RNN model architecture

3.3 Target Identification

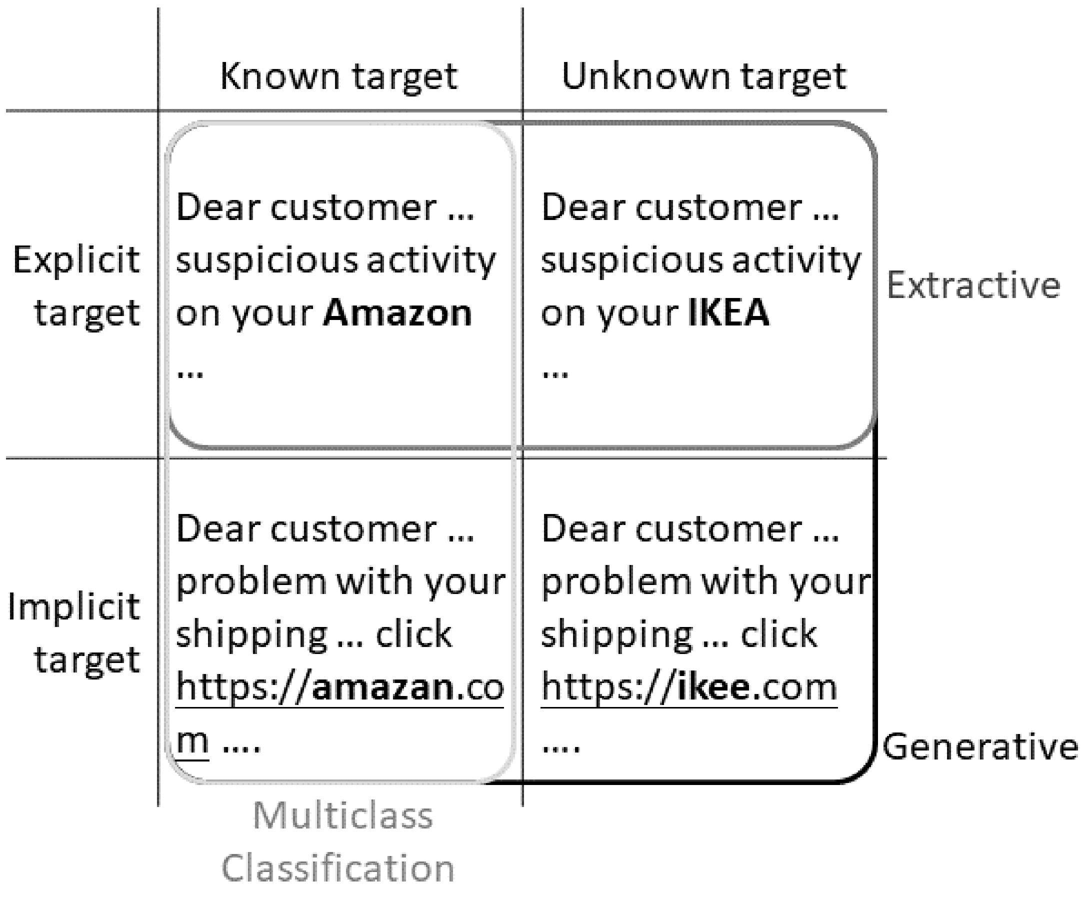

Known target
Unknown target
Explicit target
Implicit target
Dear customer ... suspicious activity on your **Amazon** ...
Dear customer ... suspicious activity on your **IKEA** ...
Dear customer ... problem with your shipping ... click https://**amazan**.com ....
Dear customer ... problem with your shipping ... click https://**ikee**.com ....
Extractive
Generative
Multiclass Classification

Figure 3.5: Illustration of four type of phishing email

Our initial strategy utilized multiclass classification models on $dataset2$ to identify phishing email targets. While effective for known targets, this approach lacked the flexibility to handle unknown targets, which was vital for generalizing to unknown phishing classes in $dataset2_{test-unk}$. Recognizing this limitation, we embarked on a deeper analysis of the phishing email characteristics. Figure 3.5 illustrates that some phishing emails explicitly named the target, such as "Amazon". In contrast, others were more subtle, impersonating targets using format, logos, imagery, or links.

Guided by these insights, we categorized the emails into four distinct categories based on visibility (explicit or implicit) and familiarity (known or unknown). Each category was tackled with models best suited for its specifics:

- **Known Targets**: We applied multiclass classification to categorize emails into pre-established classes.
- **Implicit Targets**: We leveraged summarization and generation models to deduce and generate implicit targets.
- **Explicit Targets**: We employed EQA, NER, and TC models to extract explicit targets.

For clarity, we termed models focused on extracting explicit targets (i.e. EQA, TC) as extractive models. In contrast, models that attempt to generate implicit targets (i.e. AS, TG) were referred to as generative models.

Given the inherent challenges in discerning implicit targets, we undertook manual preprocessing of specific target labels to reduce the number of implicit mentions. An example includes splitting "louisvuitton" to "louis vuitton".

Moreover, since we focus on target identification of phishing emails, we only back-propagate the loss of phishing emails, while the loss of benign emails is masked.

3.3.1 Multiclass Classification

A critical component of our methodology for target identification revolved around handling known targets. We carefully compare and select the optimal multiclass classification model to address this challenge. Drawing parallels with Section 3.2.1, the same models and preprocessing steps were taken, with slight architecture modification to suit the multiclass context. Specifically, the output size for each model is adjusted to 455 instead of 2.

3.3.2 Generative Models

In the context of unknown target identification, generative models are intuitive as they can handle both explicit and implicit targets. This work attempted to reframe unknown target identification into two generative tasks: AS and TG.

Abstractive Summarization (AS) Naively, unknown target identification can be reconsidered into a summarization task. This work uses the "pegasus-base" (PEGASUS) fine-tuned on the C4 dataset [20]. We convert the multiclass classes into strings for preprocessing and use them as the label for further fine-tuning AS models. At inference, we use the AS models to summarize the email, where predictions exceeding five tokens are pruned to improve accuracy.

Text Generation (TG) Another approach for unknown target identification is to reframe it as a text generation task. By prepending emails with the prompt, "What is the target of this email?", we leverage the semantic knowledge of the fine-tuned TG model. Particularly, this work preprocesses each email by prepending the predetermined prompt. Then, using the string of multiclass target as the label, we further fine-tune "flan-t5-base" (T5) [21] for the generation task. During inference, the same question is prepended to the email and the generated text is used as the output. Again, the generated text is truncated if it is longer than five tokens to ensure that only keyphrases are extracted.

Since both the PEGASUS and T5 model exhibit good performance even without additional fine-tuning, this work compares their performance with and without further fine-tuning.

3.3.3 Extractive Models

This work explores various extractive models and strategies for identifying explicit targets in phishing emails. Specifically, we attempted to frame the explicit target identification into TC, NER, and EQA tasks.

Token Classification (TC) The naive approach of identifying unknown explicit targets is through token classification. For each input token, we perform binary classification on whether it is part of the explicit target. Preprocessing of labels involves initializing a zero vector and assigning 1 to the explicit target indices. Then, we appended a token classification linear layer to the "bert-base-uncased" (BERT) model and fine-tuned the model. We extract continuous token spans with a size shorter than five at inference, selecting the span with the highest mean confidence as the prediction.

Named Entity Recognition (NER) Explicit target identification can also be framed as a NER task. This reframing allows for utilizing existing NER models, thereby leveraging the entity information within these models for enhanced generalization. Particularly, we utilize the "bert-base-NER" (BERT-NER) model from dslim [17], originally fine-tuned on the CoNLL-2003 NER dataset [44]. In the preprocessing stage, we create the label for the NER task by assigning organization tags (B-ORG, I-ORG, O-ORG) to explicit target indices, while other indices receive the O tag. During inference, the most frequent organization spans were extracted as the target.

Extractive Question Answering (EQA) Another method of tackling explicit target identification is EQA. Again, by prepending the predetermined prompts "What is the target of this email?" to the email, we can capitalize the extractive capabilities of already fine-tuned EQA. Specifically, we employ the "bert-base-uncased-squad1" (BERT-EQA) [17] for this task. For preprocessing, we prepend emails with the chosen prompt and use explicit target's start and end indices as labels. Any predictions exceeding five tokens were pruned to improve accuracy.

Since the three models can only extract explicit targets during fine-tuning, we only backpropagate the loss of phishing emails with an explicit target, while losses of other emails are masked. Moreover, EQA and NER models' performance with and without further fine-tuning are compared.

3.3.4 Integration Method

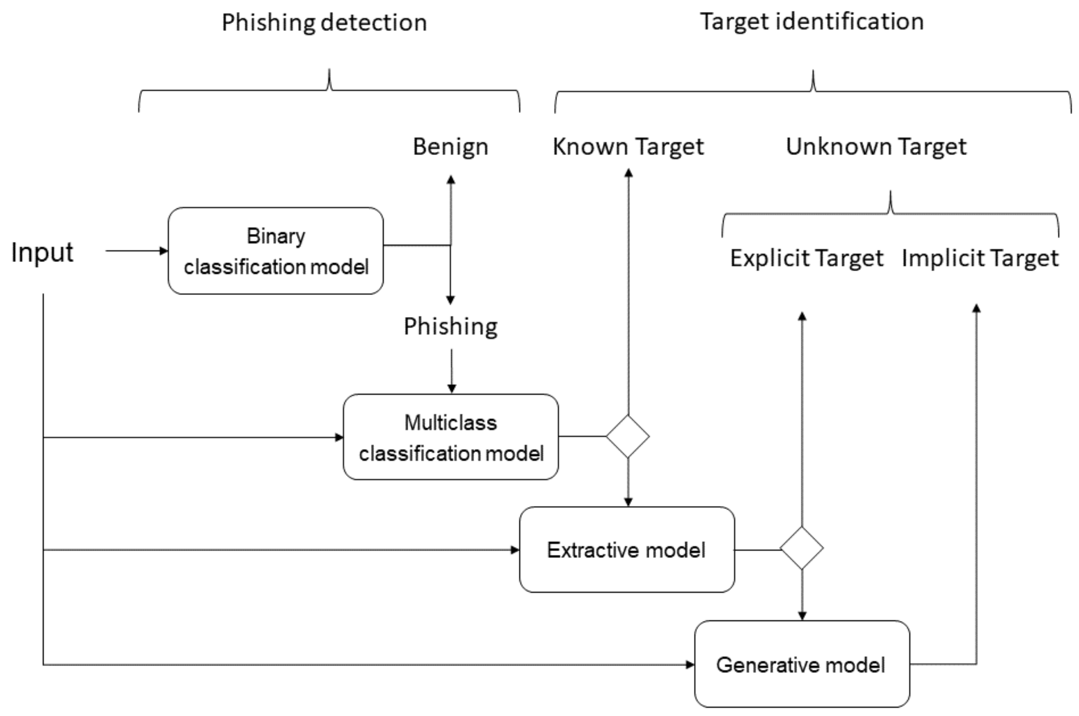


Figure 3.6: Illustration of the sequential (confidence threshold/unknown class) pipeline

This work investigates three methods for combining multiclass, extraction, and generation models for target identification: confidence threshold, unknown class incorporation, and ensemble.

Confidence Threshold Method This method integrates models in a consecutive order of inference, driven by their respective empirical performance and uses the model confidence to integrate the prediction. To maximize the combined pipeline $F1_{micro}$, we arrange the models in the order of multiclass, extractive, and generative. For each model's prediction, if the confidence falls below the predetermined threshold α, then we use the prediction from the subsequent component. For instance, if the multiclass model's prediction has low confidence ($< \alpha$), the prediction from the extractive model is used instead. However, if the extractive model also has low confidence ($< \beta$), the prediction from the generative model is used.

While this approach offers good performance and simplicity in implementation, the complexity of threshold tuning presents a significant challenge. Particularly, α and β need to be optimized on the validation dataset to determine the optimal threshold, risking overfitting and increasing computational demands.

Unknown Class Method An alternative method was explored to address the computational challenges of threshold tuning by incorporating an unknown class. Specifically, we reassigned 10% of the classes in the training data as unknown, with specific label assignments for various tasks (e.g., 456 for multiclass, "unanswerable" for EQA, [O, O...] for NER, and [0,0...] for token classification). The multiclass model's architecture was adjusted to classify into 456 categories, up from 455.

Models were subsequently fine-tuned with this newly incorporated unknown class. Again, emails are assessed sequentially, in the order of multiclass, extractive, and generative models. First, the email is fed to the multiclass model; if the multiclass model predicts "unknown", the extractive model processes the email, and if it finds no target, it is fed to the generative model. While this approach avoids threshold tuning, it necessitates including the unknown class in training, which leads to suboptimal data utilization.

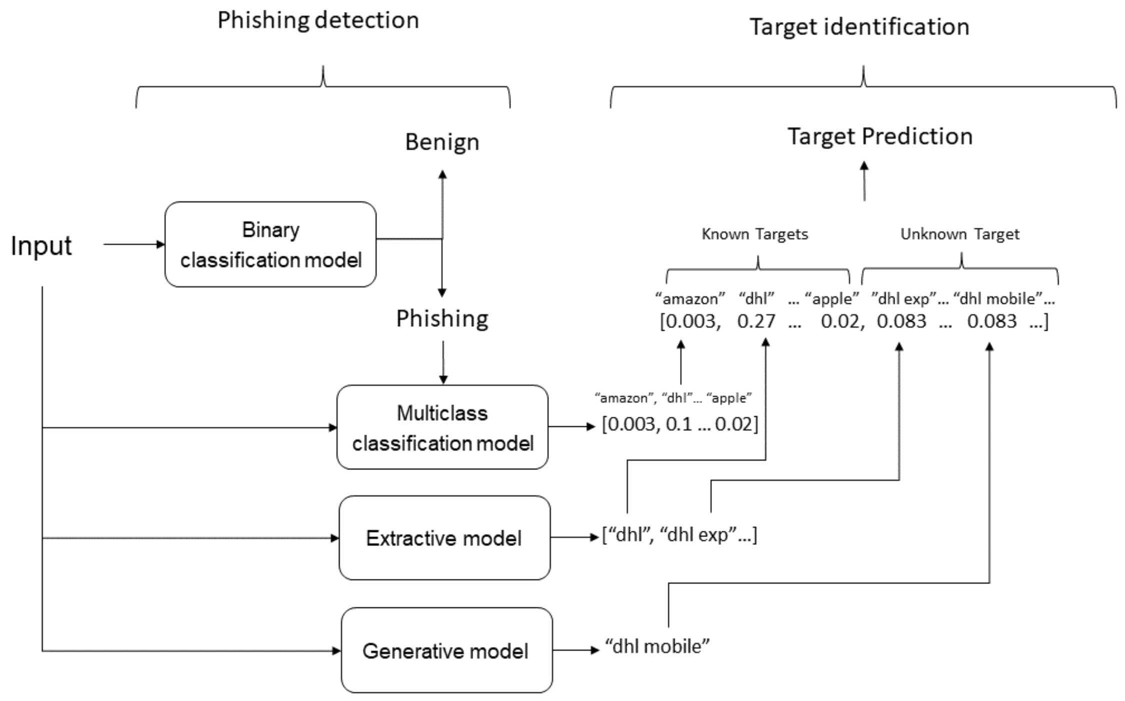


Figure 3.7: Illustration of the ensemble pipeline

Ensemble approach To expedite inference, this section explores pipeline integration with the ensemble method. This approach combines the top $k = 10$ predictions from extractive models with the prediction from generative and multiclass classification models for target identification. The combination process is graphically illustrated in Figure 3.7.

The predictions are mapped into a vector of size 1×456, with the first 445 indices corresponding to predetermined targets. Indices 446 to 456 correspond to i-th unknown targets, where $i \in \{1 \ldots 11\}$. For the i-th extractive and generative prediction, if the predicted target has not appeared before, we append that target to the existing ones at index $445 + i$. Then, we average the confidence for each of the 456 targets across the multiclass, extractive, and generative prediction. Although a weighted sum can combine the confidence instead of simple averaging, finding the weight for each model is non-trivial and requires careful tuning and thus is not used. In contrast to confidence threshold and unknown class sequential methods, the ensemble method allows parallel inferencing, reducing the inferencing time by approximately 3.2 times.

3.4 Pipeline Overview

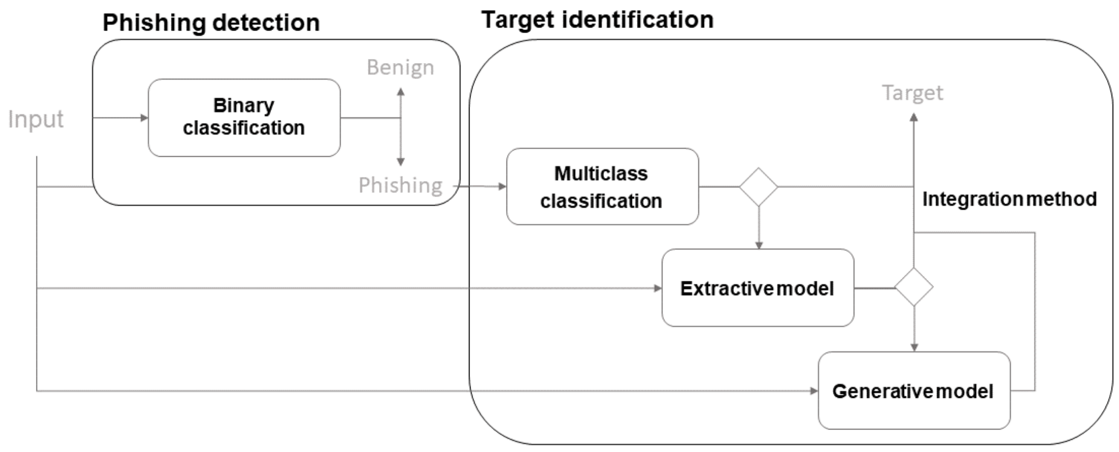


Figure 3.8: Illustration of overall pipeline

The final pipeline consists of phishing detection and target identification steps, as shown in Figure 3.8. For each email, we first use the binary classification model for phishing detection. If it is phishing, multiclass, generative, and extractive models are used for target identification.

3.5 Additional Improvements

This work attempted various methods to improve pipeline performance and tackle problems encountered:

- **Joint Training:** To reduce the long inference time
- **Hard Example Mining:** To increase the performance on minority classes
- **KNN Ensemble:** To improve unknown class generalizability
- **Domain-Specific Pretraining:** To improve pipeline performance

The details of the methods are explained in subsequent sections.

3.5.1 Joint Training

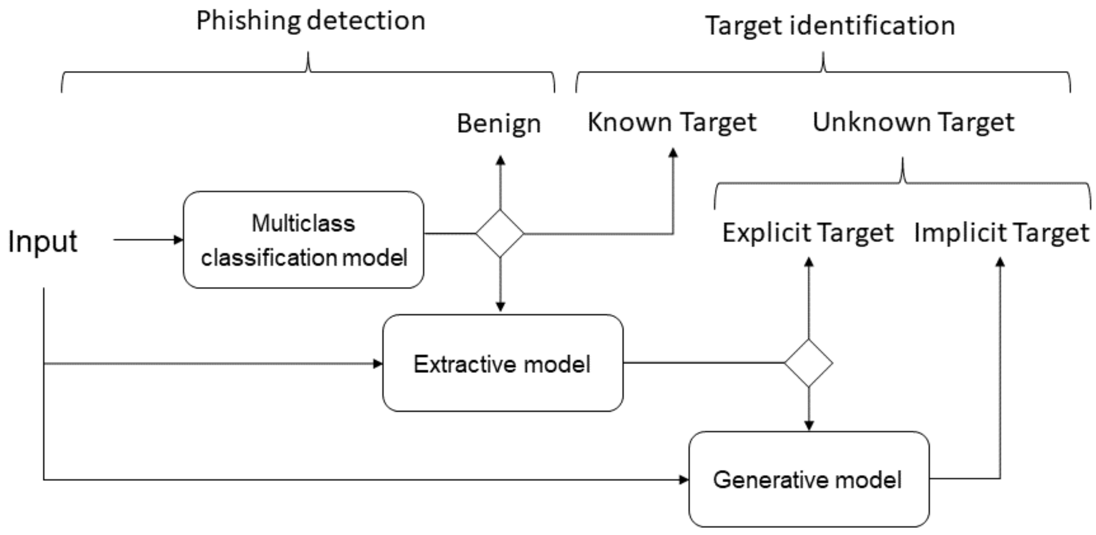


Figure 3.9: Illustration of the single multiclass model pipeline

To mitigate computational and inference costs, this work explores the joint training of models across multiple tasks. Specifically, we investigate the following strategies:

Single Multiclass This strategy replaces binary and multiclass models with a singular BERT multiclass model trained on phishing and benign data, as illustrated in Figure 3.9.

Bi-Task Joint Training In this strategy, we replace the binary and multiclass model with a single model (BERT) trained on binary and multiclass objectives. The intuition behind the fusion of the two tasks is that both tasks use the BERT encoding for classification. Thus, jointly training BERT on those two tasks may improve BERT's

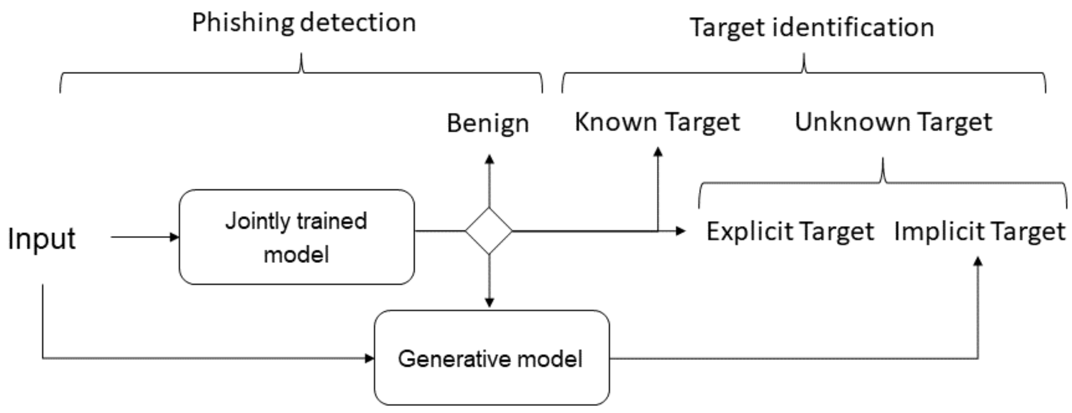


Figure 3.10: Illustration of the 3 component join training pipeline

capability in feature extraction, thus potentially improving overall performance. The combined loss for this approach is formulated as follows:

$$\alpha \times Loss_{binary} + (1 - \alpha) \times Loss_{multiclass} = Loss_{combined}, \tag{3.1}$$

where α is a predetermined hyperparameter constrained within [0,1].

Tri-Task Joint Training This strategy ventures further by jointly training models on binary classification, multiclass classification, and token classification using a single model (BERT), as illustrated in Figure 3.10. The associated combined loss is:

$$\alpha \times Loss_{binary} + \beta \times Loss_{multiclass} + \omega \times Loss_{token} = Loss_{combined}, \tag{3.2}$$

subject to $\alpha + \beta + \omega = 1$ and each parameter lies within [0,1]. The underlying intuition here is that BERT can effectively address all three tasks. Since generative objectives require decoders and are incompatible with BERT, and EQA and NER tasks benefit from pretrained information, the token classification task is chosen as the extractive task for joint training.

Additionally, we masked multiclass loss during joint training when the data was benign. We also masked token classification loss when the data does not contain an explicit target. This is because we only require the multiclass model to focus on phishing target identification and the extractive model to focus on explicit targets.

Moreover, though we initially use fixed weights for α, β, and ω, empirical observation indicates superior performance when these weights are trainable. Thus, we turn them into parameters and optimize them through backpropagation.

3.5.2 Hard Example Mining

In our endeavour to optimize model performance, we identified a predominant issue: unbalanced classes in datasets resulting in low $F1$ in minority classes. For instance, the class “novo banco” only has four samples in the training dataset. The scarcity of samples for such classes makes it challenging to correctly classify instances within them, resulting in neglect from the model in favour of more dominant classes. To address this, we introduce two hard example mining strategies: focal loss and a novel dynamic weighted loss.

Focal Loss Tailored to prioritize hard, underrepresented classes, focal loss (Loss_{fl}) is formulated as:

$$\text{Loss}_{fl} = -(1 - p_t)^{\gamma} \log(p_t) = (1 - p_t)^{\gamma}\text{Loss}_{ce} \tag{3.3}$$

Where we set γ to 2 in our implementation. By adding a factor of $(1 - p_t)^{\gamma}$ to the standard cross entropy loss (Loss_{ce}), the loss of examples with $p_t > 0.5$ is reduced, while the loss for misclassified samples with $p_t < 0.5$ remains relatively unchanged. This allows focusing on updating the loss of hard examples, increasing the model’s ability to handle hard examples.

Dynamic Loss This approach, drawing inspiration from boosting, dynamically assigns weights to samples’ losses based on its last epoch classification result ($\text{classification}_{\text{prev}}$). Initially, this approach stores the last epoch classification result and down weights losses ($Loss_{cur}$) for examples that were previously classified correctly, as described in the equation:

$$\text{Loss}_{\text{cur}} = \begin{cases} \text{Loss}_{\text{cur}} & \text{classification}_{\text{prev}} \text{ is correct} \\ \alpha \times \text{Loss}_{\text{cur}} & \text{otherwise} \end{cases} \tag{3.4}$$

Where $\alpha \in [0, 1]$ is a hyperparameter. However, explorative experimentation demonstrated that this method causes substantial fluctuation in performance and prevents loss convergence, as the update of hard examples causes the model to “forget” how to predict the new examples. Therefore, a refined method was introduced wherein the past loss $\text{Loss}_{\text{prev}_i}$ are stored as follows:

$$\text{Loss}_{\text{prev}_i} = \text{Loss}_{\text{cur}_i} + \alpha \times \text{Loss}_{\text{prev}_{i-1}} \tag{3.5}$$

with $\alpha \in [0, 1]$ as the decaying factor for the history of losses, where a higher value $\alpha > 0.5$ puts more focus on the past misclassified data. The dynamic loss ($\text{Loss}_{\text{dynamic}_i}$) for each i^{th} epoch is then computed as the sum of the current cross-entropy loss ($\text{Loss}_{\text{cur}_i}$) and the past loss ($\text{Loss}_{\text{prev}_{i-1}}$):

$$\text{Loss}_{\text{dynamic}_i} = \text{Loss}_{\text{cur}_i} + \text{Loss}_{\text{prev}_{i-1}} \tag{3.6}$$

This method adds a “momentum” of loss for each sample based on classification history. By amplifying the focus on often misclassified samples, this method mitigates the issue associated with class imbalance and hard examples, thus improving the model’s performance.

3.5.3 K-Nearest Neighbors (KNN) Ensemble

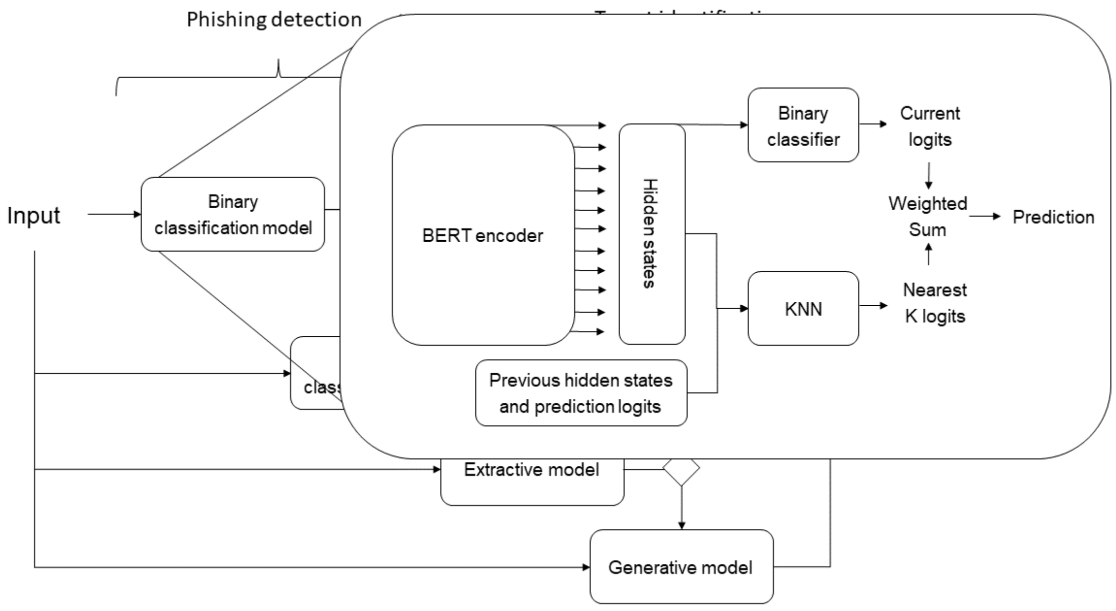


Figure 3.11: Illustration of the KNN ensemble technique

This work introduces a KNN ensemble strategy to improve the model's generalization ability. Our empirical results highlight a lack of robustness in the model's prediction. Particularly, we found misclassifications of emails that, despite sharing similarities with correctly predicted instances, are assigned wrong labels with low confidence.

The KNN ensemble approach was explored to address this issue, as illustrated in Figure 3.11. For binary and multiclass classification models, our method dynamically integrates the target prediction with its neighboring predictions with the following procedure:

Neighbor Identification Utilizing the first entry of BERT last layer hidden state (BERT encoding) of current and previous data, we calculate the cosine similarity to identify the k nearest neighbors. Due to computational limitations, the similarity is only computed between the current data and a maximum of 10,000 other data, and only the ten nearest neighbors are selected. This work explored using training data (KNN TD) and inference data (KNN ID) as neighbors. For KNN TD, the BERT encoding is stored during training, and the data label is used as the prediction. For KNN ID, the BERT encoding and its prediction vector are dynamically stored during inference instead. A basic threshold is applied to exclude the neighbors with non-positive cosine similarity.

Prediction Integration Upon the identification of the k nearest neighbors, the method proceeds to combine the neighbor prediction vector (pred_i) weighted by the cosine similarity (similarity_i) score as follows:

$$\text{pred}_{\text{knn}} = w \times \text{pred}_{\text{original}} + (1 - w) \times \sum_{i=1}^{k} \text{similarity}_i \times \text{pred}_i \qquad (3.7)$$

where $\text{pred}_{\text{original}}$ is the original prediction, pred_{knn} is the KNN ensemble prediction and $w \in [0, 1]$ is the combining hyperparameter. Using neighbors' predictions during inference time improves the model's confidence in the neighbors close to the data while decreasing the model's confidence when they are far away. Thus, it improves the threshold-based unknown classification performance and generalization ability.

While initial efforts leveraged TF-IDF vectors for neighbor identification, explorative experimentation shows that using the BERT encoding led to superior performance. Moreover, preliminary attempts ensemble BERT encoding rather than predictions. However, initial testing revealed comparable outcomes between the two strategies, with predictions ensemble being used due to their more straightforward implementation.

3.5.4 Domain-Specific Pretraining

This work applied domain-specific pretraining on the BERT model for classification, as prior research suggests this result in superior empirical performance [45]. Particularly, using the pretrained BERT [17], this work further pretrained it on $dataset2$ for five epochs, with MLM and SOP objectives, before fine-tuning it on the classification tasks. Preprocessing for domain-specific pretraining includes applying specific masks for the MLM task and necessary pairings for SOP tasks in accordance with standard practices.

3.6 Hyperparameter Tuning

A comprehensive search for hyperparameters was conducted for each model, exploring various configurations as specified in Appendix A. All models were fine-tuned for five epochs on $dataset2_{train}$, and their performances were assessed using the $F1_{micro}$ on $dataset2_{valid}$. This work selects learning rate, weight decay, dropout, optimizer, and scheduler as they significantly affect the model's performance. Though batch size is also an important parameter, GPU memory constraints mandated using a batch size of 64 with an accompanying loss accumulation strategy for all models. The tuning of these hyperparameters occurs in the order of their expected influence on model performance:

- **Learning Rate:** Given its substantial impact on the learning process, the learning rate was the first hyperparameter-tuned
- **Weight Decay and Dropout:** These parameters were tuned together as they both affect model regularization.

- **Optimizer and Scheduler:** These were tuned last in light of their relatively minor influence on model performance and more on the converging speed.

This approach ensures good performance is achieved while preserving computational resources by prioritizing the tuning of hyperparameters according to their influence on model performance. To tune the models, 4 $\times$ 24GB 3090 Nvidia GPUs with mixed precision floating point are used. The tuning time for each transformer model is 5.3 hours.

In addition to these parameters, this study also ventured into experimenting with different activation functions. Particularly for the transformer models, GeGLU and ReGLU activations were attempted, given their better performance over GeLU and ReLU counterparts [46]. The mathematical formulations for these activation functions are provided below:

$$\text{GeGLU}(x, W, V, b, c) = \text{GeLU}(xW + b) \otimes (xV + c), \tag{3.8}$$

$$\text{ReGLU}(x, W, V, b, c) = \max(0, xW + b) \otimes (xV + c). \tag{3.9}$$

Where x is the hidden state, W and V are the weight matrix, and b and c are the biases. However, empirical results demonstrate that changing the activation function of pretrained models leads to worse performance. Hence, the ReLU activation function is used instead for all models.

Chapter 4

Evaluation

This chapter analyzes and evaluates models' performance for phishing detection and target identification. Firstly, we introduce the metrics used to evaluate models' performance. Secondly, we analyze and compare models' performance for phishing detection. Thirdly, we present and compare models' performance for multiclass, generative and extractive tasks. Then, we analyze the performance of joint training, hard example mining, and KNN ensemble techniques. Lastly, this chapter conducts an ablation study for the final pipeline.

4.1 Evaluation Metrics

In this study, we use $F1_{micro}$ for classification, extractive and generative tasks. For extractive and generative prediction we consider it positive if it matches the target string and negative otherwise. Using the true positives (TP_i), true negatives (TN_i), false positives (FP_i), and false negatives (FN_i) for each i^{th} class the micro-averaged $F1_{micro}$ score is calculated as:

$$F1_{micro} = \frac{2 \times \sum_{i=1}^{n} TP_i}{2 \times \sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FP_i + \sum_{i=1}^{n} FN_i} \tag{4.1}$$

The $F1_{micro}$ score, being the harmonic mean of precision and recall, offers a more comprehensive perspective than accuracy. Despite this, the micro-averaged $F1_{micro}$ can sometimes mislead, skewing towards the majority class, especially in imbalanced datasets. As an illustration, a model might predict every instance as "dhl", yielding a high $F1_{micro}$, which seems acceptable but is fundamentally flawed. To mitigate the limitation, we supplement the analysis with the macro $F1_{macro}$ when necessary. Particularly, the macro $F1_{macro}$ score is defined as:

$$F1_{macro} = \frac{1}{n} \sum_{i=1}^{n} \frac{2 \times TP_i}{2 \times TP_i + FP_i + FN_i} \tag{4.2}$$

Essentially, the $F1_{macro}$ calculates the F1 scores for each class independently and then averages them, ensuring balanced consideration for each class. In addition to the F1 scores, this work also presents the micro-averaged precision and recall in Appendix B.

4.2 Phishing Detection

Model	Previous work	$dataset1_{test}$	$dataset2_{test}$	$dataset2_{test-unk}$
AlBERT	-	0.958	0.919	0.828
BERT	-	0.961	**0.923**	**0.830**
RF [29]	0.988	0.986	0.901	0.823
LR [29]	0.989	0.984	0.896	0.811
CNN [7]	**0.992**	**0.987**	0.911	0.824
RNN [8]	0.989	0.985	0.903	0.809

Table 4.1: Binary classification models' phishing detection $F1_{micro}$

	Phishing		Non Phishing		Total
	Count	Ratio %	Count	Ratio %	
$dataset1$	2000	50.0	2000	50.0	4000
$dataset2$	334434	42.2	458683	57.8	793117
[29]	9135	59.2	6295	40.8	15430
[7]	2279	35.4	4150	64.6	6429
[8]	4572	39.7	6951	60.3	11523

Table 4.2: Distribution of phishing and benign data in different datasets

Model	Previous Work	$dataset1_{test\_adjusted}$		
		[29]	[7]	[8]
ALBERT	-	0.963	0.945	0.950
BERT	-	0.966	0.947	0.952
RF [29]	0.988	**0.990**	0.976	0.980
LR [29]	0.989	0.981	0.967	0.971
CNN [7]	0.992	0.989	**0.982**	**0.984**
RNN [8]	0.989	0.988	0.978	0.981

Table 4.3: Adjusted binary classification model phishing detection $F1_{micro}$

This section analyzes the performance of various phishing detection models across $dataset1_{test}$, $dataset2_{test}$, and $dataset2_{test-unk}$, with their $F1_{micro}$ scores depicted in Table 4.1.

Since the data distributions between $dataset1_{test}$ and those of [8, 7, 29] is different, as highlighted in Table 4.2. To make a better comparison with results in previous work, this work adjusted the $F1_{micro}$ of $dataset1_{test}$ as follows:

$$TP_{\text{test\_adjusted}} = TP_{\text{test}} \times \frac{N_{\text{literature\_phishing}}}{N_{\text{dataset1\_phishing}}} \tag{4.3}$$

$$FN_{\text{test\_adjusted}} = FN_{\text{test}} \times \frac{N_{\text{literature\_phishing}}}{N_{\text{dataset1\_phishing}}} \tag{4.4}$$

$$FP_{\text{test\_adjusted}} = FP_{\text{test}} \times \frac{N_{\text{literature\_benign}}}{N_{\text{dataset1\_benign}}} \tag{4.5}$$

Based on the adjusted $F1_{micro}$ in Table 4.3, our model slightly underperforms compared to previous works. The RF model achieves a comparable $F1_{micro}$ of 0.990 in line with [29]'s 0.988. However, our CNN, RNN, and LR models $F1_{micro}$ are around 0.009 lower than those in [7, 29, 8]. This difference in $F1_{micro}$ is tolerable as different model architectures, datasets, and hyperparameters are used in this work. Thus, we argue that the RF, LR, CNN, and RNN implementations are sufficiently accurate, and the comparison is relatively conclusive.

Additionally, Table 4.2 reveals that models consistently achieve higher $F1_{micro}$ scores on $dataset1_{test}$ compared to $dataset2_{test}$ and $dataset2_{test-unk}$. This might be due to NetCraft's dataset's diversity in users, attack types, and targets, making classifying it more challenging.

Furthermore, models' $F1_{micro}$ on $dataset2_{test-unk}$ is only slightly lower than $dataset2_{test}$, despite half of email in $dataset2_{test-unk}$ are from unknown class. This underlines the models' good generalization ability, which can distinguish unknown phishing emails rather than guessing.

Interestingly, transformer models, while underperform traditional ML models on $dataset1_{test}$, surpass them on $dataset2_{test}$. This divergence may stem from the transformer models' effectiveness in using $dataset2_{test}$ large-scale data (793,117 entries) while potentially overfitting on the small $dataset1_{test}$ (4,000 entries). Due to the BERT model's high performance on $dataset2$, it is used as the binary classification component in the final pipeline.

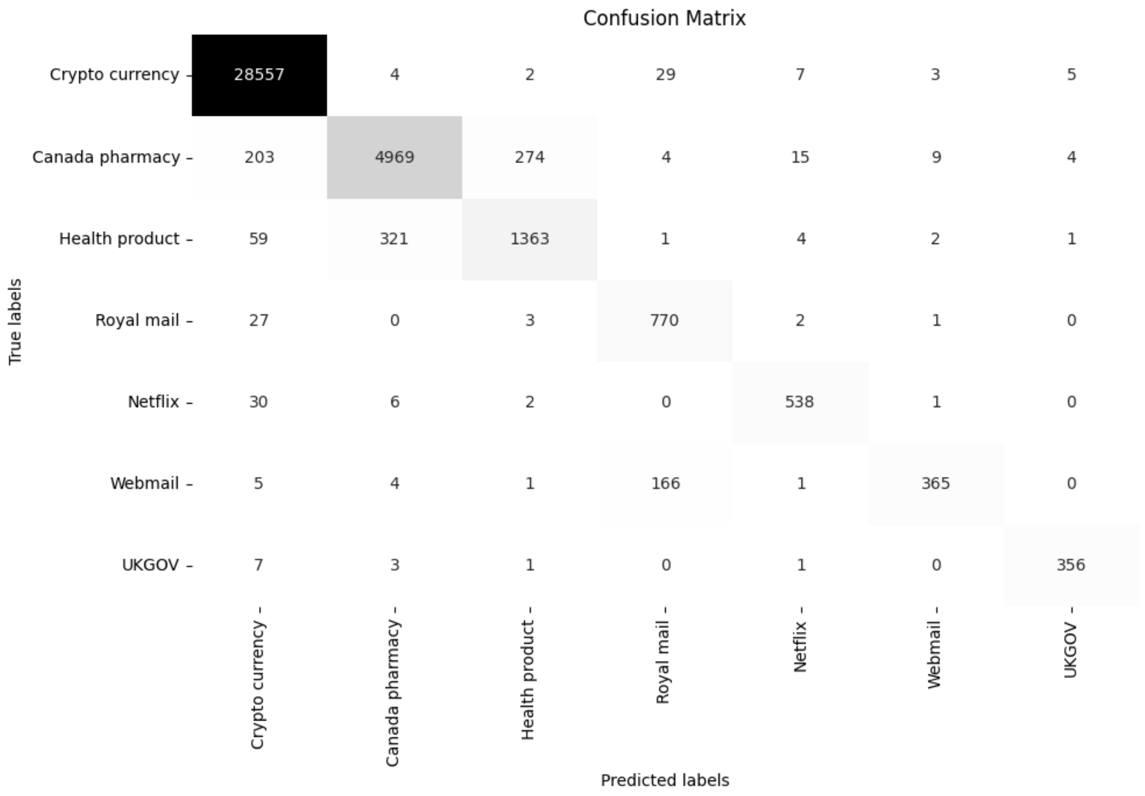
4.3 Target Identification

4.3.1 Multiclass Classification

Model	$dataset2_{test}$	$dataset2_{test-unk}$	$dataset3_{test}$	$dataset3_{reference}$
AlBERT	0.882	0.434	0.956	
BERT	**0.891**	**0.446**	**0.963**	
RF	0.864	0.422	0.936	0.954
LR	0.867	0.417	0.938	
CNN	0.875	0.424	0.945	
RNN	0.872	0.421	0.946	

Table 4.4: Multiclass classification models' target identification $F1_{micro}$

This section compares models' performance across various multiclass models. Since there is no existing phishing target identification multiclass classification dataset, we only compare models' performance on $dataset3_{test}$ with the Bi-directional



	Crypto currency	Canada pharmacy	Health product	Royal mail	Netflix	Webmail	UKGOV
Crypto currency	28557	4	2	29	7	3	5
Canada pharmacy	203	4969	274	4	15	9	4
Health product	59	321	1363	1	4	2	1
Royal mail	27	0	3	770	2	1	0
Netflix	30	6	2	0	538	1	0
Webmail	5	4	1	166	1	365	0
UKGOV	7	3	1	0	1	0	356

Figure 4.1: Confusion matrix of top 7 class for multiclass classification

LSTM results provided by Netcraft's team ($dataset3_{reference}$). We also compare model performance across $dataset2_{test}$ and $dataset2_{test-unk}$, and the micro $F1_{micro}$ is shown in Table 4.4.

First, we observe that the evaluated model achieves comparable performance on $dataset3_{test}$ compared to $dataset3_{reference}$. Particularly, we notice that RNN and CNN achieve slightly lower $F1_{micro}$, while the transformer model achieves superior performance compared to $dataset3_{reference}$. Interestingly, we observe that $F1_{micro}$ on $dataset3_{test}$ is significantly higher than that of $dataset2_{test}$. This is perhaps due to unbalanced data classes and differences in training sample size.

Table 4.4 reveals that the transformer models, AlBERT and BERT, outperform other ML models. This difference in performance highlights transformer models' effectiveness in handling complex relationships within the data.

Furthermore, we observe that the $F1_{micro}$ on $dataset2_{test-unk}$ are lower across all models compared to $dataset2_{test}$. The decreases in $F1_{micro}$ scores indicate a shared challenge in dealing with unknown or unseen classes. This aligns with expectations, as multiclass models inherently can not predict a class not seen during training, thus achieving 0 recalls for the unknown classes.

Moreover, we observe that the model performs better on binary classification than multiclass classification. This is intuitive as a data imbalance issue exists and a larger number of classes to predict in the multiclass setting, thus resulting in a more complex task.

Lastly, we plot the confusion matrix of the top 7 classes in Figure 4.1. The confusion matrix shows that the multiclass model often confuses “Canada pharmacy” with “Health product”, as shown by the respective 321 and 274 FP. Additionally, we observe that some of the “Royal mail” are predicted to be “Webmail”, as shown by the 166 FN.

Since BERT achieves the highest $F1_{micro}$ on $dataset2_{test}$ and $dataset2_{test-unk}$, this work uses it for multiclass classification in the final pipeline.

4.3.2 Generative Models

Objective	Model	Fine-Tuning Strategy	$dataset2_{test}$	$dataset2_{test-unk}$
AS	PEGASUS	Default	0.164	0.167
		Fine-Tuned	0.236	0.155
TG	T5	Default	0.356	**0.354**
		Fine-Tuned	**0.362**	0.303

Table 4.5: Generative models’ target identification $F1_{micro}$

This section compares and evaluates the performance of PEGASUS and T5. Two fine-tuning strategies have been employed for both models, where we compared the performance of the default pretrained model without fine-tuned with the model $F1_{micro}$ on the $dataset2_{test}$ and $dataset2_{test-unk}$ is shown in Table 4.5

Immediately, we observe that T5 achieve better $F1_{micro}$ compared to PEGASUS, regardless of the fine-tuning strategy. This hints that the TG task may be more suitable for the target identification. Since the summarisation models are being trained to summarize semantic meaning rather than pinpoint specific keywords.

Upon fine-tuning, both models demonstrate improved performance on $dataset2_{test}$ but worse performance on $dataset2_{test-unk}$, suggesting catastrophic forgetting. Particularly, T5 model $F1_{micro}$ increased from 0.356 to 0.362 on $dataset2_{test}$ while decreased from 0.354 to 0.303 on $dataset2_{test-unk}$.

Interestingly, T5 can achieve 0.354 $F1_{micro}$ on $dataset2_{test-unk}$ without any fine-tuning, while PEGASUS is able to achieve 0.167 on $dataset2_{test-unk}$. This highlights the utility of already using fine-tuned models for downstream tasks.

While fine-tuned models demonstrate good performance with known targets, this work uses the default T5 as the generative model for the final pipeline and in subsequent sections due to its capability to detect unknown targets.

4.3.3 Extractive Models

This section compares and evaluates the performance of various extractive models. We detail the $F1_{micro}$ of the default and fine-tuned model on the $dataset2_{test}$ in Table 4.6.

As inferred from Table 4.6, default NER and EQA models exhibit strong generalization capability. However, the fine-tuned NER and EQA show superior performance

Objective	Model	Fine-Tuning Strategy	$dataset2_{test}$	$dataset2_{test-unk}$
TC	BERT	Fine-Tuned	**0.439**	0.207
NER	BERT-NER	Default	0.338	**0.339**
		Fine-Tuned	0.351	0.324
EQA	BERT-EQA	Default	0.264	0.257
		Fine-Tuned	0.276	0.246

Table 4.6: Extractive models' target identification $F1_{micro}$

on $dataset2_{test}$ but decreased performance on $dataset2_{test-unk}$, indicating diminished generalizability. Again, the decrease in performance of the fine-tuned models on $dataset2_{test-unk}$ suggests catastrophic forgetting occurring.

Furthermore, the effect of fine-tuning varies significantly across objectives. Token classification benefits the most from fine-tuning, while EQA demonstrates the least improvement. This disparity might be due to the similarity between the previously fine-tuned task and target identification, with more similar tasks receiving more performance improvement.

While model performance appears low, it is important to mention that these models are solely trained on email with explicit targets. Given that roughly half of $dataset2_{test}$ and $dataset2_{test-unk}$ have no explicit targets, performance metrics would be substantially better when assessed against explicit-only text. For instance, while the default NER achieves 0.338 $F1_{micro}$ on $dataset2_{test}$, its $F1_{micro}$ increased to approximately 0.652 on $dataset2_{test-explicit}$.

Additionally, the bad performance of extractive and generative models may also be a result of counting only exact matches, even when the prediction from the extractive model shares a common span with the target label, such as the prediction “apple computer” with the label “apple”, it is counted as a misprediction.

Overall, this work uses the default NER model as the extractive model for the final pipeline and in the subsequent sections, as it achieves decent performance for known targets while achieving good generalization ability for unknown targets.

4.3.4 Integration Method

This section analyzes the performance across various combining strategies on $dataset2_{test}$ and $dataset2_{test-unk}$. These strategies are confidence thresholding, unknown classes, and ensemble. For each strategy, the target identification $F1_{micro}$ of integrating each component is shown in Table 4.7.

First, we observe that integrating the extractive and generative models results in $F1_{micro}$ increase for both confidence thresholding and unknown class strategies. For confidence thresholding, integrating the extractive and generative models results in 0.012 and 0.286 $F1_{micro}$ improvements on $dataset2_{test}$ and $dataset2_{test-unk}$ compared to a single multiclass model. For the unknown class strategy, integrating the extractive and generative models leads to 0.005 and 0.208 $F1_{micro}$ improvement on $dataset2_{test}$ and $dataset2_{test-unk}$, respectively.

Conversely, for ensemble strategy, integrating extractive and generative models

Integration Method	Component	$dataset2_{test}$	$dataset2_{test-unk}$
Confidence Threshold	Multiclass	0.891	0.446
	+Extractive	0.899	0.685
	+Generative	**0.903**	**0.732**
Unknown class	Multiclass	0.842	0.424
	+Extractive	0.843	0.632
	+Generative	0.847	0.696
Ensemble	Multiclass	0.891	0.446
	+Extractive	0.893	0.532
	+Generative	0.885	0.587

Table 4.7: Integration methods' target identification $F1_{micro}$

did not significantly impact the model's performance. This phenomenon is because extractive and generative models have low performance; thus, ensembling those models' predictions may skew the originally correct prediction, resulting in decreased $F1_{micro}$.

Furthermore, we observe that the unknown class method achieves the lowest $F1_{micro}$ on $dataset2_{test}$. This low performance may be due to suboptimal data utilization (as 10% of data is reassigned to the unknown class), and the assigned unknown class types may not be representative of the actual unknown class.

Interestingly, while the extractive method offers a consistent performance boost, generative integration leads to only marginal improvements, hinting at a potential overlap in correct predictions between the multiclass, extractive, and generative models.

Overall, among the evaluated strategies, the confidence threshold strategy outperforms both the unknown class handling and ensemble strategies and is thus used for the final pipeline and the subsequent sections.

4.4 Additional Improvements

This section compares the performance difference between joint training methods, hard example mining, and KNN ensemble against the baseline in our final pipeline. This pipeline comprises Binary classification (BERT), multiclass classification (BERT), extractive (BERT-NER), and generative (T5) components, with the confidence threshold integration method.

4.4.1 Joint Training

In this section, we assess the efficacy of various joint training strategies. The $F1_{micro}$ of the confidence threshold pipeline is shown in Table 4.8.

We observe that the bi-task joint training approach outperforms the single multiclass model, highlighting the advantage of decomposing complex tasks into jointly trained subtasks. Particularly, the bi-task joint training achieves $F1_{micro}$ scores of

Model	Phishing Detection		Target Identification	
	$dataset2_{test}$	$dataset2_{test-unk}$	$dataset2_{test}$	$dataset2_{test-unk}$
Baseline (No JT)	**0.933**	**0.830**	**0.903**	**0.732**
Single Multiclass	0.906	0.816	0.889	0.702
Bi-Task JT	0.926	0.821	0.893	0.717
Tri-Task JT	0.902	0.803	0.882	0.597

Table 4.8: Different joint training (JT) strategies' pipeline $F1_{micro}$

0.926 and 0.893 phishing detection and target identification on $dataset2_{test}$, respectively, surpassing the single multiclass model's scores of 0.906 and 0.889.

Despite this, the tri-task joint training strategy underperforms relative to the single multiclass model, underlining that joint training effectiveness depends on the specific sub-task and model architecture. In this case, the bi-task approach benefits as both classification tasks leverage BERT's encoding. In contrast, the tri-task strategy necessitates training the entire BERT's last layer hidden state for the token classification, resulting in underfitting. Moreover, tri-task joint training's low $F1_{micro}$ score in target identification (0.882 and 0.597) highlights its inability to fit the extractive model due to its complex objective.

Furthermore, the joint training approach leads to slightly decreased performance compared to the sequential baseline, highlighting the inherent trade-off between computational complexity and performance. For instance, the bi-task joint training strategy achieves 0.926 and 0.893 $F1_{micro}$ on $dataset2_{test}$ for phishing detection and target identification, respectively, which is behind the baseline's 0.933 and 0.903. Therefore, joint training strategies are not employed for the final pipeline to optimize performance.

4.4.2 Hard Example Mining

Loss	Phishing Detection		Target Identification	
	$dataset2_{test}$	$dataset2_{test-unk}$	$dataset2_{test}$	$dataset2_{test-unk}$
Baseline (CE Loss)	0.933	0.830	0.903	**0.732**
Focal Loss	0.932	0.821	0.900	0.727
Dynamic Loss	**0.941**	**0.832**	**0.908**	0.731

Table 4.9: Different hard example mining strategies' pipeline $F1_{micro}$

This section compares the pipeline performance with baseline CE loss, focal loss, and dynamic loss. Only binary and multiclass classification models are retrained with the new loss, and the $F1_{micro}$ of the threshold-based pipeline is shown in Table 4.9 and 4.10.

Initial observations underscore a notably low macro $F1_{macro}$ of baseline model for both known and unknown targets. Particularly, the baseline achieves 0.223 and

Loss	Target Identification	
	$dataset2_{test}$	$dataset2_{test-unk}$
Baseline (CE loss)	0.223	0.192
Focal Loss	**0.422**	**0.287**
Dynamic Loss	0.342	0.216

Table 4.10: Different hard example mining strategies' pipeline $F1_{macro}$

0.192 $F1_{macro}$ respectively, highlighting the difficulty when handling hard minority classes.

Compared to the baseline (CE loss), focal loss demonstrates a substantial improvement, achieving 0.422 and 0.287 $F1_{macro}$ on the respective dataset. Dynamic loss also enhances baseline performance, albeit less significant, with 0.342 and 0.216 $F1_{macro}$.

Despite the improvement in $F1_{macro}$, focal loss $F1_{micro}$ decreased, highlighting the trade-off between micro and macro F1. Particularly, focal loss decreased micro $F1_{micro}$ by 0.005 and 0.003 for phishing detection and target identification, respectively. In contrast, dynamic loss increased both $F1_{micro}$ and $F1_{macro}$ for target identification, albeit less significant; therefore, dynamic loss is employed for the final pipeline.

4.4.3 KNN Ensemble

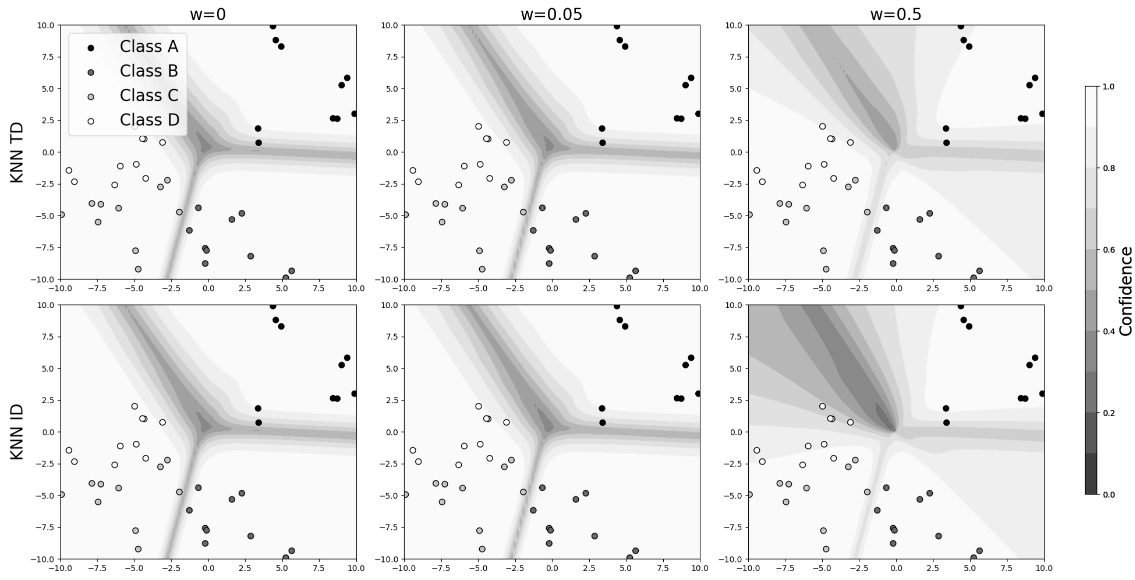


Figure 4.2: Illustration of KNN ensemble model's confidence (with known class (A, B, C) and unknown class D.)

This section delves into the performance implications of using KNN ensembles with training and inference data. Mainly, the KNN ensemble technique is employed

Inference Strategy	Phishing Detection		Target Identification	
	$dataset2_{test}$	$dataset2_{test-unk}$	$dataset2_{test}$	$dataset2_{test-unk}$
Baseline (No KNN)	0.933	0.830	0.903	0.732
KNN TD	**0.947**	0.822	**0.915**	0.726
KNN ID	0.941	**0.842**	0.907	**0.749**

Table 4.11: Different KNN ensemble strategies pipeline $F1_{micro}$

on the binary and multiclass classification models, and the $F1_{micro}$ score of the pipeline using confidence thresholding is presented in Table 4.11.

On the one hand, we observe that utilizing KNN with training data (KNN TD) yielded noticeable $F1_{micro}$ improvements (0.014 and 0.012) for known target phishing detection and target identification. For unknown targets, however, the $F1_{micro}$ decreased by 0.008 and 0.006. This highlights the limitation of KNN with training data: weak generalization for unknown classes. Conversely, despite only minor improvements on known data, employing KKN with inference data (KNN ID) significantly improves performance for known and unknown targets, highlighting its superior generalization ability.

The underlying mechanism is further elucidated in Figure 4.2. This figure illustrates the KNN ensemble model's confidence (with weight hyperparameter w) through a contour plot accompanied by a scatter plot of inference data. Notably, the first row integrates the KNN ensemble using training data, while the second row employs it with the inference data. We observe that through reweighting prediction confidence with its high cosine similarity neighbors, the KNN ensembles regularize the decision boundary into "circular sectors", assigning data with similar angular characteristics into one class. Doing so reduces the effect of outliers and improves the model's performance, as in NLP vectors with similar angular characteristics often contain similar semantic characteristics.

Additionally, KNN ensembles also reduce the confidence in the region where few neighbors are found (as shown on the bottom right and top left of Figure 4.2), thereby more data are labelled unknown and fed into subsequent components, improving pipeline unknown target detection and generalization ability.

Furthermore, the confidence of prediction explains the difference in performance between KNN with training and inference data. Mainly, since the one-hot vector of the training label is more confident than the model's predictions, KNN TD achieves overall higher confidence than KNN ID, resulting in KNN ID favoring unknown classes, while KNN TD tends to favor known classes. Moreover, KNN ID adjusts the model's prediction based on inference data distribution (i.e., if most inference data classes are A, then most of the neighbors have class A, resulting in the model favoring class A), making the model more adaptable when encountering distribution shifts.

Despite this, KNN with inference data requires the storage of N (10,000) past prediction and hidden state during inference. This can pose challenges under limited inference data, while KNN with training data mitigate this problem. Both approaches do not affect the training duration but require O(NM) additional inference time for calculating cosine similarity and O(N) data storage for hidden state, where M is the number of inference data.

Our analysis reveals that applying KNN ensembles, particularly with inference data, can yield performance and generalization gains. Hence, the KNN ensemble on inference data is employed for the final pipeline.

4.5 Ablation Study

Model	Phishing Detection		Target Identification	
	$dataset2_{test}$	$dataset2_{test-unk}$	$dataset2_{test}$	$dataset2_{test-unk}$
Single Multiclass	0.909	0.816	0.889	0.436
Binary	0.933	0.830	-	-
+ Multiclass	0.933	0.830	0.891	0.446
+ Extractive	0.933	0.830	0.899	0.685
+ Generative	0.933	0.830	0.903	0.732
+ DS Pretraining	0.941	0.842	0.911	0.736
+ Dynamic Loss	0.945	0.843	0.912	0.736
+ KNN Ensemble	**0.956**	**0.849**	**0.916**	**0.741**

Table 4.12: Ablation study showcasing pipeline $F1_{micro}$ with and without individual components, where DS Pretraining refers to Domain-Specific Pretraining

This work performs an ablation study on the final pipeline to understand the contribution of individual components and techniques. With the confidence threshold integration method, we assess the incremental value of each component, namely Multiclass (BERT), Extractive (BERT-NER), and Generative (T5), relative to a standalone multiclass model. Subsequently, we analyze enhancements like domain-specific pretraining, dynamic loss, and the KNN (ID) ensemble. Based on the $F1_{micro}$ scores in Table 4.12, the following key observations can be made:

- **Component-Wise Performance:** Compared with a singular model, the pipeline yields better results when integrating additional components. First, we observe that binary and multiclass classification models achieve higher $F1_{micro}$ than a singular multiclass model in phishing detection and target identification. Following on, integration of the extractive model significantly improves unknown target identification, from 0.446 to 0.685 F1, while only slightly improving known target identification from 0.891 to 0.899. In comparison, the performance gain of integrating the generative model is much smaller, with merely 0.004 and 0.053 $F1_{micro}$ improvement for target identification for known and unknown targets.

- **Domain-Specific Pretraining:** This work also observes that further pretraining BERT with $dataset2$ before fine-tuning it on binary and multiclass tasks leads to an average of 0.010 and 0.006 $F1_{micro}$ improvement in phishing detection and target identification respectively. This highlights the benefit of further pretraining.

- **Dyanmic Loss:** The application of dynamic loss results in a 0.004 and 0.001 $F1_{micro}$ improvement for known and unknown phishing detection. However, dynamic loss results in 0.001 and 0 $F1_{micro}$ increase for known and unknown target identification. This highlights that while dynamic loss can improve $F1_{macro}$, its effect on $F1_{micro}$ improvement is small.

- **KNN Ensmeble:** Again, this work observes that by integrating KNN techniques using inference data, the pipeline performance on known and unknown data increased. This highlights the KNN ensemble's ability to improve performance and generalization.

This ablation study underlines that integrating additional components, domain-specific pretraining, and KNN ensemble consistently improves phishing detection and target identification performance across known and unknown targets. Notably, the integration of binary and multiclass models leads to the largest performance improvement for known targets, while the integration of extractive mode leads to the largest performance improvement for unknown targets.

Despite this, the performance improvement of additional components comes with increased inference time. Specifically, the generative model and KNN ensemble result in large computation complexity. In contrast, the single multiclass method also has the fastest inference time despite exhibiting the least performance. Given this trade-off, employing those techniques should be carefully chosen based on the task and dataset considerations.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This research was driven by the need to combat the surge in phishing emails by detecting and identifying the target of phishing emails. In a landscape where traditional countermeasures like heuristics and blacklisting have shown declining efficacy, this work's exploration of phishing detection and target identification with DL methodology is pivotal.

This work developed a robust pipeline for phishing detection and target identification by comparing and combining various state-of-the-art ML, DL, and transformers-based models. The successfully realized pipeline combines classification, extractive, and generative models to achieve high performance and strong generalization.

The merit of the pipeline, however, lies not just in its detection capability but in its ability to adapt and identify unknown phishing targets. By evaluating and improving pipeline performance against unknown phishing targets, this work ensures that not only were current phishing targets addressed, but the system also remains resilient to future, unforeseen phishing attacks.

This work attempted various techniques to improve pipeline performance further. Distinctively, the proposed KNN ensemble technique significantly improves detection performance for both known and unknown targets, while the dynamic loss enhances the model's ability to handle minority classes.

Nonetheless, there are a few limitations of the model developed in this project:

- **Hyperparameter Optimization:** Due to computational limitations, hyperparameter tuning was only performed for individual baseline models and not the combined pipeline. Additionally, the absence of cross-validation led to slight variability in the model's performance, potentially affecting the comparison between models.

- **Evaluation Metrics:** This work's evaluation predominantly relied on micro-$F1_{micro}$ scores. Incorporating more metrics, such as the ROC AUC scores or weighted accuracy, would offer a more comprehensive understanding of the pipeline's capabilities.

- **Proposed Technique Evaluation:** This work proposed and evaluated the data-based dynamic loss and KNN ensemble techniques. However, their evaluation was restricted to classification tasks on $dataset2$. This limitation necessitates further exploration across varied tasks and datasets to understand their efficacy.

- **Class Imbalance and Overfitting:** The $dataset2$ exhibited class imbalance, causing the model to favor majority classes and neglect minority ones. While this work utilized focal loss and dynamic loss to solve such problems, the model still displayed overfitting tendencies, compromising its real-world applicability. Future works could explore additional hard example mining, data sampling, and augmentation strategies to alleviate this challenge.

- **Inference Time and Memory Usage:** While our primary emphasis was on model performance, computational complexity and inference times were relatively overlooked. The final pipeline, integrating four components, demands significant memory (≈ 33.2 GB) and extended inference time (3.47 seconds). This poses challenges for practical, production-level deployment.

5.2 Future work

In light of these limitations, subsequent research efforts can focus on addressing these challenges, refining the model, and ensuring more robust real-world applicability through the following options:

- **Weight Freezing:** Given the superior performance of the pretrained NER and T5 models over fine-tuned versions, future research could explore weight freezing techniques to overcome catastrophic forgetting. A promising direction could be freezing the BERT encoder while fine-tuning only the linear head. Alternatively, gradual unfreezing and weight clipping, as indicated in [47], might also avert catastrophic forgetting and improve performance.

- **Task-Wise Ensemble:** This work trained a diverse number of models for classification, extractive, and generative tasks. A logical future direction would be to delve into task-specific ensembling techniques to combine the models and enhance the pipeline's performance.

- **Inference Time Reduction:** Despite the good performance of the final model, a reduction in inference time and memory usage is necessary to meet the production standard. Potential strategies include leveraging compact architectures like Tiny BERT [48] or employing knowledge distillation techniques [49].

- **Unsupervised Learning Ensemble:** The positive outcome from the KNN ensemble indicates the potential benefits of applying unsupervised methods on the BERT hidden state. This suggests further exploring other unsupervised learning methods, such as clustering algorithms, to improve the model's performance.

- **Phishing Type Identification:** This work only focuses on phishing detection and identifying the target company of the email, while in practice, it is also important to identify the type of phishing. To address this, future research can preprocess and reframe target identification into a multilabel classification task, whereby emails are categorized based on their phishing status, target, and attack type. Furthermore, the methods used in this work can be adapted to develop a pipeline for identifying new types of phishing attacks.

Bibliography

[1] Elmer EH Lastdrager. Achieving a consensual definition of phishing based on a systematic review of the literature. *Crime Science*, 3(1):1–10, 2014. pages 2

[2] Suzanne Widup, Alex Pinto, Dave Hylender, Gabriel Bassett, and Philippe Langlois. 2022 data breach investigations report. 2022. doi: 10.13140/RG.2.2.28833.89447. URL `https://rgdoi.net/10.13140/RG.2.2.28833.89447`. pages 2

[3] Awpg, phishing activity trends reports. 2022. doi: 10.13140/RG.2.2.28833.89447. URL `https://docs.apwg.org/reports/`. pages 2

[4] Rakesh Verma, Narasimha Shashidhar, and Nabil Hossain. Detecting phishing emails the natural language way. In *Computer Security – ESORICS 2012*, pages 824–841. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-33167-1_47. URL `https://doi.org/10.1007/978-3-642-33167-1_47`. pages 2

[5] Eder Souza Gualberto, Rafael Timóteo de Sousa Júnior, Thiago Pereira de Brito Vieira, João Paulo C. L. da Costa, and Cláudio Gottschalg-Duque. From feature engineering and topics models to enhanced prediction rates in phishing detection. *IEEE Access*, 8:76368–76385, 2020. doi: 10.1109/ACCESS.2020.2989126. URL `https://doi.org/10.1109/ACCESS.2020.2989126`. pages 2

[6] Avisha Das, Shahryar Baki, Ayman El Aassal, Rakesh Verma, and Arthur Dunbar. SoK: A comprehensive reexamination of phishing research from the security perspective. *IEEE Communications Surveys &amp Tutorials*, 22(1):671–708, 2020. doi: 10.1109/comst.2019.2957750. URL `https://doi.org/10.1109/comst.2019.2957750`. pages 2

[7] Reem Alotaibi, Isra Al-Turaiki, and Fatimah Alakeel. Mitigating email phishing attacks using convolutional neural networks. In *2020 3rd International Conference on Computer Applications Information Security (ICCAIS)*, pages 1–6, 2020. doi: 10.1109/ICCAIS48893.2020.9096821. pages 2, 11, 14, 31, 32

[8] Lukas Halgas, Ioannis Agrafiotis, and Jason R. C. Nurse. Catching the phish: Detecting phishing attacks using recurrent neural networks (rnns). In Ilsun You, editor, *Information Security Applications - 20th International Conference, WISA 2019, Jeju Island, South Korea, August 21-24, 2019, Revised Selected Papers*, volume 11897 of *Lecture Notes in Computer Science*, pages 219–

233. Springer, 2019. doi: 10.1007/978-3-030-39303-8_17. URL https://doi.org/10.1007/978-3-030-39303-8_17. pages 2, 11, 14, 17, 31, 32

[9] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. Visualphishnet: Zero-day phishing website detection by visual similarity. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1681–1698, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370899. doi: 10.1145/3372297.3417233. URL https://doi.org/10.1145/3372297.3417233. pages 2

[10] A Mishra and BB Gupta. Hybrid solution to detect and filter zero-day phishing attacks. In *Proceedings of the Second International Conference on Emerging Research in Computing, Information, Communication and Applications*, pages 373–379, 2014. pages 2

[11] Jose Nazario. Phishing corpus. URL https://monkey.org/~jose/phishing/. pages 4, 13, 14

[12] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324. pages 6

[13] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2217–2225. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/shang16.html. pages 7

[14] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *CoRR*, abs/1912.05911, 2019. URL http://arxiv.org/abs/1912.05911. pages 7

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762. pages 7, 8

[16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. pages 9

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805. pages 10, 16, 20, 21, 28

[18] Patrick Xia, Shijie Wu, and Benjamin Van Durme. Which *bert? A survey organizing contextualized encoders. *CoRR*, abs/2010.00854, 2020. URL https://arxiv.org/abs/2010.00854. pages 10

[19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019. URL http://arxiv.org/abs/1909.11942. pages 10, 16

[20] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. PEGASUS: pre-training with extracted gap-sentences for abstractive summarization. *CoRR*, abs/1912.08777, 2019. URL http://arxiv.org/abs/1912.08777. pages 10, 19

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683. pages 10, 20

[22] B. B. Gupta, Nalin A. G. Arachchilage, and Kostas E. Psannis. Defending against phishing attacks: taxonomy of methods, current issues and future directions. *Telecommunication Systems*, 67(2):247–267, May 2017. doi: 10.1007/s11235-017-0334-z. URL https://doi.org/10.1007/s11235-017-0334-z. pages 11

[23] Jiwon Hong, Taeri Kim, Jing Liu, Noseong Park, and Sang-Wook Kim. *Phishing URL Detection with Lexical Features and Blacklisted Domains*, pages 253–267. 02 2020. ISBN 978-3-030-33431-4. doi: 10.1007/978-3-030-33432-1_12. pages 11

[24] Mahmood Moghimi and Ali Yazdian Varjani. New rule-based phishing detection method. *Expert Systems with Applications*, 53:231–242, 2016. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2016.01.028. URL https://www.sciencedirect.com/science/article/pii/S0957417416000385. pages 11

[25] Lizhen Tang and Qusay H. Mahmoud. A survey of machine learning-based solutions for phishing website detection. *Machine Learning and Knowledge Extraction*, 3(3):672–694, August 2021. doi: 10.3390/make3030034. URL https://doi.org/10.3390/make3030034. pages 11

[26] Said Salloum, Tarek Gaber, Sunil Vadera, and Khaled Shaalan. A systematic literature review on phishing email detection using natural language processing techniques. *IEEE Access*, 10:65703–65727, 2022. doi: 10.1109/ACCESS.2022.3183083. pages 11

[27] Grant Ho, Asaf Cidon, Lior Gavish, Marco Schweighauser, Vern Paxson, Stefan Savage, Geoffrey M. Voelker, and David Wagner. Detecting and characterizing lateral phishing at scale. SEC'19, page 1273–1290, USA, 2019. USENIX Association. ISBN 9781939133069. pages 11

[28] Abhishek Kumar, Jyotir Moy Chatterjee, and Vicente García Díaz. A novel hybrid approach of SVM combined with NLP and probabilistic neural network

for email phishing. *International Journal of Electrical and Computer Engineering (IJECE)*, 10(1):486, February 2020. doi: 10.11591/ijece.v10i1.pp486-493. URL `https://doi.org/10.11591/ijece.v10i1.pp486-493`. pages 11

[29] Somesha M. and Alwyn R. Pais. Classification of phishing email using word embedding and machine learning techniques. *J. Cyber Secur. Mobil.*, 11(3): 279–320, 2022. doi: 10.13052/jcsm2245-1439.1131. URL `https://doi.org/10.13052/jcsm2245-1439.1131`. pages 11, 14, 17, 31, 32

[30] Lukas Halgas, Ioannis Agrafiotis, and Jason R. C. Nurse. Catching the phish: Detecting phishing attacks using recurrent neural networks (rnns). *CoRR*, abs/1908.03640, 2019. URL `http://arxiv.org/abs/1908.03640`. pages 11

[31] Nikita Benkovich, Roman Dedenok, and Dmitry Golubev. Deepquarantine for suspicious mail. *CoRR*, abs/2001.04168, 2020. URL `https://arxiv.org/abs/2001.04168`. pages 11

[32] Pranav Maneriker, Jack W. Stokes, Edir Garcia Lazo, Diana Carutasu, Farid Tajaddodianfar, and Arun Gururajan. Urltran: Improving phishing URL detection using transformers. *CoRR*, abs/2106.05256, 2021. URL `https://arxiv.org/abs/2106.05256`. pages 11, 12

[33] Chenguang Wang and Yuanyuan Chen. Tcurl: Exploring hybrid transformer and convolutional neural network on phishing url detection. *Knowledge-Based Systems*, 258:109955, 2022. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2022.109955. URL `https://www.sciencedirect.com/science/article/pii/S0950705122010486`. pages 11, 12

[34] Emilin Shyni, S. Sarju, and S. Swamynathan. A multi-classifier based prediction model for phishing emails detection using topic modelling, named entity recognition and image processing. *Circuits and Systems*, 07:2507–2520, 01 2016. doi: 10.4236/cs.2016.79217. pages 12

[35] Younghoo Lee, Joshua Saxe, and Richard E. Harang. CATBERT: context-aware tiny BERT for detecting social engineering emails. *CoRR*, abs/2010.03484, 2020. URL `https://arxiv.org/abs/2010.03484`. pages 12

[36] Gang Liu, Bite Qiu, and Liu Wenyin. Automatic detection of phishing target from phishing webpage. In *2010 20th International Conference on Pattern Recognition*, pages 4153–4156, 2010. doi: 10.1109/ICPR.2010.1010. pages 12

[37] Liu Wenyin, Gang Liu, Bite Qiu, and Xiaojun Quan. Antiphishing through phishing target discovery. *IEEE Internet Computing*, 16(2):52–61, 2012. doi: 10.1109/MIC.2011.103. pages 12

[38] Samuel Marchal, Kalle Saari, Nidhi Singh, and N. Asokan. Know your phish: Novel techniques for detecting phishing sites and their targets. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 323–333, 2016. doi: 10.1109/ICDCS.2016.10. pages 12

[39] Venkatesh Ramanathan and Harry Wechsler. Phishing detection and impersonated entity discovery using conditional random field and latent dirichlet allocation. *Computers Security*, 34:123–139, 2013. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2012.12.002. URL `https://www.sciencedirect.com/science/article/pii/S0167404812001812`. pages 12

[40] Choon Lin Tan, Kang Leng Chiew, KokSheik Wong, and San Nah Sze. Phishwho: Phishing webpage detection via identity keywords extraction and target domain name finder. *Decision Support Systems*, 88:18–27, 2016. ISSN 0167-9236. doi: https://doi.org/10.1016/j.dss.2016.05.005. URL `https://www.sciencedirect.com/science/article/pii/S0167923616300781`. pages 12

[41] Apache. Apache spamassassin. URL `https://spamassassin.apache.org/`. pages 13, 14

[42] Nguyet Quang Do, Ali Selamat, Ondrej Krejcar, Enrique Herrera-Viedma, and Hamido Fujita. Deep learning for phishing detection: Taxonomy, current challenges and future directions. *IEEE Access*, 10:36429–36463, 2022. doi: 10.1109/ACCESS.2022.3151903. pages 13

[43] Tushaar Gangavarapu, C. D. Jaidhar, and Bhabesh Chanduka. Applicability of machine learning in spam and phishing email filtering: review and approaches. *Artif. Intell. Rev.*, 53(7):5019–5081, 2020. doi: 10.1007/s10462-020-09814-9. URL `https://doi.org/10.1007/s10462-020-09814-9`. pages 14

[44] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003. URL `https://aclanthology.org/W03-0419`. pages 20

[45] Anastasios Lamproudis, Aron Henriksson, and Hercules Dalianis. Developing a clinical language model for Swedish: Continued pretraining of generic BERT with in-domain data. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 790–797, Held Online, September 2021. INCOMA Ltd. URL `https://aclanthology.org/2021.ranlp-1.90`. pages 28

[46] Noam Shazeer. GLU variants improve transformer. *CoRR*, abs/2002.05202, 2020. URL `https://arxiv.org/abs/2002.05202`. pages 29

[47] Chenghao Yang and Xuezhe Ma. Improving stability of fine-tuning pretrained language models via component-wise gradient norm clipping. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4854–4859, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.322. URL `https://aclanthology.org/2022.emnlp-main.322`. pages 43

[48] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351, 2019. URL `http://arxiv.org/abs/1909.10351`. pages 43

[49] Jianping Gou, Baosheng Yu, Stephen John Maybank, and Dacheng Tao. Knowledge distillation: A survey. *CoRR*, abs/2006.05525, 2020. URL `https://arxiv.org/abs/2006.05525`. pages 43

Appendix A

Hyperparameter Settings

Model	Parameter	Search space
DL models	Learning rate	[5e-6, 5e-4]
	Weight decay	[0, 1e-2]
	Dropout	[0.1, 0.5]
	Optimizer	AdamW, Adam, Adagrad, SGD
	Scheduler	polynomial, cosine, linear
RF	N estimators	[50, 200]
	Split criterion	“gini”, “entropy”, “log_loss”

Table A.1: Hyperparameter search range of classification, extractive, and generative models

Appendix B

Additional Evaluation Results

Model	$dataset1_{test}$		$dataset2_{test}$		$dataset2_{test-unk}$	
	Precision	Recall	Precision	Recall	Precision	Recall
ALBERT	0.967	0.949	0.921	0.916	0.820	0.835
BERT	0.963	0.958	0.918	0.927	0.839	0.820
RF	0.976	0.996	0.899	0.903	0.820	0.825
LR	0.974	0.979	0.900	0.891	0.811	0.810
CNN	0.986	0.987	0.916	0.905	0.821	0.826
RNN	0.982	0.987	0.902	0.903	0.811	0.806

Table B.1: Precision and recall values for binary classification model

Model	$dataset2_{test}$		$dataset2_{test-unk}$		$dataset3$	
	Precision	Recall	Precision	Recall	Precision	Recall
ALBERT	0.881	0.882	0.437	0.430	0.964	0.948
BERT	0.890	0.891	0.442	0.450	0.962	0.962
RF	0.865	0.862	0.425	0.418	0.928	0.943
LR	0.868	0.865	0.409	0.424	0.944	0.931
CNN	0.874	0.875	0.430	0.417	0.942	0.947
RNN	0.874	0.869	0.425	0.416	0.940	0.951

Table B.2: Micro averaged precision and recall values for multiclass models

Model	Training Strategy	$dataset2$		$dataset2_{test}$	
		Precision	Recall	Precision	Recall
PEGASUS	Default	0.157	0.171	0.166	0.167
	fine-tuned	0.377	0.375	0.215	0.214
T5	Default	0.357	0.354	0.355	0.352
	fine-tuned	0.404	0.380	0.305	0.300

Table B.3: Micro averaged precision and recall values for generative models

Model	Training Strategy	$dataset2$		$dataset2_{test}$	
		Precision	Recall	Precision	Recall
BERT	fine-tuned	0.210	0.202	0.201	0.211
BERT-NER	Default	0.334	0.343	0.334	0.344
	fine-tuned	0.322	0.325	0.331	0.316
BERT-EQA	Default	0.251	0.262	0.263	0.250
	fine-tuned	0.237	0.255	0.251	0.240

Table B.4: Micro averaged precision and recall values for extractive models