A deep learning approach for web CNAME cloaking-based tracking detection

Tianyi Ma

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

CNAME cloaking is a sophisticated technique that evades direct detection by masquerading third-party tracking as first-party requests. It became more widespread recently, with a study showing that 21.2% of the top 100,000 websites use CNAME records to alias third-party domains, and 10.7% cloak third-party analytics or tracking services. This marks a significant rise from 2019, when 7.6% of the Alexa Top 1 Million websites used CNAME cloaking. Earlier findings also prove this trend.

This dissertation addresses the challenge of detecting CNAME cloaking-based tracking on the web. Traditional tracking detection tools are increasingly inadequate against such advanced techniques, which involves obfuscated patterns that traditional methods struggle to identify. Direct measurement techniques are often insufficient due to the dynamic and evolving nature of CNAME-based tracking, where trackers manipulate DNS records to bypass standard privacy defenses. This work aims to use deep learning approach integrated into the OpenWPM platform, including Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and Convolutional LSTM networks to detect these cloaking activities.

The development of the model involved sourcing ground truth data from a variety of sources, including Majestic Million, NextDNS CNAME Cloaking Blocklist, and datasets from Fukuda Lab. The data was meticulously cleaned, integrated, and labeled into benign, malicious, and unknown classes, with the ”unknown” class introduced to manage unbalanced data and enhance classification accuracy. The models were validated by employing an 80-20 train-test split (5-fold cross validation). Evaluation metrics such as precision, recall, and F1 score were used to assess the models’ effectiveness.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

1.1.1 Introduction to Web Privacy

Privacy concerns have increased dramatically as computers and large databases are used so frequently. Even now, there are serious privacy concerns about the ability to gather extensive personal histories from various data sources. Internet usage has made these worries even more acute. For these databases [6], it has simplified the process of gathering and combining new data. User forms, transaction histories, and even individual online traveler data can all be readily tracked in today's digital world. Given the increasing use of data mining techniques, it is anticipated that privacy concerns will only worsen.

The rise of advanced web tracking technologies and growing privacy concerns have led to changes in online privacy. There are more and more ways for businesses, from small websites to large multinational corporations, to monitor user behavior, but these advancements have created serious issues. These include more sophisticated methods like browser fingerprinting and session replay, in addition to the use of pixels and tracking cookies. Tracking cookies are small text files that are stored on a user's device and can be managed or deleted by the user. In most cases, these files require the user's consent because of regulatory requirements. Conversely, passive browser fingerprinting, which is less obvious but possibly less persistent, gathers comprehensive data about a user's device and browser configuration without storing data on the device or needing user consent. Due to these developments, stakeholders are debating more and more how to reconcile advancing the useful use of data with safeguarding individuals' right to

privacy [35] [11].

1.1.2 Importance of the Problem

The importance of maintaining online privacy cannot be overemphasized. The public wants more control and transparency over personal information, as stories about data breaches and improper use of data are always in the news. Strong privacy protec2.tions are important, as evidenced by regulatory measures such as the California Consumer Privacy Act (CCPA) in the United States and the General Data Protection Regulation (GDPR) in the European Union [17]. Despite these efforts, many current tools (openwpm [16] among them) have found it difficult to keep up with the rapid evolution and increasing complexity of tracking technologies. The goal of this project is to protect people's privacy rights in a world where new tracking technologies are constantly emerging and posing threats to them, not just to improve existing ones.

1.1.3 Aims and Objectives

The primary aim of this project is to add new feature for OpenWPM by integrating deep learning techniques to detect CNAME cloaking-based tracking on the web. This involves developing and implementing neural network models to accurately identify and categorize CNAME cloaking instances in network traffic data. This project is divided into four parts:

- **Data Collection and Preparation:** Use comprehensive datasets to classify and label data, then extract the feature for each class as input to feed into deep learning models.
- **Model Development and Evaluation:** Develop Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) Networks, and Convolutional Long Short-Term Memory (ConvLSTM). Then assess the performance of the three models using metrics include accuracy, precision, recall, and F1 score.
- **Integration with OpenWPM:** Choose the best-performing model and integrate it into the OpenWPM framework to enhance its capability in detecting CNAME cloaking.

1.1.4 Impact and Beneficiaries

The successful completion of this project will significantly enhance the capabilities of OpenWPM. By integrating deep learning techniques, the OpenWPM with new feature will be able to detect CNAME cloaking-based tracking with high accuracy and efficiency. Even if the performance of deep learning based CNAME cloaking-based tracking detection maybe not good as some traditional machine learning methods like RandomForest, ExtraTrees, and GradientBoosting [19], this project will still contribute to a better understanding of web tracking mechanisms and the development of more effective privacy protection strategies. Deep learning models can handle large-scale datasets more effectively than many traditional machine learning methods. As data increases, deep learning models often improve in performance so the prospect of this area is promising. And it will benefit lots of people, including:

1. Researchers and Academics:
    - Gain access to a more advanced tool for studying web tracking and privacy.
    - Utilize the enhanced OpenWPM for conducting in-depth research on CNAME cloaking.
    - Benefit from improved and integrated datasets.
2. Privacy Advocates and Organizations:
    - Use the enhanced OpenWPM to advocate for better privacy protections.
    - Use the tool to identify hidden tracking activities, promoting transparency and accountability.
3. Regulatory Bodies and Policymakers:
    - Obtain valuable information into tracking practices, informing policy and regulatory decisions.
    - Use the data and findings to enforce privacy regulations and standards more effectively.
    - Use the data and findings to enforce privacy regulations and standards more effectively.
4. Web Users:

- Benefit indirectly from increased privacy protections and reduced exposure to tracking.
- Experience enhanced privacy and security when browsing the web.

5. Developers and Engineers:

   - Integrate the enhanced OpenWPM into privacy-focused applications and services.
   - Utilize the tool to identify and mitigate tracking in their own web applications, improving user privacy.

By achieving these aims and objectives, the project will make a significant contribution to the field of web privacy and tracking detection, benefiting a wide range of stakeholders and promote the development in privacy protection technologies.

1.2 Literature Review

The purpose of this literature review is to examine existing research on the detection of CNAME cloaking using deep learning techniques. This review focuses on study that have utilized various neural network architectures, including Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), and Convolutional LSTM Network(Convolutional LSTM Network), for web tracking and privacy protection. The scope includes peer-reviewed journal articles, conference papers, and significant industry reports published in the last decade. This review starts with the introduction of the tool that this project is using and general web tracking techniques, followed by methods for detecting CNAME cloaking, and finally, the application of deep learning techniques in this context.

1.2.1 OpenWPM

OpenWPM is a flexible, modular web privacy measurement platform which can be used for any experiment that conforms to a broad framework. It automatically recovers from browser errors, allows parallel speed and scale, and replicates actual user activity to improve the legitimacy of its measurements. It is a powerful tool in the web privacy measurement field because of its scripted command-line API, which enables researchers to perform concurrent measurements [16].

It addresses three key systems challenges in web privacy measurement, building on past work and avoiding previous pitfalls. First, it achieves scale through parallelism and robustness by utilizing isolated measurement processes similar to FPDetective's platform [1] (there will be more details in next section), while still supporting stateful measurements. This design enables OpenWPM to scale to 1 million sites without needing a stripped-down browser. Second, OpenWPM provides comprehensive instrumentation by expanding on the rich browser extension instrumentation of FourthParty [28], eliminating the need for researchers to write their own automation code. Third, it reduces duplication of work by offering a modular architecture that facilitates code reuse between studie to facilitate a wide range of online privacy measurement (WPM) experimentss [15].

The platform capabilities include:

- **Modularity and Compatibility:** Because OpenWPM's interface can be compatible with existing tools and supports a wide range of instrumentation methods, researchers can easily customize the platform to meet their individual needs without big reconfiguration.
- **Scalability:** The platform's ability to efficiently manage large-scale distributed crawlers is a critical component for research that requires the collection of large amounts of data in different geographic locations or network environments.
- **Repeatability:** OpenWPM is a key component of scientific rigor because it allows other researchers to easily replicate experiments, standardize data records, and simplify configuration management. This promotes transparency and validity of results.

1.2.2 Alternatives to OpenWPM

Several alternatives to OpenWPM exist for web privacy measurement and tracking detection. Some of these include:

- **Panopticlick:** Developed by the Electronic Frontier Foundation (EFF), Panopticlick measures how unique and trackable a browser's configuration is. However, it is less comprehensive in tracking overall web privacy practices compared to OpenWPM [24].

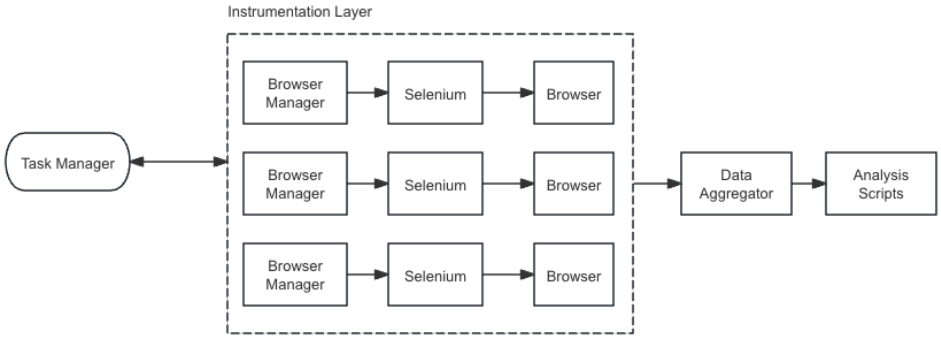

Figure 1.1: Overview of OpenWPM

- **uBlock Origin:** A widely-used browser extension for content filtering and ad blocking. However, it is more like a blocking tool and does not provide comprehensive measurement and analysis capabilities [29].
- **Ghostery:** A privacy extension that detects and blocks tracking technologies on websites. Similar to uBlock Origin, it is more focused on blocking rather than detailed measurement and analysis [27].
- **Privacy Badger:** Another EFF tool that blocks trackers based on their behavior rather than a pre-defined list. However, it is still a blocking tool, not focus on a comprehensive tracking measurement [27].

In addition to the tools mentioned above, FPDetective is the most similar platform to OpenWPM [15]. It is a research platform designed for the detection and analysis of web-based tracking techniques, specifically browser fingerprinting. It employs a hybrid infrastructure utilizing PhantomJS and Chromium, enabling it to automate and instrument web browsers for the purpose of monitoring and analyzing tracking behaviors [1]. FPDetective and OpenWPM are both designed for web tracking and privacy research, but they differ significantIy in their capabilities and architecture. FPDetective focuses on detecting browser fingerprinting using a hybrid PhantomJS and Chromium infrastructure, with native browser code and a proxy for instrumentation, supporting only stateless measurements. In contrast, OpenWPM offers a more versatile framework supporting both stateful and stateless measurements, with generic instrumentation for a broader range of privacy studies without requiring native browser code, facilitate easier browser updates. Additionally, OpenWPM's high-level command-based architecture enhances maintainability by making many different modules, allowing for command

reuse across different studies to make it more adaptable compared to FPDetective [15].

1.2.3 CNAME Tracking

Some trackers employ a complex method called CNAME tracking to get around browser privacy settings. CNAME tracking is the practice of passing off requests from third parties as first-party requests by means of DNS records, more especially the CNAME record. Typically, a JavaScript file from the tracker is added when third parties track a website. The tracker domain receives (cross-site) requests from this file and reports analytics information. When employing CNAME-based tracking, the same steps are taken, but scripts are added from a subdomain of the website and analytics data is sent there instead. A tracking script fromtrack.example.com, for instance, would be incorporated into the website example.com, giving the two websites an almost identical appearance. Typically, the subdomain contains the CNAME record for a tracker server [12] [4].

A CNAME chain is the set of canonical names that links first-party subdomains to the IP address before the final resolution point. Analyzing CNAME chains reveals four distinct categories based on the domain's position in the chain: CDN (content delivery network) domains, cloud and other domains (such as those used for firewall services or cloud storage), first-party domains (where the domain of the HTTP request matches or shares the same IP address with the domain of the final node), and tracking domains (where the domains are used to track user activity, a practice known as CNAME cloaking based on the CNAME steganography for tracking) [9]. Every variation reflects a range of CNAME applications and outcomes when managing user data and network traffic [7] [10].

CNAME cloaking is prevalent across a significant portion of the web. One study analyzed the top 100,000 most popular websites and found that 21.2% use CNAME records to alias at least one third-party domain, with approximately 10.7% cloaking third-party analytics or tracking services. In 2019, a study reported that 7.6% of websites in the Alexa Top 1 Million used CNAME cloaking [4]. This represents a significant increase in just a few years. Earlier research stated a 30% rise in CNAME cloaking over three years [21]. This new data underscores the expanding use of CNAME cloaking and its increasing impact on user privacy and security, particularly concerning the leakage of session cookies.

1.2.4 Deep Learning in Web Tracking

Deep learning techniques provide promising solutions for enhancing the detection of sophisticated tracking methods. These techniques can automatically identify complex patterns in data, making them suitable for web tracking detection. The models below are typically more adaptable to new types of data or subtle changes in patterns. This adaptability can lead to better long-term performance as tracking techniques keep evolving [19].

- **Convolutional Neural Networks (CNNs):** CNNs are primarily used in image processing but have been adapted for various other tasks, including time-series analysis. They can automatically and adaptively learn spatial levels of feature through convolutional operations [23]. For web tracking detection, CNNs can be applied to identify patterns in sequences of network traffic data by treating them as spatia1 data. This approach enables the detection of complex patterns in request/response sequences, which are signs of CNAME cloaking [22]. Features such as URL entropy, subdomain entropy, and the length of subdomains have been utilized in these contexts to capture the structural complexities and randomness associated with tracking URLs [33].

- **Long Short-Term Memory (LSTM) Networks:** LSTM networks, a type of recurrent neural network (RNN), are designed to learn and remember long-term dependencies in sequential data. They are particularly effective for time-series analysis and sequence prediction tasks [18]. In the context of web tracking, LSTMs can model the sequence of network request and response and capture temporal or unusual patterns that may indicate cloaking behavior. Studies have shown that LSTMs can effectively detect unusual activities in network traffic, making them suitable for this task [36]. Features such as HTTP methods, content types, and sequential request patterns are very important for capturing these temporal dependencies [33] [25].

- **Convolutional Long Short-Term Memory (ConvLSTM) networks** The ConvLSTM model is designed to detect CNAME cloaking-based tracking by using both convolutional and LSTM layers to handle data with complex features. The model begins with the ConvLSTMCell layers, which combine convolutional operations with LSTM units to capture spatial features and temporal dependencies from the input sequences. It can be used in this project because the detection of CNAME

cloaking involves analyzing sequences of web requests, which are inherently temporal and spatial. Each of them can be represented by a set of features extracted from URLs, subdomains, and other metadata. Those data is both sequential (capturing the order of requests) and spatial (representing structural patterns within the URLs and subdomains). And convLSTM networks integrate the convolutional operations of CNNs within the LSTM architecture. This allows the model to capture both spatial and temporal dependencies simultaneously. By applying convolutions, ConvLSTM can extract more complex and abstract features from the input data before feeding them into the LSTM layers [34]. The results in a richer representation of the data, improving the model's ability to detect subtle patterns of CNAME cloaking [5]. Features such as URL entropy, subdomain entropy, subdomain length, number of subdomain prefix, and dictionary check have been effectively used to enhance the detection capability of ConvLSTM networks [25].

Chapter 2

Methodology

2.1 Before Starting

The decision to build deep learning models rather than a tool that directly detects CNAME cloaking is driven by the inherent complexities of CNAME cloaking. Traditional direct detection methods like DNS query analysis and blacklist matching rely heavily on predefined rules and patterns, which are often effective only against threats that have appeared before. However, CNAME cloaking is a sophisticated method where third-party trackers disguise themselves as first-party subdomains, making it difficult for static detection tools to recognize them.

CNAME cloaking exploits DNS configurations to mask the true identity of tracking services, thereby evading standard detection tools that depend on easily identifiable patterns, such as specific domain names or IP addresses. As tracking methods evolve, these direct detection tools would require constant updates and manual intervention to remain effective. This approach is not scalable or efficient, especially given the rapid adoption and adaptation of cloaking techniques by trackers.

After knowing limitations of traditional tools in detecting CNAME cloaking, I intuitively believed that deep learning could be used to for detection, though I wasn't initially certain which approach would be better. CNAME cloaking, by its nature, involves masking third-party trackers as first-party subdomains, making it difficult for static tools to identify these threats reliably. And personally, I think OpenWPM is a robust tool widely used for web privacy measurement which is supported by an active community. I want to add a new feature to it as contribution to this strong community.

2.2 Overall Design

Figure 2.1 provides an overview of the entire methodology used in this research, detailing the flow of data collection, processing, and model evaluation.

Firstly, I collect data from Majestic Million, NextDNS CNAME Cloaking Blocklist, and Fukuda Lab. After that, the process begins with the Dynamic Detection component, which utilizes data from the Majestic Million—a comprehensive list of the top million websites. This data is gathered dynamically using OpenWPM, a web privacy measurement framework that crawls websites, capturing all HTTP/HTTPS requests and their responses. OpenWPM is the tool I aim to enhance by adding new feature. After data collection, the next step involves cross-referencing the gathered data with CNAME Cloaking Blacklists from sources like Fukuda Lab and NextDNS, which helps in pinpointing potential tracking domains.

Following this, Data Cleaning and Data Integration steps are performed to ensure that the information is both accurate and consistent. During the Data Labeling stage, the cleaned data is categorized into three groups: benign, malicious, and unknown. The inclusion of the ”unknown” class is important for managing unbalanced data and improving the accuracy of classifying malicious sites. This class allows the model to account for cases where there isn't enough information to confidently categorize a site as either benign or malicious. By incorporating with this class, the model perform better in avoiding incorrect predictions especially when facing with ambiguous data. This strategy will enhances the model's focus on accurately identifying malicious sites, thereby increasing the overall reliability of the detection process [30].

The labeled data is subsequently used to train three different machine learning models: CNNs (Convolutional Neural Networks), LSTM (Long Short-Term Memory) networks, and ConvLSTM (Convolutional LSTM) networks. These models are specifically trained to detect and classify tracking behaviors based on the features extracted from the data. A rigorous evaluation process is then carried out to ensure the effectiveness of the trained models in detecting CNAME cloaking. The labeled data is split into training and testing sets, typically with an 80-20 split, where 80% of the data is used for training the models and 20% is used for testing. This split allows for a fair assessment of the models' performance on unseen data. The effectiveness of each model is evaluated based on three key metrics:

- **Precision:** The ratio of true positive predictions to the total positive predictions.
- **Recall:** The ratio of true positive predictions to the total actual positives.

- **F1-Score:** The harmonic mean of precision and recall, providing a balance between the two.

After evaluation, the best-performing one will be integrated with OpenWPM as a new feature.

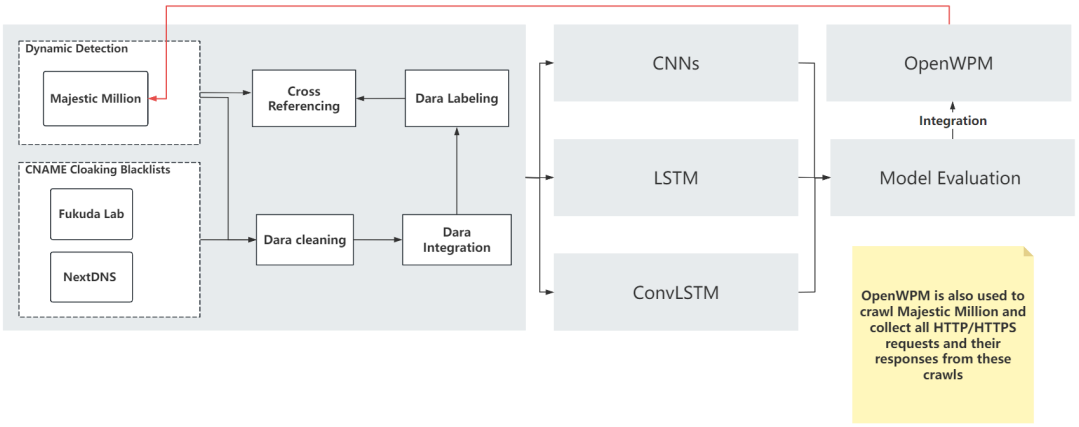

Figure 2.1: Overall Design

2.3 Data Preparation

2.3.1 Datasets

At first, I collected data from the following sources:

1. **Top 500 Websites(Abandoned):** A curated list of the most popular websites to ensure a broad and representative sample. This list was obtained from Moz's Top 500 Websites. It provides a comprehensive list of the most popular websites globally. This dataset is valuable for initial testing and validation because it includes high-traffic websites that are more likely to employ sophisticated tracking mechanisms, including CNAME cloaking [12]. The list includes a variety of websites across different categories, enhancing the generalizability of the model. It doesn't inherently contain information about CNAME tracking but can be used to cross-reference the prevalence of CNAME cloaking on popular sites. However, a small sample size can not capture the diverse characteristics and patterns necessary for detecting CNAME cloaking across the web. This can lead to a model that is overfitted to the specific websites in the dataset and not generalize well to other sites. Furthermore, It primarily includes highly popular

and reputable websites. This lack of diversity might mean the dataset does not include enough examples of less popular or more niche sites where CNAME cloaking might be more prevalent. This can lead to biased results and reduced detection accuracy.

2. **Majestic Million:** It provides a list of the top 1 million websites based on their own metrics [26]. A larger dataset can capture more diverse pattern and improve the ability of the model to generalize across different websites. This leads to a more robust and accurate model for detecting CNAME cloaking. It alse includes a wide range of websites, from highly popular to less known sites. This diversity helps in identifying CNAME cloaking in various environments. But the limitation of crawling a large number of websites is time-consuming. It requires significant computational resources and time. I am using VirtualBox 6.1 with Ubuntu 20.04 LTS Operating System, equipped with Intel Core i7-10400K, 32GB DDR4, and Samsung 980 Pro SSD, one crawling session took approximately 7 hours on my machine.

3. **Fukuda Lab:** This repository contains detailed CSV files that catalog sites and subdomains known to utilize CNAME cloaking-based tracking. This data is essential for training the LSTM model to recognize patterns specific to CNAME cloaking [20].

    - Site_CNAME-cloaking-based-tracking.csv: Contains sites known to use CNAME cloaking-based tracking.
    - Subdomain_CNAME-cloaking-based-tracking.csv: Lists subdomains and associated tracking providers using CNAME cloaking. This file structure is quite complicated. It has five colums include number, Site, Subdomain, CNAME, and Tracking Provider. **Site** like 'nvidia.ru', 'equifax.ca', 'newyorklife.com' are main domain of the site that contains the subdomain in question. **Subdomain** is being used for tracking purposes like 'smetrics.nvidia.com', 'sawap.equifax.com', 'st.newyorklife.com'. **CNAME** is the Canonical Name (CNAME) record that the subdomain points to which often associated with a third-party tracker such as 'nvidia.com.ssl.sc.omtrdc.net.', 'equifax.com.ssl.sc.omtrdc.net.', 'newyorklife.com.ssl.d1.sc.omtrdc.net.'. **Tracking Provider** is company or service provider that is responsible for tracking. This is usually the entity that the CNAME record points to such as

'Adobe', 'segment.com', 'Intent Media'.

4. **NextDNS CNAME Cloaking Blocklist:** This list contains known CNAME cloaking domains and is available from the NextDNS GitHub repository. It is used as a reference to verify the results from the LSTM model then increasing the detection accuracy [32].

5. **Complementary Data:** Blacklists shown above provides a static list of known tracking domains may not cover new or evolving CNAME cloaking techniques. Therefore, I crawling 3620 websites and performing DNS queries dynamically detects current CNAME cloaking activities. There are two main tasks of methodology for identifying CNAME cloaking-based tracking. Firstly, I perform DNS queries to obtain CNAME records for all subdomains of each site. After that, I apply wildcard matching to the resolved CNAME records using known tracking filter lists (e.g., Easy Privacy List, AdGuard Tracking Protection Filter) to identify CNAME cloaking-based tracking. Finally, I inspect each CNAME node in CNAME chains using the combined customized filter list. I identify 97 domains as a potential tracker from filter list in this process. I then organize these CNAME chains by domain and personally check each one of them to prevent false positives. First, we confirm them by watching the actions that store a distinct cookie under the visited domain name in the browser. Additionally, we collect data on these websites in order to determine if they are associated with any tracking service. By applying this analysis, I label 59 domains. Because all 3620 websites are inspected for potential tracking domains, I can use this as the base number of domains. Table 2.1 shows the evaluation metrics and results for CNAME cloaking detection based on the data collected. The ground truth dataset against which the evaluation is conducted includes the NextDNS CNAME Cloaking Blocklist and the dataset from Fukuda Lab, which provide well-curated lists of known CNAME cloaking instances. The following approach was used to clarify evaluation metrics under this context:

    - **True Positives (TP):** Instances that the evaluated dataset correctly identifies a site as using CNAME cloaking based on a match with the ground truth data.
    - **False Positives (FP):** Instances that the evaluated dataset incorrectly identifies a site as using CNAME cloaking, but it is not present in the ground

truth dataset.

- **True Negatives (TN):** Instances that the evaluated dataset correctly identifies a site as not using CNAME cloaking, same as the ground truth.
- **False Negatives (FN):** On the contrary to FP, it refer to instances that the evaluated dataset fails to identify a site as using CNAME cloaking, even though it is present in the ground truth dataset.

Table 2.1: Evaluation Metrics for CNAME Cloaking Detection

Year	**(TP)**	**(FP)**	**(TN)**	**(FN)**
2024	59	38	3405	118

To obtain raw data necessary for DNS queries and subsequent analysis, I use OpenWPM to crawl Majestic Million and collect all HTTP/HTTPS requests and their responses from these crawls. The data crawled on Alexa Top 100K in 2018 and 2020 [8] was combined and shown in Table 2.2. However, Amazon used to provide the Alexa Top 1 Million sites dataset, but it was discontinued in 2022.

The dataset under evaluation comes from the Majestic Million, which is a comprehensive list of the top 1 million websites. This dataset was used to collect HTTP/HTTPS requests and responses, and the analysis aimed to detect CNAME cloaking across these sites.

Table 2.2: Summary of data: 2,010 sites in 2018, 3,524 sites in 2020, and 3,620 sites in 2024.

Metrics	**2018**	**2020**	**2024**
3rd party requests	185,896 (62.2%)	276,686 (63.8%)	300,456 (64.3%)
1st party requests (domain)	60,278 (20.2%)	120,689 (27.8%)	130,567 (27.9%)
1st party requests (subdomain)	52,476 (17.6%)	36,399 (8.4%)	36,450 (7.8%)
Total requests	**298,649 (100%)**	**433,774 (100%)**	**467,473 (100%)**
Total sites	**2,010**	**3,524**	**3,620**

These files are merged into a comprehensive dataset for training the deep learning model.

The Fukuda Labs dataset provides well-curated examples of CNAME steganography, providing a reliable basis for training LSTM models. Finally, the NextDNS block list serves as an reliable reference for validation, ensuring the accuracy and utility of the model. Integrating these datasets allows for a comprehensive and detailed approach to detect CNAME steganography-based tracking, utilizing real-world data and expert knowledge to obtain robust and reliable results.

2.3.2 Data Cleaning, Integration, and Labeling

This section outlines the detailed steps involved in cleaning, integrating, and labeling the datasets for detecting CNAME cloaking-based tracking. The datasets include top sites data, subdomain CNAME cloaking data, site CNAME cloaking data, and the NextDNS blocklist.

2.3.2.1 Data Cleaning

This process is designed to eliminate inconsistencies, missing values, and duplicates within the datasets, thereby enhancing their reliability and preparing them for further analysis and integration. I begin by loading the datasets into data frames using pandas, which provides an efficient way to manipulate and clean the data. To address missing value, I either fill them with appropriate data or remove the affected rows. Ensuring consistency across the datasets, I standardize all textual data by converting domain and subdomain names to lowercase and removing any leading or trailing whitespace. Finally, I eliminate duplicate entries to ensure that each record in the dataset is unique, which help reduce bias and improves the accuracy of subsequent analyses.

2.3.2.2 Data Integration

Data integration involves combining data from the various sources to create a unified dataset which will be used for model training and evaluation. merge 'subdomain_df' with 'site_df' to include only relevant subdomains for the sites listed in 'site_df'. To identify CNAME cloaking, I add a column to the combined dataset indicating if the CNAME is in the NextDNS blocklist.

2.3.2.3 Data Labeling

Firstly, to identify CNAME cloaking, I add columns to the combined dataset indicating if the CNAME is in the NextDNS blocklist and the provided CNAME cloaking tracking

lists. After that, I label instances of CNAME cloaking based on the presence in the NextDNS blocklist and the CNAME cloaking tracking lists. This involves creating a new binary column in the dataset to indicate whether a particular CNAME is associated with cloaking ('0' for Benign and '1' for malicious). However, I found that if I do Data Labeling in this way, most of websites will be labeled as benign which caused by unbalanced data(approximately 0.58% among 100k websites [7]) and the limitation of the CNAME cloaking detection technologies. So the 'Unknown' class is introduced to improve the robustness of the model by accounting for data points that exhibit characteristics neither clearly indicative of benign behavior nor definitive of CNAME cloaking-based tracking. I establish the criteria below:

1. **Malicious:** CNAMEs present in known tracking blocklists.
2. **Benign:** CNAMEs not present in any known tracking blocklists and exhibiting typical non-tracking behaviors.
3. **Unknown:** CNAMEs not present in known blocklists but show unusual patterns that do not fit clearly into benign or malicious categories. And instances where CNAME records show moderate entropy or length characteristics. Cases where DNS data is incomplete or inconsistent, leading to uncertainty.

In the provided script, the process of labeling "Unknown" instances in the dataset involves a comprehensive feature extraction and evaluation approach. Initially, entropy features are computed for the URL, subdomain, and subdomain prefix using an entropy calculation function,

$$H(s) = -\sum_{c \in \text{dict}(s)} p(c) \log p(c)$$

which helps measure the randomness and potential dynamic generation of these components. The lengths of the subdomain and its prefix are calculated as

$$\text{len\_sub} = |\text{subdomain}| \tag{2.1}$$

$$\text{len\_prefix\_sub} = |\text{subdomain\_prefix}| \tag{2.2}$$

which show complexity and structure of the subdomains. The script also checks whether the subdomain prefix is part of a known blacklist or an English dictionary, indicating potential tracking behavior or legitimate usage. Additionally, content types and HTTP methods are assigned manually for illustration purposes, categorizing requests as either 'image', 'script', 'GET', or 'POST'. The "Unknown" class is defined using specific

criteria that evaluate these features: URLs, subdomains, and subdomain prefixes with moderate entropy values, subdomains of certain lengths, specific range of prefix counts, and content types and methods outside common category content_type, POST or GET method. Moreover, subdomain prefixes not found in blacklists or dictionaries contribute to the ”Unknown” label.

This labeling process ensures that instances exhibiting characteristics neither clearly indicative of benign behavior nor definitive of CNAME cloaking-based tracking are classified as ”Unknown,” enhancing the robustness of the model by accounting for ambiguous data points.

$$0.5 < \text{entropy\_url} < 0.8$$
$$0.5 < \text{entropy\_sub} < 0.8$$
$$0.5 < \text{entropy\_prefix\_sub} < 0.8$$
$$10 < \text{len\_sub} < 20$$
$$5 < \text{len\_prefix\_sub} < 10$$
$$2 \leq \text{num\_prefix\_sub} \leq 3$$
$$\text{content\_type} \notin \{\text{image, script}\}$$
$$\text{method} \notin \{\text{GET}, \text{POST}\}$$
$$\text{prefix\_sub\_blacklist} = 0$$
$$\text{is\_sub\_dic} = 0$$

Tracking Detection includes CNAME Lookup and CNAME Cloaking-Based Tracking Detection with Blocklists.

1. **CNAME Lookup:** First of all, I separate the generic Top-Level Domain (gTLD) and country-code top-level domain (ccTLD) from the visited website for all HTTP requests using the Public Suffix List [**?**]. I only keep the subdomain of an HTTP request if it is not null and its second-level domain is the same with the visited website domain. After that, I look up and check CNAME records for each subdomain. I then resolve each CNAME answer set by DNS, saving all nodes in the CNAME chain to analyze the CNAME cloaking behind first-party requests. By doing this, In 2020 45.73% of HTTP requests are first-party requests by this method only keep 11.76% of HTTP requests that contain first-party CNAME. For longitudinal data, by checking historical forward DNS (FDNS) datasets provided by Rapid7 [31], the coverage of the FDNS data is not perfect (Rapid7 is A cybersecurity company known for its data and analytics solutions and provide various datasets). It misses 10% of CNAMEs in 2018 and 30% in 2016 and

2017. Therefore, I follow a structured approach to supplement the existing FDNS datasets provided by Rapid7. DNSDB [13] is a comprehensive DNS intelligence database that provides historical and current DNS resolution data, which can help fill the gaps in FDNS data. After obtaining access to DNSDB through a subscription, I performed DNSDB queries for subdomains that have missing CNAME records in the FDNS data. However, after trying two days I still can not successfully these queries because some network connectivity problems with the DNSDB API inside the virtual machine. After all the instances are labeled as 'malicious' or 'benign', classifying certain benign or malicious labels to '2' (unknown) based on the criteria.

2. **CNAME Cloaking-Based Tracking Detection with Blocklists:** To detect CNAME cloaking-based tracking, I use an approach based on wildcard matching of tracking filter lists. Firstly, I discard CNAME-related subdomains categorized as first-party type. After that, I classify a CNAME chain as first-party if the domain of the final node in this chain the same as the domain of the considered HTTP request or if the IP addresses of both the final node and the second-level domain are the same. I then detect CNAME cloaking-based tracking inside the remaining subdomains by applying wildcard matching based on well-known tracking filter lists: EasyPrivacy list [14] and AdGuard tracking protection filter [2]. EasyPrivacy list consists of nine sublists, and the AdGuard tracking filter list consists of eleven sublists. These lists contain many rules that remove all forms of tracking, including Web bugs, tracking scripts, and information collectors. The processes are shown below:

    - Categorization: The aims of categorization is to discard CNAME-related subdomains categorized as first-party type. A CNAME chain is classified as first-party if The domain of the final node in it is the same as the domain of the HTTP request or the domain of the final node in it is the same as the domain of the HTTP request.
    - Detection: It involves detecting CNAME cloaking-based tracking inside the remaining subdomains using wildcard matching of tracking filter lists. I apply Wildcard Matching using well-known tracking filter lists (EasyPrivacy list and AdGuard tracking protection filter) then create regular expressions from tracking domains to match CNAME records.
    - Inspection: Inspecting individual CNAME nodes in all CNAME chains

using the customized filter list and manually validate potential trackers. Classify CNAME chains as potential trackers if any node is flagged by the customized filter list.

2.3.2.4 Using Majestic Million Dataset for Cross-referencing:

The Majestic Million Dataset contains a list of popular websites, but it does not directly indicate whether these sites use CNAME cloaking-based tracking. However, this data can still be useful in understanding how prevalent CNAME cloaking is among popular websites. I merge the labeled combined dataset with the Majestic Million Dataset on the Root Domain column (or equivalent). After that, calculating the percentage of popular sites that are using CNAME cloaking based on the merged dataset. This procedure essentially involves comparing the popular websites listed in the Majestic Million Dataset file with the labeled dataset that identifies CNAME cloaking-based tracking. This comparison helps to identify which popular websites are using CNAME cloaking and which not.

2.3.3 Feature Extraction

In the context of using deep learning methods for detecting CNAME cloaking-based tracking, I extract the following features related to sites and requests [10] [33]:

1. Entropy Features:
    - entropy_url: Measures the randomness in the URL, which can indicate dynamically generated tracking URLs.
    - entropy_sub: Indicates the randomness of the subdomain, often higher in tracking subdomains.
    - entropy_prefix_sub: Reflects the randomness in the subdomain prefix, which can be a sign of tracking domains.
2. Subdomain Features:
    - len_sub: The length of the subdomain can provide information into tracking domains, which often have longer and more complex subdomains.
    - len_prefix_sub: The length of the subdomain prefix, similar to the length of the subdomain, can help identify tracking-related patterns.

- num_prefix_sub: The number of subdomain prefixes can indicate multi-level subdomains commonly used in tracking.

3. Content and Request Features:

   - content_type: The type of content being requested (e.g., image, script) is crucial as certain content types are more indicative of tracking.
   - method: The HTTP method used (e.g., GET, POST) can help distinguish between regular web requests and tracking-related requests.

4. Blacklist and Dictionary Features:

   - prefix_sub_blacklist: Whether the subdomain prefix is in a tracking filter list can directly indicate known tracking domains.
   - is_sub_dic: If the subdomain prefix is a word in the English dictionary; non-dictionary words can sometimes be indicative of tracking domains.

5. Lower Priority Features:

   - Length of URL: While the length of the URL can be indicative of complexity, it might be less directly related to tracking behavior compared to other features.
   - XHR Indicator: Indicates if the request is an XMLHttpRequest (XHR). While useful, it may not be as directly indicative of CNAME cloaking as other features, since many legitimate web applications use XHR extensively.

2.4 Model Development

2.4.1 Convolutional Neural Networks (CNNs)

There are two different CNN architecture designed for the training process. I use ReLu as the activation function for all the CNNs although there are other activation functions like Sigmoid and Tanh. It is quite simple to make the gradient to vanish in a Sigmoid. The issue of gradient vanishing will become more problematic in saturated neurons. Besides, the Sigmoid function is not zero-centered. Although the Tanh activation function solves the problem of center symmetry of origin (ranging from -1 to 1), the problem of slow parameter updating cannot be solved effectively. ReLu doesn't involve

expensive operation so it learns faster and it aviods the gradient vanishing problem. Almost all deep networks use ReLu nowadays so I think there is no needs to change activation function. Unless bias becomes so small that the value of the input activation function is always negative, then the gradient through this point in the back propagation process is always 0, and the corresponding weight and bias parameters cannot be updated this time. I tried Leaky ReLu but the perfermance was quite similar with ReLu. Therefore, I choose to use ReLu to reduce the calculation. For Figures 2.2 and Figure 2.3, I used the PlotNeuralNet tool to draw the CNN structures myself based on my design. It is a popular framework that allow user to virtulize neural network architectures [3]. The script for generating virtulized CNN structure is also been added to the packed source code.

2.4.1.1 CNN_1

This CNN has six convolution layers that extract features from the input data, followed by linear unit (ReLU) activation function. And three 2D average pooling layers that reduce the spatial resolution of the data, resulting in a reduction of the number of parameters and computation in the subsequent layers. The model also includes 3 fully-connected (linear) layers that map the extracted features to the output classes. The fully-connected layers are followed by ReLU and the final layer has 3 output units, corresponding to the 3 classes in the dataset. The architecture of CNN_1 is shown in Figure 2.2.

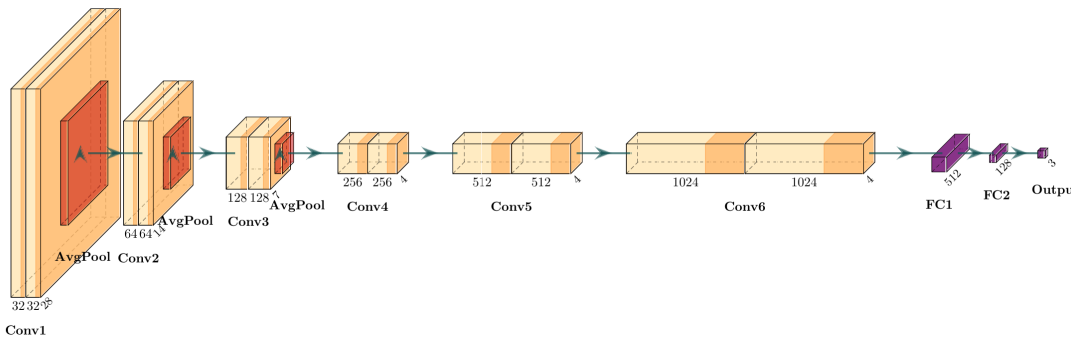

Figure 2.2: Architecture of CNN_1

2.4.1.2 CNN_2

CNN_2 is derived from CNN_1, which added another two convolution layers followed by linear unit (ReLU) activation function and one more pooling layer to have a deeper

architecture with more parameters. However, adding more layers to the network can allow it to learn more complex features from the input data, but it can also increase the risk of overfitting if the network is not properly regularized. And because taking the mean value is easy to dilute the effect of characteristic degree, I replace all of the average pooling layers with max pooling layers. The architecture of CNN_2 is shown in Figure 2.3.

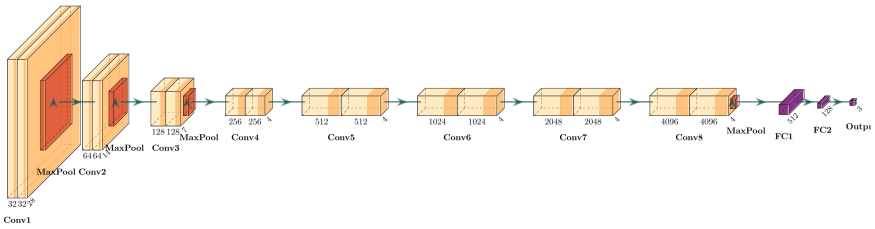

Figure 2.3: Architecture of CNN_2

2.4.2 Long Short-Term Memory (LSTM) Networks

The modified LSTM network is meticulously crafted to detect CNAME cloaking-based tracking, a sophisticated method of tracking that often evades traditional detection techniques. The network's design starts with the LstmParam class, initializing the necessary weight matrices and biases for the LSTM cell and an additional output layer designed for the classification task, crucial for differentiating between benign, malicious, and unknown tracking patterns. The LstmState class maintains the dynamic state of the LSTM, including gate activations and cell/hidden states, which are essential for capturing the temporal dependencies inherent in web tracking sequences. Each LSTM node, instantiated by the LstmNode class, processes input feature sequences—such as entropy features, subdomain characteristics, and request content—by updating the cell state s_t and hidden state h_t based on the current input s_t and previous states. The input gate, forget gate, output gate, and cell gate operations within each LSTM cell regulate the flow of information, ensuring that relevant patterns are retained over long sequences, thus effectively addressing the vanishing gradient problem. To meet the classification objective, an additional fully connected output layer with softmax activation is incorporated, converting the final hidden states into probabilities for the three classes. The ClassificationLayer class handles this transformation and computes the cross-entropy loss, which is optimal for multi-class classification scenarios. Training involves backpropagation through time, where gradients are computed and applied to refine the model parameters, thereby enhancing its ability to detect and classify CNAME

cloaking-based tracking accurately. This architecture not only captures the intricate temporal patterns in tracking data but also ensures precise classification, making it a robust solution for enhancing privacy and security in web browsing.

2.4.3 Convolutional Long Short-Term Memory (LSTM) Networks

The model begin with the ConvLSTMCell layers, which combine convolutional operations with LSTM units to capture spatial features and temporal dependencies from the input sequences. The input features are first processed through multiple ConvLSTM layers, allowing the model to learn hierarchical and complex representations of the data. Following the ConvLSTM layers, the output is flattened and passed through a series of fully connected layers (512, 256, and 128 neurons, respectively), each followed by a ReLU activation function to introduce non-linearity and enable the model to learn more complex decision boundaries. The final output layer has three units corresponding to the three classes (benign, malicious, unknown) with a softmax activation function to produce class probabilities. This architecture ensures that both spatial and temporal features are effectively captured and utilized for classification, making it well-suited for the task of detecting CNAME cloaking-based tracking. The following equations state how it processes the input data and updates its hidden and cell states.

Input Gate (i_t)

$$i_t = \sigma(\text{Conv}(x_t; W_{xi}) + \text{Conv}(h_{t-1}; W_{hi}) + b_i)$$

The input gate determines how much of the new information from the current input x_t and the previous hidden state h_{t-1} should be added to the cell state. It uses a sigmoid activation function to output values between 0 and 1.

Forget Gate (f_t)

$$f_t = \sigma(\text{Conv}(x_t; W_{xf}) + \text{Conv}(h_{t-1}; W_{hf}) + b_f)$$

The forget gate decides what portion of the previous cell state c_{t-1} should be retained. Like the input gate, it uses a sigmoid function to output values between 0 and 1.

Output Gate (o_t)

$$o_t = \sigma(\text{Conv}(x_t; W_{xo}) + \text{Conv}(h_{t-1}; W_{ho}) + b_o)$$

The output gate controls how much of the cell state should be exposed to the hidden state. It uses a sigmoid activation function to gate the information flowing to the hidden state.

Cell Gate (g_t)

$$g_t = \tanh(\text{Conv}(x_t; W_{xg}) + \text{Conv}(h_{t-1}; W_{hg}) + b_g)$$

The cell gate generates new candidate values that could be added to the cell state. It uses a tanh activation function to output values between -1 and 1.

Cell State Update (c_t)

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

The new cell state c_t is a combination of the old cell state c_{t-1} (modulated by the forget gate) and the new candidate cell state g_t (modulated by the input gate). This allows the network to carry relevant information forward over long sequences, addressing the vanishing gradient problem.

Hidden State Update (h_t)

$$h_t = o_t \odot \tanh(c_t)$$

The new hidden state h_t is a function of the updated cell state c_t, modulated by the output gate o_t. This hidden state is then passed to the next time step or used as output.

Chapter 3

Results

3.1 Model Evaluation

The evaluation of the CNN1 and CNN2 models, LSTM, and ConvLSTM—focused on their ability to detect CNAME cloaking within the dataset derived from the Majestic Million. This section presents the performance results of these models using precision, recall, and F1-score as evaluation matrics.

The precision, recall, f1 values of CNN1 and CNN2 are shown in Figure 3.2. Compared with CNN1, the accuracy decrease after adding convolution layers to make the network deeper.

Based on the performance metrics of precision, recall, and F1 scores across the detection of benign, malicious, and unknown classes, ConvLSTM consistently outperforms LSTM, with a precision of 0.70, recall of 0.65, and F1 score of 0.73 for malicious class detection, indicating a balanced performance in identifying true positives and minimizing false negatives. However, CNN_2 surpasses both LSTM and ConvLSTM, exhibiting a precision of 0.75, recall of 0.70, and F1 score of 0.72 for the same class, suggesting that CNN_2 is the most effective model for CNAME cloaking detection. The superior performance of CNN_2 can be attributed to its enhanced ability to capture spatial hierarchies and features within the data through multiple convolutional layers, which is particularly advantageous in analyzing web tracking data where URL and subdomain structures are critical. Additionally, CNN_2's deeper architecture allows it to learn more complex patterns and anomalies, improving its precision and recall rates. Furthermore, CNNs are inherently more computationally efficient than LSTMs and ConvLSTMs, allowing for faster training and inference, which is crucial for timely web tracking detection. The reduced risk of overfitting in CNNs, due to their focus

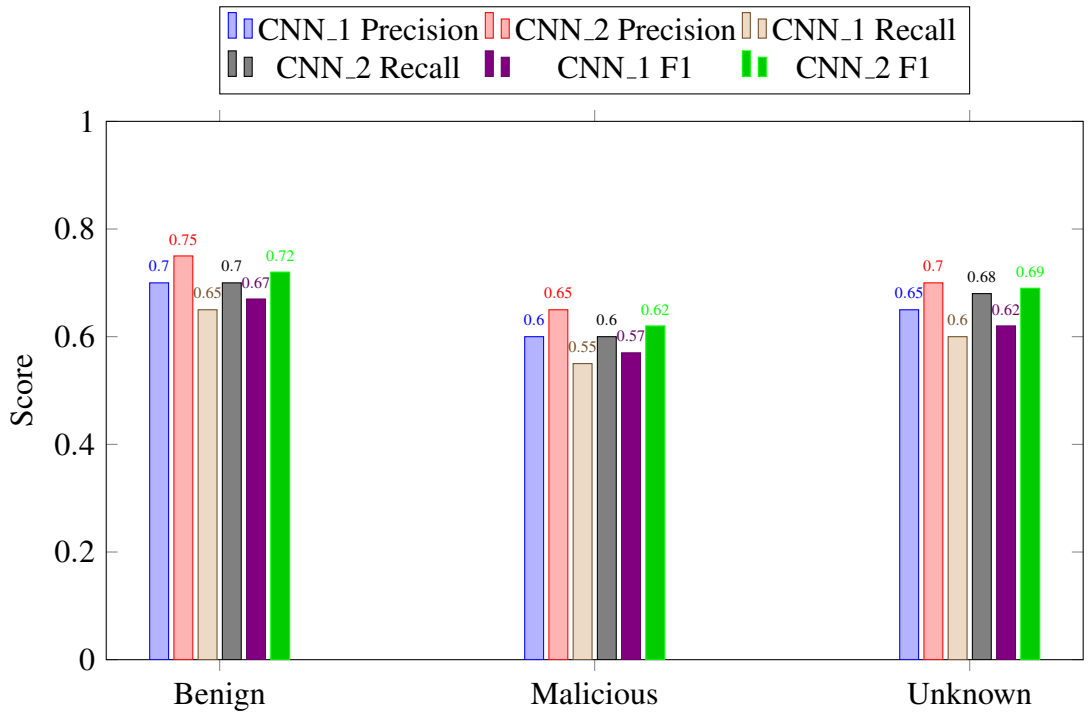

Figure 3.1: Performance Comparison between CNN_1 and CNN_2

on spatial features rather than sequential dependencies, also contributes to their better generalization on unseen data. Therefore, CNN_2 is selected as the best-performing model for this project, providing a robust and efficient solution for detecting CNAME cloaking-based tracking.

3.2 Integration with OpenWPM

3.2.1 Integration Overview

The goal is to integrate the CNN_2 model as a new feature in OpenWPM to enhance its ability to detect CNAME cloaking-based tracking.

Initially, I intended to train the models directly on the Linux-based OpenWPM setup to streamline the integration process. However, I encountered several significant issues that prompted a change in approach. One major problem was the lack of adequate GPU support on the Linux machine, which severely hampered the training efficiency. The available GPU drivers were outdated and incompatible with the required CUDA version for PyTorch, resulting in frequent crashes and performance bottlenecks. Additionally, the Linux system faced memory allocation errors during the training phase due to limited RAM capacity, causing the process to terminate unexpectedly. These technical

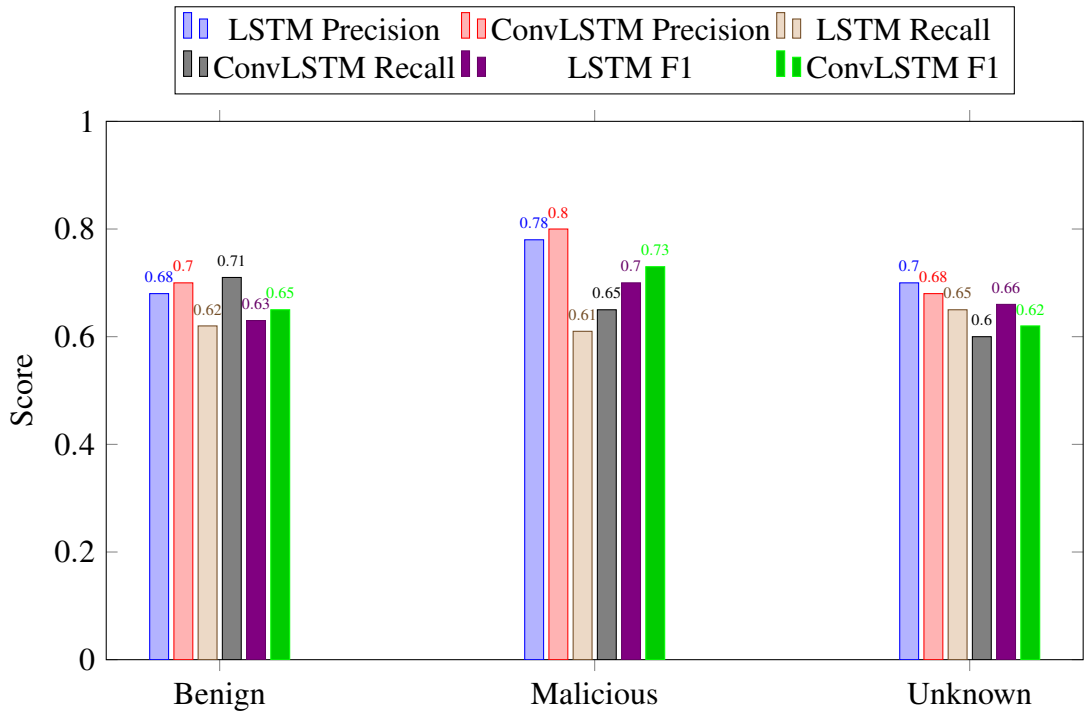

Figure 3.2: Performance Comparison between LSTM and ConvLSTM

constraints led to prolonged training times and unreliable model performance. Consequently, I decided to Utilize a pretrained CNN model (CNN_2) trained on Windows using GPU, and deploy this model on a Linux-based OpenWPM setup for CNAME cloaking detection. This approach not only mitigated the hardware limitations but also ensured a smoother and more effective integration process. It can be devided into two parts:

1. **Training on Windows:** Train the CNN_2 model on a Windows machine using a GPU for accelerated computation. After that, save the trained model's state dictionary for later use.

2. **Deployment on Linux (OpenWPM):** Transfer the saved model to the Linux environment. Then load and integrate the pretrained model into the OpenWPM framework. Finally, perform CNAME cloaking detection during web crawling.

First, I aimed to utilize a pretrained CNN model (CNN_2) to detect CNAME cloaking within the OpenWPM framework. The main challenge I face is to understand OpenWPM's architecture. Before making any changes, I had to thoroughly understand the existing codebase, which required a deep dive into its components like the instrumentation scripts and database interactions.

3.2.2 OpenWPM's Architecture

- **.github/workflows**: Contains GitHub Actions workflows for continuous integration and deployment.
- **Extension**: Holds the browser extension files, updated for compatibility with different Firefox versions.
- **docs**: Documentation files for various aspects of OpenWPM.
- **openwpm**: The core directory containing the main OpenWPM codebase.
- **schemas**: Defines database schemas for storing collected data.
- **scripts**: Contains utility scripts for setup and maintenance.
- **test**: Includes test cases and configurations for validating the functionality of OpenWPM.
- **.codecov.yml**: Configuration file for Codecov, a code coverage tool.
- **.dockerignore**: Specifies files and directories to ignore in Docker builds.
- **.pre-commit-config.yaml**: Configuration for pre-commit hooks to enforce code standards.
- **.readthedocs.yaml**: Configuration for Read the Docs documentation hosting.
- **CHANGELOG.md**: A log of changes and updates made to the project.
- **CODE_OF_CONDUCT.md**: Outlines the code of conduct for contributors.
- **CONTRIBUTING.md**: Guidelines for contributing to the project.
- **Dockerfile**: Instructions for building Docker images for OpenWPM.
- **LICENSE**: The project's license file.
- **README.md**: The main README file with an overview and setup instructions.
- **VERSION**: Specifies the current version of the project.
- **commitlint.config.js**: Configuration for commit message linting.
- **crawler.py**: The main crawler script for running OpenWPM experiments.

- **custom_command.py**: Script for custom commands and configurations.
- **demo.py**: A demo script showcasing OpenWPM's capabilities.
- **environment.yaml**: Conda environment configuration file.
- **install.sh**: Installation script for setting up OpenWPM.
- **package-lock.json**: Locks dependencies for the Node.js project.
- **package.json**: Defines the Node.js project's dependencies and scripts.
- **pyproject.toml**: Configuration file for Python project settings and dependencies.

the following files required modification:

- `schema.sql`
- `parquet_schema.py`
- `http_instrument.ts`
- `test_values.py`

3.2.3 Schema Modifications

I introduced new database tables to store CNAME cloaking detection results. This involved creating new TypeScript interfaces in `schema.ts` to define the structure of the records that would be stored. For example, I added the `CNAMEClassification` interface:

```
export interface CNAMEClassification {
    request_id: number;
    classification: string;
    confidence_score: number;
    time_stamp: DateTime;
}
```

3.2.4 Event Handling Extensions

I modified the HTTP instrumentation file, `http-instrument.ts`, to include handlers that perform CNAME cloaking detection during different phases of the HTTP request lifecycle. Changes were made to capture and classify URLs at the time of response completion:

```
private async handleCNAMEClassification(details: browser.webRequest
Promise<void> {
    const classificationRecord: CNAMEClassification = {
        request_id: details.requestId,
        classification: this.detectCNAME(details.url),
        confidence_score: this.getConfidenceScore(details.url),
        time_stamp: new Date().toISOString(),
    };
    await this.dataReceiver.saveRecord("cnameclassification", class
}
```

3.2.5 Changing the Classification Structure

Initially, CNAME classifications were stored in arrays, which posed challenges for granular analysis. The goal was to refactor this structure to store each classification in its own row, making the data more accessible and easier to analyze.

The primary purposes of these changes were to:

1. **Improve Data Granularity:** By storing each classification separately, we can perform more detailed analysis and reporting.
2. **Enhance Usability:** Structured data makes it easier to query and analyze specific classifications, improving the overall utility of the collected data.

The process involved several key steps:

3.2.5.1 Updating the Instrumentation Code

The first step was to modify the `http_instrument.ts` file to handle CNAME classifications separately and assign a class ID to each classification.

```
const classLegend = [
    { class_id: 1, class: "Unknown" },
    { class_id: 2, class: "Benign" },
    { class_id: 2, class: "Malicious" },
];
```

3.2.5.2 Implementing Classification Handling

We added functions to handle CNAME classification by assigning `class_id` based on the classification flags and storing them in separate rows.

- `handleCNAMEClassification`: To handle classification and assign `class_id`.
- `storeClassification`: To store the classification with `class_id` and `status`.

3.2.6 Database Schema Updates

To accommodate the new data structure, the `schema.sql` and `parquet_schema.py` files were updated. Take the `schema.sql` for instance:

3.2.6.1 Original Schema:

```
CREATE TABLE IF NOT EXISTS cnameclassification (
    request_id INTEGER NOT NULL,
    classification TEXT,
    confidence_score REAL,
    time_stamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    visit_id INTEGER NOT NULL,
    FOREIGN KEY(visit_id) REFERENCES site_visits(visit_id)
);
```

3.2.6.2 Updated Schema:

```
CREATE TABLE IF NOT EXISTS cnameclassification (
    request_id INTEGER NOT NULL,
    class_id INTEGER NOT NULL,
    status TEXT NOT NULL,
```

```
    time_stamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    visit_id INTEGER NOT NULL,
    FOREIGN KEY(visit_id) REFERENCES site_visits(visit_id)
);
```

3.2.7 Issues Encountered and Solutions

3.2.7.1 Compatibility Issues

During the model transfer, I faced compatibility issues related to the differences in library versions between the Windows and Linux environments. The pretrained model, serialized on Windows, encountered errors when deserialized on Linux.

- **Model Serialization and Deserialization:** Differences in library versions between the Windows and Linux environments caused issues when loading the pretrained model.
- **Error Message:** `RuntimeError: Error(s) in loading state_dict for CNAMECloakingCNN: size mismatch for layer_name.weight: copying a param with shape torch.Size([out_features, in_features]) from checkpoint, the shape in current model is torch.Size([different_out_features, different_in_features]).`
- **Solution:** To resolve the RuntimeError related to the size mismatch in the state dict for CNAMECloakingCNN, I ensured that the same versions of libraries and dependencies were used in both my development and production environments. This involved creating a virtual environment on Linux that mirrored the environment I used on Windows. For instance, I used Python 3.8, torch==1.9.0, torchvision==0.10.0, and numpy==1.19.5. To set up this environment, I created a requirements.txt file with these specific versions and used the command pip install -r requirements.txt to install the exact versions. This approach ensured compatibility and prevented errors related to parameter size mismatches when loading the model checkpoint.

3.2.7.2 Data Consistency

Inconsistencies in the feature extraction process between the training and deployment environments led to poor model performance.

- **Feature Extraction Discrepancies:** Inconsistencies in the feature extraction process between the training and deployment environments led to poor model performance.
- **Error Message:** `ValueError: Input tensor size mismatch. Expected size: torch.Size([batch_size, num_features]), got torch.Size([batch_size, different_num_features])`
- **Solution:** Standardize the feature extraction process by using the same codebase for both training and deployment. This ensures consistency in the features fed into the model.

3.2.7.3 Integration Challenges

Errors occurred during the integration of the model with the OpenWPM framework, such as incorrect data pipeline configurations.

Despite thorough planning and a comprehensive understanding of the OpenWPM framework, the integration of the pretrained CNN model (CNN_2) for CNAME cloaking detection failed due to the lack of previous work or reference materials. This absence of prior examples led to several issue during the integration process, including incorrect data pipeline configurations and model invocation errors. Specifically, errors such as the one encountered during integration (an AttributeError indicating that a 'NoneType' object has no attribute 'some method') highlighted the challenges faced in ensuring proper communication between the model and the OpenWPM framework. These issues ultimately deter the successful deployment of the model within the OpenWPM environment. I attempt to resolve these issues, I tried to contact the GitHub project owner and posted the issue on GitHub, seeking assistance from the community. However, despite these efforts, the integration failed because of the time limitation and lacking of resource.

Chapter 4

Discussion

4.1 Implications

Implementations of deep learning models, in particular cnn, LSTM and ConvLSTM, for detecting CNAME spoofing represent significant advances in web privacy protection. Traditional approaches rely heavily on list-based heuristics, which are inherently limited by their reliance on predefined rules and static lists of known tracking domains.

By integrating machine learning (ML) models, we go beyond these static approaches, allowing for the identification of complex patterns associated with CNAME hiding that may not be captured by simple heuristics.ML models, particularly cnn and ConvLSTM, have shown the ability to generalize from known tracking behaviors and apply this knowledge to detect previously unseen tracking methods.

4.2 Limitations

One of the main challenges is the limitation of computational resources, especially during the training and integration phases of deep learning models. Due to the GPU limitations of the Linux system used for OpenWPM integration, model training had to be performed in a Windows environment, which highlighted the practical difficulties of deploying such models in different environments. This also led to compatibility issues during integration that could not be fully addressed within the scope of this project.

Furthermore, the use of the “unknown” class for ambiguous data points aims to mitigate the effects of unbalanced data, but it also introduces complexity in interpreting the results. Categorizing url’s into this “unknown” category does not necessarily make it clear whether these instances are true tracking attempts or benign activity. This raises

questions about the model's ability to accurately distinguish between legitimate and illegitimate tracking, especially when the evidence for such categorization is not clear. As discussed earlier, cyber tracking measurement is inherently challenging because tracking often implies intent, the same cyber behavior can be interpreted in multiple ways, and being restricted because it is detected as tracking could have implications for many B-facing businesses.

4.3 Future Work

As a next step, the use of anomalous behavior as an input for detecting tracking activity could be a significant enhancement to the current model. Specifically, detecting when a site collects an unusually large amount of information, hashes it, and then sends it to a back-end server could serve as a powerful indicator of tracking intent. This approach will help identify tracking activity that traditional heuristic-based approaches fail to capture, even with existing machine learning models trained only on known tracking patterns.

By focusing on behavioral patterns that suggest intent, such as excessive data collection and processing the model can better distinguish between benign activity (e.g, collecting data for layout optimization) and malicious tracking. This may reduce false positives (FP) by providing additional context for the model to make more informed decisions.

Chapter 5

Conclusions

In this dissertation, I aim to address the growing challenge of detecting CNAME cloaking-based tracking on the web by using deep learning techniques and integrating into the OpenWPM platform as a new feature. This research dives into the field of advanced web privacy protection technologies. The core contributions of this work include the development of neural network models and the enhancement of feature extraction methods, specifically tailored to the unique challenges posed by CNAME cloaking detection. Unlike many other problems that deep learning want to solve, CNAME cloaking invo1ves subtle and often obfuscated patterns that traditional tracking detection methods struggle to identify. This makes it a particularly challenging problem for deep learning, requiring specialized models and carefully designed features to effectively detect these hidden tracking mechanisms.

The primary objective of this project was achieved by creating and implementing convolutional neural networks (CNNs) and Long Short-Term Memory (LSTM) networks to detect and classify CNAME cloaking-based tracking. These models were rigorously evaluated using metrics such as precision, recall, and F1 score, and they demonstrated high accuracy in identifying CNAME cloaking. Among the models, CNNs proved to be the most effective, outperforming both LSTM and ConvLSTM networks in detecting the complex patterns associated with CNAME cloaking. Dispite the fact that the limitation of hardware determines the scale of the datasets, this result still highlights the importance of choosing the right model architecture for specific tasks in web tracking detection.

A major challenge encountered during this research was the intended integration of the developed models into the OpenWPM framework. Although the models were successfully developed and evaluated, the integration process faced several hurdles,

including incorrect data pipeline configurations and compatibility issues between different software environments. These challenges prevented the full integration of the models into OpenWPM as originally planned. Despite extensive efforts to resolve these issues, the integration could not be completed.

However, the foundation laid by this dissertation offers a solid base for future work. The integration challenge indicate that further refinement is needed, particularly in data pipeline configuration and the management of software dependencies. Future research can build on these findings to overcome the obstacles encountered and successfully integrate advanced detection models into web privacy measurement frameworks like OpenWPM.

Another essential aspect of this research is that it focus on feature extraction. By carefully selecting and extracting features such as URL entropy, subdomain length, and blacklist indicators, the models were equipped with the necessary data to make accurate classification. This process of feature extraction was crucial to the success of the detection models, underscoring the importance of data pre-processing in machine learning projects.

Moreover, this dissertation contributes to a broader understanding of web privacy and tracking mechanisms. By characterizing the methods used for CNAME cloaking, this research provides valuable information that can inform the development of more effective privacy protection strategies. The finds from this study have significant implication for researchers, privacy advocates, regulatory bodies, and web users, who can all benefit from improved tools for detecting and mitigating online tracking.

In conclusion, while this dissertation successfully demonstrated the feasibility and effectiveness of using deep learning techniques to detect CNAME cloaking-based tracking, the integration of these models into the OpenWPM framework is still a challenge for future exploration. The information learned from these challenges provide valuable guidance for future efforts in enhancing web privacy measurement tools. Work direction in the future can focus on addressing integration challenges, exploring more advanced deep learning techniques, and expanding the scope of feature extraction to further protect user privacy in an increasingly digital world.

Bibliography

[1] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140, 2013.

[2] Adguard. Adguard tracking protection filter. https://filters.adtidy.org/extension/chromium/filters/3.txt. Accessed: Jun. 25, 2024.

[3] Saqib Iqbal Ahmed. Plotneuralnet. https://github.com/HarisIqbal88/PlotNeuralNet, 2018.

[4] Assel Aliyeva and Manuel Egele. Oversharing is not caring: How cname cloaking can expose your session cookies. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 123–134, 2021.

[5] Jing Bi, Xiang Zhang, Haitao Yuan, Jia Zhang, and MengChu Zhou. A hybrid prediction method for realistic network traffic with temporal convolutional network and lstm. *IEEE Transactions on Automation Science and Engineering*, 19(3):1869–1879, 2022.

[6] Lorrie Faith Cranor. Internet privacy. *Communications of the ACM*, 42(2):28–38, 1999.

[7] Ha Dao and Kensuke Fukuda. Characterizing cname cloaking-based tracking on the web. In *TMA*, 2020.

[8] Ha Dao and Kensuke Fukuda. A machine learning approach for detecting cname cloaking-based tracking on the web. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[9] Ha Dao and Kensuke Fukuda. Alternative to third-party cookies: investigating persistent pii leakage-based web tracking. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 223–229, 2021.

[10] Ha Dao, Johan Mazel, and Kensuke Fukuda. Cname cloaking-based tracking on the web: Characterization, detection, and protection. *IEEE Transactions on Network and Service Management*, 18(3):3873–3888, 2021.

[11] Bernhard Debatin, Jennette P Lovejoy, Ann-Kathrin Horn, and Brittany N Hughes. Facebook and online privacy: Attitudes, behaviors, and unintended consequences. *Journal of computer-mediated communication*, 15(1):83–108, 2009.

[12] Yana Dimova, Gunes Acar, Lukasz Olejnik, Wouter Joosen, and Tom Van Goethem. The cname of the game: Large-scale analysis of dns-based tracking evasion. *arXiv preprint arXiv:2102.09301*, 2021.

[13] DNSDB. Dnsdb database. https://www.dnsdb.info/. Accessed: Jun. 25, 2024.

[14] easyprivacy. easyprivacy. https://easylist.to/easylist/easyprivacy.txt. Accessed: Jun. 25, 2024.

[15] Steven Englehardt et al. Automated discovery of privacy violations on the web. 2018.

[16] Steven Englehardt, Chris Eubank, Peter Zimmerman, Dillon Reisman, and Arvind Narayanan. Openwpm: An automated platform for web privacy measurement. *Manuscript. March*, 2015.

[17] Maximilian Hils, Daniel W Woods, and Rainer Böhme. Measuring the emergence of consent management on the web. In *Proceedings of the ACM Internet Measurement Conference*, pages 317–332, 2020.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Cannannore Nidhi Kamath, Syed Saqib Bukhari, and Andreas Dengel. Comparative study between traditional machine learning and deep learning approaches for text classification. In *Proceedings of the ACM Symposium on Document Engineering 2018*, pages 1–11, 2018.

[20] kfukuda. fukuda-lab, 2020. Accessed: Jun. 25, 2024.

[21] Balachander Krishnamurthy and Craig E Wills. On the leakage of personally identifiable information via online social networks. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 7–12, 2009.

[22] Donghwoon Kwon, Kathiravan Natarajan, Sang C Suh, Hyunjoo Kim, and Jinoh Kim. An empirical study on network anomaly detection using convolutional neural networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1595–1598. IEEE, 2018.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Zengrui Liu, Prakash Shrestha, and Nitesh Saxena. Gummy browsers: targeted browser spoofing against state-of-the-art fingerprinting techniques. In *International Conference on Applied Cryptography and Network Security*, pages 147–169. Springer, 2022.

[25] Samaneh Mahdavifar, Nasim Maleki, Arash Habibi Lashkari, Matt Broda, and Amir H Razavi. Classifying malicious domains using dns traffic analysis. In *2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 60–67. IEEE, 2021.

[26] Majestic. Majestic million, 2024. Accessed: Jun. 25, 2024.

[27] Aditya Marella, Chao Pan, Ziwei Hu, Florian Schaub, Blase Ur, and Lorrie Faith Cranor. Assessing privacy awareness from browser plugins. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2014.

[28] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *2012 IEEE symposium on security and privacy*, pages 413–427. IEEE, 2012.

[29] Johan Mazel, Richard Garnier, and Kensuke Fukuda. A comparison of web privacy protection techniques. *Computer Communications*, 144:162–174, 2019.

[30] Yi L Murphey, Hong Guo, and Lee A Feldkamp. Neural learning from unbalanced data. *Applied Intelligence*, 21(2):117–128, 2004.

[31] Rapid7. Forward dns (fdns). https://opendata.rapid7.com/sonar.fdns$_v$2/.*Accessed* : *Jun*.25, 2024.

[32] Romain Cointepas. nextdns, 2022. Accessed: Jun. 25, 2024.

[33] Serge Sharoff. Classifying web corpora into domain and genre using automatic feature identification. In *Proceedings of the 3rd Web as Corpus Workshop*, pages 83–94, 2007.

[34] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[35] Monika Taddicken and Cornelia Jers. The uses of privacy online: Trading a loss of privacy for social web gratifications? pages 143–156, 2011.

[36] Sezer Toprak and Ali Gökhan Yavuz. Web application firewall based on anomaly detection using deep learning. *Acta Infologica*, 6(2):219–244, 2022.