DEFENDING AGAINST STEALTHY MOBILE MALWARE

KAIFA ZHAO

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University
Department of Computing

Defending Against Stealthy Mobile Malware

Kaifa Zhao

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
April 2023

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: _____

Name of Student: Kaifa Zhao

Abstract

The widespread popularity of Android, as one of the most widely-used mobile operating systems, is attributed to its ability to provide users with a wide range of convenient and entertaining options through its functional apps. Nevertheless, mobile users may face a risk to their privacy and property from potentially harmful apps that can be installed on their devices.

This thesis focuses on combining Android static analysis, artificial intelligence techniques, and natural language processing techniques to investigate app behavior, discover vulnerabilities in Android malware detection systems, and understand Android apps' privacy policies. To safeguard user privacy from potentially harmful apps, we propose the following measurements: (1) Investigating the vulnerability of Android malware detection systems under evolving structural attacks and proposing defense solutions. (2) Analyzing whether Android app privacy policies meet regulatory requirements. and (3) Empirically evaluating the capacity of pre-trained large language models to identify regulation-required components in Android privacy policies.

For (1), to investigate the vulnerability of Android malware detection (AMD) systems under evolving attacks and design effective defense solutions, we propose a Heuristic optimization model integrated with Reinforcement learning framework to optimize our structural ATtack, namely HRAT, which is the first problem-pace structural attack designed to deceive Android malware detection systems. HRAT employs four types of graph modification operations and corresponding bytecode manipula-

tion techniques to generate executable adversarial apps that can evade detection. HRAT bridges the research gap between feature-space attacks, which generate only adversarial features to deceive machine learning models, and problem-space attacks, which generate complete adversarial objects, i.e., executable Android apps in our scenario. Our extensive experiments demonstrate that HRAT demonstrates effective attack performance and remains robust against obfuscation methods that do not affect the app's function call graph. In addition, we propose potential defense solutions to improve the robustness of AMD against such advanced attack methods.

For (2), we construct a benchmark dataset for Android privacy policies, i.e., a novel large-scale human-annotated **C**hinese **A**ndroid a**pp**lication **p**rivacy **p**olicy dataset, namely CA4P-483. Following a manual inspection of regulatory articles, we identify seven types of labels that are relevant to the regulatory requirements for apps' access to user data. We design a two-step annotation process to ensure label agreement, and our evaluation showed that our annotations achieved a Kappa value of 77.20%, indicating substantial agreement for CA4P-483. In addition, we evaluate robust and representative baseline models in our dataset and present our findings and potential research directions based on our results. Finally, we conduct case studies to explore the potential application of CA4P-483 in protecting user privacy.

For (3), we empirically evaluate three widely used pre-trained large language models on the CA4P-483 dataset. This work aims to explore the capacity of LLMs in processing Chinese privacy policies and to uncover their potential to address compliance issues that are challenging for traditional NLP techniques. Building on our previous work with CA4P-483, we leverage the semantic understanding capabilities of pre-trained LLMs and apply carefully crafted prompts according to established prompt engineering principles to maximize the models' inference performance. Our evaluation reveals that state-of-the-art pre-trained LLMs still fall short of achieving satisfactory performance on the Chinese privacy policy dataset. The limitations may stem from the complexity of the language environment, the intricate cross-relationships among

elements within privacy policies, and the models' current generalization capabilities. Based on our evaluation results, we also propose potential future research directions that include leveraging long-context LLMs to analyze privacy policies holistically and achieve overall semantic consistency, as well as training a dedicated large-scale privacy policy analysis model that incorporates multilingual datasets to address privacy policies across different languages and platforms.

Publications

1. Jianfeng Li, Kaifa Zhao, Yajuan Tang, Xiapu Luo, and Xiaobo Ma, "Inaccurate Pre-diction Is Not Always Bad: Open-World Driver Recognition via Error Analysis", in *2021 IEEE 93rd Vehicular Technology Conference (VTC), 2021.*

2. Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo, "Structural Attack against Graph Based Android Malware Detection", in *Proceedings of ACM Conference on Computer and Communications Security (CCS), 2021.*

3. Kaifa Zhao, Le Yu, Shiyao Zhou, Jing Li, Xiapu Luo, Yat Fei Aemon Chiu, and Yutong Liu, "A Fine-grained Chinese Software Privacy Policy Dataset for Sequence Labeling and Regulation Compliant Identification", in *The 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2022.*

4. Le Yu, Yangyang Liu, Pengfei Jing, Xiapu Luo, Lei Xue, Kaifa Zhao, Yajin Zhou, Ting Wang, Guofei Gu, Sen Nie, and Shi Wu, "Towards Automatically Reverse Engineering Vehicle Diagnostic Protocols", in *Proceedings of the 31st USENIX Security Symposium (USENIX Security), 2022.*

5. Lei Xue, Yangyang Liu, Tianqi Li, Kaifa Zhao, Jianfeng Li, Le Yu, Xiapu Luo, Yajin Zhou, and Guofei Gu, "SAID: State-aware Defense Against Injection Attacks on In-vehicle Network", in *Proceedings of the 31st USENIX Security Symposium (USENIX Security), 2022.*

6. Yulin Zhu, Yuni Lai, Kaifa Zhao, Xiapu Luo, Mingquan Yuan, Jian Ren, and Kai Zhou, "Binarizedattack: Structural poisoning attacks to Graph-based anomaly detection", in *38th International Conference on Data Engineering(ICDE), 2022.*

7. Shuohan Wu, Jianfeng Li, Hao Zhou, Yongsheng Fang, Kaifa Zhao, Haoyu Wang, Chenxiong Qian, Xiapu Luo, "CydiOS: A Model-Based Testing Framework for iOS Apps", in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, 2023.*

8. Yulin Zhu, Yuni Lai, Kaifa Zhao, Xiapu Luo, Mingquan Yuan, Jun Wu, Jian Ren, Kai Zhou, "From Bi-Level to One-Level: A Framework for Structural Attacks to Graph Anomaly Detection", in *IEEE Transactions on Neural Networks and Learning Systems, 2024.*

Acknowledgments

In this thesis, I would like to take this opportunity to express my deepest appreciation to my parents for their unwavering support and encouragement throughout my academic journey. Their love, guidance and sacrifices have been instrumental in helping me overcome challenges and achieving my academic goals. I am forever grateful for their unwavering belief in me, and I hope to continue making them proud in all my future endeavors. I would like to express my heartfelt gratitude to myself for demonstrating unwavering perseverance and determination in overcoming the unwanted and unnecessary challenges I faced during my Ph.D. studies, especially those man-made malicious events, I firmly believe that what I have experienced will make me who I am. Completing this journey with a sound mind and body is a testament to my strength and resilience, and I am proud to have achieved this milestone.

I would also thank my supervisor, professor Xiapu Luo. He has provided me with the opportunity to pursue a doctoral degree and a platform to conduct research in security and software engineering. He has dedicated a lot of time to guiding me on how to identify meaningful research topics and how to ensure that my work contributes to the field. I have learned a great deal from his research approach and am grateful for his mentorship.

Additionally, I would like to thank Prof. Hao Zhou, Prof. Lei Xue, Prof. Jing Li, Prof. Kai Zhou, Prof. Jianfeng Li, and Prof. Xiaoming Wu for their constructive feedback and suggestions, which have helped me recognize limitations in my work. I

am grateful to Yu Yang, Yangyang Liu, Shiyao Zhou, Shuohan Wu, Wenying Wei, and Zhiyuan Wen for the interesting discussions and memorable moments during my Ph.D. study.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The escalating prevalence and extensive utilization of smartphones have rendered them a prime target for cybercriminals. Due to its extensive usage, Android, being the most widely used mobile operating system[131], is particularly vulnerable to malicious software (malware) attacks. Malware on Android devices can cause a range of problems [130], from stealing sensitive information to corrupting data and disabling system functions. Concealing users' data access behaviors in the privacy policy will deprive users of the right to know about the processing of private data, thereby leading to the potential leakage of users' privacy. Comprehending the malevolent conduct of Android applications (apps) is critical to detecting Android malware and safeguarding the privacy of users.

Recent studies demonstrate the importance of analyzing the behavior of potentially harmful apps (PHA) for detecting Android malware [9, 79, 68, 152, 91, 25, 24, 18, 177, 163, 65, 5, 120, 61, 158, 42], investigating potential vulnerabilities of Android malware detection (AMD) systems [15, 26, 56, 77, 107, 60], and identifying consistency between Android apps' behavior and privacy policy statements [6, 7, 165, 166, 97, 178, 146]. Android malware detection systems are designed to identify malicious apps before malware is published in the market [49, 62] and to further prevent apps from harming

mobile device users. Unfortunately, previous research has revealed that AMD can be circumvented with relative ease through adversarial attacks [15, 26, 56, 77, 107, 60] that manipulate the features extracted from apps, thereby deceiving machine learning models utilized in detection systems. As a consequence, app markets [49, 62] have imposed regulations [50, 63] to govern the conduct of apps published in their markets. App markets, in particular, require developers to explicitly disclose their app's behavior, including the collection, usage, sharing, or storage of users' information, in a privacy policy document upon uploading the app. However, existing research [6, 7, 165, 166, 97, 178, 146] has revealed that the behavior of apps is frequently inconsistent with their privacy policy statements. Despite the numerous studies proposed to analyze the behaviors of potentially harmful Android apps [15, 26, 56, 77, 107, 60, 9, 79, 68, 152, 91, 25, 24, 18, 177, 163, 65, 5, 120, 61, 158, 42] and their privacy issues [6, 7, 165, 166, 97, 178, 146], the research community still lacks comprehensive studies on investigating the vulnerability of Android malware detection systems under structural adversarial attacks; and creating a benchmark dataset that meets the regulatory requirements for app privacy policy statements.

1.1 Android Malware Detection

Android malware detection systems are developed to recognize malevolent apps, and current detection methodologies typically rely on classification techniques. Such methods extract features from both benign and malicious apps and subsequently employ them to train a machine-learning model with the capacity to differentiate between the two. Notably, many systems [9, 79, 68, 152, 91, 25, 24, 18, 167, 42] utilize static analysis to extract features, including requested permissions [9, 79, 68] and function call relations [91, 152]. Of these systems, function call graph (FCG)-based approaches [152, 91, 25, 24, 18] have demonstrated promising performance as FCGs contain abundant semantic information, such as calling relationships. In an FCG,

each node represents a method, and each directed edge depicts a calling relationship between two methods. State-of-the-art malware detection tools extract potential malicious features, such as sensitive APIs, from FCGs to represent malware's malevolent behaviors. For instance, Malscan [152] extracts centralities of sensitive nodes in FCGs to train classifiers. Mamadroid [91] abstracts the nodes of FCGs into different states and uses the transition probability between states as features. Cai et al. [18] use the graph neural network to extract features from FCGs for malware detection.

1.2 Adversarial Attack Against Android Malware Detection

The goal of adversarial attacks against Android malware detection systems [56, 15, 77, 26, 107] is to evade detection by generating adversarial samples that incorporate perturbations into extracted feature vectors, in order to deceive the classifiers of target systems. For instance, Grosse et al. [56] apply the Jacobian matrix to modify features in Drebin [9] to generate adversarial examples and achieve a 69% evasion rate. AndroidHIV [26] streamlines the attack process by manipulating the features in Mamadroid [91] and Drebin [9] using various algorithms, such as $C\&W$ based methods [26]. Traditional adversarial attacks against AMD [56, 15, 77], however, can solely create adversarial features to escape detection and are confined to producing executable adversarial applications. Hence, traditional adversarial attacks fail to evaluate the susceptibility of actual Android malware detection systems since these systems identify whether an app (usually in the form of an *apk* file) is malicious based on its physical form rather than its feature representations, such as permissions utilized in the apps. The growing emphasis on producing authentic evasion samples has led to an increasing number of systems [26, 107] developing algorithms for generating executable adversarial Android malicious apps, also known as problem-space attacks. Existing problem-space attacks are limited to inserting non-functional methods or in-

vocations in order to evade detection while preserving the semantics of the generated adversarial malware. Due to the inverse feature-mapping problems, the feature mapping transformations between problem space and feature space are neither injective nor surjective [110], meaning that some information can be lost or misrepresented in the mapping process [26, 107].

1.3 Android Privacy Policy

The proliferation of mobile devices has elevated the necessity for privacy and security measures to safeguard personal information. The Android operating system, being the dominant platform for mobile devices, holds a vast amount of sensitive information about its users, such as personal and financial data, location, and online activity. As a result, the need for privacy and security measures to safeguard this information has become increasingly important. This vast amount of sensitive information available on Android, the most widely used mobile operating system, makes it a valuable target for cybercriminals. As a result, it is imperative to have effective privacy policies in place to govern the behavior of Android apps and safeguard user data.

To normalize privacy-related behaviors and prevent privacy leakage, various privacy-related regulations (e.g., California Consumer Privacy Act [144], California Privacy Rights Act [101], General Data Protection Regulation [47], International Covenant on Civil and Political Rights [52], Code of Federal Regulations [113]) have been promulgated to protect people's personal information from being abused. A privacy policy is a legal document [47, 108, 98] written in natural language that outlines the types of information collected, how it is used, and who it is shared with. The Android privacy policy undergoes constant evolution to keep up with the rapidly changing technological landscape and to address emerging privacy concerns as they arise. Privacy policies play a crucial role in providing users with a clear understanding of how their personal data will be used and enabling them to make informed decisions about

using a product. They serve as a tool for enhancing transparency and accountability and help users assess the risks associated with sharing their personal information. As such, privacy policies have emerged as a key aspect of data protection and privacy regulation in the mobile app ecosystem. However, privacy policies are tedious, making it hard for users to read and understand them [128].

1.4 Android Apps Behavior Analysis

Android app analysis can be broadly classified into two categories, namely static analysis [54, 190, 53, 64, 40] and dynamic analysis [39, 135, 179, 159, 112, 164, 138, 33, 66]. The static analysis evaluates apps' code and resources without executing it. Static analysis is a powerful tool for identifying potential security vulnerabilities and privacy violations in an app, as it allows researchers to identify issues that may not be evident during normal app execution. Static analysis can be performed quickly and easily on large numbers of apps without executing apps. Moreover, static analysis excels in code coverage, tamper resistance, cost-effectiveness, early detection, and repeatability.

Dynamic analysis, on the other hand, involves executing an app on a device or emulator and monitoring its behavior during runtime. This method provides a more realistic view of an app's behavior and allows researchers to identify issues that may not be visible in the code, such as network communication and data usage. Dynamic analysis is particularly useful for identifying malicious behavior, such as data theft and unauthorized access to sensitive information. However, dynamic analysis requires significant computational resources, including a high-performance device or emulator, memory, and storage space. Besides, dynamic analysis is a time-consuming process, especially for large or complex apps. This can make it difficult to scale dynamic analysis for large numbers of apps. Dynamic analysis can only provide an approximation of an app's behavior, as it does not take into account all the possible ways an app

can behave in the real world. This can result in false negatives or a limited view of app behavior. Dynamic analysis can interfere with the normal functioning of an app, as the analysis process can modify system settings or disrupt normal app behavior. This can result in inaccurate or misleading results.

1.5 Pre-trained Large Language Models for Privacy Policy Analysis

Large language models (LLMs) [2, 102, 8, 32] demonstrate exceptional ability in understanding and processing natural language. Benefiting from LLMs' advanced natural language understanding abilities, researchers have begun to apply LLMs to analyze privacy policies across various domains [23, 90, 116, 139], including the Internet of Things (IoT), web applications, LLM plugins, and beyond. The application of LLMs to downstream tasks primarily relies on prompt engineering, where researchers craft detailed instructions that include task descriptions and data samples to be analyzed. These instructions are used to query target LLMs, which then infer new content based on the provided context to complete the tasks.

1.6 Our Work

To safeguard individuals' privacy and property, our focus is on analyzing Android apps that may potentially be harmful and preventing apps' malicious activities. We first investigate the vulnerability of Android malware detection systems, focusing on the design of the first problem-space structural attack against Android malware detection systems. Second, we construct the first Chinese Android privacy policy dataset to identify the consistency between apps' behavior and privacy policy statements. Finally, we investigate the capabilities of pre-trained large language models

for identifying regulation-required components in privacy policies.

1.6.1 Vulnerability Investigation of Android Malware Detection Systems

First, we propose HRAT, the first problem-space structural attack against Android malware detection systems, to investigate the robustness of detection systems under evolving threats. Android malware detection techniques achieve great success with deeper insight into the semantics of malware. Among existing detection techniques, function call graph (FCG) based methods achieve promising performance due to their prominent representations of malware's functionalities, i.e., the nodes in FCG denote the methods in apps and the edges in FCG denote the invocation relations between methods. Meanwhile, researchers propose adversarial attacks to investigate the vulnerability of detection systems and propose corresponding defense methods to make the systems more robust in malware detection. However, existing adversarial attacks against Android malware detection systems focus on perturbing the feature vectors, that are extracted from the apps' components or function call graphs, to escape detection, and ignore the new attack face exposed in function call graph-based detection systems, such as the structure of function call graphs as is shown in Figure 3.2 in Chapter 3 Section 3.2.1. Furthermore, suffering from the attack interface, i.e., perturbing the feature vector, existing adversarial attacks can not guarantee the success of generating runnable adversarial apps based on perturbations on feature vectors, which is known as inverse transformation limitation [107].

In this thesis, we design a Heuristic optimization model integrated with Reinforcement learning framework to optimize our structural ATTack, namely HRAT. Compared with existing adversarial attacks against Android malware detection systems, HRAT has three important capabilities, namely a) a novel attack channel, i.e., the structure of Android apps' function call graph, b) filling the research gap between problem-

space attack and feature-space attack, and c) a useful framework for Android app manipulation while maintaining the functionalities.

Specifically, HRAT implements four types of graph modification methods to perturb the structure of Android apps' function call graph, namely adding edges, rewiring, inserting nodes, and deleting nodes. We craft four types of graph modifications while maintaining the connection relations of nodes which may be affected by the structure modification. This design guarantees the invocation relations of methods in apps will not be affected at a theoretical level and provide the basis for transforming perturbations in feature space to problem space. To optimize the attack process, we apply a reinforcement learning algorithm to learn the optimal graph modification sequence based on the criteria of minimum perturbations. To make sure the function call graph modification can be transformed into apps' code manipulation and does not affect the apps' functionalities, we craft four types of Android app manipulation methods accordingly, namely adding function calls, rewiring function calls, inserting methods, and deleting methods. HRAT implements the Android app manipulation tool based on static analysis and requires no access to apps' source code.

We evaluate HRAT on over 30,000 Android apps including apps from different time period [13] and using different obfuscation techniques [37]. HRAT demonstrates outstanding attack performance on both feature space and problem space and illustrates that combining multiple graph modifications strategically makes the attack more effective and efficient. Besides, our experiments also demonstrate that HRAT is resilient to obfuscations that do not affect or hide invocation relations in apps. After accessing the vulnerability of Android malware detection systems, we also propose defense strategies to make the detection systems robust against evolving attacks.

1.6.2 Curation of Chinese Privacy Policy Benchmark

The Android privacy policy is a legal document written in natural language that discloses how and why a controller, who determines the purposes for which and the means by which personal data is processed, collects, shares, uses, and stores user information [47, 108, 98]. Regulation department [47, 108, 98] and Android application management [49, 62] ask developers to provide a privacy policy to clearly state how they deal with user information to help users understand and be aware of whether their privacy will be abused and decide whether to use the product. However, privacy policies are tedious, making it hard for users to read and understand them [128]. Fortunately, Natural language processing techniques achieve great success in understanding document semantics [161, 150, 36]. However, applying natural language processing to understand privacy policies still requires a large amount of labeled data to train the semantic understanding model.

In this thesis, we first construct a fine-grained Chinese software privacy policy dataset to fill the research gap and prompt research on understanding the consistency between apps' behavior and privacy policy statements. Although there exist English versions of privacy policy datasets, such as Online Privacy Policies (OPP-115) [151] and Android app privacy policies (APP-350) [193], the annotations in the dataset are coarse-grained, i.e., being labeled in sentence level, and cannot satisfy regulation requirements in China [98, 108, 100, 27]. Thus, we construct the first large-scale human-annotated **C**hinese **A**ndroid a**pp**lication **p**rivacy **p**olicy dataset, namely `CA4P-483`. Specifically, we manually visit the software markets, such as Google Play [49] and AppGallery [62], check the provided privacy policy website, and download the Chinese version if available. We finally collect 483 documents. To determine the labels in the privacy policy analysis scenario, we read through Chinese privacy-related regulations and summarize seven components (§4.3.2). We annotate all occurrences of components in 11,565 sentences from 483 documents. Unlike paragraph-level annotations in existing privacy policy datasets [151], `CA4P-483` annotates character-level

corpus.

Second, based on `CA4P-483`, we summarize representative baseline algorithms for Chinese sequence labeling, which aim to identify the labels of Chinese characters within a given privacy policy sentence, based on predefined regulatory components. In detail, we first evaluate the performance of several classic sequence labeling models on our dataset, including Conditional Random Forest (CRF) [70], Hidden Markov Model (HMM) [94], BiLSTM [55], BiLSTM-CRF [74], and BERT-BiLSTM-CRF [34]. Recent work shows lattice knowledge improves the performance of Chinese sequence labeling tasks. We also evaluate a lexicon-based model specifically designed for Chinese NLP scenarios, such as Lattice-LSTM [180].

Third, we investigate potential applications of `CA4P-483`. Combining knowledge of regulations, we first identify whether the privacy policy violates regulation requirements based on `CA4P-483`. We also identify whether the app behaves consistently with privacy policy statements by combing software analysis [184, 189].

1.6.3 Investigating Pre-trained Large Language Models for Privacy Policy Analysis

Pre-trained large language models (LLMs) exhibit remarkable capabilities in understanding the semantics of natural language. These models are trained on vast collections of diverse natural language resources, including Wikipedia, publicly available news, books, and programming code, among others [185, 102]. Consequently, LLMs are equipped with general natural language understanding abilities, such as document summarization [175, 22], sentence completion [99], etc. In addition to general-purpose capabilities, LLMs fine-tuned on specific datasets for particular tasks have shown exceptional performance. For example, LLMs trained on programming-related data excel in tasks such as code completion[81, 95], code generation[104, 86], and code summarization[1, 75]. However, achieving such performance often requires access

to extensive domain-specific datasets for pre-training[140] or fine-tuning[84], which can be resource intensive. Moreover, research highlights that LLMs perform sub-optimally on specialized downstream tasks when they are not explicitly trained for those purposes. Tasks such as summarizing legal documents or drafting scripts exemplify these limitations. To address these challenges and enhance the utility of LLMs for domain-specific tasks, researchers and industry professionals have increasingly adopted prompting-based approaches [46, 82]. These approaches leverage the inherent generative capabilities of LLMs to tackle a wide range of tasks without the need for extensive retraining or fine-tuning.

In this work, we design an LLM-based system for Chinese privacy policy analysis, termed LLMPP, to empirically evaluate the performance of two publicly available LLMs [160, 141] and one widely used commercial LLM [102] on our curated Chinese privacy policy dataset [183], i.e., CA4P-483. LLMPP takes as input the privacy-related sentences, specifically, those containing terms related to data collection and sharing [6, 183], in order to address the context length limitations of LLMs and prevent them from losing focus when processing an entire lengthy and complex privacy policy document. Besides, LLMPP employs carefully crafted prompts, leveraging prompt engineering techniques [46, 115, 59], to enhance the performance of popular general-purpose LLMs on downstream tasks—in our case, Chinese privacy policy analysis. To gain deeper insights into the performance of LLMPP, we evaluate the performance of LLMPP on CA4P-483 [183] and perform comprehensive ablation studies on various prompt engineering techniques and assess the hallucination risks across multiple LLMs. Our findings highlight persistent challenges in applying LLMs to downstream tasks involving Chinese privacy policy analysis. Building on these experimental results, we propose potential research directions to advance user privacy protection in this domain.

1.7 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 introduces the most related work. Chapter 3 proposes the first structural attack to investigate the vulnerability of Android malware detection systems. Chapter 4 crafts the first Chinese privacy policy dataset with fine-grained annotations. Chapter 5 empirically evaluates the performance of LLMs on analyzing privacy policy documents. Finally, Chapter 6 concludes the thesis and discusses future work.

The primary research outputs that emerged from the thesis are as follows:

- Kaifa Zhao, Le Yu, Shiyao Zhou, Jing Li, Xiapu Luo, Yat Fei Aemon Chiu, and Yutong Liu, "A Fine-grained Chinese Software Privacy Policy Dataset for Sequence Labeling and Regulation Compliant Identification", in *The 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2022.*

- Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo, "Structural Attack against Graph Based Android Malware Detection", in *Proceedings of ACM Conference on Computer and Communications Security (CCS), 2021.*

In summary, this thesis makes the following contributions:

- We propose the first structural attack against Android malware detection systems to investigate the vulnerability of detection systems. We also release the code and data to other researchers by responsibly sharing a repository (https://github.com/zacharykzhao/HRAT).

- We develop a new tool to transform perturbations on feature space to problem space while maintaining the functionality of Android apps and publish the tool to facilitate research in this community.

- We create and publish the first fine-grained Chinese privacy policy dataset, namely CA4P-483, with word-level annotation. Our dataset and code are publicly available in https://github.com/zacharykzhao/CA4P-483

- We evaluate popular baseline algorithms on CA4P-483, summarize difficulties in our dataset, provide findings and further research topics on our dataset, and conduct a case study to demonstrate the potential application of our dataset in identifying privacy compliance studies.

- We empirically evaluate the performance of pre-trained large language models for analyzing compliance in privacy policies. We meticulously designed prompts to utilize pre-trained large language models to identify entities related to legal and regulatory requirements in Chinese privacy policies. We also assessed the effectiveness of three publicly popular large language models on this task. We also make all our data, scripts and results publicly available in https://github.com/zacharykzhao/CA4P-483/tree/main/LLMPP

- We systematically analyze the evaluation performance of LLMs and identify key challenges in applying them to the analysis of Chinese privacy policies. Based on our findings, we propose future directions for applying LLMs to such downstream tasks. These include leveraging long-context models to address the task in an end-to-end manner and pretraining a new LLM to explore relationships between privacy policies, enabling the model to handle privacy policies written in different languages and originating from various platforms.

Chapter 2

Literature Review

We introduce related work that is most related to our studies, including adversarial attacks against Android malware detection (in §2.1), privacy policy dataset (in §2.2), and Android privacy policy analysis (in §2.3).

2.1 Adversarial Attack against Android Malware Detection

Many attack methods [56, 26, 119, 107, 77, 15, 171] have been proposed to evade Android malware detection systems. From the perspective of whether the adversary generates real Android apps, adversarial attacks can be categorized into *problem-space attacks* [26, 107, 77] and *feature-space attacks* [56, 26, 119, 107, 77, 15]. *Feature-space attacks* only modify the features to deceive the classifiers used by detection systems. Grosse1 et al. [56] propose to replace the classifier in *Drebin* with a neural network to improve detection performance. They also propose Jacobian matrix-based (JM) methods to modify the features to escape detection. Similarly, Shahpasand et al. [119] use a generative adversarial neural network to generate adversarial features

for evading *Drebin*. However, neither of them considers the modification of APK, which means that the adversarial samples they generate can only evade the detection of the classifier, and cannot generate new malware that has escaped system detection and flowed into the software market. *Problem-space attacks* to generate real adversarial examples. AndroidHIV [26] builds the transformation relation between feature modification and APK manipulation to transform features (i.e., invocation probability) to the number of invocations. Then, AndroidHIV applies optimization algorithms to guide perturbations on the feature vectors. Based on feature perturbations, AndroidHIV inserts the corresponding number of call relations between target methods to generate adversarial malware. Pierazzi et al. [107] extract benign components from the training set and then insert those benign components into malicious samples to generate adversarial malware to evade Drebin's [9] detection. Although these studies aim to generate real APK, they just insert no-op code. Li et al. [77] attack Drebin by inserting and removing components, in which removal is achieved by renaming components, in the feature vectors. However, their modification only renames the corresponding string content to deceive the feature extraction methods.

From the perspective of the attack surface for perturbing graph-based systems, adversarial attacks can be categorized into feature attacks [56, 26, 119, 107, 77, 15] and structural attacks [136, 30, 89, 147, 191]. Feature attacks [26, 152] perturb the feature vectors extracted from the graph to deceive the target algorithm. Structural attacks [136, 30, 89, 147, 191] modify the structure of graphs or the features of nodes to evade detection. Except for simply inserting and deleting edges, ReWatt [89] proposes rewiring edges to preserve the graph characteristics. ReWatt uses reinforcement learning to select the most influential edges for modification. The middle nodes in ReWatt's rewiring are restricted to the second-order neighbors of nodes in the original edge to make their modification "unnoticeable" to GCN. Wang et al. [147] deceive the graph convolutional neural network classifier by inserting nodes selected by the generative adversarial neural network. Note that existing studies usually ignore deleting

nodes. Moreover, existing attacks against malware detection only insert dead code because removing nodes will cause relevant edges to be deleted and may lead to side effects [107].

2.2 Privacy Policy Dataset

Prior privacy policy datasets are all in English and omit other languages. OPP-115 [151] collects 115 privacy policies in English websites and makes annotations at the sentence level. OPP-115 designs labels based on previous works [92, 128] and ignores the regulation requirements at the latest time. APP-350 [193] gathers Android apps' privacy policies written in English. APP-350 only conducts limited annotations, including two types of data controllers, namely first party and third party, thirteen types of specific data, and two types of modifiers, i.e., do and do not.

Existing Chinese sequence-labeling datasets are generally gathered from News [173, 170, 134] and social media [106, 149, 180]. The datasets include abundant corpus, but their annotations are limited to location, person name, and organization. Even though CLUENER2020 [154] expands the labels, such as the game, government, and the book, the datasets are still hard to be applied in specific downstream tasks. On the other hand, CNERTA [134] includes another media data, i.e., voice data, to improve the sequence labeling performance.

2.3 Android Privacy Policy Analysis

XFinder [146] identifies the cross-library data harvesting in Android apps with dynamic analysis. XFinder identifies third-party libraries' usage by comparing the caller's and callee's package names. XFinder also restores reflection invocations using two predefined patterns. For conflict identification, XFinder manually parses the

terms-of-service of 40 TPLs and then uses NLP techniques to extract data-sharing policies. Nguyen et al. [97] investigate whether apps achieve users' consent before sharing personal information. The authors use dynamic analysis to identify the network traffic and data-sharing behaviors. They identify whether the shared data are identifiable personal data by comparing the same traffic collected from different times or the same traffic from different devices. The ablation experiments are designed to determine whether the data-sharing action achieves users' explicit consent.

PAMDroid [178] analyzes the impact of misconfigurations of analytical services in Android. After analyzing 1000 popular apps, PAMDroid finds 52 of 120 apps misconfigured the services and led to a violation of either the service providers' terms-of-service or the app's privacy policy. PPChecker [165] also identifies the conflict in apps' privacy policies, but only identifies whether apps' privacy policies provide TPLs' privacy policy links and interactions of five permission-related personal information with 81 TPLs. POLICHECK [7] identifies the app's data sharing with third parties using dynamic analysis. POLICHECK finds that 49.5% of apps disclose their third-party sharing practices using vague terms and 31.1% of data flows as omitted disclosures. Existing works ignore analyzing whether TPLs satisfy the regulation of requirements.

2.4 Pre-trained Large Language Model for Privacy Policy Analysis.

CLEAR [23] is designed to enhance user awareness of privacy policies and potential risks when interacting with pre-trained large language model applications. CLEAR is capable of identifying sensitive information, summarizing relevant privacy policies, and highlighting potential risks in a contextual and just-in-time manner. Through co-design workshops and user studies, CLEAR demonstrates its effectiveness in im-

proving users' understanding of data practices and privacy risks and encouraging more cautious data-sharing behaviors. However, CLEAR's applicability is currently limited to specific platforms, such as ChatGPT and the Google Gemini plugin, lacking support for a broader range of applications and modalities. Additionally, CLEAR relies on the Microsoft Presidio library for identifying personally identifiable information (PII), which may not always accurately or promptly identify all types of personal identification information, potentially leaving user privacy at risk.

Maliyetty et al. [90] evaluate the performance of quantized LLaMA models in analyzing Internet of Things (IoT) privacy policies. The authors design various prompts to guide different pre-trained large language models in generating privacy policy language and assess the output using semantic similarity metrics such as ROUGE-Lsum [83] and BERT precision [174]. The authors' experiments demonstrate that the quantized models exhibit comparable performance to the base model. However, the authors do not differentiate model performance across various types of privacy policy statements, nor do they include fine-tuning for specific privacy policy texts. This absence may limit the models' ability to handle nuanced or domain-specific privacy concerns effectively.

Rodriguez et al. [116] evaluate the performance of pre-trained large language models, specifically ChatGPT and LLaMA 2, in analyzing privacy issues within the MAPP dataset. The authors assess various prompts using these LLMs to analyze coarse-grained privacy issues in privacy policies, such as whether the policy involves user-specific types of information, including financial data, location data, etc. However, the proposed methods overlook the input limitations of large language models. When the length of the privacy policies exceeds the input limits of the target LLM, the proposed methods may truncate the content of the privacy policy, which can further affect the performance of these methods.

LLM-PBE [80] is designed for the systematic evaluation of data privacy risks in Large Language Models (LLMs), addressing a critical gap in comprehensive privacy assess-

ments for these models. LLM-PBE employs a diverse array of attack and defense strategies, including data extraction, membership inference, and jailbreaking attacks, to analyze privacy vulnerabilities across different LLMs, data types, and metrics. LLM-PBE reveals that larger LLMs are susceptible to data extraction, underscoring the need for further research into prompt protection and privacy-preserving mechanisms. However, LLM-PBE only focuses on existing attack and defense methods, suggesting that future advancements may yield different findings. Moreover, the exploration of dynamic text data management strategies for evolving LLMs remains an open challenge.

PolicyGPT [139] implements a framework that leverages pre-trained large language models, such as ChatGPT and GPT-4, for the automated categorization of privacy policies. PolicyGPT addresses the challenge of comprehending verbose and complex legal documents. The framework employs a zero-shot learning approach to evaluate performance on OPP-115 and PPGDPR datasets. However, PolicyGPT relies on predefined categories and potential inefficiencies with few-shot prompts, as observed during the A/B testing phase.

Chapter 3

Structural Attack against Graph Based Android Malware Detection

3.1 Overview

Malware detection techniques achieve great success with deeper insight into the semantics of malware. Among existing detection techniques, function call graph (FCG) -based methods achieve promising performance due to their prominent representations of malware's functionalities. Meanwhile, recent adversarial attacks not only perturb feature vectors to deceive classifiers (i.e., feature-space attacks) but also investigate how to generate real evasive malware (i.e., problem-space attacks). However, existing problem-space attacks are limited due to their inconsistent transformations between feature space and problem space.

Existing feature attacks [56, 15, 77, 26, 107] generated adversarial samples by adding perturbations to extracted feature vectors to deceive classifiers in target systems. For instance, Grosse et al. [56] apply the Jacobian matrix to modify features in Drebin [9] to generate adversarial examples and achieve the 69% evasion rate. AndroidHIV [26] aims to optimize their attack process, which perturbs the features in Mamadroid [91]

and Drebin, through several algorithms (e.g., $C\&W$ based methods [26]).

Recent studies designed problem-space attacks to generate real adversarial malware [26, 107]. Unfortunately, to preserve the semantics of generated adversarial malware, existing problem-space attacks are restricted to inserting non-functional methods or calls [26, 107]. Due to the inverse feature-mapping problems, the feature-mapping functions between problem space and feature space are neither injective nor surjective [110]. Thus, existing problem-space attacks have to take extra processes to deal with side effects [26, 107].

We propose a novel and practical structural attack method against FCG-based AMD systems, which tackles the limitations (**L1-3**) of existing attack methods.

L1-system-specific attack methods. Existing attack approaches [56, 15, 77, 26, 107], especially those problem-space ones [26, 107], highly depend on the feature extraction methods of target systems. For example, AndroidHIV[26] implements a problem-space attack on Mamadroid by transforming the features (i.e., call probability) to invocation numbers. Hence, if the features change, the transformation relation must also be adjusted accordingly. By contrast, since our structural attack perturbs the graph structures rather than feature vectors, feature extraction methods have no impact on our attack flow. That is, our structural attack is general to all FCG-based systems.

L2-limited software modification operations. To maintain functional consistency, existing App modification methods are limited to inserting dead code, such as no-op API calls [117, 26, 107]. By contrast, we design four types of software manipulation operations: *inserting methods*, *removing methods*, *adding call relations*, and *rewiring call relations* (§3.4).

L3-inconsistent transformation relation. The adversarial App generation methods of existing approaches are guided by the transformation from feature perturbations to App modifications [26, 107]. Although they can randomly modify features as

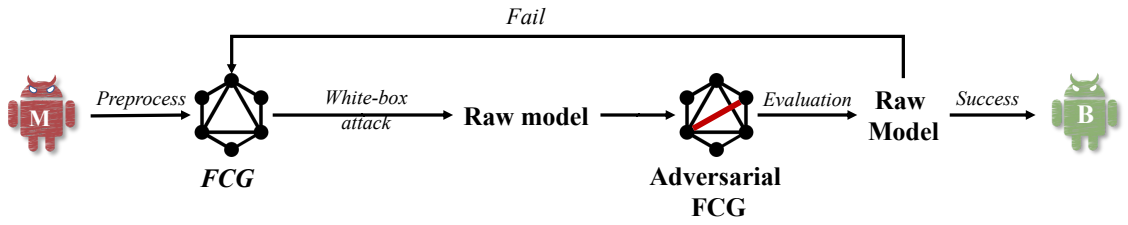


Figure 3.1: Overview of HRAT

needed when perturbing features, they can only insert no-op code to restricted methods [26, 107] when generating adversarial apps. To bridge this gap, we design each structural attack action by considering the characteristics of apps (§3.3.2). As nodes and edges in the CFGs correspond to methods and call relations in apps, respectively, our attack method ensures that the modification of the graph is consistent with the manipulations of apps.

Although our structural attack solves the aforementioned limitations in existing methods, there remain two challenges in our system design, i.e., how to determine a manipulation operation type and how to select the most effective objects (nodes or edges) to modify. To this end, we design a Heuristic optimization integrated Reinforcement learning ATtack (`HRAT`) algorithm (§3.3.3) to solve them. `HRAT` consists of two phases: a) determining an action type according to the current graph state and b) selecting optimal edges or nodes to conduct the modifications on the graph. Leveraging reinforcement learning, HRAT learns how to select effective action types based on the current graph state (§3.3.3) through interacting with the target environment [137]. Then, with the determined action type, HRAT uses the gradient search to select the most influential edges or nodes for modification. In this way, HRAT learns the modification sequences on the target graph, which allows the modified App to bypass the detection. Finally, HRAT automatically generates adversarial apps based on graph modification sequences. Figure 3.1 gives the overview of HRAT.

Our major contributions are summarized as follows:

(1) A Novel Structural Attack. To our best knowledge, we propose the first structural attack on Android malware detection systems, namely HRAT. HRAT includes four types of graph modification operations and uses reinforcement learning to optimize the attack process.

(2) Fill Research Gap. Our method fills the gap that adversarial features cannot be effectively mapped to real apps. Besides, since our attack method works on graphs directly, it could be extended to other graph-based detection systems.

(3) A Useful Tool. We develop an automated tool that can manipulate the structure of Android apps without affecting the original functionality. HRAT can manipulate apps according to graph modification sequences. We release the source code and data set to other researchers by responsibly sharing a private repository. The project website with instructions to request access is at: https://sites.google.com/view/hrat.

(4) Valid Evasion Effects. We evaluate the effectiveness of our attack on the two latest AMD systems and one enhancement system. The results of extensive experiments show that our attack can achieve over 90% of the overall attack success rate in feature space and up to 100% of the attack success rate in problem space.

3.2 Preliminaries

In this section, we present the necessary knowledge on the difference between feature attacks and structural attacks, our target Android malware detection (AMD) systems (i.e., Malscan [152] and Mamadroid [91]), and one AMD enhancement method (i.e., APIGraph [177]). Both two AMD tools use function call graphs (FCGs) to detect Android malware. Besides, we present basic knowledge about reinforcement learning to ease the explanation of our methods later.

3.2.1 Feature Attacks and Structural Attacks

Android malware detection (AMD) [9, 79, 68, 152, 91, 25, 24, 18, 177, 61, 158] has attracted much attention from both industry and academia. One popular technique applied in AMD is signature-based methods [38] that are limited to a tedious extracting process and easy evasion properties [78]. Thus, machine learning (ML) -based systems are widely used for malware detection. ML-based detection techniques first extract features from benign and malicious Apps and then train the detection model. In particular, many systems [9, 79, 68, 152, 91, 25, 24] use static analysis to extract features, such as requested permissions [9, 79, 68] and function call relations [91]. Among these systems, FCG-based methods [152, 91, 25, 24, 18] achieve promising performance as FCGs contain rich semantic information, e.g., calling relationships. In a function call graph, each node represents a method, and each directed edge represents a calling relationship between two methods. The state-of-the-art malware detection tools extract potential malicious features (e.g., sensitive APIs) from FCGs to represent malware's malicious behaviors. For example, Malscan [152] extracts centralities of sensitive nodes in FCGs to train classifiers. Mamadroid [91] abstracts the nodes of FCGs into different states and uses the transition probability between states as features. Cai et al. [18] use the graph neural network to extract features from FCGs for malware detection.

Recent studies [152, 26] demonstrate the possibility of attacking FCG-based detection methods[152, 91] through adversarial samples. Unfortunately, their methods are almost limited to perturbing extracted feature vectors from FCGs, i.e., feature attacks. By contrast, in this paper, we investigate the vulnerability of FCG-based detection methods from their attack surfaces relevant to edges and nodes [153, 182, 89, 147, 136, 30, 110] and propose a new structural attack method.

Figure 3.2 shows the differences between feature attacks and structural attacks. The former adds perturbations to extracted feature vectors, whereas the latter directly

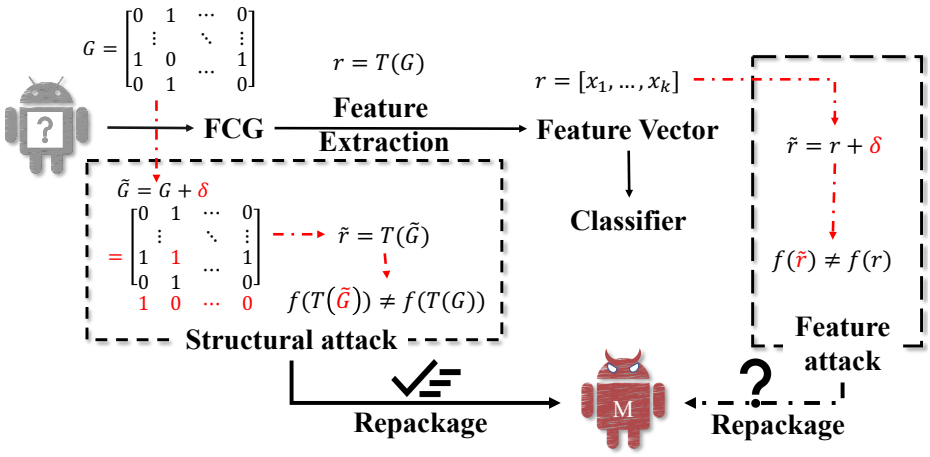


Figure 3.2: Feature attacks vs. structural attacks

modifies the nodes and edges in graphs. Graph-based detection methods extract features from graphs and use these features to train classifiers for detection. By contrast, structural attacks directly modify the graph features, and thus they are more intrinsic and effective [145, 192]. Besides, as nodes in FCGs correspond to methods in software and edges correspond to call relations, structural attacks could address the inverse-transformation problems [107] between feature-space and problem-space attacks.

3.2.2 Target Android Malware Detection Systems

Malscan. Malscan extracts FCGs (G) from Apps and uses centralities [43, 67] of sensitive nodes in G as features. Those sensitive nodes correspond to Android's sensitive APIs, which reflect the malicious properties [12, 152] of apks. Let $c(G)$ denote the centralities of all nodes in the target function call graph G. The label of the target G in Malscan is formulated as:

$$y = f\left(I_{cen} \times c\left(G\right)\right), \tag{3.1}$$

where $I_{cen} = [i_1, ..., i_N] \in R^{1\times N}$ is the sensitive index of nodes in G, N is the number of nodes, $i_k \in \{0, 1\}$ and $i_k = 1$ means that node k is sensitive, $G \in R^{N\times N}$ is the

adjacency matrix of FCG, $f(\cdot)$ is the pre-trained classifier (i.e., kNN in Malscan).

Mamadroid. Mamadroid [91] extracts function call relations from FCGs as features. Based on the characteristic of Android method signature, it first abstracts methods into different states according to the package name or family name. By doing so, it will be resilient to API changes in the Android framework [91]. Next, Mamadroid extracts call the probabilities between states, i.e., families or packages of target methods, as features and trains classifiers like kNN to detect malware. The label of G is represented as:

$$y = f\left(T_{cp}\left(S \times G\right)\right), S = \{s_{i,j}\} \in R^{N_s \times N}, \tag{3.2}$$

where N is the number of nodes in G and N_s is the number of states in Mamadroid, $s_{i,j} \in \{0, 1\}$ and $s_{i,j} = 1$ denotes that node j belongs to state i, $T_{cp}(\cdot)$ is the call probabilities among states, and $f\left(\cdot\right)$ is pre-trained classifier.

APIgraph. APIgraph [177] is a framework to enhance AMD. It enhances the representation ability of features and uses the characteristics of Android to aggregate APIs with similar semantics. Specifically, APIgraph first collects API documents from the Android official website and then builds the connections among APIs using a relation graph. Based on the relation graph, a graph embedding algorithm is applied to get each API's embedding vector. APIgraph uses a clustering algorithm, like k-means, to aggregate APIs with similar semantics into one cluster. To enhance the target AMD, APIgraph uses a specific API to denote all APIs in one cluster during the feature extraction process.

3.2.3 Reinforcement Learning

Reinforcement learning (RL) [137] learns what to do and how to take actions based on current situations through interacting with the target environment and maximizes the reward from the environment's feedback. Different from supervised learning, which learns from the training set, corresponding knowledge (labels of samples in

the training set), and an external supervisor (objective function), RL learns without prior knowledge. Unlike unsupervised learning, which targets leveraging a hidden structure in the unlabeled data set, such as the distributions or representations, RL targets maximizing a reward signal by interacting with the target environment. Although evolutionary algorithms (EAs) could approach RL problems, EAs are more suitable to solve problems whose policy space is small and can be structured or the problems whose learning agent cannot accurately sense the environment. Moreover, EAs neither learn from the environment nor formulate the relation between actions and the environment's states[137].

Reinforcement learning contains four elements: an **action** set, a **state** set, a **reward** function and a **policy** model.

• The **state** set stores all the possible states of the target environment. For example, in the maze problem, the state set saves all possible positions of the player in the maze.

• **Action** set contains all the actions that the learner can take. For instance, the maze problem includes the directions the player can move forward at each step.

• **Reward** function defines the environment feedback for the current state. For example, in the maze problem, if the player walks out of the maze, the greatest reward is given. In middle states, which refer to any positions between the entrance and the exit, the closer the player is to the exit, the higher the reward will be given.

• **Policy** model defines how current action influences current state.

An essential property of reinforcement learning is that the problem to be solved should conform to the Markov Decision Process (MDP) [76]. Only when this condition is met, the action and reward in reinforcement learning can be formulated as a function of the current state.

3.3 Attack Model

This section presents our threat model (§3.3.1), attack formulation (§3.3.2), and optimization process (§3.3.3). We also theoretically prove the effectiveness and advantages of our structural attack (§3.3.4).

3.3.1 Threat Model

In our attack scenario, the adversary possesses white-box access to target systems. That is, the adversary has access to the dataset, feature space, and model parameters of target systems. This setting follows Kerckhoffs' principle [107] and ensures that a defense does not rely on "security by obscurity" by unreasonably assuming some properties of the defense that can be kept secret [19]. Our attack aims at modifying the function call graph of a malicious app to evade the target system's detection. Two FCG-based AMDs, i.e., Malscan [152] and Mamadroid [91], and one AMD enhancement method, namely APIGraph [177], are used to evaluate the effectiveness of our attack. We select Malscan and Mamadroid because they are state-of-the-art FCG-based AMDs and are published in top conferences, i.e., ASE 2019 and NDSS 2017 respectively, with influential impact. They report outstanding malware detection performance (98% detection accuracy for Malscan and 99% F1 score for Mamadroid).

3.3.2 Attack Formulation

Our structural attack $\mathcal{K}$ takes in FCG (adjacency matrix) and modifies the nodes and edges in the graph to deceive the detection systems. Our attack is defined as:

$$\tilde{G} = \mathcal{K}(G) = G + \delta, \tag{3.3}$$

where G is input graph, $\tilde{G}$ is the adversarial graph, and δ is the perturbations to the adjacency matrix of graph. We use $f(G, \theta)$ to denote a malware detection system,

which takes in G and uses the pre-trained parameters θ to determine the category to which G belongs. Since our attack only modifies the structure of G without changing parameters θ, the detection system can be simplified to $f(G)$. The goal of our attack is to modify G to make the system $f(G)$ misclassify the malicious G as benign, i.e., $f(G) \neq f(\tilde{G}) = f(\mathcal{K}(G))$.

To ensure that the modified app works normally and preserves the same functionality as the original one, we design four types of modification actions - adding edges, rewiring, inserting nodes, and deleting nodes, tailored to app characteristics. Next, we first introduce constraint definitions, followed by action definitions and properties.

Definition 1. *(Constraints)* $\mathbb{C} = [c_1, .., c_N] \in R^{N \times 1}$ *where* N *is the number of nodes in graph* G, $c_i \in \{0, 1\}$ *denotes the modifiability of node* i. $c_i = 1$ *in dicates that the node* i *is modifiable, otherwise* $c_i = 0$.

Definition 1 defines the edges and nodes that cannot be modified during our attack process. They refer to the scenarios when some specific methods in apps (§3.4.1) cannot be modified. For example, we cannot modify Android framework APIs used in app [26]. For better formulation, we use $c_n \notin \mathbb{C}$ to denote $c_n = 1$, i.e., the node n is modifiable.

Definition 2. *(Adding edge) An adding edge action* A_a *involves two nodes* $A_a = \{v_{beg}, v_{tar}\}$, *where* $v_{beg}, v_{tar} \notin \mathbb{C}$ *are the caller and callee of the edge to be added respectively. The adding edge operation builds an invocation relation (directed edge) from* v_{beg} *to* v_{tar}.

Definition 3. *(Rewiring) Rewiring removes an edge from the graph and finds another intermediate node to maintain the connectivity of nodes in the deleted edge. A rewiring action* A_r *involves three nodes* $A_r = \{v_{beg}, v_{end}, v_{mid}\}$, *where* $v_{beg}, v_{end} \notin \mathbb{C}$ *are the caller and callee in deleted edge respectively, and* $v_{mid} \notin \mathbb{C}$ *is the intermediate node. The rewiring action deletes the edge from* v_{beg} *to* v_{end}, *and creates a new edge from* v_{beg} *to* v_{mid}, *and from* v_{mid} *to* v_{end}.

Definition 3 aims to preserve the app's functionality. That is, after deleting an edge that denotes a call relationship (e.g., $A \Rightarrow B$), we must find an intermediate node C to maintain the connection between A and B (i.e., modify the call relationship to $A \Rightarrow C \Rightarrow B$). Although it is possible to insert multiple nodes between A and B to maintain their connectivity after removing the edge between A and B, HRAT inserts one intermediate node in each rewiring action for the ease of implementation. Note that inserting multiple nodes can be achieved by several rewiring operations.

Definition 4. *(Inserting node) An inserting node action A_i involves one node $A_i = \{v_{caller}\}$, where $v_{caller} \notin \mathbb{C}$ denotes the caller to inserted node. It creates a new node v_{new} on the graph and then builds an invocation relation from $v_{caller} \notin \mathbb{C}$ to v_{new}.*

Definition 5. *(Deleting node) A deleting node action A_d involves three types of node $A_d = \{v_{tar}, \hat{v}_{caller}, \hat{v}_{callee}\}$, where $v_{tar} \notin \mathbb{C}$ denotes a node to be removed, $\hat{v}_{caller} \notin \mathbb{C}$ is the set of nodes that call v_{tar}, and $\hat{v}_{callee}$ is the set of nodes called by v_{tar}. The deleting node action deletes nodes v_{tar} in the graph and builds call relations from all nodes in $\hat{v}_{caller}$ to each node in $\hat{v}_{callee}$.*

According to Definition 5, when a node is deleted, the connectivity of the remaining nodes in the graph keeps unchanged.

So far, we have defined all four operations of the attack. Then, our attack process is formulated as:

$$\begin{aligned} &\mathcal{K}(G) \Leftrightarrow (a_1, a_2, ..., a_m)\, G, \\ &\text{where } a_i \in \{A_{ae}, A_{rewi}, A_{in}, A_{dn}\}. \end{aligned} \tag{3.4}$$

3.3.3 Heuristic Optimized Reinforcement Learning based Structural Attack

Our structural attack process consists of a sequence of attack actions on a target graph. Specifically, the decision of each attack action involves two phases: *determining an action type* and *selecting attack objects*. The former resolves the attack

action type (adding edge, rewiring, inserting node, or deleting node), while the latter determines the specific edges or nodes to be modified according to the action type. Our target is to find the modification sequence with minimum modifications on the graph rather than the hidden structure, e.g., the distributions of the dataset, in the target graph. Since supervised or unsupervised learning cannot be used to optimize our attack process [137], we leverage reinforcement learning [89, 93, 137] – an iterative learning algorithm that learns how to take actions based on the current environment. Our structural attack aims at modifying malware to escape target detection systems' detection with minimal modifications: modifying the fewest edges and nodes. Our attack considers four types of attack actions, and the decision of each action consists of *determining an action type* and *selecting attack objects*. For the greedy algorithm that chooses locally optimal solutions and concretes them together to approximate the optimal global solution, it will always select *adding edge* at each step because *adding edge* modifies one edge each time, *rewiring* modifies two edges, *inserting nodes* modifies one edge and one node, and *deleting nodes* modifies at least one node and one edge. In this way, adding edges alone cannot always achieve optimal perturbations. For example, if decreasing the degree centrality of certain nodes is locally optimal for the state, adding edges cannot achieve it because adding edges can only increase the degree of centrality. Evolutionary algorithm (EA) effectively solves problems whose policy spaces (i.e., attack action set) are small or can be structured [137]. In our scenario, the policy space is enormous (number of nodes and edges in the target graph) and cannot be structured because each attack action consists of a) selecting an action type and b) determining attack objects. The determination of b) depends on a), and a) is related to previous actions (§ 3.3.3). HRAT uses reinforcement learning, specifically deep Q-learning, to determine the attack action on the target graph. Our *policy model* involves two parts. The first part uses a neural network, specifically a two-layer fully connected neural network, to learn the relation between state and action type with the loss function based on reward. Then, with the determined action type, HRAT uses the gradient search to determine attack objects. Besides, to evalu-

ate the effectiveness of each attack action, our *reward function*, i.e., Equation 3.7, is designed as the number of modified nodes and edges. This reward function evaluates the impact of both action type and attack objects. For example, for adding one edge, the reward value is -1 as only one edge is added, no matter which edge the gradient search selects. For rewiring, the reward is -3 because one edge is removed and 2 edges are added. For inserting nodes, the reward is -2 because one node and one edge are inserted. For deleting nodes, the value of the reward depends on the node the gradient search selects. Since deleting nodes will remove one node from the target graph and build the connections from each of the deleted node's callers to all of its callees, the reward value depends on the number of deleted nodes' callers and callees.

For a target system $f(G)$ that takes in graph G and outputs the decision of G, HRAT aims to modify the structure of the graph, $\hat{G} = \mathcal{K}(G)$ so that the output $(f(\hat{G}))$ differs from the original one, i.e., $f(\hat{G}) \neq f(G)$. Following reinforcement learning, our structural attack progress is described as a series of decision-making processes: $P = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \pi\}$, where $\mathcal{S} = \{s_t\}$ is the state set that contains the intermediate and final state of the target environment, $\mathcal{A} = \{a_t\}$ is the set of actions that consists of all possible actions in state s_t, $\mathcal{R}$ is a reward function that evaluates the reward of taking action a_t on state s_t, and π is the deterministic policy that describes how to determine a_t and how the action a_t changes the state s_t .

HRAT adopts one attack action at each step to modify the target malicious graph until target systems flip their decision on the graph to benign. The structure of the current graph depends on the state of all previous graphs and the modification actions on the graph, i.e., $s_t \mapsto \pi(s_{t-1}, a_{t-1}, ..., s_0, a_0)$. According to HRAT's termination condition (i.e., the target system regards the FCG as benign), the intermediate states s_{mid} are all predicted to the same label as the original FCG, i.e., malicious. Thus, every step in our attack can be regarded as modifying a new graph. For example, when crafting malware to deceive the target system, HRAT extracts FCG and modifies its structure (methods and call relations). After one modification, HRAT can regard the

next modification as modifying a new version of the malware. In other words, the current state only depends on the latest state and action, i.e., $\pi(s_{t-1}, a_{t-1}, ..., s_0, a_0) = \pi(s_{t-1}, a_{t-1})$, which satisfies the Markov Decision Process (MDP) [137].

To choose the influential attack action in each step, we use reinforcement learning to learn the attack process. Reinforcement learning optimizes the decision-making process by continually interacting with the target environment and obtaining rewards. We will introduce each element of reinforcement learning in HRAT.

State Space

During the attack process, the state space holds all intermediate states, and HRAT selects feature vectors extracted from the corresponding intermediate graphs to store as states. In the case of attacking Malscan, the state space would store all intermediate centrality features of sensitive APIs. HRAT only stores the latest modified graph's adjacency matrix to facilitate subsequent modifications. This setting has the following advantages:

• **Easy handling:** The feature vector is of fixed length and ideal for training the policy network with states, actions, and rewards. As we do not cover the dynamic handling of changes in the graph scale, we avoid using the adjacency matrix in this thesis.

• **Easy storing:** It is easy to store the feature vector. Since the FCG of an app can have hundreds of thousands and even millions of nodes and edges, a sufficiently large amount of memory is required if the intermediate graphs are stored directly. Fortunately, the features extracted from the function call graphs by these target systems are usually a one-dimensional vector with tens of thousands of dimensions, which greatly saves memory. Since the current state only depends on the previous state and action (§3.3.3), storing the latest graph structure is sufficient for determining the next action.

Algorithm 1: Deep Q-learning for structural attack

Input: Target classifier f, training dataset $\{X_t, Y_t\}$, target FCG G, memory capacity N, probability ϵ, maximum modification times M, feature-space transformation T, node constraints $\mathbb{C}$

Output: action sequence a_{seq}, adversarial graph G'

Initialize replay memory D to capacity N

Initialize action-value network Q with random weights θ

$Obj_{adv}(G) = \sum_{i=1}^{m} \omega_i \cdot \sigma\left(\|x_i - T\left(G\right)\|_2\right)$

$y_t = f(G), y_p = f(G), G_i = G$

while $y_p == y_t$ *and* $i < M$ **do**

  $tmp = random\_probability$

  if $tmp < \epsilon$ **then**

    $a_i = argmax_a Q(G_i, a; \theta)$

  else

    $a_i = random\_action$

  Calculate gradient of each edge: $\partial_G = \bigtriangledown_{G_i} Obj_{adv}(G_i)$

  Execute action a_i on G_i: $G_{i+1}, r_i = ATT\_OBJ(G_i, \partial_{G_i}, \mathbb{C})$

  $D \leftarrow (T(G_i), a_i, r_i, T(G_{i+1}))$

  $$y_j = \begin{cases} r_j, \text{ if } i \textit{ terminates at } j+1 \\ r_j + \max_{a'} Q\left(G_{i+1}, a'; \theta\right), otherwise \end{cases}$$

  Perform gradient descent: $\bigtriangledown_\theta \left(y_j - Q(G_i, a_i; \theta)\right)$

  Store action in action sequence: $a_{seq} \leftarrow a_i$

  $y_p = f(G_{i+1}), \quad i++$

if $y_p \neq y_t$ **then**

  return a_{seq}, G'

Policy Model

The policy model determines the attack action that consists of *determining an action type* and *selecting attack objects*. HRAT determines an attack action based on reinforcement learning, specifically deep Q-learning [93].

Algorithm 1 sketches the flow of the deep Q-learning process for the structural attack. We first initialize (Algorithm 1 lines 1-2) a memory list with capacity N to store the experience (i.e., the action type, graph state, and rewards) for further learning with an action-value network Q (a two-layer fully connected neural network). With probability ϵ (0.95 as default [93]), HRAT determines the *action type* using Q; otherwise, HRAT randomly selects one action type(Algorithm 1 lines 7-10). With a determined action type, HRAT uses the gradient search to select the optimal attack objects (i.e., nodes/edges). If the target system uses kNN, which is non-differentiable, as its classifier, we first transform kNN into a differentiable version [125]:

$$Obj_{adv}(G) = \arg min_{\delta} \quad \sum_{i=1}^{m} \omega_i \cdot \sigma\left(\|x_i - T\left(G\right)\|_2\right), \tag{3.5}$$

where m is the number of instances in the training set. $\omega_i = 1$ if the label of x_i equals y_{adv}, otherwise $\omega_i = -1$. $\sigma\left(x\right) = \frac{1}{1+e^{-x}}$ is a sigmoid function, and $T(\cdot)$ is the transformation function that transforms FCGs to feature vectors in the target system. Differentiating the objective function with respect to the edges in graph G_i, we obtain the gradient of each edge (Algorithm 1 line 11). With *action type* and the gradient of each edge, HRAT conducts current action on G_i and obtains the modified graph G_{i+1} and the corresponding reward § 6 (Algorithm 1 line 12). Next, HRAT stores the experience (Algorithm 1 line 13) for further learning and optimizing Q (Algorithm 1 lines 13-15). Each step's experience is potentially used in weight updates, which allows for greater data efficiency [93] (Algorithm 1 lines 7-8). Besides, HRAT stores features $T(G)$ and the latest N experience tuples in the replay memory. This setting that stores feature vectors instead of graphs saves memory to a great extent and

stores the interaction experience. Next, we will introduce how to utilize the gradient search to guide the modification of the graph according to each action type.

Algorithm 2: Adding edge

Input: current graph G, node constraints $\mathbb{C}$

Output: action sequence a_{seq}, new graph G'

```
1  a_seq ← [−1, −1, −1, −1];
2  a_type ← max(NN(s_t));
3  calculate the gradient ∂ of adding edge in G;
4  ∂ = argsort(∂) ;
5  for each edge (v_beg, v_end) in ∂ do
6      if (v_beg ∉ ℂ and v_end ∉ ℂ then
7          connect v_beg and v_end in G; G' = G;
8          a_seq = [0, v_beg, v_end, −1];
9          break;
10 return a_seq, G' ;
```

• **Adding edge.** We first calculate the gradient of each edge in the adjacency matrix. Then, we extract the gradients of all adding edge actions and select the edge, denoted as a two-tuple of nodes $\{v_{beg}, v_{end}\}$, with maximum gradients to add. Notably, when we select $\{v_{beg}, v_{end}\}$, if one node is in the constraints $\mathbb{C}$, i.e., $v_{beg} \in \mathbb{C}$ or $v_{end} \in \mathbb{C}$, we select the edge with the second-largest gradient until both nodes are not in $\mathbb{C}$ (line 5-9 of Algorithm 2). In this way, we can guarantee that the edges in the output modification sequence, a_{seq}, can be modified in the app.

• **Rewiring.** After obtaining the gradient of each edge, we first select the edge, $\{v_{beg}, v_{end}\}$, with the maximum gradient to remove (lines 4-6 of Algorithm 3). Similar to adding edges, we need to make sure $v_{beg} \notin \mathbb{C}$. It is worth noticing that we do not modify the callee (v_{end}) during manipulation. Thus, it does not matter whether v_{end} is in $\mathbb{C}$ or not. Then, to maintain the app's functionality according to Definition 3,

Algorithm 3: Rewiring

Input: current graph G, node constraints $\mathbb{C}$

Output: action sequence a_{seq}, new graph G'

1 $v_{end} \leftarrow -1, v_{tar} \leftarrow -1$;
2 calculate the gradient ∂ of deleting edge in G;
3 $\partial = argsort(\partial)$;
4 **for** *each edge* (v_{beg}, v_{end}) *in* ∂ **do**
5 **if** ($v_{beg} \notin \mathbb{C}$ **then**
6 break;
7 find the intermediate node v_{mid} linking v_{beg} and v_{beg} with the maximum gradient;
8 **if** $z_{mid} \notin \mathbb{C}$ **then**
9 disconnect v_{beg} and v_{end} in G;
10 connect v_{mid} and v_{end} in G;
11 connect v_{beg} and v_{mid} in G;$G' = G$;
12 $a_{seq} = [1, v_{beg}, v_{end}, v_{mid}]$;
13 break;
14 return a_{seq}, G' ;

we select an intermediate node, v_{mid} that has no connection to v_{beg} and v_{end} and is not in $\mathbb{C}$, with the maximum gradient sum of $\{v_{beg}, v_{mid}\}$ and $\{v_{mid}, v_{end}\}$ (lines 8-13 of Algorithm 3).

• **Inserting node.** We create a new method (v_{new}) and calculate the gradient from each manipulable node ($v_{candi} \notin \mathbb{C}$) to v_{new}. Then, the edge $\{v_{candi}, v_{new}\}$ with the maximum gradient is built (line 2-4 of Algorithm 4). Building edge from an existing node to the inserted node guarantees that the static analysis will not exclude the inserted node (method) as the dead code. Finally, we update $\mathbb{C}$ by adding v_{new} with a modifiable index (line 6 of Algorithm 4).

Algorithm 4: Inserting node

Input: current graph G, node constraints $\mathbb{C}$

Output: action sequence a_{seq}, new graph G'

insert a new node v_n to G;

calculate the gradient ∂ of all edges to v_n;

find the edge (v_{beg}, v_n) with the largest gradient and $v_{beg} \notin \mathbb{C}$;

connect v_{beg} and v_n in G; $G' = G$;

$a_{seq} = [2, v_{beg}, v_n, -1]$;

update $\mathbb{C}$;

return a_{seq}, G' ;

Algorithm 5: Deleting node

Input: current graph G, node constraints $\mathbb{C}$

Output: action sequence a_{seq}, new graph G'

calculate the gradient ∂ of all nodes;

find the nodes $v_{tar} \notin \mathbb{C}$ with largest gradient;

remove v_{tar} in G; connect each caller of v_{tar} to all callee of v_{tar} in G; $G' = G$;

$a_{seq} = [3, v_{tar}, -1, -1]$;

update $\mathbb{C}$;

return a_{seq}, G' ;

• **Deleting node.** When deleting a node (v_{tar}), we need to maintain the connectivity between the methods calling v_{tar} and the methods called by the v_{tar}. According to Definition 5, the gradient ($g(\cdot)$) of node i, is defined as:

$$g(i) = \sum_j v_{ij} \cdot g(v_{ij}) + \sum_j (v_{ji} \cdot \sum_k (1 - v_{jk}) \cdot g(v_{jk})), \tag{3.6}$$

where $v_{ij} = 1$ denotes there are existing connections from node i to j, otherwise $v_{ij} = 0$. The first item computes the sum of gradients of all edges leading to $node_i$. The second item computes the sum of gradients of all edges originating from $node_j$, which invokes $node_i$, to $node_k$, which are called by $node_i$. For the node that has the

maximum gradient and is not in the constraint set, we remove it from the adjacency matrix and build the connections from each of its callers to all of its callees (line 3 of Algorithm 5). Finally, we update $\mathbb{C}$ by removing v_{tar}.

Action Space

The action space stores the operations to modify the graph. Each action is represented as a four-tuple that stores the action type (the first element) and action objects (the remaining three elements).

Reward Function

The reward function evaluates the effect of the selected action on the current state, i.e., graph. Since the goal of our attack is to make the system's decision on the modified graph different from the decision on the original graph $f(\hat{G}) = f(\mathcal{K}(G)) \neq f(G)$ by modifying graphs as few as possible, our reward function is designed as follows:

$$\mathcal{R}(s_t, a_t) = \begin{cases} 1 & \text{if } f(\hat{G}) \neq f(G) \\ -(\triangle N_{node} + \triangle N_{edge}) & \text{if } f(\hat{G}) = f(G) \end{cases}, \tag{3.7}$$

where $\triangle N_{node}$ and $\triangle N_{edge}$ denote the differences between the number of nodes and the number of edges in the current graph and the original graph, respectively.

This reward function assesses the impacts of attack action types and attack objects. The reward of adding edge is -1 because only one edge is modified, no matter which edge the gradient search selects. The reward of rewiring and inserting nodes are -3 and -2, respectively. For removing nodes, the reward depends on the node that the gradient search selects. Removing nodes deletes one node v_{tar} from G and builds connections between each of v_{tar}'s callers to all of v_{tar}'s callee. The reward of removing one node is obtained by:

$$\mathcal{R}(\cdot)_{rn} = 1 + N_{caller}^{v_{tar}} \cdot N_{callee}^{v_{tar}} \quad , \tag{3.8}$$

where $N_{caller}^{v_{tar}}$ and $N_{callee}^{v_{tar}}$ are the number of callers and callees of v_{tar}, respectively.

3.3.4 Structural Attack Analysis

We first analyze how graph-based algorithms learn and extract features from FCGs. Malscan uses centralities of sensitive nodes (sensitive APIs [12]) in FCGs to represent graph semantics. Malscan shows that centrality can measure the significance of sensitive nodes in graphs, which can help identify malicious behavior in apps. On the other hand, Mamadroid utilizes function call probability in FCGs as features, and to standardize the size of the resulting vector, the functions are abstracted into various clusters based on families or packages. To unify the size of the extracted vector, Mamadroid abstracts functions into different clusters based on the families or packages.

According to the algorithms adopted by target systems, we analyze the influence of each structural modification on them. For Malscan, we take degree centrality (d_{cen}) as an example:

$$d_{cen\_i} = \frac{d_i}{N_v - 1}, \tag{3.9}$$

where d_i denotes the degree of node i and N_v denotes the number of nodes in a FCG. When we add one edge (method invocation) between from i to any of the other nodes, the deg_i increases, and then the degree centrality of node i increases; and vice versa for deleting one edge; when we add one node to an FCG, N_v increases, and then the degree centrality of node i decreases and vice versa for deleting one node.

For Mamadroid, the function call probability (f_{cp}) is calculated by:

$$f_{cp}(i,j) = \frac{f_N(i,j)}{\sum_{k=1}^{N} f_N(i,k)}, \tag{3.10}$$

where $f_N(i,j)$ denotes the number of callers from state i to j, and N is the total number of states (i.e., the number of method families). In this way, when we insert one edge from state i to j, the numerator and denominator increase by 1 and the

```
1. <packageName.className returnType
   methodName(paraList)>
```

Figure 3.3: Android method signature in soot

$f_{cp}(i, j)$ increases and vice versa for deleting one edge; when we delete one node k from state i, all callers to k will lose. To preserve the functionality, those callers integrate the code of k and invoke the callees of k. In this way, $f_N(i, j)$ decreases, $\sum_{k=1}^{N} f_N(i, k)$ increases and then the probability from state i to j decreases.

Structural attacks bridge the gap between feature-space attacks, which only perturb feature vectors to deceive the classifier, and problem-space attacks, which generate adversarial objects, i.e., real Android apps. It is also known as the inverse feature-mapping problem [107]. Existing problem-space attacks [107, 110, 26] are limited by inverse feature-mapping problems (i.e., the optimized feature-space attacks cannot be perfectly mapped into problem-space attacks), which can also cause side effects and decrease the attack success rate. Structural attacks modify the nodes and edges in the FCG that contain the methods and call relations in an app. Hence, our four FCG modification actions correspond to the manipulations on apps (§3.4). Meanwhile, the extracted FCGs from the crafted software are consistent with the corresponding modified FCGs. In other words, based on the modification sequence calculated by our attack algorithm, we locate the methods or call relations in bytecodes and modify them correspondingly. Compared with the existing methods, our structural attack considers not only the operations of the inserting method and call relations, but also the operations of deleting, which makes our attack more comprehensive and feasible at the programming level.

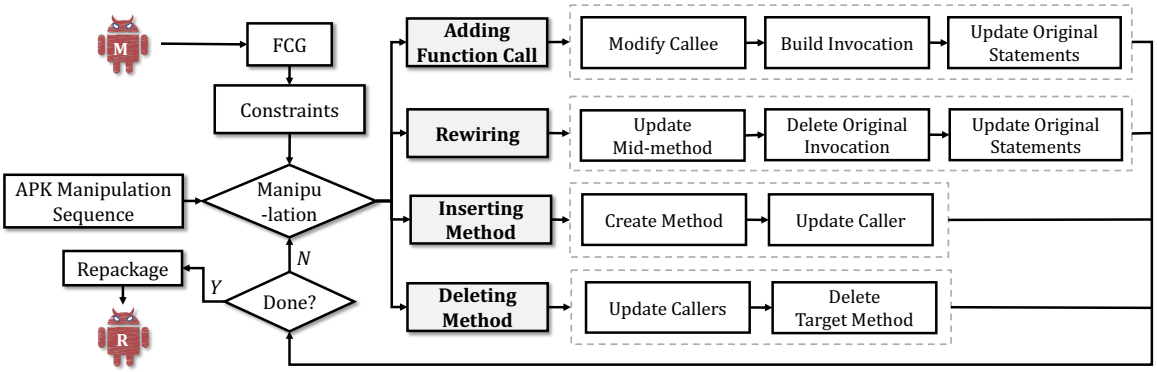


Figure 3.4: Work flow of APPMOD

3.4 Android Application Manipulation

Our Android app modifier (APPMOD) automatically manipulates an app according to the graph modification sequence following two principles: *a)* ***Functional Consistency***: the app's functionality before and after modifications should be consistent; and *b)* ***Valid Modifications***: the inserted code will not be identified and removed by static analysis. Specifically, static analysis can detect dead code that will never be executed [107] and remove it. In this case, the graph extracted from the manipulated app will not include the corresponding nodes or edges, i.e., the modifications are invalid.

The prototype of app modifier in HRAT (i.e., APPMOD) is built on *soot* [142]. APPMOD modifies apps using soot, which translates Android bytecode to an intermediate representation without the need for apps' source code. According to method signatures, as shown in Figure 3.3, APPMOD locates the methods in the app and conducts the modification accordingly. Figure 3.4 sketches the workflow of APPMOD. Next, we introduce how APPMOD implements the four manipulation operations: adding function call (adding edge), rewiring function calls (rewiring), inserting method (inserting node), and deleting method (deleting node). We first introduce how to determine the constraints, which define whether the methods are modifiable by *soot*.

3.4.1 Constraints Determination

Constraints list the methods in apps and their modifiability. When HRAT modifies nodes and edges in FCGs, the constraints guide HRAT to only modify modifiable methods and call relations. We determine unmodifiable methods according to the properties of Android, *Flowdroid* [10], and *soot*. The following methods are unmodifiable:

• **Framework APIs**: Android framework APIs are pre-defined sets of classes, interfaces, and protocols provided by the Android operating system to help developers build mobile applications. These APIs allow developers to access system-level functions and services, such as location services, telephony, camera, and storage. The APIs are defined and implemented in the Android system rather than the app, so they are unmodifiable by soot when analyzing apps.

• **Lifecycle methods**: Android lifecycle methods are a set of callback methods provided by the Android framework to manage the lifecycle of an activity or fragment. These methods allow developers to perform certain actions at different stages of an activity's or fragment's lifecycle, such as when it is first created, started, resumed, paused, stopped, or destroyed. By using these methods, developers can manage the memory usage of their apps, save and restore the state of the user interface, and handle different configuration changes, among other things. Lifecycle methods can be invoked by the Android system. If we delete or add the connection to the lifecycle method, such modifications may lead to the app crash.

• **Flowdroid methods**: Since FlowDroid introduces additional methods (e.g., fake *main* method [11]) to facilitate the analysis, the FCGs will include these methods. However, it is worth noting that the apps under investigation do not include such auxiliary methods.

```
1. packageName1.className1 returnType1
   caller(parameterList1){
2.     callee(parameterList2, True);
3.     ... raw caller method body  ... }
```

(a) Modified target caller.

```
1.  packageName2.className2 returnType2
    callee(parameterList2, FLAG){
2.   if (FLAG == True){
3.      return defaultValueofReturnType2;
4.   }else{
5.      ... raw callee method body ... }}
```

(b) Modified callee.

```
1.  returnTypei oriCaller(paraListi){ ...
2.    tmp = callee(paraList2);
3.    tmp = callee(paraList2, False);...}
```

(c) Modified callers of original callee.

Figure 3.5: Pseudo code of adding function call.

3.4.2 Adding Function Calls

Adding function calls manipulates two methods (i.e., a *caller* and a *callee*) and all statements that invoke *callee*. The pseudo-code for adding a function call is illustrated in Figure 3.5, where the blue code indicates the inserted code, and the gray code represents the removed code. To add a function call, we insert an extra parameter **FLAG** in *callee* (line 1 of Figure 3.5(b)), and then insert a statement to invoke *callee* in *caller*'s method body. To keep the functional consistency, APPMOD inserts a conditional expression on **FLAG** in *callee*'s method body (line 2 of Figure 3.5(a)). If **FLAG** equals **True**, it indicates the invocation is an inserted call, and *callee* directly returns a default value that has the same type as that of *callee*'s return value (line 2-3 of Figure 3.5(b)). If **FLAG** equals **False**, it indicates the invocation is an original call, and the *callee* runs as usual. Besides, to maintain functional consistency, APPMOD

modifies all statements that initially invoke *callee* and sets **Flag** to **False** (e.g., line 2-3 in Figure 3.5(c)).

```
1.  class newType{
2.   returnType1 rt1;  returnType2 rt2; }
```

(a) Insert new data type.

```
1.   ~~returnType1~~newType imm(paraL3,
    paraL2, FLAG){
2.    vNt = new newType;                ...
3.    if (FLAG == True){
4.        vRt.rt2 = callee(paraL2);
5.        return vRT; }
6.    else{ raw caller method body; } ...
7.    ~~return vRt1;~~   vNt.rt1 = vRt1;
8.    return vNT;  }
```

(b) Modified intermediate method.

```
1.  returnType2 caller(paraL){  ...
2.     ~~v1 = callee(paraL2);~~
3.     vtmp = imm(paraL3, paraL2, True);
4.     v1 = vtmp.rt2;            ...    }
```

(c) Modified caller.

```
1.  returnType2 oneImmCaller(paraL1){ ...
2.     ~~tmp = imm(VarL3);~~
3.     vNt = new newType;
4.     vNt = imm(VarL3,defaultRt2,False);
5.     tmp = vNt.rt1;    ... }
```

(d) Modified original callers of intermediate method.

Figure 3.6: Pseudo code of rewiring.

3.4.3 Rewiring Function Calls

According to Definition 3, we implement rewiring by modifying three methods: a *caller*, a *callee*, and an *intermediate method* (*imm*). This operation includes three

steps: **a)** build call relations between *imm* and *callee*, **b)** replace call relations between *caller* and *callee*, and **c)** update the original call statements.

• **Step a).** Figure 3.6(b) illustrates a modified intermediate method. APPMOD adds an extra parameter **FLAG** to *imm*'s parameter list to determine whether the invocation is an original one or an intermediate invocation (i.e., *caller* invokes *imm*). If it is an original invocation, *imm* runs its original method body (line 6 of Figure 3.6(b)); otherwise, it invokes *callee* (line 3-5 of Figure 3.6(b)). Note that the *imm*'s return type and *callee*'s return type may be inconsistent. To handle this issue, APPMOD introduces a new data type (i.e., *newType* in Figure 3.6(a)) that includes both *imm*'s and *callee*'s return types. When *imm* needs to return data, APPMOD assigns the original data to the object of *newType* and then returns the object to the caller (line 8 in Figure 3.6(b)). To ensure that *callee* works as original, we extend the parameter list of *imm* and pass the *caller*'s variables used for invoking *callee* to *imm* when invoking *imm* (line 3 of Figure 3.6(c)).

• **Step b).** APPMOD modifies *caller*'s function body and replaces the statements that originally invoke *callee* with new statements to invoke the intermediate method. Then, APPMOD sets **FLAG** to **True** to indicate that the invocation is an original one. Then, *caller* obtains the return value of *callee* (line 3 of Figure 3.6(c)).

• **Step c).** As the method signature (i.e., parameter list and return type) of *imm* is changed, APPMOD finds all statements that invoke *imm* and updates them accordingly. Specifically, APPMOD first locates the invocation statements and adds the parameters of *callee* with their default value, e.g., 0 for *Integer* (line 4 of Figure 3.6(c)), to the end of the *imm*'s parameter list. Then, APPMOD sets **FLAG** to *False*, which indicates an original invocation (line 4 of Figure 3.6(d)). In this way, when those methods invoke *imm*, *imm* runs as usual (line 6 of Figure 3.6(b)), and the functional consistency of *imm* is preserved.

```
1.  packName.className int
    newMethod_i(int i1,int i2){
2.      int i3;    i3 = i1 + i2;
3.      return i3; }
```

(a) Create a new method.

```
1.  returnType caller(paraList){
2.      int i;
3.      i = newMethod_i(1, 1);
4.      i = i + 1;
5.      ... raw caller method body  ... }
```

(b) Modified caller to invoke new method.

Figure 3.7: Pseudo code of inserting methods.

3.4.4 Inserting Methods

This operation first creates a new method and then finds one existing method (i.e., *caller*) to invoke it. APPMOD creates a method that performs simple mathematical calculations and returns the results (Figure 3.7(a)). Then, APPMOD inserts an invocation statement in *caller*'s method body (line 3 in Figure 3.7(b)). The *caller* gets the returned value of the inserted invocation, and uses the returned value to perform mathematical calculations in *caller*'s method body (line 4 in Figure 3.7(b)), so that the inserted method will neither be excluded as dead code nor affect the app's functionalities.

3.4.5 Deleting Methods

This operation removes the target method (*tarMethod*) and modifies all methods that invoke it. Figure 3.8 shows the pseudo-code for deleting *tarMethod*. First, APPMOD finds all methods that invoke *tarMethod* and locates the corresponding invocation statements. Then, APPMOD replaces the invocation statements with *tarMethod*'s method body. More specifically, APPMOD firstly creates local variables

```
1.  returnType1 tarMethod(para1, para2){
2.      var1 = para1;  var2 = para2;  ...
3.      { method body of tarMethod }  ...
4.      return var3;                   }
```

(a) Target method to be deleted.

```
1.  returnType2 oneOfCaller(paraL){ ...
2.      v_rt = tarMethod(v1, v2);
3.      var1 = v1;          var2 = v2;
4.      { method body of tarMethod }
5.      v_rt = var3;          ...       }
```

(b) One of modified methods that calls target method.

Figure 3.8: Pseudo code of deleting methods.

(*lv_caller*) in *caller* that include *tarMethod*'s parameter variables (*pv_tar*) and local variables (*lv_tar*). Then, APPMOD assigns the variables used for invoking *tarMethod* to *pv_tar* (lines 2-3 in Figure 3.8(b)). Since we recreate *tarMethod*'s variables in *caller*, which are different from those originally used by *tarMethod*, APPMOD needs to rewrite the *tarMethod*'s statements rather than directly copying the statements from *tarMethod* to *callers*. For example, APPMOD uses soot's APIs *newAssign-Stemt()*, *newInvokeStmt()* to rewrite the assign statements and invoke statements with *newVar*, respectively. Moreover, if *tarMethod* has a return value, APPMOD replaces the return statement with an assignment statement (line 4 of Figure 3.8(a) to line 5 of Figure 3.8(b)). Since the return statements will end the invocation, if the precondition of one return statement is met, the program will end the invocation directly. Besides, one method may contain multi-return statements. To avoid affecting the invocation logic, when APPMOD replaces one return statement, APPMOD inserts a *goto* statement to let the program jump to the next statement of the statements (i.e., line 5 in Figure 3.8(b)) that initially invoke *tarMethod*.

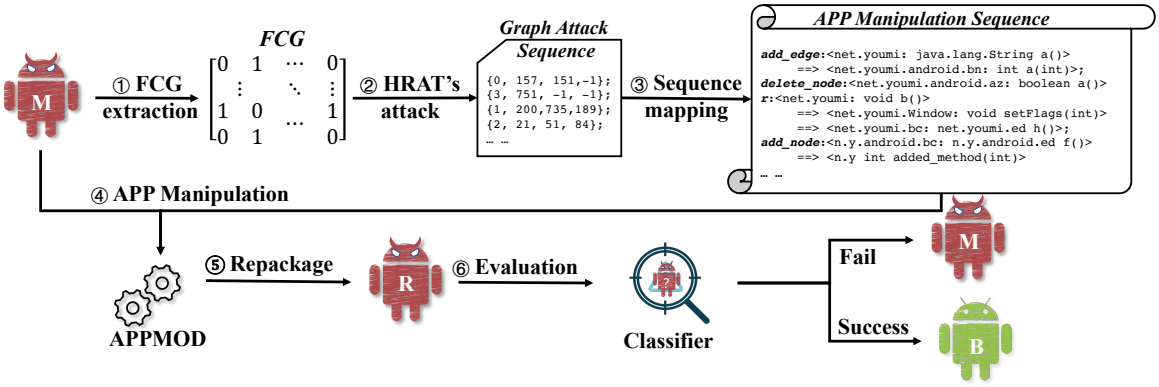


Figure 3.9: Work flow of HRAT

3.5 Evaluation

Figure 3.9 sketches the attack flow of HRAT. Given an app, HRAT first extracts its FCG ① and generates a graph modification sequence ②, where each item indicates a perturbation on the FCG's nodes or edges. Then, we convert a graph modification sequence to the manipulation sequence where each item denotes a manipulation operation on app ③. APPMOD is a process of modifying an existing mobile application by using a sequence of manipulations ④ to create a new, adversarial version of the app ⑤. This process involves analyzing the original app for weaknesses and vulnerabilities, and then using various techniques to modify its code. Once the app has been modified, it is repackaged to create a new app package that includes the modified code, as well as any additional components required for the app to function. The resulting adversarial app is then distributed to unsuspecting users, with the aim of exploiting their devices and stealing their personal information. Finally, we evaluate whether target systems (i.e., Malscan, Mamadroid, and APIGraph enhanced Malscan) can detect the adversarial malware ⑥. We evaluate the performance of HRAT and investigate potential defense methods against HRAT by answering the following seven research questions.

• **RQ1: Effectiveness analysis.** *How effective is HRAT against the state-of-the-art*

AMD techniques?

• **RQ2: Modification efficiency comparisons.** *Compared with other attack methods, how is the modification efficiency of HRAT?*

• **RQ3: Effectiveness of IMA.** *How effective is individual modification action (IMA)?*

• **RQ4: Resilience to code obfuscation.** *How is the resilience of HRAT to malware with different obfuscation techniques?*

• **RQ5: Functional consistency assessment.** *Do the adversarial apps generated by HRAT preserve the functionality as the original ones?*

• **RQ6: Influence of key parameters.** *Will the parameters influence HRAT's attack performance?*

• **RQ7: Defense against HRAT.** *How to defend against HRAT's attack?*

Dataset. We adopt the dataset that includes 11,613 benign Android apps and 11,583 malicious Android apps from 2011 to 2018 in Malscan [152] to evaluate HRAT (for RQ1-3&RQ5). All apps are collected from AndroZoo [4], and each sample has been detected by several antivirus systems in VirusTotal [143] to determine its label. For RQ4&5, we use a dataset from previous work [37] to evaluate the effectiveness of HRAT on malware using different obfuscation. This dataset includes apps from different malware families, and 6,586 malware are obfuscated by variable renaming [114], 1,090 malware are obfuscated with string encryption [41], and 1,172 malware include reflection [37].

Metrics. To evaluate the effectiveness of the attacks on both feature space and problem space, we use three types of attack success rates (ASRs), i.e., Initialization ASR ($Init\_ASR$), Relative ASR ($Rela\_ASR$) and Absolute ASR (Abs_ASR), as our

evaluation metrics, which are defined as follows:

$$\begin{aligned} Init\_ASR &= \frac{N_g}{N}, \\ Rela\_ASR &= \frac{N_p}{N_g}, \\ Abs\_ASR &= \frac{N_s}{N_p}. \end{aligned} \tag{3.11}$$

The detailed definitions are as follows:

• *Initialization ASR* (*Init_ASR*) evaluates HRAT's effectiveness on feature space. The feature-space attack denotes only modifying the structure of FCGs to escape the detection of classifiers in target detectors. Given N malware samples, HRAT perturbs the FCGs of N_g malware samples with at most 500 modifications and makes them successfully escape detection. The threshold (i.e., 500 modifications) selection can refer to RQ2. We found that nearly 100% malware samples can escape detection within 500 manipulations by HRAT. Note that not all modified FCGs can be repackaged due to the anti-repackage protection [169].

• *Relative ASR* (*Rela_ASR*) reflects the rate of successful repackaged malware. That is, among N_g malware samples, N_p samples can be successfully repackaged into app files.

• *Absolute ASR* (*Abs_ASR*) evaluates HRAT's effectiveness on problem space, i.e., whether the repackaged samples can evade the detection and keep the functionality. Among N_p repackaged malware samples, N_s samples run successfully and evade the detector.

Baselines. In addition to HRAT, we also employ evolutionary algorithms to design structural attacks and implement the attack process. These methods serve as baselines for comparison, allowing us to evaluate the effectiveness of different attack strategies and identify areas for improvement. We also evaluate evolutionary algorithms, specifically simulated annealing [72], hill-climbing [127] and evolutionary programming[162], for comparisons. Evolutionary algorithms are suitable for scenar-

ios whose policy space is small or can be structured [137]. However, in our attack scenario, the policy space, i.e., the attack action set, is enormous and unable to be structured. We adopt the idea of evolutionary algorithms and adapt them to our scenarios. Combining gradient search, we design three evolutionary algorithms-based structural attacks: 1) Simulated Annealing based structural attaCK(SACK) (Algorithm 6), 2) Hill-climbing based structural AttaCK (HACK) (Algorithm 7) and 3) Evolutionary Programming based structural AttaCK (EPACK) (Algorithm 8). Next, we use HACK as an example to introduce how we use evolutionary algorithms to guide a structural attack.

During the initialization phase, the system's behavior is determined by several critical parameters, including the initial temperature, target temperature, and temperature drop ratio. These parameters must be carefully chosen to ensure that the system functions effectively. To initialize the system's state, we begin by randomly selecting an action type from among four candidate types. We then use the gradient search to modify the system's graph based on the selected action type, ensuring that the system is primed for optimal performance. Unlike the original simulated annealing (SA) algorithm, where each step randomly selects a state from among its neighbors, our approach involves a more targeted initialization process that leverages gradient search and candidate action types to create a more effective system. To identify the system's neighbors, we follow a three-step process. First, we extract the corresponding representation feature for the latest graph. Second, we conduct all possible modifications to the graph, generating a set of candidate states. Third, we extract features from all modified graphs, creating a candidate state set. The next step in the process involves selecting the nearest neighbor from the candidate state set using a predefined distance formula, such as Euclidean distance. However, this method is not feasible in practice, as the number of all possible modifications is very large, even when the number of nodes is not increased. Assuming that there are N nodes in the

Algorithm 6: SACK: simulated annealing-based structural attack

Input: Target classifier f, training dataset $\{X_t, Y_t\}$, target FCG G, feature-space transformation T, maximum modification times M, node constraints $\mathbb{C}$, cooling ratio r

Output: action sequence a_{seq}, adversarial graph G'

Initial and final temperature T_i and T_f

Initialize: $i = 0, \quad T = T_i$, $G_i = G$, $\ y_t = f(G), y_p = f(G)$, $cost_i = min(dist(f(G), X_t))$

while $T > T_f$ *and* $i < M$ *and* $y_t == t_p$ **do**

- $a_i = random\_type$
- Calculate gradient of each edge: $\partial_G = \bigtriangledown_{G_i} Obj_{adv}(G_i)$
- Execute action a_i on G_i: $G_{i+1} = ATT\_OBJ(G_i, \partial_{G_i}, \mathbb{C})$
- $cost_{i+1} = min(dist(f(G_{i+1}), X_t))$
- $prob = exp(-(cost_{i+1} - cost_i)/T)$
- **if** $cost_{i+1} < cost$ *or* $rand\_num < prob$ **then**
  - Store action in action sequence: $a_{seq} \leftarrow a_i$
  - $G_i = G_{i+1}, \ cost = cost_{i+1}$
- $T = T * r$
- $y_p = f(G_i), \quad i++$

if $y_p \neq y_t$ **then**

- **return** a_{seq}, G'

target graph, the number of all possible modifications is:

$$N_{pmod} = C^1_{N\times N} \times (C^1_{N\times N} \times C^1_{(N-2)\times(N-2)}) \times (C^1_N), \tag{3.12}$$

where C^j_i is the combination. The first item calculates the number of possible situations of *adding edges*, the second item calculates the number of possible situations of *rewiring*, and the third for *deleting nodes*. Considering the malware that has 1,000 nodes and 20% of nodes are modifiable, we have over 12 quadrillion possible mod-

Algorithm 7: HACK: hill-climbing based structural attack

Input: Target classifier f, training dataset $\{X_t, Y_t\}$, target FCG G, maximum modification times M, node constraints $\mathbb{C}$

Output: action sequence a_{seq}, adversarial graph G'

1 Initialize: $i = 0, \quad G_i = G, \;\; y_t = f(G), y_p = f(G), \;\; cost_i = min(dist(f(G), X_t))$
2 **while** $i < M \;\; and \; y_t == t_p$ **do**
3 $\quad a_i = random\_type$
4 $\quad$ Calculate gradient of each edge: $\partial_G = \bigtriangledown_{G_i} Obj_{adv}(G_i)$
5 $\quad$ Execute action a_i on G_i: $G_{i+1} = ATT\_OBJ(G_i, \partial_{G_i}, \mathbb{C})$
6 $\quad cost_{i+1} = min(dist(f(G_{i+1}), X_t))$
7 $\quad$ **if** $cost_{i+1} < cost$ **then**
8 $\quad\quad$ Store action in action sequence: $a_{seq} \leftarrow a_i$
9 $\quad\quad G_i = G_{i+1}, \;\; cost = cost_{i+1}$
10 $\quad y_p = f(G_i), \quad i++$
11 **if** $y_p \neq y_t$ **then**
12 $\quad$ **return** a_{seq}, G'

ifications. Besides, as we can insert arbitrary numbers of nodes, the scale of the candidate solution set is infinite. Thus, we randomly select an action type and use the gradient search to conduct the action on the graph, which also ensures the fairness of comparison with HRAT. We define the *cost* as the nearest distance from the modified graph to benign graphs in the training set (Algorithm 6 line 7). This setting follows the intuition that our target is to deceive kNN classifiers. Then, if the latest solution is better than the previous one, i.e., $cost_{i+1} < cost_i$, SACK will adopt the attack action. Otherwise, SACK will adopt the attack action with a probability less than $exp(-(cost_{i+1} - cost_i)/T)$. SACK continues to modify the graph based on the previous state. Different from SACK, HACK adopts all actions that have positive impacts on the state. In each epoch, EPACK randomly generates the mutation prob-

Algorithm 8: EPACK: evolutionary programming based structural attack

Input: Target classifier f, training dataset $\{X_t, Y_t\}$, target FCG G, maximum modification times M, node constraints $\mathbb{C}$, number of action types n_a

Output: action sequence a_{seq}, adversarial graph G'

Initialize: $i = 0, \quad G_i = G, \;\; y_t = f(G), y_p = f(G), \;\; cost_i = min(dist(f(G), X_t))$
while $i < M \;\; and \;\; y_t == t_p$ **do**
    Randomize mutation probability: $p_m = p_1, ..., p_{n_a}$
    $G_{i+1} = G_i$, tmp_{act}
    for $a_i = 0 \;\; to \;\; a_i < n_a$ **do**
        if $p_{a_i} > 1/n$ **then**
            Calculate gradient of each edge: $\partial_G = \bigtriangledown_{G_i} Obj_{adv}(G_{i+1})$
            Execute action a_i on G_i: $G_{i+1} = ATT\_OBJ(G_{i+1}, \partial_{G_{i+1}}, \mathbb{C})$
            $tmp_{act} \leftarrow a_i$
    $cost_{i+1} = min(dist(f(G_{i+1}), X_t))$
    if $cost_{i+1} < cost$ **then**
        Store action in action sequence: $a_{seq} \leftarrow tmp_{act}$
        $G_i = G_{i+1}, \;\; cost = cost_{i+1}$
    $y_p = f(G_i), \quad i++$
if $y_p \neq y_t$ **then**
    return a_{seq}, G'

ability of each attack action. Then, EPACK adopts all attack actions whose mutation probability is larger than $1/n_a$. n_a is the number of all action types.

3.5.1 RQ1: Effectiveness Analysis

Experimental Setup. We divide the dataset into training sets and testing sets. As the goal of HRAT is to modify malware to evade the target detection system, the

Table 3.1: ASRs of HRAT towards Malscan, Mamadroid, and APIGraph enhanced Malscan

Algorithm	Training	Testing	Init ASR	Rela ASR	Abs ASR
Malscan	TRo	TEo	82.50%	91.31%	100%
		TE1	94.23%	96.71%	100%
		TE2	97.83%	93.86%	100%
	TR2	TE1	91.50%	97.81%	100%
APIGraph + Malscan	TRo	TEo	89.67%	97.50%	100%
		TE1	99.57%	98.93%	100%
Mamadroid	TRo	TEo	71.42%	87.88%	100%
		TE1	99.94%	94.95%	100%
		TE2	100%	88.79%	100%

testing sets consist solely of malware samples. To ensure that our adversarial attack method is effective, we exclude misclassified malware samples from our dataset by using pre-trained classifiers, as modifying these samples would not be useful. We design three dataset [152] settings for effectiveness analysis. In the first setting, the training dataset (TRo) and the testing dataset (TEo) are collected during the same period. In the second setting, the testing data (TE1 and TE2) was collected after the training set. This setting emulates the situation that the malware detectors are trained with known malware and use pre-trained classifiers to detect malware. In this case, since there may be concept drift [13] in the malware samples, detectors are suggested to retrain their classifiers to deal with new malware. The third setting uses the latest malware to train the classifier (TR2) and uses older malware (TE1) for testing. In this setting, we aim to evaluate whether our attack can renew outdated malware. We use each training set to train target AMDs (i.e., Malscan, APIGraph enhanced Malscan and Mamadroid). Given malware in testing sets, we use HRAT to

modify it and then evaluate whether it can evade target AMDs and record each ASR.

We also compare the performance of HRAT and that of other approaches. One is AndroidHIV [26], which is the state-of-the-art attack against Mamadroid and also considers problem-space attacks. Since AndroidHIV has not been released to the public, we implement AndroidHIV by strictly following the description and configurations in the manuscript[26]. We also design attack strategies based on evolutionary algorithms [72] (i.e., simulated annealing-based structural attack, SACK, hill-climbing-based structural attack, HACK, and evolutionary programming-based structural attack, EPACK, as baseline algorithms for comparisons with reinforcement learning adopted by HRAT.

Results. Table 3.1 lists the results of the effectiveness comparison of different attacks. *Init ASR* represents the ratio of malware that escapes detection after at most 500 modifications have been applied. According to our analysis in §3.3.4, the characteristics of malicious apps will eventually be diluted so that they will be regarded as benign ones as long as HRAT keeps adding useless vertices. But the unlimited modifications will increase the attack's computational complexity. We apply HRAT to 50 randomly selected apps and find that after 500 modifications, these apps were still unable to evade detection. We then continued to apply HRAT without any restrictions on the number of modifications until the apps were successfully able to evade detection. The result shows that these apps can successfully evade detection after more modifications (i.e., from 635 to 4,091).

Due to the limitations of soot and flowdroid [107], some apps cannot be successfully repackaged. Thus, we utilize *Rela_ASR* to denote the ratio of apps that can evade the detection at the algorithm level but cannot be repackaged successfully. It is worth noting that the failure of app repackaging is typically due to the anti-repackaging strategies implemented by the apps themselves, rather than any shortcomings in our manipulation techniques.

Comparing the Init ASR and Rela ASR of Malscan and APIGraph enhanced Malscan in Table 3.1, we can see that HRAT is more effective on APIGraph enhanced detector than the original detector. The reason for this is that APIs that were originally unmodifiable may become modifiable through the use of APIGraph, which uses a unique API to represent APIs with similar functionalities. Furthermore, we have observed that it is easier to obscure the distinguishing features of malware and evade detection when the number of features is smaller. For example, the feature number of Mamadroid (121) is much smaller than that of Malscan (43,972), and the Init ASR on Mamadroid in Table 3.1 is better than Malscan.

Table 3.2 illustrates the ASR of different algorithms using TRo for training and TE1 for testing. We can see that for feature-space attack (Init ASR), AndroidHIV can achieve over 95% attack success rate. However, in the case of problem-space attacks, the performance of AndroidHIV drops to 37%, as it can only modify a limited number of methods in malware and cannot maintain consistency between perturbations on features and modifications made to apps. Furthermore, the initial ASRs for SACK, HACK, and EPACK are lower compared to reinforcement learning-based algorithms, since evolutionary algorithms direct structural attacks to choose attack action types randomly. Benefiting from structural attacks, EA-based methods achieve the same absolute ASRs as reinforcement learning-guided attacks.

Next, we explore the impact of randomness on the experiments. HRAT uses a k-nearest neighbor (kNN) algorithm with k=1 as the classifier for the target detection system. The way the training data is split significantly influences the attack performance. For instance, the label assigned to a test sample is determined by its nearest neighbor in the training set, meaning any changes in the training data can alter classification results. Consequently, variations in the testing data split also affect the attack performance. HRAT's action determination model is randomly initialized, and actions are selected based on a predefined threshold, even if the model is well-trained. This inherent randomness can lead to varied performance outcomes when

HRAT is applied in real-world scenarios. For comparison, SACK employs a random selection for the action space due to the vast initialization settings, as discussed in *Section3.5-Baselines*. This randomness in action selection introduces variability in performance during application. Similarly, EPACK uses a random mutation probability for determining attack actions, which also introduces randomness. Different target detection systems exhibit varying sensitivities to randomness. In Malscan, the features are sparse, so perturbations may differently affect the sparse features' representation. Mamadroid relies on the dependency probability between different method families' invocations. A single manipulation may impact these probabilities by altering both the numerator and the denominator in Mamadroid's feature calculation. For APIGraph-enhanced Malscan, feature clusters are used to reduce Malscan's feature vector. Random perturbations in nodes and edges can change the centrality of multiple clusters, impacting the overall performance. The inherent randomness in data splitting, action determination, and feature perturbations can significantly influence the performance of HRAT's attacks and the comparison algorithms, leading to varied outcomes in different application scenarios.

Answer to RQ1: HRAT achieves up to 99.94% init ASR within 500 modifications. Without restriction on the number of modifications, HRAT achieves a higher ASR of problem-space attack. HRAT outperforms evolutionary algorithms that demonstrate the effectiveness of optimization strategies in HRAT.

3.5.2 RQ2: Modification Efficiency Comparison

Experimental Setup. We measure the modification efficiency of an attack using the number of modifications required to let a malicious app evade detection. We compare the modification efficiency of HRAT and that of AndroidHIV and SACK on the collected dataset by recording the escaping number of modifications for each app toward the different attack approaches. Note that HRAT and the aforementioned

Table 3.2: Effectiveness comparison of different attacks

Systems	Algorithms	Init ASR	Rela ASR	Abs ASR
Malscan	HRAT	**94.23%**	96.71%	100%
	SACK	**87.13%**	62.56%	100%
	HACK	75.80%	97.89%	100%
	EPACK	76.63%	94.21%	100%
Mamadroid	HRAT	**99.94%**	94.95%	100%
	SACK	81.05%	84.30%	100%
	HACK	87.40%	81.01%	100%
	EPACK	72.60%	99.17%	100%
	Android HIV	**96.02%**	87.64%	**37.67%**

evolutionary attacks (i.e., SACK, HACK, and EPACK) work against both Malscan and Mamadroid, whereas AndroidHIV only targets Mamadroid.

Results. Figure 3.10 shows the cumulative distribution (CDF) of the required number of modifications for evasion. We can see that even SACK achieves comparative ASR with HRAT (§3.5.1), SACK requires more modifications to make target malware escape detection. Specifically, when attacking Malscan through SACK, more than 10% of malware needs more than 50 modifications. By contrast, this ratio is only about 5% under HRAT's attack. When attacking Mamadroid through HRAT, 90% of the malware needs at most 50 modifications to evade detection. However, SACK requires at least 150 modifications to achieve the same ratio. This difference may be caused by the different learning strategies of those two algorithms. More precisely, HRAT uses reinforcement learning to learn and decide the action type by interacting with the target environment, whereas SACK randomly selects the action type and decides whether to adopt the action by checking if the selected action has a positive impact. For AndroidHIV, nearly 30% of adversarial malware escapes detec-

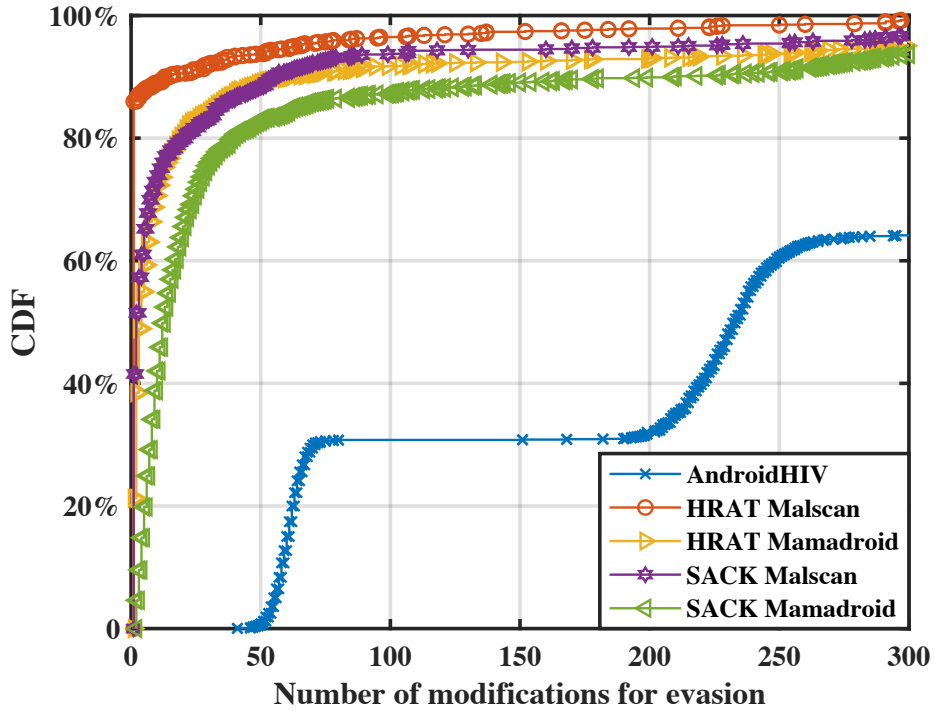


Figure 3.10: CDF of the required number of modifications

tion within 80 modifications, and only 65% of malware deceives Mamadroid within 300 modifications.

Answer to RQ2: HRAT needs fewer number of modifications than other methods to let malicious apps evade detection.

3.5.3 RQ3: Effectiveness of IMA

Ratio of individual actions

To evaluate the effectiveness of individual manipulation action (IMA), we compute the ratio of each attack action to all modifications applied to all adversarial apps that successfully evade the target systems (i.e., Mamadroid, Malscan and APIGraph enhanced Malscan) under the aforementioned data sets and configurations. For example, to obtain the ratio of *insert node* in *Malscan_TRo_TE1*, we first count the

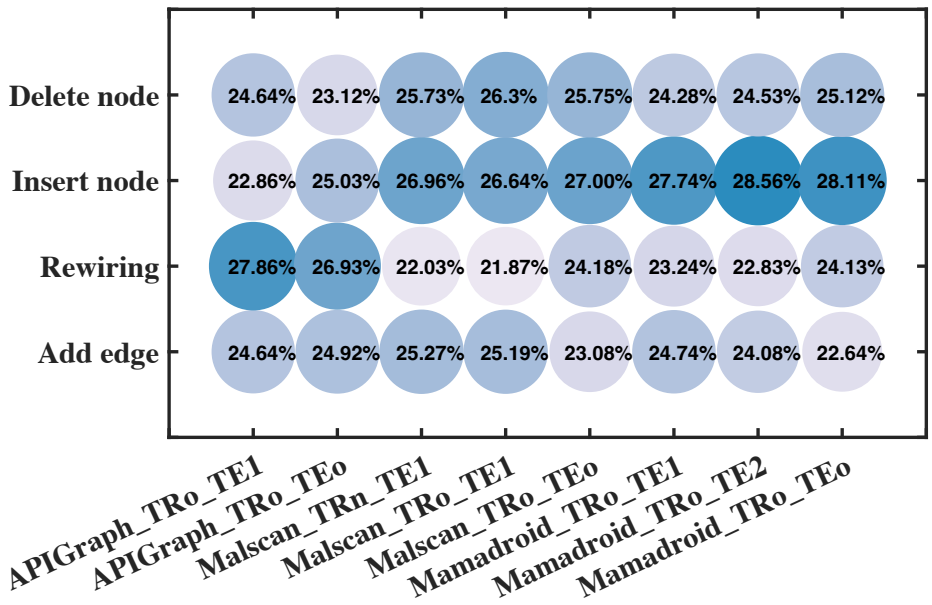


Figure 3.11: The ratio of each attack action.

number of *insert node* (denoted as N_{an}) applied to those adversarial apps and the total number of modifications (denoted as N_m) and the ratio will be computed using N_{an}/N_m.

Results. Figure 3.11 illustrates the ratio of each attack action in adversarial samples. The deeper the shade, the greater the ratio. We can see that for evading Malscan, *inserting nodes* and *deleting nodes* actions account for a large proportion (over 26%) because adding edges and deleting nodes can effectively decrease the degree centrality of nodes (§ 3.3.4). It is consistent with the analysis of Malscan in [152] that suggests the degree centralities of benign apps are smaller than that of malware. For API-Graph enhanced Malscan, rewiring action accounts for a large ratio (over 27%). The reason may be that as APIGraph clusters methods with similar semantics into one class, rewiring action can effectively decrease the degree centrality of target clusters by replacing the connections between different clusters with connections within one cluster. As Mamadroid abstracts methods into different families and uses invocation probabilities as features, inserting nodes to specific families could be more effective

to perturb the features. Besides, as the action ratios of all actions exceed 20%, it suggests that HRAT actively selects attack actions to achieve the trade-off between ASR and modification efficiency.

ASR of individual actions

To evaluate whether malicious apps can escape detection with only one type of attack action, we compare the ASR of HRAT and that of IMA attacks against the aforementioned detectors. TRo and TE1 are adopted as training and testing sets, respectively. More precisely, for each attack action, we just use it to perturb the structure of target graphs and record its ASR. We also compute the average number of modifications (*Avg. Mod*) required by each IMA by first counting the number of modifications (N_{mod}) applied to N_ad adversarial samples that successfully evade the detectors in both feature space and problem space and then calculating the ratio of N_{mod}/N_{ad}.

Results. Table 3.3 shows that using *adding edges* alone to attack Malscan can achieve over 90% *Init ASR* but requires 11.03 average modifications that are nearly double of the number of modifications required by HRAT (5.58). Regarding Mamadroid, both *adding edges* and *inserting nodes* can achieve over 90% *Init ASR* in feature space, while *rewiring* and *deleting nodes* only achieve 83.95% and 78.49% *ASR*, respectively. However, *adding edges* and *inserting nodes* require more average modifications than *rewiring* and deleting nodes. Combing different attack actions together, HRAT achieves nearly 100% *Init ASR* against Mamadroid with much better modification efficiency. As APIGraph clusters similar APIs into one class to enhance target AMD, perturbations on fewer APIs can let malware escape the enhanced detector. In other words, when APIGraph enhances target systems, it also introduces new vulnerabilities. As all IMAs follow our graph structure modifications, the performance of those attack methods of the problem-space attacks is the same as that

Table 3.3: Comparisons between individual attack strategies and HRAT

Algorithm		Init ASR	Rela ASR	Abs ASR	Avg. Mod
Malscan	HRAT	**94.23%**	96.71%	100%	**5.58**
	Add edge	**96.97%**	82.31%	100%	11.03
	Rewiring	67.14%	79.58%	100%	**7.54**
	Insert node	81.48%	98.65%	100%	14.68
	Delete node	74.17%	93.76%	100%	8.06
Mamadroid	HRAT	**99.94%**	94.95%	100%	**34.55**
	Add edge	93.77%	94.90%	100%	57.75
	Rewiring	83.95%	75.53%	100%	**27.64**
	Insert node	**96.76%**	98.29%	100%	64.26
	Delete node	78.49%	83.17%	100%	**23.24**
APIGraph + Malscan	HRAT	**99.57%**	98.93%	100%	**1.63**
	Add edge	91.51%	98.00%	100%	3.39
	Rewiring	**95.61%**	85.46%	100%	**2.57**
	Insert node	92.20%	95.80%	100%	2.81
	Delete node	94.62%	96.07%	100%	4.29

of the feature-space attacks.

Effectiveness of HRAT on malware that fails to escape detection using individual actions

To evaluate whether combining multiple attack actions is more effective than individual attack actions, we use HRAT to modify apps that fail to evade the target systems (i.e.,Malscan, APIGraph enhanced Malscan, and Mamadroid) using individual attack actions. If N_s apps fail to deceive target systems using individual actions but N_h out

of N_s apps successfully escape the detection under HRAT's modifications, we define the effective ratio as N_h/N_s to quantify HRAT's effectiveness.

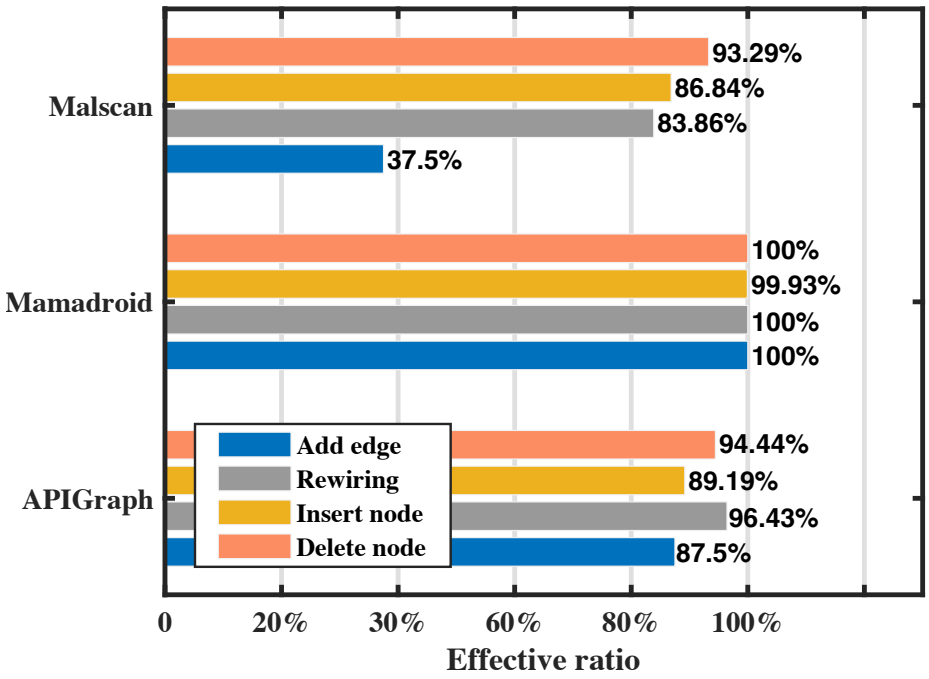


Figure 3.12: Effectiveness of HRAT over malware that fails to escape detection under individual attack action

Results. Figure 3.12 shows the effective ratio of HRAT over apps that fail to evade detection systems using individual attack actions. For attacking Malscan, since *adding nodes* action achieves comparative *Init ASR* with HRAT, the malware that fails to deceive Malscan using adding nodes has overlapping with malware that fails to evade the detection under HRAT's modification, leading to a 37.5% effective ratio. For Mamadroid and the enhanced detector, as HRAT achieves nearly 100% *Init ASR* (Table 3.3), HRAT can also achieve nearly 100% and around 90% effective ratio, respectively.

Next, we discuss the randomness in individual manipulation attacks. Individual manipulation attacks use a greedy strategy that always selects the edges or nodes with the maximum gradient to modify. However, a locally optimal solution is not necessar-

ily a globally optimal one. Since individual manipulation attacks also use the same substitute model as the target detection systems, the randomness inherent in HRAT's data splitting and action determination affects these attacks as well. Therefore, any variability introduced by the dataset can influence the performance of individual manipulation attacks. Additionally, the impact of manipulations on Mamadroid, Malscan, and APIGraph, as discussed in Section3.5.2, contributes to the variability in the performance of individual manipulation attacks. The way these systems process and respond to perturbations can significantly alter the outcomes of such attacks.

Answer to RQ3: Individual attack actions are not always effective in attacking all AMDs. Combining multiple attack actions, HRAT is much more effective and modification efficient.

3.5.4 RQ4: Resilience to Obfuscation Techniques

Experimental Setup. To evaluate the resilience of HRAT against malware adopting different obfuscation techniques, we use a dataset in [37], which includes malware from different families obfuscated by three different obfuscation techniques (i.e., identifier renaming, string encryption, and reflection) We use TRo as the basic training set. For each malware family, we randomly select 100 samples from the dataset and add them to the training set to improve the classifier's performance. To construct the testing set, we randomly select 500 samples that are correctly identified as malware by target systems. We apply HRAT to above malware for evading target AMD systems(i.e., Malscan and Mamadroid), and define $evasion\ rate = N_e/N_t$, where N_e is the number of malware that escapes the detection and N_t is the number of test samples, to quantify the resiliency of HRAT against these commonly used obfuscation techniques.

Results. As shown in Table 3.4, the *evasion rate* of Malscan and Mamadroid are 100%, meaning that our approach is resilient to identifier renaming. The reason is that

Table 3.4: Evasion rate of AMD systems by adversarial apps whose original apps belong to different families and adopt three different obfuscation techniques.

	Without attack		**With attack**	
	Malscan	**Mamadroid**	**Malscan**	**Mamadroid**
Renaming	0.00%	0.00%	100%	100%
Encryption	0.00%	0.00%	93.94%	87.11%
Reflection	0.00%	0.00%	92.08%	91.30%

the renaming obfuscation can only change the names of parameters or identifiers into meaningless strings or hash values [114], which do not affect the structure of FCGs. Thus, HRAT achieves desirable performance on malware using identifier renaming. For string encryption, the *evasion rate*s of Malscan and Mamadroid are 93.94% and 87.11%, respectively, meaning that this obfuscation technique can affect our approach. Our manual analysis reveals that string encryption may affect the graph structure because malware may use encrypted strings to replace original invocation statements and restore them at run-time [37], and thus, some nodes and edges are missed during the modification. For example, when deleting a node, if the callee's name is encrypted in one call relation, HRAT fails to replace the method invocation statement with the deleted method body in the corresponding method. It may cause HRAT to break the functionality of target malware and make the performance of the problem-space attack(*evasion rate*) lower than 100%. If the reflection is used, the *evasion rate* of Malscan and Mamadroid are 92.08% and 91.30%, respectively, meaning that reflection may affect our approach. Reflection may make it difficult to conduct static analysis on apps, and some invocation relations may be missed.

Next, we discuss how different obfuscation techniques introduce randomness that affects HRAT's performance. For Malscan, renaming methods have minimal impact on the overall Function Call Graph (FCG) centrality evaluation, as it only changes the names without affecting the structure. Similarly, encryption has a comparable

effect to renaming, altering method names without significant structural changes. However, reflection has a more substantial impact, as it can modify the raw FCG's invocation relationships, thereby influencing the centrality of key nodes in the graph. For Mamadroid, which abstracts methods into different families and uses invocation probabilities between families as features, renaming and encryption randomly affect method names. This leads to variability in Mamadroid's performance under attack with these obfuscation methods. Since reflection can alter the raw FCG's invocation relationships, it also affects the invocation probabilities, introducing further randomness in the attack's impact on Mamadroid.

Answer to RQ4: If an obfuscation technique neither affects the structure of FCGs nor impedes the static analysis and manipulation on apps, it will not affect our approach.

3.5.5 RQ5: Functional Consistency Assessment

We conduct static analysis and dynamic analysis to assess whether the adversarial apps generated by HRAT preserves the functionality as the original ones. We randomly select 40 malicious apps and apply HRAT to them. During this process, we also insert `log` into the modified methods of original and modified apps to collect information for assessment.

•**Static analysis assessment.** For these app pairs, we conduct static analysis on them to ensure that the modifications have been correctly imposed. Specifically, we check whether the added invocations and methods exist and whether the deleted methods are correctly modified. Besides, we also compare the scale (number of nodes and edges) of FCGs and extracted features of the modified app and those obtained by HRAT.

Results. The results show that the FCGs extracted from modified apps are the same as the FCGs computed by the algorithm. Besides, the number of nodes and edges

in the FCGs and extracted features from modified apps are also the same as those calculated by HRAT.

• **Dynamic analysis assessment.** For the dynamic assessment, we install the apps before and after modifications on two Android virtual machines (AVMs) with the same configuration, respectively. For apps (35/40) that have been modified by less than ten times, we manually analyze their FCGs to learn how to trigger the modified methods. Then, we conduct the same operations to run an app pair on the two AVMs and record the run-time UI. To check whether the modified methods are triggered, we insert *log* functions to the modified methods to print the method's all parameters and the *callers*'s signature. For example, when testing *adding function call*, we insert the log to print the parameters of *callee* and its callers signature between lines 1 and 2 of Figure 3.5(b). To check whether the modified app works the same as the original one, we insert *log* to print the parameters of methods to be modified in the original app. In this way, besides manually checking whether the user interface gives the same feedback (e.g., same activity transition, same pop-up window, same text rendered on the window, etc.), we also compare the values of the methods' parameters before and after modification. For apps (5/40) that are hard to find the activation paths because of heavy code obfuscation, we use a popular Android testing tool Monkey [51] to conduct the dynamic exploration on them. We configure Monkey to let it ignore crashes and timeouts and set the duration of each event to 300ms. The execution time is set to 20 minutes. We collect the `logs` to identify the invoked methods and check whether the apps before and after modification have the same functionality by comparing the log information.

Result. For apps that have been examined by us manually, the coverage rate ($N_{triggered\_methods}/N_{modified\_methods}$) of modified apps is 100%, because we have pre-analyzed their FCGs, and 33/35 apps show the same user interface as original apps. Specifically, when we conduct the same operations on the apps, both the modified apps and original apps output the same UI feedback. Besides, the log messages show

the expected results. For example, for inserted nodes, the expected invocation sequence exists in the corresponding caller. However, two of thirty-five apps crashed during the testing process, whereas their original apps run smoothly. By manually inspecting these two apps, we find that when HRAT modifies the methods called by reflection, it cannot make the corresponding modifications to the reflection invocation, thus leading to app crash. Thus, we cannot verify their functional consistency. For other apps tested using Monkey, the coverage rate drops to 24.87% as Monkey randomly interacts with the app to trigger methods. For these trigger paths, we find that all modifications do not affect the functionality.

Answer to RQ5: HRAT keeps the functional consistency of the modified apps in most cases. It may break the functionalities of apps using obfuscation techniques that hinder static analysis.

3.5.6 RQ6: Influence of Key Parameters

Two key parameters will influence the effectiveness and efficiency of HRAT: a) probability, which determines how likely HRAT is to adopt an action type learned by deep Q-network or randomly select an action type; and b) memory capacity, which determines the frequency at which HRAT interacts with the environments. We evaluate the parameter influence of HRAT on Mamadroid and Malscan with 500 randomly selected malware. Figure 3.13 shows the influence of these two key parameters on HRAT. We can see that our attack on Mamadroid is not sensitive to the parameters. For Malscan, as storage capacity increases, ASR drops, because the increase of storage capacity means that the frequency of interaction between HRAT and the environment is reduced and thus the system cannot better determine its behavior based on the environment. Similarly, this will also lead to more modifications with the increase of memory capacity. Since HRAT takes a random attack action type with $1 - Probability$, when the $Probability$ decreases, the probability of our system

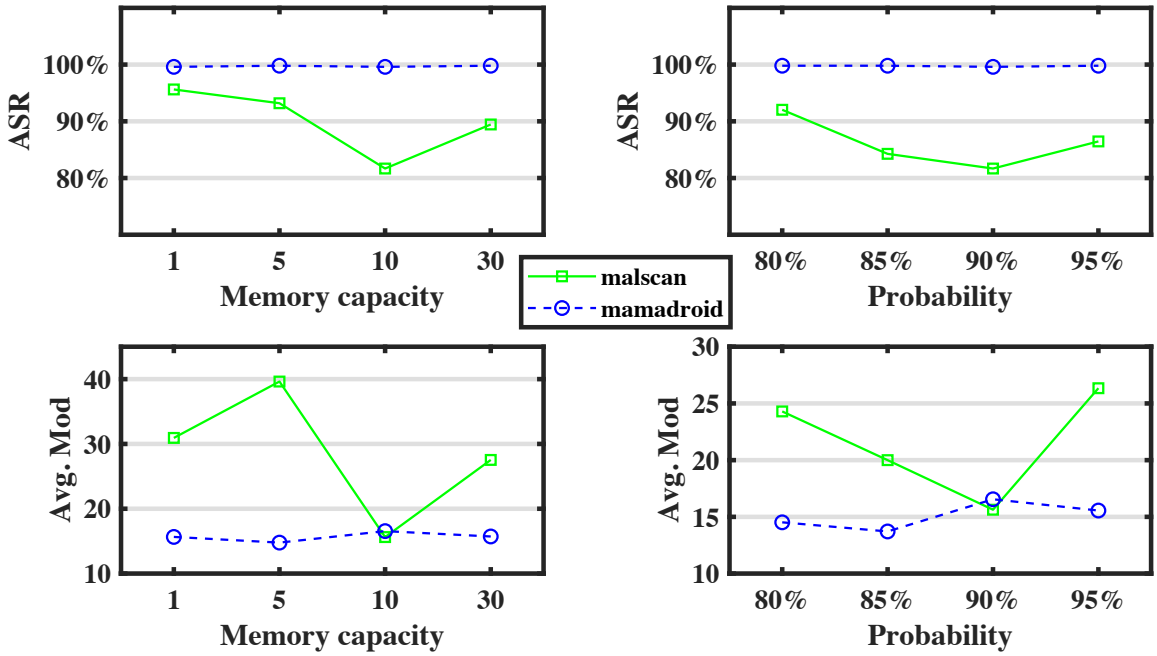


Figure 3.13: Parameter analysis

taking random attack behavior will increase. This will result in a decrease in the ASR of the system and an increase in the number of modifications taken.

For other parameters in our system, such as *maximum modification times* M, m in Eq 3.5, we set M to 500 and m as 75, which follows the settings in [125]. We set the value of M to 500, because as M increases, ASR will also be increased, but the optimization time consumption will also increase. The results of our experiments in § 3.5.1 demonstrate that if the number of M is not limited, the app can be successfully attacked in the feature space. Similarly, the CDF of the number of app modifications § 3.5.2 also shows that most apps can be successfully modified at most 50 times. For m, it is only used to solve the distance between the target graph and the m samples in the training set during the optimization process. When evaluating whether the algorithm was successfully attacked, we still used all the samples from the original training set.

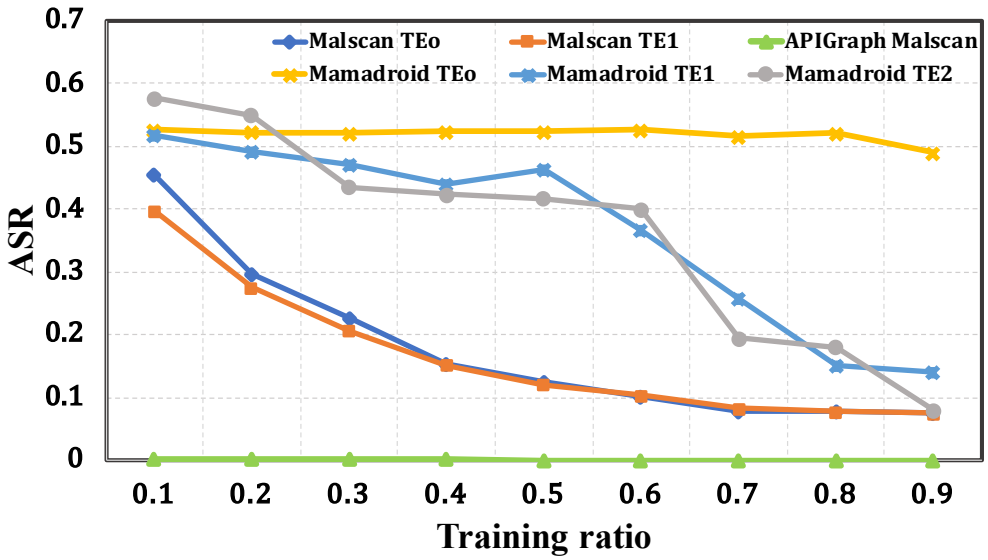


Figure 3.14: Attack success rate against retraining.

Answer to RQ6: The parameter setting's impact on HRAT varies for victim detection systems.

3.5.7 RQ7: Defense against HRAT

We evaluate two potential approaches for defending against HRAT, including adversarial training and ensemble learning[133], which have been used to defend against adversarial attacks in other domains (e.g., computer versions).

Adversarial retraining is regarded as one of the most effective defense methods against adversarial attacks [48]. We randomly select 500 samples that evade the detection in both feature and problem space and divide them into training and testing sets. Figure 3.14 shows that with the increase of training ratio (i.e., the ratio of retraining samples to all adversarial samples), *ASR* drops accordingly. However, Mamadroid trained with the TRo dataset cannot achieve good defense performance even when enough adversarial samples are available. It may be due to the limited ability of Mamadroid to learn extracted features for malware detection. Similarly, when the

retraining ratio is less than 0.5, the attack success rate of Mamadroid TE1 and TE2 remains more than 40%. It is worth noting that Malscan enhanced by *APIGraph* only needs a small number of retraining samples to successfully detect adversarial apps. For attacking Malscan, when the training ratio is limited to 10% to 40%, the ASR drops dramatically, while the training ratio grows to more than 50%, and the defense effectiveness does not improve a lot. This could be caused by the limitation of Malscan's defense performance as the F1-score of raw Malscan's detection performance is limited to 98.8% [152]. In summary, retraining would be an effective defense method against HRAT if there are enough adversarial samples for retraining. However, it is worth noting that if many samples are used for retraining, the time consumption and computational workload will also increase.

We also evaluate the effectiveness of ensemble learning. Ensemble learning has been used to defend against adversarial attacks [133]. To evaluate the defense effectiveness of ensemble learning algorithms, we adopt four ensemble learning algorithms: *Bagging, Adaboost, gradient boosting decision tree, and Voting*. *Bagging* [69] forms a class of algorithms that will be trained using a random subset of the original training set and then aggregate their individual predictions to get a final prediction. *Adaboost* [45] utilizes a sequence of weak learners on repeatedly modified versions to improve the performance of weak classifiers. *Gradient boosting decision tree* [105] (GBDT) integrates a set of regression trees and uses a generalization of boosting to arbitrary differentiable loss functions. *Voting* strategy simply combines different machine learning classifiers and uses votes to predict the class labels. We use sklearn [105] to implement those embedding algorithms and the parameters of each algorithm are set as default (see Table 3.5).

To conduct the experiments, we use ensemble learning algorithms to replace the kNN classifier in original detection systems. Then, we use those pre-trained ensemble classifiers to identify the label of adversarial malware. Table 3.6 shows that ensemble learning for APIGraph enhanced system cannot effectively defend HRAT's attack.

Table 3.5: Parameter settings of ensemble algorithms

Algorithms	Parameter settings
Bagging	base_estimator: 1NN max_samples: 0.5 max _features: 0.5
Adaboost	base_estimator: 1NN n_estimators:100
GBDT	n_estimators: 100 learning_rate: 1.0 max_depth: 1 random_state: 0
Voting	base_estimators: SVM, 1NN, DT voting:majority voting

SVM: support vector machine; 1NN: 1st nearest neighbors; DT: decision tree

For Malscan, *Adaboost* and *GBDT* are promising defense strategies that could benefit from their boosting strategies. Since Boosting strategy uses weighted methods to combine weak classifiers, the ensemble classifier is supposed to integrate the advantages of weak classifiers, thus making the defense more effective. For Mamadroid, ensemble learning could achieve at least 60% ASR decrease. As the feature number in Mamadroid is small, the learning strategy of bagging (using the subset of the training set) can effectively exclude outliers in the training set. The results show that it could be a promising defense strategy for Mamadroid. However, it is not always effective for Malscan. Therefore, different defense strategies should be adopted for different detection systems to defend against HRAT.

Answer to RQ7: Different defense strategies should be adopted for different detection systems to defend against HRAT.

Table 3.6: Evaluation of ensemble learning based defense methods

	Bagging	**Adaboost**	**GBDT**	**Voting**
Malscan TEo	91.50%	18.42%	15.79%	100%
Malscan TE1	92.31%	15.03%	13.99%	100%
APIGraph +Malscan	100%	99.84%	97.28%	100%
Mamadroid TEo	14.86%	32.32%	33.19%	34.20%
Mamadroid TE1	30.91%	36.75%	35.14%	24.64%
Mamadroid TE2	14.00%	31.88%	32.64%	36.25%

3.6 Discussion

3.6.1 Applicability of HRAT

The algorithm design of HRAT, which includes constraints and four elaborate graph modification actions, is versatile enough to be applied to attack function call graphs in other software platforms, such as Windows or PDF. When modifying the FCG, HRAT maintains method connectivity and call sequence, which is not limited to any specific programming language. Furthermore, the constraints set by HRAT ensure that in other software platforms, it will not modify the unmodifiable nodes or edges (methods and call relations). In this way, HRAT maintains the functionalities of the target software. When applying HRAT to other systems, the adversary only needs to consider the limitations of modifiable functions in target scenarios and then add them to the algorithm constraints.

3.6.2 Limitations

HRAT's *optimization progress* leads to high computational consumption because, given a new graph, HRAT needs to calculate the gradient of each edge to select the influential nodes or edges. In future work, we will investigate the transferability of HRAT. In other words, we will first use HRAT to attack one detection system and generate adversarial malware and then check whether the generated malware can escape another detection system. Besides, to address the limitation of gradient-related methods, we will explore involving information entropy in modification selection.

Since HRAT relies on static analysis, its attack may break the app's dynamic features, such as reflection [37], dynamic class loading, etc. We can add related methods into constraints and set them as unmodifiable to avoid modifying such dynamic features. Moreover, HRAT could not handle heavily obfuscated malware [181, 157, 155, 156, 186], such as packaged apps, because static analysis may just access the Dex file of the shell rather than the real functional Dex file.

Chapter 4

A Fine-grained Chinese Software Privacy Policy Dataset for Sequence Labeling and Regulation Compliant Identification

4.1 Overview

The Android privacy policy is a legal document written in natural language that discloses the purposes and mechanisms by which a controller —the entity determining the purposes and means of processing personal data—collects, shares, uses, and stores user information [47, 108, 98]. Regulatory authorities [47, 108, 98] and Android application platforms [49, 62] require developers to provide clear privacy policies to inform users about how their personal data is handled. This enables users to understand and assess whether their privacy may be at risk, thereby helping them make informed decisions about using the application. However, privacy policies are often lengthy and complex, making them difficult for users to read and comprehend [128].

Natural Language Processing (NLP) techniques have achieved significant success in understanding document semantics [161, 150, 36]. Therefore, it is essential to apply NLP techniques to analyze privacy policies, identify compliance between privacy policy statements and regulation requirements, and assist users in understanding the privacy practices of mobile applications [166, 7, 168]. However, applying NLP methods to this domain requires a large amount of annotated corpus to train models that can identify who is responsible for collecting or sharing user data, as well as with which parties or organizations such data is shared. Currently, available datasets for this task only include English-based privacy policy corpora [151, 193], while there remains a lack of publicly available datasets specifically tailored for Chinese privacy policies.

Chinese **s**oftware **p**rivacy **p**olicy **p**rocessing (CSP^3) task is a sequence labeling problem that recognizes privacy-related components in the sentences. CSP^3 has two main unique features. First, privacy policies contain an amount of information inside [165], such as how the app stores user data and how to contact the app developers. In our dataset, we concentrate on data access-related sentences as the sentences are directly related to user privacy. Second, privacy policies are written in legally binding professional language and contain software jargon. Thus, it requires a strong background [187, 188] to understand the statements inside. Both characteristics prevent users from understanding privacy policies. A well-annotated dataset can facilitate the building of automatic privacy policy analysis tools and further help users protect their privacy.

Although privacy policy datasets have been proposed recently [151, 193], labels in existing datasets are coarse-grained (i.e., sentence-level annotations [151]) and those data set only involve few privacy practices [193]. Besides, existing datasets only include English privacy policies, which limits the application of these datasets in regions with other languages. We construct a fine-grained Chinese dataset for software privacy policy analysis.

In this work, we focus on Android application privacy policies because Android possesses the largest share of mobile operating systems [129], and a large number of Android privacy data leaks have been revealed [123, 126]. Unlike previous work [151, 193], we deal with the problem using sequence labeling methods and pay special attention to the Chinese privacy policies. The motivations come from the following four aspects:

First, worldwide regulation departments enact laws [98, 108, 47, 144, 27] to regulate the software's behaviors and protect users' privacy. The laws require the software to clarify how and why they need to access user data. Analyzing privacy policies can help users understand how app process their data and identify whether apps comply with laws. However, privacy policies are written using professional legal and software jargon that prevents users from reading and understanding them. Thus, it is necessary to apply NLP techniques to analyze and help users understand privacy policies. Second, for sequence labeling tasks, CSP^3 aims to identify how and why the software collects, shares, and manages users' data according to regulations. CSP^3 can be abstracted as identifying components in the privacy policy documents, such as data type and the purpose of using user data. NLP techniques can help automatically analyze privacy policies. Third, existing privacy policy analysis research is limited to English and totally omits other languages. With over 98.38 billion app downloads [132] and privacy-related regulations enacted in China, it is necessary and urgent to research CSP^3. Last but not least, recent research in other communities, such as software engineering [166, 96] and cybersecurity [7, 6], demonstrates requirements for analyzing privacy policies to help the analyst identify whether the apps' behavior is consistent with privacy policies.

In this work, we make the following efforts to advance CSP^3 task:

First, we construct a novel large-scale human-annotated **C**hinese **A**ndroid a**pp**lication **p**rivacy **p**olicy dataset, namely CA4P-483. Specifically, we manually visit the software markets, such as Google Play [49] and AppGallery [62], check the provided privacy

policy website, and download the Chinese version if available. We finally collect 483 documents. To determine the labels in the privacy policy analysis scenario, we read through Chinese privacy-related regulations and summarize seven components (§4.3.2). We annotate all occurrences of components in 11,565 sentences from 483 documents. Unlike paragraph-level annotations in existing privacy policy datasets [151], `CA4P-483` annotates character-level corpus.

Second, based on `CA4P-483`, we summarize families of representative baselines for Chinese sequence labeling. In detail, we first evaluate the performance of several classic sequence labeling models on our dataset, including Conditional Random Forest (CRF) [70], Hidden Markov Model (HMM) [94], BiLSTM [55], BiLSTM-CRF [74], and BERT-BiLSTM-CRF [34]. Recent work shows that lattice knowledge improves the performance of Chinese sequence labeling tasks. We involve lexicon-based models, such as Lattice-LSTM [180].

Third, we investigate potential applications of `CA4P-483`. Combining law knowledge, we first identify whether the privacy policy violates regulation requirements based on `CA4P-483`. We also identify whether the app behaves consistently with privacy policy statements combining software analysis [184, 189].

The contributions of this work are three-fold:

- To the best of our knowledge, we construct the first Chinese privacy policy dataset, namely CA4P-483, integrating abundant fine-grained annotations.
- We experimentally evaluate and analyze the results of different families of sequence labeling baseline models on our dataset. We also summarize difficulties in our dataset and provide findings and further research topics on our dataset.
- We investigate potential applications of `CA4P-483` to regulate privacy policies with law knowledge and program analysis technologies.

4.2 Preliminaries

4.2.1 Android Privacy Policy

A privacy policy is a legal document written in natural language that discloses how and why a controller collects, shares, uses, stores, and protects user data [47, 108, 98]. Privacy policies help users understand whether their privacy will be abused and decide whether to use the product. Android application markets, such as Google Play [49] and Huawei Gallery [62], require developers to upload the app's privacy policy when they submit their apps to markets.

4.2.2 Sequence Labeling

The sequence labeling task recognizes components of interest, which are predefined in specific applications, in the sentence. One classic example of sequence labeling tasks is part of speech tagging, which aims at assigning each word a part of speech in given sentences or documents.

4.3 Dataset Construction

4.3.1 Dataset Collection

We manually collect the Chinese privacy policies from Android application markets. According to application market requirements [63, 50], developers must provide privacy policies to claim their user data access behavior and to ensure apps will not violate laws or regulations. Since privacy policies are publicly available for users to understand the apps' access to personal data, the three authors of this paper manually access the most popular apps in markets and visit their privacy policy websites

provided at the moment (January 2021). We use *html2text* [3] to extract context. Finally, we use *tagtog* [20] for document annotation.

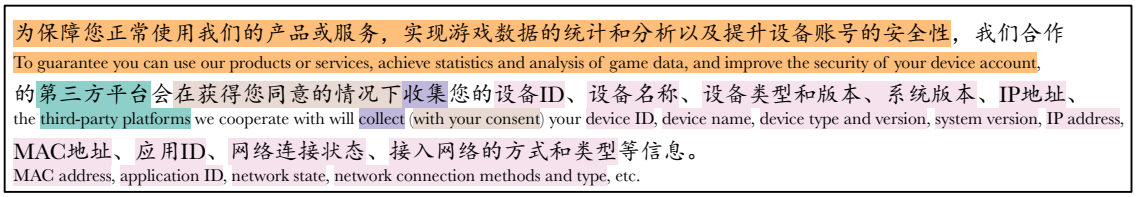
Next, we annotate `CA4P-483` based on the law requirements. Specifically, we analyze Chinese privacy-related laws and regulations [98, 108, 100, 29], and find requirements for apps' privacy process behavior. For example, GB/T41391-2022 Article 4.n) claims that "*developers should expressly state the purpose of applying or collecting information to the subject of personal information.*" Finally, we summarize seven types of labels related to requirements for apps' access to user data.

4.3.2 Fine-grained Annotations

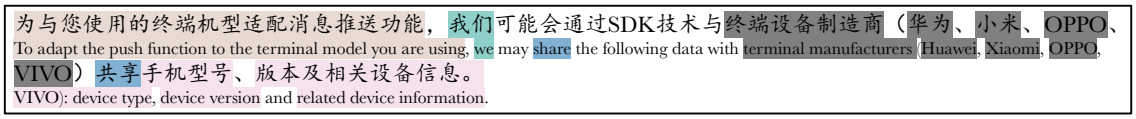
For each privacy policy, we concentrate on the sentences that describe the data process behavior. After locating the sentences, we annotate seven components, i.e., the controller, data entity, collection, sharing, condition, purpose, and receiver.

Data controller. According to regulation requirements, the data controller is the party that determines the purpose and means of personal data processing. A data controller could be the app (first party) or the third party. As is shown in Figure 4.1, data controllers are "*third-party platforms*" in Figure 4.1(a) while that is "*we*" in Figure 4.1(b). Thus, we annotate data controllers according to sentence semantics, i.e., who is responsible for processing the data.

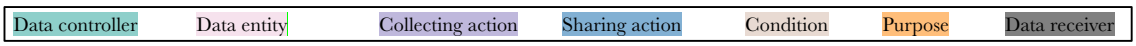
Data entity. Data entities are any information that can identify or reflect the activities of a natural person [108]. Recent research [17, 122] demonstrates the probability of combining various information to infer and even locate a specific person. Thus, we annotate all data nouns or noun phrases that are requested in privacy policies, including sensitive information, such as device id, and normal information, such as device type.



(a) Demo 1.



(b) Demo 2.



(c) Annotation legend.

Figure 4.1: Annotation demos from `CA4P-483`. We translate the statements into English for illustration.

Collection. Collection actions are verbs that describe how controllers access data, such as gather (收集) and obtain (获取).

Sharing. Sharing actions are verbs that indicate whether the data controller will distribute data to others. Although both Sharing and Collection describe how the party access user data, we differentiate them according to the requirements of laws on the action, such as Article 5 and 9.2 in [108].

Condition. The condition describes the situation where the data controller will access personal data. Laws require data controllers to inform users under what conditions their data will be processed. For example, bank apps may require the users' identification information when activating the bank account. Figure 4.1(a) also demonstrates that under the condition of the user's consent, the third-party platforms (TPP) access users' data. Another semantics in Figure 4.1(a) also indicates that the TPP cannot access those data without users' consent, which can help

users and analysts understand whether the app violates laws.

Purpose. The purpose should claim why the data controller processes user data. Laws enact specific requirements for user data access. For example, PISS Article 4.d) requires controllers to clearly state the purpose of processing data. Purpose can also help the users understand why the app collects their data and further determine whether to give consent as is shown in Figure 4.1(a).

Data receiver. The data receiver describes the parties that receive user data. Laws not only ask apps to clarify who will get shared data [108] but also restrict the data receivers' behavior [98], such as why processing user data.

4.3.3 Human Annotation Process

Our privacy policy annotation consists of two phases: coarse-grained annotation and fine-grained annotation. Coarse-grained annotation labels privacy policies at the paragraph level following previous work [151]. Fine-grained annotation labels our defined components at the word level based on coarse-grained annotation.

For the first phase, three authors of this paper, who have researched privacy policies and software engineering for over eight and three years, label ten privacy policies for reference and record a video instruction to guide annotators. Then, we hire thirty undergraduates from our university to annotate the dataset. The three instructors train each annotator for at least four hours to become familiar with the dataset and requirements. Students are asked to annotate 1,000 Android apps' privacy policies in Chinese, and each privacy policy should be analyzed for at least 30 minutes to ensure quality. Each privacy policy is allocated to at least four annotators. Finally, three instructors inspect each annotation.

For the second phase, we select two undergraduates who coarse-grained annotate

the documents with high precision to conduct the fine-grained annotation. Specifically, we select 483 documents that are well coarse-grained annotated after inspection. Instructors first annotate ten documents to lead undergraduates to annotate. The annotators also keep discussing with instructors when the role of components in sentences is unclear. Each annotator is required to label each privacy policy for at least 30 minutes to guarantee the dataset quality.

Finally, the instructors analyze the annotations and use Fleiss' Kappa metrics [28, 151] to evaluate the agreements. Table 4.1 shows that the average Kappa value (77.20%) satisfies the substantial agreement, i.e., the Kaapa value lies in 0.61-0.80, and four components achieve almost perfect agreement (0.81-1.00). The *Condition*, which only gets a moderate agreement, is caused by the overlap between labels. For example, the statements may claim that "*under the condition of accessing your contact, we will send your location to emergency contact*" one annotator only annotates the "*contact*" as *data* without annotating the whole clause.

4.3.4 Dataset Statistics and Comparison

We conduct statistical analysis and show the results in Table 4.1. `CA4P-483` is split into training, development, and test set. Table 4.1 also gives details of the number of different labels in each set. Table 4.1 shows that the average length of condition and purpose is much longer than other corpora as the two types are generally in the form of clauses.

We compare `CA4P-483` with related datasets in Table 4.2. We first compare our corpus with Chinese sequence labeling datasets, such as MSRA [173], OntoNotes [149], Weibo [106], PeopleDiary [170], Resume [180], CLUENER2020 [154], and CNERTA [134]. We also involve widely used English sequence labeling datasets, namely Twitter-2015 [172] and Twitter-2017 [88]. We also consider privacy policy datasets, namely Online Privacy Policies (OPP-115) [151] and Android app privacy policies (APP-

Table 4.1: The statistics of CA4P-483. Here, "Avg" denotes *average*, "ann" denotes *annotation*, "len" denotes *length*, "#" denotes *the number of*.

# doc					483	
# sentences					11,565	
# sentences with ann					3,385	
Avg sentences len					79.06	
Type	Num	Train	Dev	Test	Avg len	Kappa
Data	21,241	18,925	2,521	2,331	4.68	85.39%
Collect	5,134	4,133	576	528	2.03	73.78%
Share	4,976	3,989	533	505	2.10	84.87%
Controller	8,424	6,085	815	782	2.49	82.22%
Condition	4,917	5,477	716	713	14.41	50.07%
Receiver	3,202	2,776	360	350	4.29	89.88%
Purpose	4,683	6,442	860	867	19.24	74.18%
Total	52,577	47,827	6,381	6,076		

350) [193].

We first compare the size and classes in different datasets. Table 4.2 shows that CA4P-483 contains abundant semantics, i.e., CA4P-483 has seven annotation classes that are larger than most other datasets (seven out of nine). Table 4.2 also compares the CA4P-483 with other privacy policy datasets. For privacy policy-related datasets, the comparison is conducted with the number of documents as one privacy policy corresponds to one app. OPP-115 annotates at the sentence level, and APP-350 only annotates data controllers, data entities, and modifiers. Since APP-350 specifies data entities into 16 categories, APP-350 exhibits more number of classes than CA4P-483. To summarize, CA4P-483 is the first and largest Chinese Android privacy

Table 4.2: A comparison between CA4P-483 and other popular sequence labeling datasets. # denotes *"number"*. *"doc"* denotes *"documents"*.

Dataset	# Train	# Dev	# Test	Size	Language	# Class
MSRA	41,728	4,636	4,365	50K	Chinese	3
PeopleDairy	20,864	2,318	4,636	23k	Chinese	3
Weibo	1,350	270	270	2k	Chinese	4
Resume	3,821	463	477	2k	Chinese	8
CLUENER2020	10,748	1,343	1,345	13K	Chinese	10
CNERTA	34,102	4,440	4,445	42,987	Chinese	3
Twitter-2015	6,176	1,546	5,078	12,784	English	4
Twitter-2017	4,290	1,432	1,459	7,181	English	4
CA4P-483	14,678	2,059	1,842	18,579	Chinese	7
Dataset	# Train doc	# Dev doc	# Test doc	Size	Language	# Class
OPP-115	75 doc	/	40 doc	115 doc	English	12
APP-350	188 doc	62 doc	100 doc	350 doc	English	18
CA4P-483	386 doc	48 doc	49 doc	483 doc	Chinese	7

policy dataset with abundant semantic labels.

4.4 Task and Experiment Setup

4.4.1 Task Description

CSP^3 figures out who collects or shares what kind of data to whom, under which kind of condition, and for what. The underlined words correspond to each type of annotation. As CSP^3 concentrates on data access-related sentences, we first locate the sentences based on data collection and sharing words [7, 166]. We summarize

Table 4.3: Data access word list

Sharing	收集 (collect), 获取 (obtain), 接受 (get), 接收 (receive), 保存 (save), 使用 (use), 采集 (gather), 记录 (record), 存储 (store), 储存 (store)
Collection	披露 (reveal), 分享 (share), 共享 (share), 交换 (exchange), 报告 (report), 公布 (public), 发送 (send), 交换 (exchange), 转移(transfer), 迁移 (migrate), 转让 (make over), 公开 (public), 透露 (disclose), 提供 (provide)

the word list based on laws, app market requirements, and previous works [166, 6, 7]. Table 4.3 gives data sharing and collection word list, that is summarized from laws [98, 47, 108], app market requirements [50, 63], and previous works [166, 6, 7]. With those words, researchers can locate data access-related sentences and conduct further analysis to get interested entities, such as data controller, data entity, collection, sharing, condition, purpose, and the data receiver. Given the sentences $C = c_1, c_2, ..., c_n$ and its labels $L = l_1, l_2, ..., l_n$, where c_i denotes the i-th Chinese characters and l_i denotes the c_i's label, the task is to identify sequence labels.

4.4.2 Model Summaries

This section introduces baseline methods for sequence labeling tasks on `CA4P-483`.

Probabilistic models

Hidden Markov Model (HMM): HMM[1] [44] is one of the most classic probabilistic models and is applied as our baseline.

[1] https://github.com/luopeixiang/named_entity_recognition

Condition Random Field (CRF): CRF[2] [73] aggregates the advantages of HMM and counters the label bias problems.

Neural network models

BiLSTM: BiLSTM[1] [55] uses the neural network to learn a mapping relation from sentences to labels through the nonlinear transformation in high-dimensional space.

BiLSTM-CRF: BiLSTM-CRF[1] uses BiLSTM as an encoder to map the sentences into a high dimension vector and uses CRF as a decoder.

BERT-BiLSTM-CRF: Since BiLSTM-CRF is still limited to the word vector representation, BERT-BiLSTM-CRF[3] [31] uses BERT as a feature extractor and takes advantage of BiLSTM and CRF for sequence labeling.

Lattice enhanced models

As Chinese words are not naturally separated by space, character-based methods omit the information hidden in word sequences. Thus, lattice-based methods that integrate lattice information are proposed for Chinese sequence labeling and achieve the promised performance.

LatticeLSTM: LatticeLSTM[4] [180] takes inputs as the character sequence together with all character subsequences that match the words in a predefined lexicon dictionary.

[2] http://crfpp.sourceforge.net/
[3] https://github.com/macanv/BERT-BiLSTM-CRF-NER
[4] https://github.com/LeeSureman/Batch_Parallel_LatticeLSTM

4.4.3 Setup and Implementation Details

We evaluate baselines on an Ubuntu 20.04 server with 5 NVIDIA GeForce 3090 (24 GB memory for each), 512 GB memory, and an Intel Xeon 6226R CPU. Next, we present our implementation details. For HMM, the number of states, i.e., class number in our dataset with the BIO tag, is set as 22, and the number of observations, i.e., the number of different characters, is set as 1756, which is the default value[1]. For CRF, we use the default settings in CRF++[2]. For BiLSTM and BiLSTM-CRF, the embedding size is 128, the learning rate is 0.001, and we train models using 30 epochs with a batch size of 64. For BERT-BiLSTM-CRF[3], we use the Chinese bert-base[5] pre-trained model and fine-tune it on our training data. The BiLSTM is set with 128 hidden layers and a learning rate of $1e^{-5}$. BERT-BiLSTM-CRF model is trained on our dataset with default settings[3] where the batch size is 64, the learning rate is $1e^{-5}$, the dropout rate is 0.5, gradient clip is 0.5, and early stop strategy is "*stop if no decrease*". For Lattice-LSTM, we use the same lattice provided in [180].

4.5 Evaluation

4.5.1 Main Results

In this section, we evaluate baseline methods on all 18,579 sentences that are divided into training, development, and testing sets as detailed in Table 4.2. Following previous research [151, 134], we apply the following metrics to evaluate baseline methods in `CA4P-483`: precision (P), recall (R), and F1-score (F1).

Table 4.4 presents the performance of different baseline models on `CA4P-483`, with each cell displaying the mean and variance of results obtained from five runs. Table 4.4 shows that BiLSTM-CRF achieves the most promising performance, which

[5] https://github.com/google-research/bert

Table 4.4: Overall performance of baseline methods on our dataset.

	Precision	Recall	F1
HMM	77.47%(±0.00%)	66.11%(±0.00%)	69.63%(±0.00%)
CRF	85.52%(±0.00%)	86.28%(±0.00%)	85.63%(±0.00%)
BiLSTM	85.61%(±0.45%)	86.26%(±0.37%)	85.57%(±0.57%)
BiLSTM-CRF	86.26%(±0.32%)	86.46%(±0.41%)	86.25%(±0.38%)
BERT-BiLSTM-CRF	49.05%(±5.82%)	41.22%(±1.97%)	44.07%(±2.12%)
Lattice-LSTM	67.93%(±0.44%)	68.16%(±0.50%)	68.04%(±0.03%)

may benefit from the enhanced presentation ability of bidirectional LSTM and CRF for capturing the context information. The zero variance of the results for HMM and CRF over five runs may denote that the two algorithms with few parameters overfit our dataset. The BERT-BiLSTM-CRF performed poorly on the dataset, with the lowest mean and highest variance over five runs. This could be caused by the fact that the model is designed with a large number of parameters, and our dataset size is insufficient to train the model effectively. Future work may focus on incrementing the dataset size to improve the performance of BERT-based models in this context. Lattice-LSTM performs a strong representation of capturing lattice information, while some clauses in our labels may mislead the model in learning the patterns.

We analyze the identification performance of each component to investigate the challenges and limitations of `CA4P-483`. Table 4.5 demonstrates the detailed performance of baselines, i.e., HMM, CRF-based models, BERT-based models, and Lattice-based models, and gives the mean and variance of results obtained from five runs. Besides, we also compare the performance with *manual agreements* to demonstrate task difficulties. Table 4.5 demonstrates that BiLSTM-CRF and Lattice-LSTM achieve over 70% performance on *data* with relatively low variance (i.e., lower than 3%) because the *data* possesses few overlaps with other labels and is in the format of words. *Collect* and *share* only achieve around 60% F1-score because the two types of entities

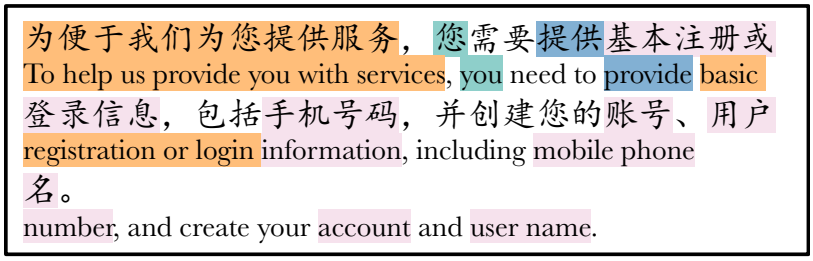


Figure 4.2: Overlapping between components. Differences between ground truth and prediction.

perform some overlapping, as is shown in Figure 4.1 and Figure 4.2. Table 4.5 shows that BiLSTM-CRF achieves better precision on *Condition* than Lattice-LSTM, which may be caused by the fact that *Condition* and *Purpose* are mainly in the format of attributive clauses rather than words.

Next, we analyze the confusion matrix of BiLSTM-CRF results that performs the best on `CA4P-483`. In Figure 4.3, the depth of the background color denotes the proportion of classification; the darker the color, the higher the proportion, and the digit denotes the number of classification results. Figure 4.3 indicates that most of the misclassified samples are related to *Condition*.

To have a deep understanding of divergences between ground truth and predictions, we inspect the misclassifications. We find that the algorithm may fail to identify *Condition*s, which are in the adverbial clause as shown in Figure 4.4(a) where the highlighting for Chinese is ground truth and highlighting for English is prediction results. Besides, when the data controller is the user, as is shown in Figure 4.4(b), the algorithms fail to distinguish *Purpose* and *Condition*. Our experiments also reveal that models need to be well designed to learn deep semantic information, such as distinguishing overlapping among components and distinguishing *Purpose* in modifiers.

Next, we show the prediction results of the algorithm and some common problems. These problems could be the limitations of existing models and also be challenges for designing algorithms for our data scenario.

Table 4.5: Evaluation performance of three types of methods on our dataset. "O" denotes *others*.

	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
	HMM			BiLSTM			BiLSTM-CRF		
Collect	26.03% (±0.00%)	61.98% (±0.00%)	36.30% (±0.00%)	77.14% (±0.81%)	51.54% (±5.13%)	59.75% (±3.79%)	73.90% (±1.42%)	58.81% (±2.67%)	64.87% (±1.73%)
Condition	24.90% (±0.00%)	44.39% (±0.00%)	31.90% (±0.00%)	55.86% (±2.85%)	45.64% (±4.57%)	49.86% (±1.74%)	53.23% (±2.32%)	54.24% (±2.74%)	53.63% (±1.20%)
Data	40.32% (±0.00%)	69.73% (±0.00%)	51.09% (±0.00%)	82.94% (±1.31%)	65.97% (±2.49%)	73.41% (±1.13%)	81.34% (±1.00%)	69.19% (±1.74%)	74.75% (±0.72%)
Handler	21.09% (±0.00%)	51.84% (±0.00%)	29.47% (±0.00%)	78.11% (±1.79%)	49.28% (±7.90%)	59.37% (±5.54%)	76.19% (±1.44%)	58.72% (±1.82%)	66.10% (±0.99%)
Purpose	32.66% (±0.00%)	45.41% (±0.00%)	37.97% (±0.00%)	65.66% (±1.91%)	52.08% (±5.21%)	57.66% (±3.26%)	62.01% (±1.35%)	59.74% (±2.48%)	60.77% (±1.39%)
Share	20.50% (±0.00%)	79.98% (±0.00%)	32.56% (±0.00%)	72.25% (±2.31%)	50.50% (±5.02%)	58.34% (±3.05%)	71.13% (±2.69%)	58.00% (±1.89%)	63.34% (±1.60%)
Receiver	17.56% (±0.00%)	56.40% (±0.00%)	26.76% (±0.00%)	73.86% (±1.16%)	43.67% (±6.09%)	54.40% (±4.64%)	66.69% (±0.32%)	49.63% (±4.04%)	56.72% (±2.83%)
O	90.00% (±0.00%)	67.75% (±0.00%)	77.31% (±0.00%)	89.41% (±0.94%)	94.51% (±0.97%)	91.88% (±0.19%)	90.92% (±0.25%)	92.94% (±0.27%)	91.92% (±0.26%)
Average	77.47% (±0.00%)	66.11% (±0.00%)	69.63% (±0.00%)	85.61% (±0.45%)	86.26% (±0.37%)	85.57% (±0.57%)	86.26% (±0.32%)	86.46% (±0.41%)	86.25% (±0.38%)
	BERT-BiLSTM-CRF			Lattice-LSTM			Manual Agreements		
Collect	61.65% (±14.80%)	52.57% (±8.52%)	55.25% (±6.70%)	79.28% (±0.59%)	81.10% (±1.39%)	80.17% (±0.38%)	96.30%	92.07%	94.14%
Condition	28.23% (±6.08%)	29.91% (±7.46%)	28.85% (±6.26%)	42.57% (±0.61%)	46.97% (±0.21%)	44.66% (±0.24%)	93.53%	84.50%	88.79%
Data	60.54% (±3.90%)	56.97% (±4.82%)	58.40% (±1.39%)	75.47% (±0.45%)	75.54% (±0.97%)	75.50% (±0.26%)	96.20%	91.79%	93.94%
Handler	68.61% (±2.95%)	48.79% (±1.80%)	56.99% (±1.78%)	77.27% (±0.02%)	74.65% (±0.29%)	75.94% (±0.14%)	96.96%	90.18%	93.45%
Purpose	31.43% (±11.66%)	26.35% (±1.44%)	27.22% (±7.44%)	55.18% (±0.57%)	48.09% (±0.62%)	51.39% (±0.10%)	95.64%	92.61%	94.10%
Share	54.32% (±9.75%)	37.70% (±6.03%)	44.37% (±7.14%)	73.78% (±0.16%)	83.01% (±0.68%)	78.12% (±0.21%)	96.10%	94.71%	95.40%
Receiver	38.25% (±6.38%)	32.93% (±1.99%)	35.14% (±3.10%)	54.24% (±3.58%)	55.62% (±0.28%)	54.87% (±1.95%)	97.33%	85.00%	90.75%
O	49.37% (±7.67%)	44.54% (±2.22%)	46.32% (±2.86%)	85.64% (±1.37%)	80.34% (±1.38%)	83.71% (±1.57%)	/	/	/
Average	49.05% (±5.82%)	41.22% (±1.97%)	44.07% (±2.12%)	67.93% (±0.44%)	68.16% (±0.50%)	68.04% (±0.03%)	96.01%	90.12%	92.94%

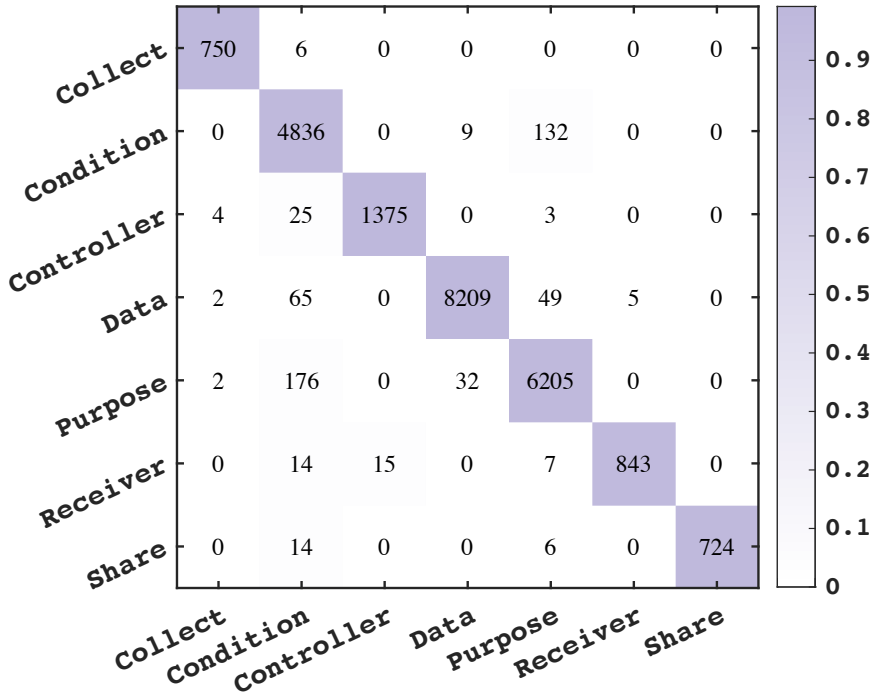


Figure 4.3: Confusion matrix of BiLSTM-CRF results on `CA4P-483`.

本公司可在以下事项中使用用户的个人隐私信息。
The company may use the user's personal privacy information in the following matters.

(a) Missing condition.

用户在申请使用开发人员网络服务时，必须向开发
The user must provide the developer with accurate personal data
人员提供准确的个人资料。
when applying to use the developer‘s network services.

(b) Error prediction when controller is user.

Figure 4.4: The visualization of divergence between ground truth and prediction.

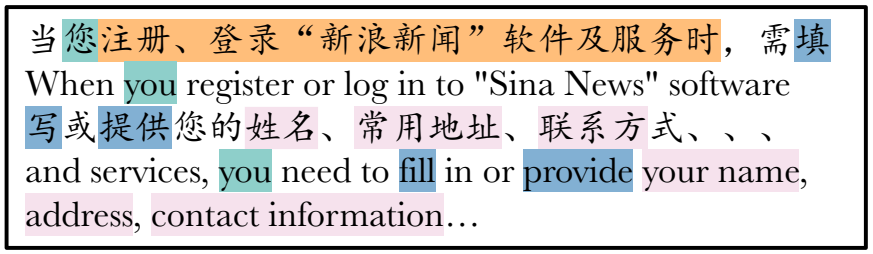


Figure 4.5: The visualization of divergence between ground truth and prediction for missing Purpose.

Figure 4.2 illustrates the scenario where there exists overlapping between components, i.e., the "*basic registration or login information* (基本注册或登录信息)". Exactly, "basic registration or login information" should be one data as is highlighted in the Chinese version, i.e., the ground truth. However, the algorithm will predict "*basic registration or login* (基本注册或登录)" as Purpose and "*information*(信息)" as Data, as is highlighted in the English version. The meaning of color for different categories can be referred to Figure 4.1. Figure 4.5 shows that the pre-trained algorithm may misclassify *Purpose* as *Condition* when we show the prediction results of the algorithm and some common problems. These problems could be the limitations of existing models and also be challenges for designing algorithms for our data scenario.

Figure 4.2 illustrates the scenario where there exists overlapping between components, i.e., the "*basic registration or login information* (基本注册或登录信息)". Exactly, "basic registration or login information" should be one data as is highlighted in the Chinese version, i.e., the ground truth. However, the algorithm will predict "*basic registration or login* (基本注册或登录)" as Purpose and "*information*(信息)" as Data as is highlighted in the English version. The meaning of color for different categories can be referred to Figure 4.1. Figure 4.5 shows the pre-trained algorithm may misclassify *Purpose* as *Condition* when the data controller is the user data controller is the user.

4.5.2 Case Study

In this section, we will present cases of potential applications of `CA4P-483`, such as whether privacy policies comply with regulatory requirements and whether privacy policies are consistent with the apps' functionalities.

Regulation compliance identification. Chinese privacy-related laws [108, 98, 27] ask developers to clearly claim purpose conditions for processing user privacy data. We first investigate the distribution of annotations in `CA4P-483`. Fig.4.6 sketches the box plot of the frequency of components in each privacy policy. Fig.4.6 indicates that some privacy policies claim data processing without clarifying the purpose and condition, i.e., the minimum frequency of *Data* is positive while that of *Purpose* is zero. We manually inspect privacy policies. We find that the privacy policies, whose package name is *com.yitong.weather*, claim the app collects users' data while omitting to give the purposes or conditions of data access, which violates regulation requirements. Thus, `CA4P-483` can facilitate the research in the area of privacy compliance identification [6, 14].

App behavior consistency identification. To improve the security of the Android community, researchers design systems [7, 165] to identify the consistency between privacy policies and app behaviors to prevent apps from abusing user data or conducting malicious behavior. One popular method to check the app's behavior is dynamic analysis [159], i.e., running the app on the device and checking the *log* information. To investigate the application of `CA4P-483` in the security community, we first identify the privacy policies without *purpose* or *condition* components. Then, we install the app on one smartphone, manually interact with the app, and try our best to trigger all possible functions in the app by clicking every visible button. We use *logcat* to capture the app's running information. We find that the app (id: *com.chengmi.signin*) requests device storage to use the app's functionalities, while no condition-related statements are claimed in its privacy policy. With more intelligent automatic soft-

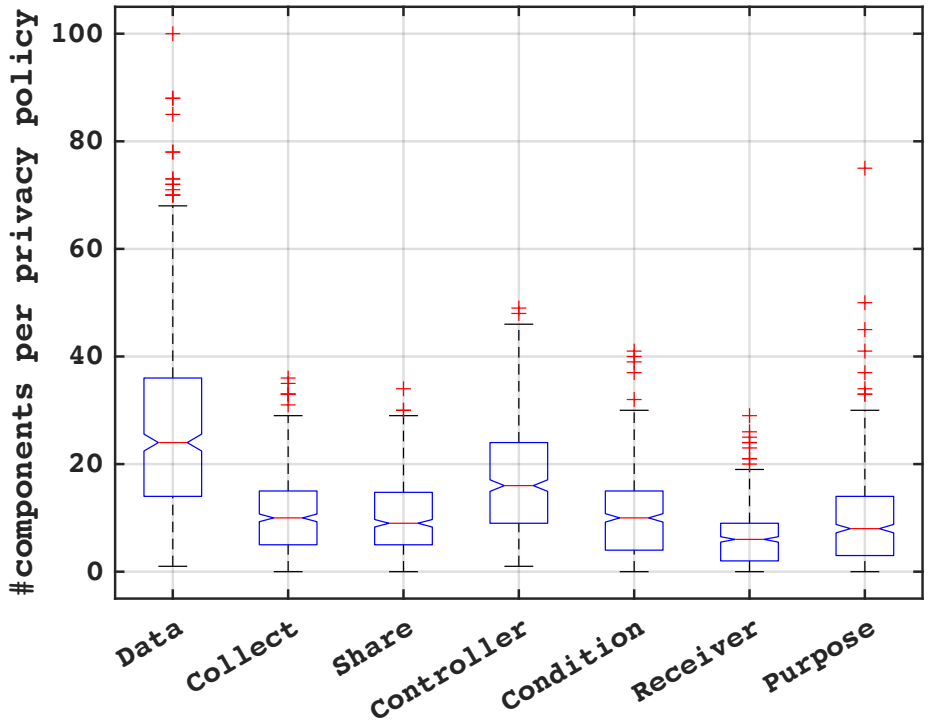


Figure 4.6: Components distribution of CA4P-483.

ware engineering techniques, CA4P-483 can facilitate research in this area, and more vulnerabilities in the consistency between app behavior and privacy policy could be investigated.

4.6 Discussion

In this section, we first discuss difficulties in CA4P-483. Then, we propose potential research topics on CA4P-483. Finally, we discuss the limitations of CA4P-483. Besides, we also discuss ethical concerns.

4.6.1 Dataset Difficulties

Based on evaluation results in §4.5 and related work, we raise the following difficulties: 1) How do we distinguish overlaps between components? 2) How to effectively deal with the length variation of components? 3) Difficulties in semantic analysis.

Different from traditional sequence labeling tasks, components in our data set may contain other components. One scenario is that the Purpose or Conditions may be used to decorate the data, for example, "*We will collect your login information* (我们会收集您的登录信息)" where the *login* may be understood as the purpose of *information*. Since traditional sequence labeling methods predict one character with one label, it is hard to distinguish component overlaps. One possible solution is using multi-model algorithms [134] that demonstrate effectiveness for distinguishing boundaries between entities. Similar to traditional news or social media datasets that use voice or images as additional information, integrating apps' analysis results helps distinguish different components.

Second, existing sequence labeling tasks mainly concentrate on entity recognition, while practical applications may require labeling clauses for further analysis. Table 4.1 shows that the average length of components in `CA4P-483` varies from 2.03 to 19.24. `CSP`^{3} not only requires identifying words but also asks the models to identify the role of clauses.

The semantic analysis of privacy policies is still difficult. Laws require apps to clearly clarify how apps collect and share user data. Privacy policies can claim that apps will *share* data with third parties or that third parties will *collect* user data. In this way, it becomes essential to understand the context to distinguish the controller and action type. It could be a solution to use multi-model algorithms integrating program analysis to improve the performance; however, identifying the third party and the app itself remains a challenge in program analysis.

4.6.2 Limitations

CA4P-483 provides detailed annotations for data access statements in privacy policies. However, analyzing privacy policies using CA4P-483 depends on the performance of locating data access-related sentences. We use data collection and sharing words to locate the sentences. However, some *Purpose* and *Condition* claims may be given as an enumeration format, such as "*we will not share your personal data under the following conditions*". CA4P-483 is limited when capturing information in the enumeration format.

Privacy policies possess timeliness. App developers should provide privacy policies when publishing the apps. When the apps' functionality updates, the privacy policies ought to be updated accordingly. The data set is limited to the timestamp we collected. When combining our dataset with program analysis, this factor should be considered.

The varying regulatory requirements for privacy policies across regions pose significant challenges. The label design in CA4P-483 is based on Chinese regulatory frameworks [27, 100, 98]. However, regulatory authorities impose stringent and distinct requirements for app user data access in different regions, exemplified by the General Data Protection Regulation (GDPR) in Europe [47]. This necessitates the development of adaptable privacy policy analysis systems that can account for region-specific regulatory nuances.

4.6.3 Ethical Consideration

CA4P-483 is a dataset constructed by gathering publicly available privacy policy websites without posing any ethical problems. First, privacy policies are publicly accessible in multiple ways. According to the application market's requirements, developers or companies are asked to provide those privacy policy websites once they publish

their apps. Privacy policies also ought to be given when the users use apps for the first time according to law requirements [108]. Second, we do not collect any privacy-related information. Besides, the `CA4P-483` is proposed to prompt research for protecting user privacy.

For the annotations, we hired part-time research assistants from our university to label the dataset. They are compensated with 9 USD/hour and at most 17.5 hours per week.

Chapter 5

Investigating Pre-trained Large Language Models for Chinese Privacy Policy Analysis

5.1 Overview

Pretrained large language models (LLMs) achieve great success in understanding the semantics of natural language. Pre-trained LLMs are trained on vast collections of diverse natural language resources, including Wikipedia, publicly available news, books, and programming code, among others [185, 102]. Consequently, LLMs are equipped with general natural language understanding abilities, such as document summarization [175, 22], sentence completion [99], etc. In addition to general-purpose capabilities, LLMs fine-tuned on specific datasets for particular tasks have shown exceptional performance. For example, LLMs trained on programming-related data excel in tasks such as code completion[81, 95], code generation[104, 86], code summarization[1, 75], etc. However, achieving such performance often requires access to extensive domain-specific datasets for training[140] or fine-tuning[84], which can be resource intensive.

In addition, recent research highlights that LLMs perform suboptimally on specialized downstream tasks when they are not explicitly trained for those purposes. Tasks like summarizing legal documents or drafting scripts exemplify these limitations. To address these challenges and improve the capabilities of LLMs for domain-specific tasks, researchers and industry professionals have increasingly adopted prompting-based approaches [46, 82]. These methods leverage the inherent generative capabilities of LLMs to tackle a wide range of tasks without the need for extensive retraining or fine-tuning.

Existing applications of Large Language Models (LLMs) for privacy policy analysis primarily focus on summarizing privacy policies [139, 80], identifying privacy issues within LLM applications [23], and analyzing privacy policies in specific domains with a coarse-grained approach [90, 116]. Although these studies provide valuable information, the studies often limit their scope to specific domains, thus restricting their applicability to various real-world scenarios. Besides, existing work of accessing capabilities of LLM for privacy policies analysis focuses on English privacy policies. Furthermore, while LLMs offer powerful capabilities for natural language processing, the application of LLMs is still constrained by practical limitations, such as input length restrictions.

In this work, we aim to investigate the potential of applying pre-trained large language models (LLMs), which have demonstrated powerful natural language semantic understanding capabilities, to the task of analyzing Chinese privacy policies. This task presents two primary challenges: (1) Pretrained LLMs are predominantly trained on English corpora, which may limit their performance in Chinese-language environments, and (2) Privacy policies are often lengthy, with the same phrases and nouns potentially serving different roles across different sentences. To address these challenges, we adopt the following strategies: (1) Decomposing the end-to-end task, which refers to analyzing the entire privacy policy and generating a final analysis report, into multiple sentence-level analysis tasks, and (2) Applying prompt engineering techniques,

such as few-shot learning, to enhance the performance of LLMs on analyzing Chinese privacy policies. We empirically evaluate the capabilities of Large Language Models (LLMs) for analyzing regulation-required items in privacy policies. Specifically, we carefully craft prompts with advanced prompt engineering techniques to query various popular pre-trained LLMs, aiming to identify regulation-required items within sentences of privacy policies. Our experiments on both publicly available LLMs, i.e., LLaMA and Qwen, and the popular commercial LLM, i.e., ChatGPT, demonstrate the models' effectiveness in analyzing privacy policy tasks. We also systematically analyze the results and provide further directions for leveraging the capabilities of LLMs to protect user privacy.

The main contributions of this work are summarized as follows:

(1) We empirically evaluate the performance of both popular publicly available LLMs and commercial LLMs in identifying regulation-required items in privacy policies. Our evaluations employ advanced prompt engineering techniques to instruct LLMs in handling downstream tasks effectively.

(2) We systematically analyze the evaluation results and provide further directions for leveraging the capabilities of LLMs to protect user privacy. Our analysis highlights key insights into the strengths and limitations of LLMs in this context, offering recommendations for future research and practical applications aimed at enhancing privacy protection.

5.2 Preliminaries

In this section, we present the necessary knowledge on applying pre-trained large language models for downstream tasks, including a basic introduction to pre-trained large language models and prompt engineering.

5.2.1 Pre-trained Large Language Models

Pre-trained large language models (LLMs) revolutionize the field of natural language processing (NLP) by enabling significant advancements across a wide range of downstream tasks. LLMs are trained on massive corpora using self-supervised learning, allowing the models to capture rich contextual representations of language. Notable LLMs include BERT [35], which introduced bidirectional pretraining for language understanding, and the GPT series [16], which demonstrated remarkable generative capabilities using autoregressive modeling. Additionally, more recent models, such as T5 [111], have emphasized the versatility of sequence-to-sequence architectures for both understanding and generation tasks. LLMs are adapted to specific domains, including legal [21] and biomedical [57] text, illustrating their potential for domain-specific applications. However, challenges remain in applying these models to tasks requiring deep contextual understanding or domain adaptation, such as analyzing privacy policies.

5.2.2 Prompt Engineering

Prompt engineering emerges as a critical technique for optimizing the performance of pre-trained large language models (LLMs) across various natural language processing (NLP) tasks. By designing effective prompts, researchers aim to guide LLMs to generate more accurate and contextually appropriate outputs, even without extensive fine-tuning. Early work on prompt engineering, such as PET [118], demonstrates how template-based prompting could be used to adapt LLMs for few-shot learning scenarios. The introduction of zero-shot and few-shot prompting paradigms in GPT-3 [16] further highlights the importance of carefully crafting prompts to elicit desired behavior. AutoPrompt [121] explores automated approaches to generate optimal prompts. Additionally, chain-of-thought prompting [148] shows that structuring prompts to encourage step-by-step reasoning can significantly improve performance on complex

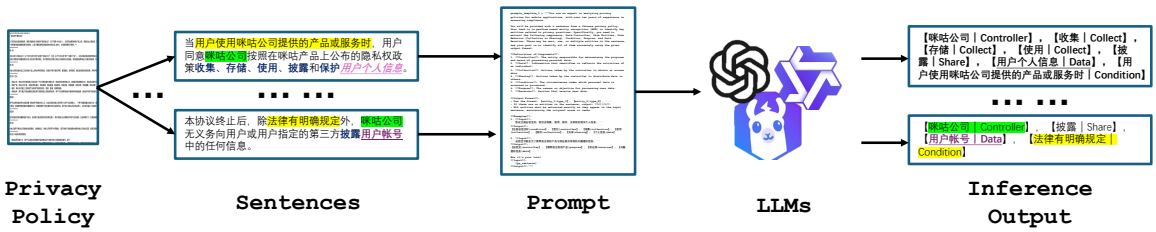


Figure 5.1: Framework of LLMPP

tasks. Existing work underscores the significance of prompt engineering as a flexible and effective method for leveraging the capabilities of LLMs in both general and domain-specific contexts.

5.3 Framework

In this section, we introduce the framework for our LLM-based Privacy Policy analysis (LLMPP). Figure 5.1 illustrates the framework of LLMPP. Given a Chinese privacy policy document, LLMPP first segments the document into sentences containing specific keywords of interest (§5.3.2). Next, each sentence is embedded into a carefully crafted prompt (§5.3.3). Then, the prompts will be used to query different pre-trained LLMs. Each sentence is used to query each LLM individually every time. Finally, LLMPP retrieves the generated content from the LLMs, parses the output, and evaluates the performance of the LLMs in identifying key entities and entity types within the sentences.

In the following of this section, we first define the tasks that LLMPP is designed to address. Next, we explain the process of analyzing privacy policies and how they are prepared for querying LLMs to infer the desired outputs. Finally, we introduce the prompt design, which incorporates prompt engineering techniques.

5.3.1 Task Description

LLMPP is designed to identify regulation-required elements within privacy policies. Specifically, LLMPP is required to determine who is responsible for data-related actions, such as collection or sharing, under what conditions, and for what purposes. Additionally, LLMPP identifies the parties that will receive the shared data if applicable. Given the token limitations of large language models (LLMs), such as 4K tokens for ChatGPT-3.5 [102] and 128K tokens for QWen [160], LLMPP processes privacy policies on the sentence level. Moreover, as LLMs have demonstrated unreliable performance in numeric counts [124, 176], LLMPP is designed to directly output the identified entities and entity types in the provided sentences, rather than labeling individual characters within the sentences which is a kind of numeric counting task. For example, the sentence "我们会收集你的邮箱地址" will be labeled as "B-Controller, E-Controller, O, B-Collect, E-Collect, O, O, B-Data, I-Data, I-Data, E-Data" in traditional named entity tasks such as experiments in CA4P-483. When an LLM is tasked with producing the same number of labels as the number of characters in a sentence, the output becomes highly unreliable and difficult to control [124, 176]. We also conducted a case study (§5.4.6) to evaluate the performance of LLMs when the task is defined as labeling each character in sentences. The results demonstrate poor performance. Thus, LLMPP is designed to output a more structured result: "【我们|Controller】, 【收集|Collect】, 【邮箱地址|Data】."

5.3.2 Privacy Policy Preprocessing

To enable LLMPP to accurately identify regulation-required items in privacy policies, we first extract sentences of interest using the same methodology as CA4P-483. Following previous research [6, 183, 58], we iterate through each sentence in the privacy policy and determine whether it contains keywords related to sharing or collection, based on a predefined word list as is given in Table 4.3 [183]. Sentences contain-

ing predefined keywords will be embedded in prompts to query LLMs for identifying regulation-required entities and corresponding entity types.

5.3.3 Prompt Design

Existing work demonstrates that a well-designed prompt is highly effective in leveraging the capabilities of Large Language Models (LLMs) for resolving downstream domain-specific tasks. Drawing on advanced prompt engineering techniques, we carefully craft our prompt to harness the full potential of LLMs in identifying privacy components within sentences of Chinese privacy policies. Our prompt design is detailed in Figure 5.2. Our approach to prompt design focuses not on discovering the optimal prompts to achieve the highest performance metrics for pre-trained large language models, but rather on applying advanced prompt engineering techniques to create effective prompts tailored for handling privacy policies. We recommend that users adapt our prompt by substituting specific definitions to better fit their particular use cases and scenarios. Next, we provide a detailed introduction to our prompt and explain the rationale behind its design.

Our prompt begins by **assigning a role** to the LLM: "You are an expert in..." that aims to make the LLM analyze privacy policies in a more professional manner. Assigning a role[115] to the LLM has been shown to establish a behavioral framework for the model, influencing its tone, style, and level of domain-specific expertise. Additionally, this approach can increase domain relevance and ensure that the output reflects professional-level analysis.

Following by, we explicitly describe the definition of the task according to [185], ensuring that LLMs understand the purpose of the task. Specifically, we first describe the type of data that will be input for analysis that corresponds to "You will be provided with a sentence from a Chinese privacy policy" in our prompt. The task then requires the LLM to perform a named entity recognition (NER) task, i.e., "Your

```
LLM_PP_Prompt = """You are an expert in analyzing privacy policies
for mobile applications, with over ten years of experience in
assessing compliance.

You will be provided with a sentence from a Chinese privacy policy.
Your task is to perform named entity recognition (NER) to identify
key entities related to privacy practices. Specifically, you need to
extract the following components: Data Controller, Data Entities,
Data Behavior (Collection or Sharing), Condition, Purpose, and Data
Receiver. There may be zero, one, or multiple entities in the
sentence, and your goal is to identify all of them accurately using
the given output format.

**Definitions of Components**:
1. **Controller**: The entity responsible for determining the
purposes and means of processing personal data.
2. **Data**: Information that identifies or reflects the activities
of an individual.
3. **Collection**: Actions taken by the controller to obtain or
access data.
4. **Sharing**: Actions taken by the controller to distribute data
to others.
5. **Condition**: The circumstances under which personal data is
accessed or processed.
6. **Purpose**: The reason or objective for processing user data.
7. **Receiver**: Parties that receive user data.

**Output Format**:
- Use the format: 【entity_1|type_1】, 【entity_2|type_2】, ...
- If there are no entities in the sentence, output: **</|/>**.
- ALL entities must be extracted exactly as they appear in the input
sentence, maintaining the original nouns or verbs.

**Examples**:
1. **Input**:
```您在注册启信宝时，我们会收集、使用、保存、共享您的相关个人信息。```
**Output**:
【注册启信宝时|condition】, 【我们|controller】, 【收集|collection】, 【使
用|collection】, 【保存|collection】, 【共享|sharing】, 【个人信息|data】

2. **Input**:
```启信宝可能会为了推荐适合您的产品与供应商共享您的兴趣爱好信息。```
**Output**:
【启信宝|controller】, 【推荐适合您的产品|purpose】, 【供应商|receiver】,
【兴趣爱好信息|data】

Now it's your turn!
**Input**:
```{pp_sentence}```
**Output**:"""
```

Figure 5.2: Prompt Design for Leveraging LLM for Analyzing Privacy Policies
```

task is to perform named entity recognition (NER) to identify key entities related to privacy practices" in our prompt. The LLM is tasked with identifying seven types of components within the provided privacy policy sentences. A clear task description provides the model with a precise and unambiguous understanding of the downstream tasks, minimizing confusion and hallucinations, and ensuring the model focuses on the specified tasks. This approach enhances the accuracy and reliability of the LLM's output by guiding it to concentrate on relevant information.

After describing the tasks, we apply **context-aware prompting** [16] to clearly highlight the definition of each component to be analyzed. For instance, "Controller" is described in terms specific to data governance. By providing such definitions, we aim to ensure that the LLMs clearly understand the scope of each component, referencing LLM's general knowledge from training data. Context-aware prompting [185, 16] has been shown to enhance the accuracy of outputs. Moreover, context-aware prompting enables the model to better generalize when dealing with nuanced or ambiguous terms prevalent in specialized fields. By enhancing the model's understanding of domain-specific terminology, context-aware prompting improves the reliability and applicability of the LLM's output in complex scenarios.

Following the clear task description and component definitions, we further leverage **Instruction-Tuned Prompting** [103, 87] that states the expected output structure in our prompt. Specifically, we ask LLM to use 【$entity\_1|type\_1$】 for entity annotations or $**<//>**$ when no entities are found to facilitate further parsing of the results of generated content by LLMs. We specify the output format for the LLMs. Clear descriptions of the output format aim to unify the generated content from the LLMs, facilitating further analysis of LLMs' performance and simplifying the result parsing process.

Next, we apply few-shot learning techniques [16] to provide the LLM with a few examples to better understand the task definitions and the required output format. Few-shot learning has been demonstrated to effectively enhance task comprehension,

particularly in specialized domains, by providing the model with concrete examples of context, patterns, and output structure.

When using the prompt to query LLMs for analyzing privacy policies, the placeholder {*pp_sentence*} will be replaced by the specific sentence to be analyzed.

5.4 Experiments

5.4.1 Model Summaries

In this study, we empirically evaluate the three most popular large language models. The first two models are publicly accessible pre-trained models, i.e., Qwen [160] and LLaMA [141], while the third is the commercially dominant generative AI product, i.e., ChatGPT [102, 71]. Specifically, we evaluate Qwen 2.0 7B, LLaMA 3.1 8B, and ChatGPT 3.5 Turbo-0613.

LLaMA 3.1 8B [141], released in April 2024, features 8 billion parameters and was trained on an extensive dataset of over 15 trillion tokens, enabling it to handle contexts of up to 8,000 tokens. Qwen 2.5 7B [160], launched on 19 September 2024, has 7 billion parameters and is fine-tuned for instruction-following tasks, although the precise size of its training dataset is not disclosed. ChatGPT 3.5 [102], which does not officially disclose its parameter count, is estimated to have approximately 175 billion parameters and was trained on a dataset containing 300 billion tokens, with updates made until April 2023.

Each of these models represents a notable advancement in large language model technology, with LLaMA 3.1 8B and Qwen 2.5 7B being more recent releases, and ChatGPT 3.5 is widely recognized for its extensive training and broad capabilities.

5.4.2 Experiment setup

To assess the effectiveness of large language models (LLMs) in processing Chinese privacy policies, we conduct a series of experiments to evaluate the performance of three widely adopted models on the CA4P-483 dataset [183], which contains a comprehensive collection of Chinese privacy policy documents accompanied by detailed annotations. CA4P-483 provides Chinese privacy policy texts and annotates sentences related to data collection and sharing. Each annotation specifies who is responsible for collecting or sharing what type of personal data, with whom, and under which conditions or for what purposes.

For the LLMs, the configuration details of each model are as follows: For ChatGPT 3.5, we utilize the Python *requests* library to interact with the GPT-3.5 API, specifying only the user role and populating the predefined prompt in the *content* field. For Qwen 2.5 and LLaMA 3.1, we evaluate the performance of these LLMs on an Ubuntu 20.04 server equipped with four NVIDIA A100 GPUs (each with 80 GB of memory), 1 TB of RAM, and an Intel(R) Xeon(R) Platinum 8358 CPU running at 2.60 GHz.

Our experiments are finally designed to answer the following three research questions:

- **RQ1: Main results.** What is the effectiveness of models in identifying privacy components in privacy policies?

- **RQ2: Ablation study.** What is the impact of different prompt engineering techniques on model performance?

- **RQ3: Hallucination analysis.** What are hallucinations in LLMPP, and how does prompt engineering mitigate hallucinations in LLMPP?

Table 5.1: Main Results

Entity Type	GPT			Llama			Qwen		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
O	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Data	87.02%	74.10%	80.04%	83.79%	58.69%	69.03%	77.99%	17.24%	28.24%
Controller	71.34%	48.08%	57.44%	75.61%	54.72%	63.49%	52.45%	23.19%	32.16%
Collection	69.69%	28.94%	40.89%	74.26%	42.79%	54.30%	31.35%	9.05%	14.05%
Sharing	89.16%	32.65%	47.80%	90.24%	23.66%	37.50%	60.78%	5.08%	9.37%
Condition	45.84%	30.01%	36.27%	36.15%	29.87%	32.71%	4.20%	7.01%	5.26%
Purpose	45.52%	33.27%	38.44%	61.00%	21.28%	31.55%	46.29%	5.68%	10.12%
Receiver	63.50%	58.94%	61.13%	71.25%	47.59%	57.07%	75.11%	10.93%	19.08%
Overall	49.02%	49.02%	49.02%	44.24%	44.24%	44.24%	13.26%	13.26%	13.26%

5.4.3 RQ1. Effectiveness of LLMs in analyzing privacy policies.

Table 5.1 presents the main results of three LLMs on all 18,579 sentences from the CA4P-483 dataset. In line with previous research [183], we evaluate the results using the Precision, Recall, and F1 score. Since LLMs may not always adhere to the instructions to produce outputs in the predefined format, we exclude results that do not align with the required output format. Specifically, ChatGPT-3.5 strictly follows the predefined output format, while LLaMA generates 238 instances, and QWen 2.5 produces 365 instances that deviate from the expected format. Additionally, the entity type *O* in Table 5.1 denotes entities that are not specified in the ground truth but are identified by the LLM as belonging to one of the predefined entity types. To parse the results generated by the LLM, we only consider outputs that strictly adhere to the required format. Specifically, entities and their corresponding types must follow the format "【entity | entity_type】", where each item represents a single entity, and multiple entities are separated by commas. For example, acceptable outputs include

“【个人信息 | Data】, 【共享 | Sharing】”.

Table 5.1 presents the main results and reveals that all large language models (LLMs) achieve higher precision than recall across all entity types. This suggests that while LLMs are effective in correctly identifying entities as belonging to their defined types, LLMs often misclassify entities of other types as belonging to these categories. Among the entity types, all LLMs demonstrate strong performance in identifying data entities. However, the performance of LLMs drops significantly for condition and purpose entities, with Qwen achieving an F1-score as low as 5.26% for condition entities. The results align with the analysis presented in Table 4.5, as conditions and purposes are inherently challenging to distinguish, even for human analysts, without sufficient context. In addition, overall metrics are notably lower than average metrics for individual entity types. This discrepancy indicates that a substantial number of entities are incorrectly classified as $\boldsymbol{O}$ (i.e., non-entities). To gain deeper insights into these misclassifications, we conduct a detailed analysis of the confusion matrices for each LLM.

Figures 5.3, 5.4, and 5.5 present the confusion matrices for ChatGPT-3.5, LLaMA, and Qwen, respectively. Since the LLMPP task definition does not include O as a valid entity type in the ground truth, the first rows of all confusion matrices are entirely zeros. Another phenomenon observable from the confusion matrices is that the LLMs pretend to misclassify entities of known types as non-predefined entity types rather than identify the entities as known types. This suggests that LLMs may not have fully learned the definitions of all entity types but could correctly distinguish the difference between different entity types from the definition. Additionally, a plausible explanation for this phenomenon could be that LLMs are primarily pre-trained on English corpora and thus struggle to generalize effectively to Chinese sentences. Since the CA4P-483 dataset was released in July 2022, it is worth noting that even though evaluated models were released after the dataset became available, LLMs still perform poorly on these tasks. This underscores the challenges inherent in privacy policy

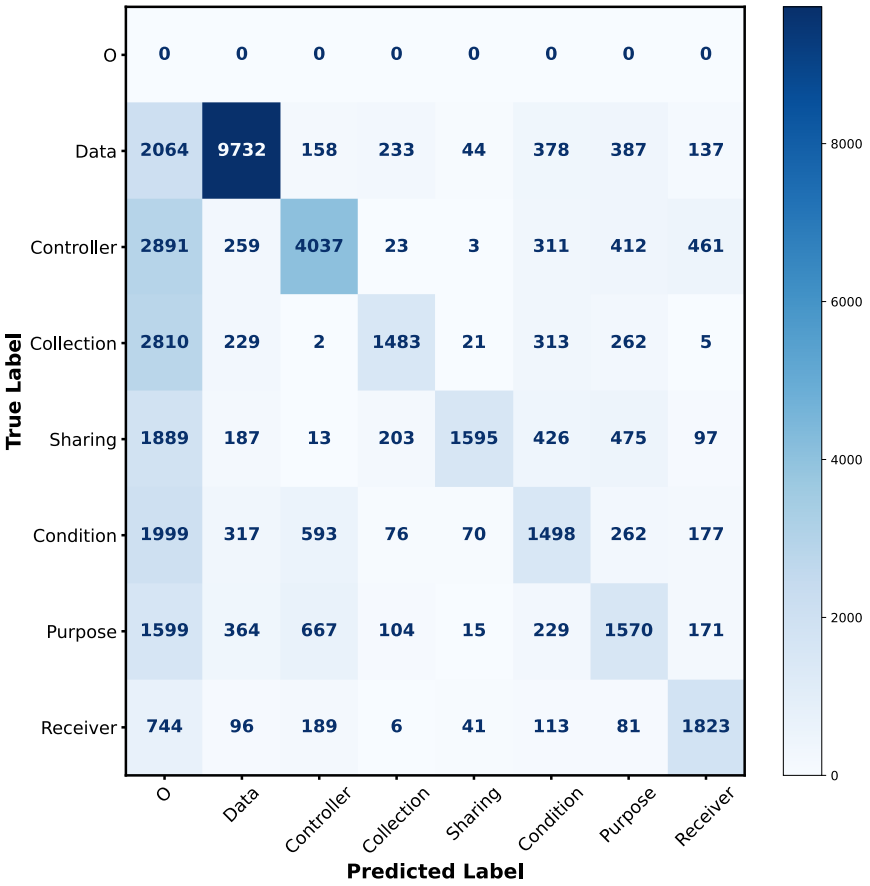


Figure 5.3: Confusion metrics of ChatGPT 3.5.

analysis tasks.

Answer to RQ1: Identifying privacy components remains a challenging task for general pre-trained language models. While LLMs demonstrate the capability to correctly identify privacy-related components in privacy policies, they also exhibit a tendency to misclassify non-privacy-related components as privacy-related ones.

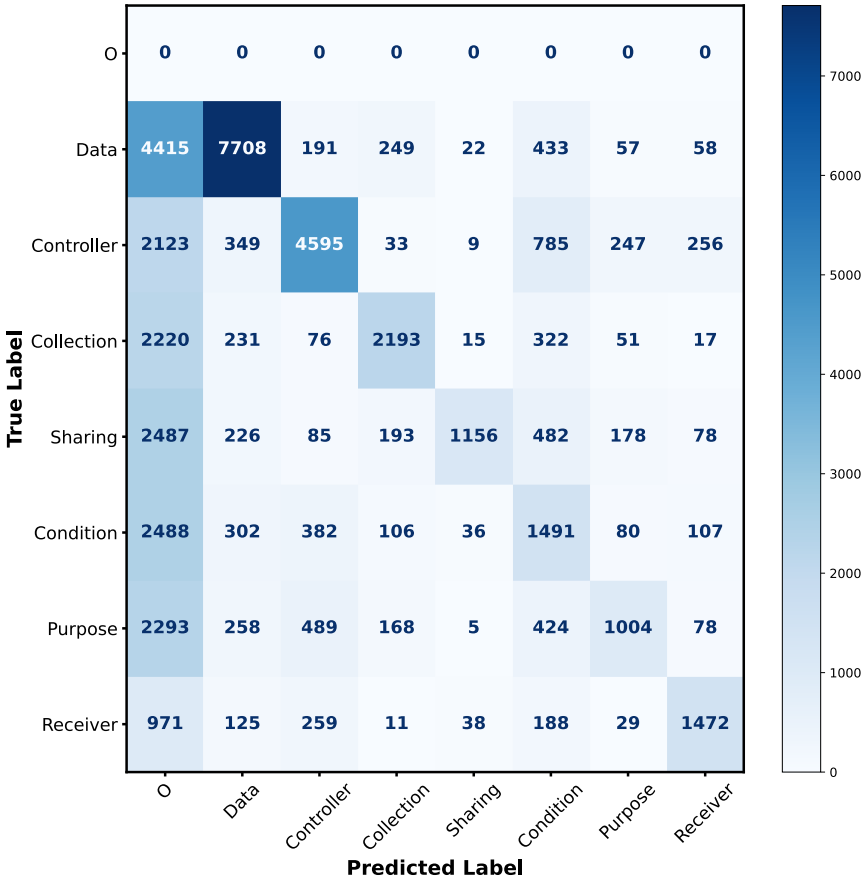


Figure 5.4: Confusion metrics of LLaMA.

5.4.4 RQ2: Impact of Prompt Engineering Techniques on Model Performance

This research question conducts an ablation analysis to evaluate the impact of various prompt engineering techniques applied in our prompt design on the performance of LLMPP. While the primary goal of this work is not to identify optimal prompts for maximizing LLM performance, the techniques incorporated into our prompt design are derived from existing reports, research, and documented methods that have demonstrated effectiveness in general scenarios. The ablation study aims to evaluate whether prompt engineering techniques enhance performance in the specific context of privacy policy analysis. Additionally, the ablation study seeks to provide researchers

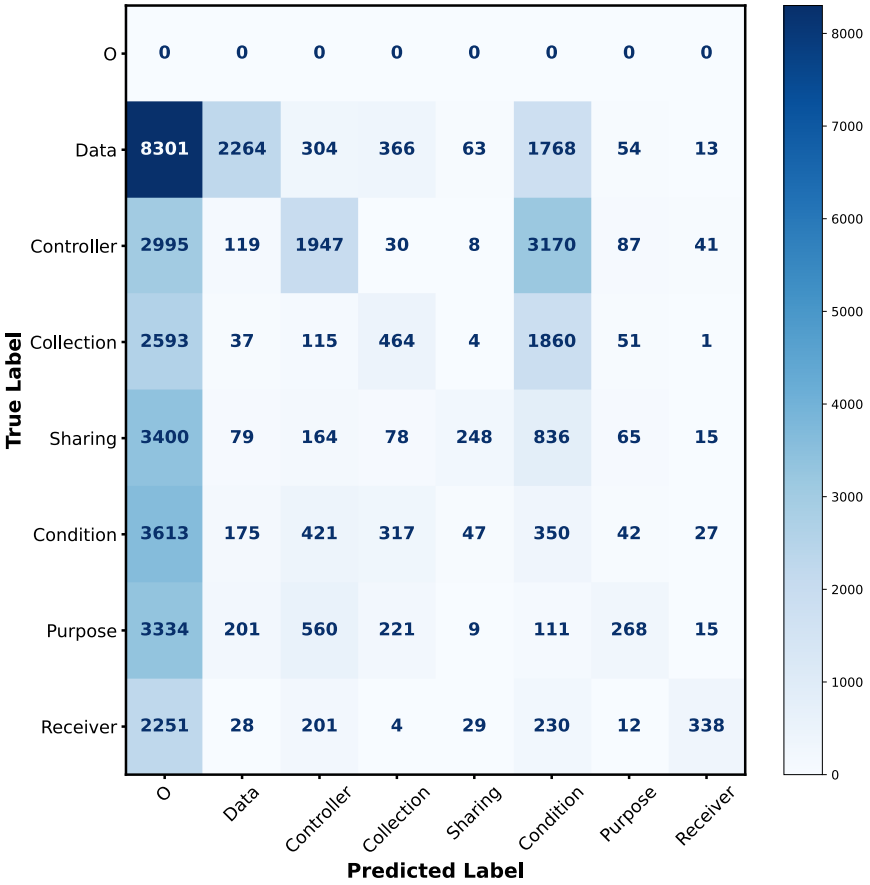


Figure 5.5: Confusion metrics of QWen.

with practical insights and references for designing prompts tailored to their application scenarios.

To address this research question, we conduct an ablation study by isolating and evaluating each prompt engineering technique individually. The prompt techniques include assigning a role prompting (ARP), task description (TDP), context-aware prompting (CAP), instruction-tuned prompting (ITP), and few-shot prompting (FSP). To evaluate each technique, we use all other techniques collectively to query the target Large Language Model (LLM) and analyze its performance on privacy policy sentences. To ensure fairness and consistency, we apply a standardized results parser format across all outputs generated by the LLMs. Specifically, the results are parsed using the

Table 5.2: Ablation study results of prompt engineering techniques in LLMPP

		ARP			TDP			CAP			ITP			FSP		
	Class	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
GPT	O	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Data	85.04%	74.65%	79.51%	87.08%	72.49%	79.12%	85.47%	74.81%	79.79%	87.49%	68.89%	77.08%	0.00%	0.00%	0.00%
	Controller	71.85%	48.14%	57.65%	74.04%	55.73%	63.60%	69.66%	48.12%	56.92%	72.41%	49.57%	58.85%	0.00%	0.00%	0.00%
	Collection	68.41%	27.84%	39.58%	63.70%	33.21%	43.66%	67.97%	22.07%	33.32%	73.78%	28.60%	41.23%	0.00%	0.00%	0.00%
	Sharing	88.39%	30.40%	45.24%	87.18%	34.25%	49.18%	90.96%	20.39%	33.31%	89.44%	28.43%	43.15%	0.00%	0.00%	0.00%
	Condition	47.52%	29.17%	36.15%	42.54%	35.94%	38.96%	46.02%	29.31%	35.81%	44.78%	24.50%	31.67%	0.00%	0.00%	0.00%
	Purpose	47.91%	32.87%	38.99%	41.97%	36.47%	39.02%	52.74%	33.46%	40.94%	38.50%	29.29%	33.27%	0.00%	0.00%	0.00%
	Receiver	66.08%	60.52%	63.18%	69.10%	59.88%	64.16%	63.99%	57.23%	60.42%	69.30%	52.12%	59.49%	0.00%	0.00%	0.00%
	Overall	48.79%	48.79%	48.79%	51.74%	51.74%	51.74%	0.469173	46.92%	46.92%	45.74%	45.74%	45.74%	0.00%	0	0
	Class	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Llama	O	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Data	84.23%	52.40%	64.60%	82.69%	59.91%	69.48%	78.74%	59.81%	67.98%	85.91%	52.97%	65.53%	0.00%	0.00%	0.00%
	Controller	75.40%	52.58%	61.96%	70.71%	58.34%	63.93%	71.84%	49.82%	58.84%	78.66%	48.62%	60.09%	0.00%	0.00%	0.00%
	Collection	73.83%	37.78%	49.98%	61.26%	51.76%	56.11%	74.37%	32.30%	45.04%	74.27%	46.54%	57.23%	0.00%	0.00%	0.00%
	Sharing	89.36%	19.18%	31.58%	79.44%	33.48%	47.11%	93.22%	18.15%	30.38%	79.92%	24.59%	37.60%	0.00%	0.00%	0.00%
	Condition	34.91%	30.83%	32.74%	32.27%	36.98%	34.46%	35.86%	24.37%	29.02%	23.19%	36.81%	28.45%	0.00%	0.00%	0.00%
	Purpose	61.87%	20.29%	30.56%	54.71%	25.10%	34.41%	63.89%	18.18%	28.31%	60.79%	25.32%	35.75%	0.00%	0.00%	0.00%
	Receiver	71.72%	42.97%	53.74%	66.68%	49.11%	56.56%	71.31%	49.70%	58.57%	71.44%	41.00%	52.10%	0.00%	0.00%	0.00%
	Overall	40.58%	40.58%	40.58%	48.72%	48.72%	48.72%	0.410468	41.05%	41.05%	42.67%	42.67%	42.67%	0.00%	0	0
	Class	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Qwen	O	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Data	80.01%	11.03%	19.39%	81.68%	11.49%	20.14%	83.21%	11.81%	20.68%	77.66%	7.69%	13.99%	0.00%	0.00%	0.00%
	Controller	60.73%	20.08%	30.19%	52.86%	19.28%	28.26%	44.97%	16.55%	24.19%	45.01%	10.71%	17.31%	0.00%	0.00%	0.00%
	Collection	42.53%	6.20%	10.82%	44.28%	8.27%	13.94%	34.13%	7.17%	11.85%	22.10%	4.88%	7.99%	0.00%	0.00%	0.00%
	Sharing	73.82%	3.47%	6.62%	76.87%	6.18%	11.44%	71.68%	4.30%	8.12%	55.88%	2.69%	5.13%	0.00%	0.00%	0.00%
	Condition	8.64%	5.15%	6.45%	10.63%	10.17%	10.40%	5.14%	8.08%	6.28%	4.80%	7.73%	5.92%	0.00%	0.00%	0.00%
	Purpose	47.12%	3.57%	6.63%	42.82%	4.59%	8.29%	46.37%	4.06%	7.47%	35.62%	3.37%	6.15%	0.00%	0.00%	0.00%
	Receiver	72.26%	7.63%	13.81%	70.37%	9.74%	17.10%	74.29%	10.94%	19.08%	66.24%	6.92%	12.52%	0.00%	0.00%	0.00%
	Overall	9.65%	9.65%	9.65%	11.01%	11.01%	11.01%	0.100427	10.04%	10.04%	6.88%	6.88%	6.88%	0.00%	0	0

format 【*entity*_1|*type*_1】. This uniform approach is essential because the results parser is an integral part of our prompt design. It facilitates efficient and accurate parsing of the model outputs, ensuring that the results can be reliably interpreted and compared across different scenarios.

Table 5.2 presents the results of the ablation study on prompt engineering techniques in LLMPP. Assigning a Role Prompting (ARP) slightly improves the performance of all three models. Without ARP, the F1-score for GPT drops from 49.02% to 48.79%, and for Llama, the F1-score decreases from 44.24% to 40.58%. The F1-score of Qwen also decreases from 13.26% to 9.65%. Task Description Prompting (TDP) enhances the performance of Qwen, increasing its F1-score from 11.67% to

19.45%. However, TDP demonstrates side effects on the performance of GPT and Llama. Without TDP, the F1-score of GPT increases from 49.02% to 51.74%, and the F1-score of Llama increases from 44.24% to 48.72%. Both Context-Aware Prompting (CAP) and Instruction-Tuning Prompting (ITP) demonstrate consistent and effective improvements across all three models too. The performance of CAP and ITP suggests that providing clear definitions of each privacy component and the expected output format help the models better understand the task, leading to improved performance. This highlights the importance of precise and context-aware prompt design in guiding LLMs for specific tasks.

Another phenomenon is that without few-shot prompting (FSP), all three models achieve the worst performance. This indicates that providing examples to LLMs can effectively help LLMs understand the requirements of the task, the definition of each privacy component, and the output format requirements. We manually analyze the results of few-shot prompting. We observe that without FSP, the output of LLMs cannot follow the requirements of the output format. Some outputs do not strictly adhere to the required format, such as failing to enclose content within brackets 【】 or omitting the use of vertical bars — for separation. For example, outputs like "个人信息 | data" or "【电话号码， data】" do not fully comply with the specified format. Additionally, some outputs begin with the entity type followed by a colon and the corresponding entities under that type, such as "data: 个人信息，电话号码". The first format may arise from the LLM interpreting the brackets "【】" as merely a presentational element rather than a strict formatting requirement. Another format may stem from the type definition structure used in Task Description Prompting (TDP). To further analyze the performance of LLMs without FSP, we incorporate parsing mechanisms with both of the aforementioned two formats and reanalyze the results accordingly.

Table 5.3 presents the results of the ablation analysis for few-shot prompting, incorporating two additional text parsing formats. Compared to Table 5.2, the ex-

Table 5.3: Additional Analysis of Few-Shot Prompting Ablation Results

Entity Type	GPT			Llama			Qwen		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
O	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Data	45.81%	17.78%	25.62%	41.00%	20.60%	27.42%	44.39%	7.40%	12.68%
Controller	74.78%	2.01%	3.92%	75.60%	4.35%	8.23%	47.95%	3.00%	5.65%
Collection	54.90%	2.19%	4.20%	28.12%	0.56%	1.10%	15.12%	0.32%	0.63%
Sharing	42.21%	1.72%	3.30%	53.66%	0.48%	0.95%	25.00%	0.08%	0.15%
Condition	24.98%	7.13%	11.10%	33.49%	4.48%	7.90%	39.76%	1.67%	3.21%
Purpose	28.98%	9.56%	14.37%	34.91%	4.65%	8.21%	14.53%	0.67%	1.28%
Receiver	55.73%	3.46%	6.51%	100.00%	0.10%	0.20%	77.27%	0.69%	1.36%
Overall	8.15%	8.15%	8.15%	8.04%	8.04%	8.04%	3.12%	3.12%	3.12%

tra two parsing formats enable the identification of partial results generated by the LLMs. Additionally, the precision scores for all three models are significantly higher than their recall scores. This indicates that the entities identified by the LLMs and correctly parsed are more likely to align with the pre-defined types. However, the low recall scores suggest that the LLMs frequently misclassify non-relevant components as belonging to the target types. This observation aligns with the conclusions drawn from the main results. To further investigate, we manually examined a subset of the LLM outputs. Without few-shot prompting, we observed instances where the LLMs either repeated the entire instruction or generated outputs in the format of "entity_type—entity_type". Additionally, despite incorporating and summarizing two additional parsing formats, the performance of few-shot prompting remains suboptimal. Moreover, integrating extra parsing formats introduces additional manual analysis efforts. The consistency and ease of parsing output formats are critical when leveraging LLMs for downstream tasks.

Answer to RQ2: Ablation analysis demonstrates that prompt engineering techniques used in our prompts improve LLM performance to different extents. Notably, few-shot prompting proved to be the most effective in helping LLMs understand task requirements and output formats.

5.4.5 RQ3: Hallucinations Analysis in LLMPP

This research question investigates the phenomenon of hallucinations in LLMPP. Hallucinations are primarily discussed in the context of chatbot applications, where they refer to instances where LLMs generate content that appears plausible but is factually incorrect or unfounded. In LLMPP, we define and identify hallucinations from two perspectives: entity hallucination (H_e) and entity type hallucination (H_t). Under the condition that the LLM generates results strictly adhering to the required format and the results are correctly parsed into entities and entity types, H_e measures the extent to which the entities are not derived from the given privacy policy sentences but are instead randomly inferred by the LLM. H_e is formulated as:

$$H_e = \frac{I_e}{G_e}, \tag{5.1}$$

where I_e represents the number of entities generated by the LLM that do not appear in the given privacy policy sentences, and G_e represents the number of entities correctly parsed from the LLM's outputs. Similarly, entity-type hallucination (H_t) measures the proportion of entity types generated by the LLM that do not belong to the predefined set of entity types. H_t is calculated using the following formula:

$$H_t = \frac{I_t}{G_t}, \tag{5.2}$$

where I_t represents the number of entity types generated by the LLM that are not within the given set of predefined entity types, and G_t represents the total number of entity types correctly parsed from the LLM's outputs.

Table 5.4: Hallucination analysis in LLPP

	Main Prompt		ARP		TDP	
	H_e	H_t	H_e	H_t	H_e	H_t
GPT	3.56%	0.86%	3.24%	1.55%	4.12%	0.33%
Llama	5.96%	17.65%	6.09%	20.03%	5.68%	5.37%
Qwen	30.75%	21.79%	18.41%	33.69%	40.61%	48.86%
	CAP		ITP		FSP	
	H_e	H_t	H_e	H_t	H_e	H_t
GPT	3.51%	3.74%	7.13%	1.81%	65.00%	60.97%
Llama	10.73%	15.90%	7.96%	16.96%	25.37%	81.27%
Qwen	42.97%	39.03%	40.20%	23.72%	16.15%	54.23%

Table 5.4 presents the analysis of hallucination in LLMPP, where the "main prompt" refers to the use of the full prompt, and other categories represent ablation settings that exclude specific prompting techniques. GPT introduces the least hallucination in both entities and entity types compared to the other two models, while Qwen exhibits the highest level of hallucination in its generated content. Notably, GPT demonstrates strong task understanding capabilities, as evidenced by its entity type hallucination rate of only 0.86% under the main prompt setting. Table **??** demonstrates that Assigning a Role Prompting (ARP) and Few-Shot Prompting (FSP) significantly reduce hallucination in both entities and entity types across all three models. Specifically, without ARP and FSP, the H_e and H_t metrics increase for all models. This improvement can likely be attributed to two factors: ARP guides the LLMs to focus on relevant knowledge within their training corpora, and FSP provides correct examples that help the models learn and generalize more effectively. Task Description Prompting (TDP) only slightly reduces entity hallucination for GPT and Qwen but increases entity type hallucination. This may occur because the LLMs can still learn the task effectively through Context-Aware Prompting (CAP) and Few-Shot Learning (FSL), making additional descriptions redundant. Furthermore, excessively long prompts re-

sulting from detailed descriptions may confuse the LLMs, potentially degrading their performance. Context-Aware Prompting (CAP) effectively reduces hallucination in both entity and entity type recognition for GPT and Qwen, while also alleviating entity hallucination for Llama. This demonstrates that providing clear descriptions of each component helps LLMs better understand the scope and boundaries of the entities, leading to improved accuracy and reduced errors. Instruction-Tuned Prompting (ITP) is primarily designed to restrict the output format of LLMs, making it easier for users to parse the results. As a result, ITP demonstrates only a limited ability to alleviate hallucination across all three models.

Answer to RQ3: Experimental results indicate that models with stronger foundational capabilities exhibit fewer hallucinations in the LLMPP scenario. Among the techniques evaluated, Few-Shot Learning (FSL) proves to be the most effective in mitigating hallucinations. Additionally, prompt engineering strategies such as clear task descriptions and well-defined instructions also contribute to reducing hallucinations, albeit to a lesser extent.

5.4.6 Case Study

This section presents a case study on our prompt design and false recognition in the main results. Specifically, for the prompt design case study, we begin with an initial prompt that mimics traditional Named Entity Recognition (NER) tasks to label each character in given sentences. This allows us to evaluate the effectiveness of our approach in the early stages. For the false case analysis, we conduct a case study to investigate the reasons behind incorrect component identification, providing insights into potential improvements.

Prompt design. When designing the LLMPP, the intuitive approach is to mimic traditional Named Entity Recognition (NER) tasks and ask LLMs to identify the label of each character in privacy policy sentences, as introduced in §5.3.1. To achieve this,

we evaluate the prompt as is given in Figure 5.6. We use the prompt to query LLMs, asking LLMs to label each character in the provided sentence in order to identify privacy components within those sentences.

You are an expert in mobile app privacy policy analyzer and have over ten years experience in analyzing the compliance in privacy policies.

You will be given one sentence from a Chinese privacy policy (PP). You should analyze the sentence to identify the key components in the privacy policy. Specifically, you need to identify the data controller, data entities, data behavior (collect or share), condition, purpose and data receiver. The definition of the seven components are
as follows:

1. Data controller: (noun) the party that determines the purpose and means of personal data processing.
2. Data entities: (nouns) any information that can identify or reflect the activities of a natural person.
3. Collection: verbs that describe how controllers access data.
4. Sharing: verbs that describe whether how the controller distribute the entities to others.
5. Condition: the situation where the data controller will access personal data.
6. Purpose: why the data controller processes user data.
7. Data receiver: the parties that receive user data.
0. Others: that are not related to aforementioned components.

Your need to output the label of each character in the provided sentence. The length of output label must be strictly the same with the character length of provide sentence. For example:
Example 1:
"sentence": "您在注册启信宝时，我们会收集、使用、保存、共享您的相关个人信息.".
"label": "0 5 5 5 5 5 5 0 0 1 1 0 3 3 0 3 3 0 3 3 0 4 4 0 0 0 0 2 2 2 2 0"
Example 1:
"sentence": "我们可能会为了推荐适合您的产品与供应商共享您的兴趣爱好。"
"label": "1 1 0 0 6 6 6 6 6 6 6 6 6 6 6 0 7 7 7 5 5 0 0 2 2 2 2 0""

Now, it is your turn:
"sentence": "{sentence}"
"label": "

Figure 5.6: Initial prompt for LLMPP

Figure 5.7(a) and 5.7(b) give two examples of using the prompt in Figure 5.6 to query LLMs for analyzing privacy components in privacy policy sentences. The index shows the position of each character in the sentence. "GroundTruth" is the label for each character. A "/" means the LLM did not generate the required output. "+num" indicates extra characters generated by the LLM but not shown in the figure. The figures clearly show that all three models almost entirely generate incorrect labels for each character, often producing more labels than the number of characters in the sentence. Additionally, we provide more cases in our GitHub repository to support

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Setence	您	还	可	以	根	据	自	身	需	求	选	择	填		性	别	、	生	日	、
GroundTruth	1	0	0	0	0	0	0	0	0	0	0	0	4	4	2	2	0	2	2	0
GPT	0	0	0	0	0	0	5	5	5	0	0	0	0	0	0	0	6	6	6	6
Llama											/									
Qwen	of	5	5	5	5	5	5		a	5	5	5	5	5	5	5	5	5	5	5
Index	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	···
Setence	地	区	及	个	人	介	绍	来	完	善	您	的	信	息	。					
GroundTruth	2	2	0	2	2	2	2	0	6	6	6	6	6	6	0					
GPT	6	6	7	7	7	7	7	7	3	3	3	3	0	6	+5					
Llama											/									
Qwen		5	5	5	5	5	5	5	5	5	5	5	5	5	+77					

(a) Example 1.

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Setence	未	经	您	许	可	,	梦	映	不	会	向	第	三	方	公	开	、	透	露	您
GroundTruth	0	5	5	5	5	0	1	1	0	0	0	7	7	7	4	4	0	4	4	0
GPT	0	0	0	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	2	2
Llama	0	0	5	0	5	5	0	5	5	5	5	5	0	5	5	5	0	0	0	1
Qwen											/									
Index	21	22	23	24	25	26	0													
Setence	的	个	人	信	息	。														
GroundTruth	0	2	2	2	2	0														
GPT	2	3	3	3	3	3	+38													
Llama	1	0	0	3	3	0	+52													
Qwen				/																

(b) Example 2.

Figure 5.7: Results of prompt case study.

the case study of the prompt design. Our analysis reveals that nearly all cases exhibit this same problematic behavior. This phenomenon is consistent with existing research showing LLMs' poor performance in number-counting tasks, as discussed in §5.3.1. This issue may stem from the tokenization methodologies [109] employed by LLMs. The embedding methodologies often group multiple characters into a single token, which can interfere with the model's ability to accurately count individual characters within a sentence. This grouping can lead to inaccuracies in tasks that require precise character-level processing.

False recognition analysis. Figure 5.8 provides an illustrative example of the inference results generated by LLMs for a privacy policy sentence, and additional cases are available in our GitHub repository. In the ground truth, we observe one condition component that specifies the situation in which the owner of the privacy policy will conduct data access and one purpose statement that explains why the data access will

be performed. GPT partially correctly identifies the condition in the sentence but incorrectly labels an unrelated phrase as a condition. Additionally, GPT fails to identify the purpose component in the sentence. Llama correctly identifies two subsets of the condition components but incorrectly labels four unrelated phrases as conditions. Additionally, Llama fails to identify the purpose component in the sentence. Qwen correctly identifies the purpose in the sentence and a subset of the condition components. For the data controller, GPT correctly identifies the data controller, while the other two LLMs (Llama and Qwen) fail to do so. Qwen even incorrectly identifies the data controller as a data entity. For data entities, both GPT and Llama correctly identify all data entities, although GPT mistakenly labels an additional definition of data as a data entity. Qwen only correctly identifies one data entity but misses all other data entities. In the context of data operation behavior, since the subject performing the action is the user ("您"), the verb describing this action should be labeled as a sharing behavior. This distinction is crucial for accurately categorizing and understanding the nature of data operations within privacy policies. GPT fails to identify the behavior component. Llama incorrectly labels the action phrase as a condition, and Qwen wrongly identifies the action behavior as a collection. Additionally, the results reveal that LLMs tend to incorrectly classify unrelated phrases into given categories, leading to low recall metrics in the main results (Table 5.1). These unrelated phrases are statistically labeled as "O" during the metrics computation process.

Based on our case study of LLMs' inference results, we observe that several types of errors can be attributed to the LLMs' inability to correctly parse the sentence structure. For instance, LLMs often misidentify data action verbs. Additionally, some errors stem from the LLMs' incomplete understanding of the provided context, which adversely impacts their ability to accurately identify key phrases, such as the purpose and conditions within the sentence.

Sentence	在您使用身份认证的功能或相关服务所需时，根据相关法律法规，您可能需要提供您的真实身份信息（真实姓名、身份证号码、电话号码）以完成实名验证。该等信息属于个人敏感信息，您可以拒绝提供，如果拒绝提供您将可能无法获得相关服务，但不影响其他功能与服务的正常使用
Ground Truth	【在您使用身份认证的功能或相关服务所需时，根据相关法律法规|**condition**】，【您|controller】，【提供|sharing】，【真实身份信息（真实姓名、身份证号码、电话号码）|data】，【以完成实名验证|**purpose**】
GPT	【身份认证的功能或相关服务所需时|**condition**】，【您|controller】，【提供|collection】，【真实身份信息|data】，【真实姓名|data】，【身份证号码|data】，【电话号码|data】，【个人敏感信息|data】，【拒绝提供|condition】，【相关服务|**purpose**】，【无法获得相关服务|**condition**】，【其他功能与服务的正常使用|**purpose**】
Llama	【您|data_entities】，【真实身份信息|data_entities】，【真实姓名|data_entities】，【身份证号码|data_entities】，【电话号码|data_entities】，【身份认证的功能|**condition**】，【相关服务|**condition**】，【相关法律法规|condition】，【我们|controller】，【您|data_entities】，【拒绝提供|**condition**】，【相关服务|**condition**】，【其他功能与服务|**condition**】，【我们|controller】，【可能|**condition**】，【无法获得|**condition**】，【启信宝|controller】
Qwen	【启信宝|controller】，【真实身份信息|data】，【提供|collection】，【根据相关法律法规|**condition**】，【完成实名验证|**purpose**】，【无法获得相关服务|receiver

Figure 5.8: Case study of inference results in LLMPP.

5.5 Discussion

In this section, we discuss the limitations of this work and propose directions for future research on applying LLMs to privacy policy analysis.

In this work, we empirically evaluate the performance of popular pre-trained LLMs in identifying regulation-mandated entities within Chinese privacy policies. By referencing existing work [139, 185] and employing established prompt engineering techniques [46, 102], this work focuses on evaluating effective prompts tailored for handling privacy policies in general scenarios, rather than seeking to identify the optimal prompts for all LLMs.

We evaluate two publicly available LLMs and one widely used commercial LLM. In future work, we will evaluate a wider range of LLMs, particularly those specifically trained to process Chinese content, to provide deeper insights into the performance of LLMs in this challenging task. Additionally, to ensure fairness, we evaluate all LLMs using a unique and consistent prompt format. Existing research has shown that different prompt templates [59], such as XML format, Markdown format, or plain text, may influence the performance of LLMs. Future work will explore methods to quan-

tify the impact of prompt formats and expand the evaluation to include more LLMs. Additionally, our ablation study and analysis of hallucination in LLMs have demonstrated that different prompt engineering techniques can improve the performance of LLMs and reduce hallucinations in LLMPP. One promising direction to enhance the performance of LLMs and mitigate hallucinations remains improving their fundamental capabilities. Given the inherent randomness in the inference process of LLMs, which may impact the reproducibility of this work, we provide all prompts and results generated by the LLMs to facilitate a deeper understanding of our findings and ensure that other researchers can reproduce and build upon our results.

In future work, one potential direction could involve exploring the application of long-context models to implement end-to-end privacy policy analysis tasks. Since privacy policies are typically composed of lengthy, detailed descriptions, long-context models may be better suited for processing such content. Long-context models may be capable of effectively summarizing key points and extracting downstream requirements from the context. While labeling privacy policies across different languages and regions is labor-intensive, it is feasible to pre-train a large language model specifically designed to analyze privacy policies from diverse platforms. Such a model can help safeguard user privacy by accurately identifying key components like data-handling practices and consent requirements. It can also mitigate the risk of policy abuse by providing clearer insights into how entities draft and enforce these policies. This approach not only enhances privacy protection but also promotes greater accountability and transparency in privacy policy management.

Chapter 6

Conclusions and Suggestions for Future Research

6.1 Conclusions

The main objective of this thesis is to address the risks posed by potentially harmful Android apps that could compromise user privacy once published in app markets. In this thesis, we tackle several technical challenges by developing innovative approaches and constructing a benchmark dataset to serve three main objectives. As an initial step, this thesis investigates the vulnerability of Android malware detection systems by proposing a novel problem-space structural attack against existing malware detection systems. Furthermore, the first work explores potential defense strategies to address this issue. Secondly, we provide an Android app privacy policy dataset to promote research in the field of Android app privacy policies. In addition, we evaluate the performance of LLMs for analyzing Chinese privacy policies. The target of our work is to enhance the security and privacy of Android app users by offering useful insights and resources for developing more effective detection, prevention, and mitigation strategies.

6.1.1 Investigating Vulnerability of Android Malware Detection

To investigate the vulnerability of Android malware detection systems and develop effective defense methodologies, we propose HRAT, a novel structural attack against function call graph-based Android malware detection systems. By conducting this attack, we aim to uncover weaknesses in the systems and explore potential countermeasures to mitigate the risks associated with Android malware. HRAT is unique in that it leverages the correlation between function call graphs and software to bridge the gap between feature-space attacks and problem-space attacks. By leveraging the capabilities of HRAT, we are able to effectively exploit vulnerabilities in Android malware detection systems that would not be accessible through traditional feature-based attacks. This enables us to identify new avenues for improving the security of these systems, ultimately enhancing their ability to detect and prevent potentially harmful apps from compromising user privacy and security. Compared to heuristic methods, our attack proves to be more effective and efficient in terms of modifying and interacting with target systems. By using our proposed structural attack, we are able to generate subtle but significant modifications to Android malware samples that evade detection by state-of-the-art detection systems. Our attack highlights the value of HRAT in testing the robustness of Android malware detection systems and identifying areas for improvement. Our experiments show that combining multiple attack actions is significantly more effective than using a single action alone. By integrating several attack techniques, we are able to achieve higher success rates in evading detection by Android malware detection systems. This highlights the importance of using a multi-pronged approach to testing the security of these systems and developing more robust defenses against Android malware. Notably, our methodology is not limited to the Android platform and can be adapted to other systems as well. Our approach, which leverages the structural properties of software and identifies vulnerabilities in malware detection systems, has the potential to improve the

security of a wide range of software systems, demonstrating the broad applicability and versatility of our proposed methodology.

6.1.2 Introducing a Comprehensive Android Application Privacy Policy Dataset

In order to facilitate research into analyzing Android app privacy policy issues, we present the CA4P-483 dataset, the first comprehensive dataset of Chinese Android application privacy policies. Our dataset was constructed using a rigorous data collection and corpus annotation process, ensuring that it is of high quality and can be used as a reliable benchmark for research purposes. The CA4P-483 dataset includes fine-grained annotations that align with the requirements of privacy-related laws and regulations, ensuring that it provides a comprehensive and accurate representation of the privacy policies of Chinese Android applications. These annotations enable researchers to easily identify and analyze the specific privacy practices of individual applications, facilitating a deeper understanding of the privacy landscape of the Android app ecosystem. By providing a large-scale dataset of Chinese Android application privacy policies with fine-grained annotations, the CA4P-483 dataset has the potential to advance natural language processing research on practical downstream tasks. Our dataset can be used to develop and evaluate machine learning models for a range of tasks, such as automatic policy analysis and summarization, as well as to explore the relationship between privacy policy content and actual data collection practices. This makes the dataset a valuable resource for researchers working on improving the privacy and security of mobile applications. In addition, we perform experimental evaluations of several popular baselines on the CA4P-483 dataset and present the results of our analysis. Our evaluation provides a benchmark for future research on privacy policy analysis and highlights the strengths and weaknesses of current state-of-the-art approaches. Based on our findings, we propose potential research

directions for improving the accuracy and effectiveness of privacy policy analysis, with the goal of enhancing user privacy and security in the Android app ecosystem. In addition to our experimental evaluations, we perform several case studies to explore the potential applications of our dataset in software engineering and cybersecurity. Our analysis demonstrates the utility of the CA4P-483 dataset for a range of downstream tasks, including developing tools for automated privacy policy analysis and generating user-friendly summaries of privacy practices. These applications have the potential to enhance the privacy and security of mobile applications, improving the user experience and fostering greater trust in the app ecosystem.

6.1.3 Application of LLMs for Analyzing Privacy Policies

With the growing popularity of LLMs, many downstream tasks, particularly those related to natural language processing, have achieved significant success with the assistance of LLMs. We empirically evaluate the performance of popular LLMs in identifying regulation-required entities and their types in Chinese privacy policies. Our ablation study of prompt engineering techniques and analysis of hallucinations in LLMPP both demonstrate that these techniques can improve the performance of LLMs and alleviate hallucinations in LLMPP. Our evaluation highlights the challenges of applying LLMs to Chinese privacy policies. Specifically, pre-trained LLMs often fail to recognize defined entity types within given sentences, even when advanced prompt engineering techniques and well-crafted prompts are employed.

6.2 Future Work

Having completed three major works related to the analysis of potentially harmful Android apps and the protection of user privacy, there are several promising directions for our future research.

First, we can develop more effective and efficient attack strategies against Android malware detection systems. Currently, HRAT is only applicable under white-box settings, meaning that it requires access to the machine learning models used in target detection systems. In recent years, machine learning researchers have demonstrated the feasibility of training a student model to mimic a target model using only the target model's output, without access to the model itself. Thus, in cases where it is not possible to access the machine learning models, we can train a student model to mimic the target model and use HRAT to deceive the student model into achieving the same level of detection evasion as the target model.

Secondly, as CA4P-483 opens up avenues for research in natural language processing, privacy protection, and cybersecurity, we plan to explore more potential application scenarios based on our dataset. For example, we can conduct an emotional analysis of privacy policies based on CA4P-483. Previous studies [6] have found that privacy policies can present conflicts when used in different contexts. Several existing methods [6, 7, 165] use negative language to detect potential conflicts in privacy policies but do not account for complications such as double negatives. In the Chinese privacy policy, negative representations are more complicated [85]. Thus, emotional analysis can help analysts better understand the semantics of privacy policies.

Finally, a promising direction for future research is the development of a privacy policy-specific large language model capable of processing privacy policies written in various languages and sourced from diverse platforms, including mobile apps, web applications, and Internet of Things (IoT) devices. This endeavor would begin with the construction of a high-quality dataset tailored to the complexities of privacy policies. Additionally, given that existing research highlights the significant impact of tokenization methods on LLM performance, designing an effective tokenization strategy would be a critical step. Such an approach could enhance the model's ability to handle the diverse linguistic and contextual nuances of privacy policies, thereby advancing the field of privacy policy analysis.

References

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. *arXiv preprint arXiv:2005.00653*, 2020.

[2] DeepSeek AI. Deepseek: Advanced ai capabilities. https://www.deepseek.com/en, 2024. Accessed: 2024-06-18.

[3] Alir3z4. html2text. https://github.com/Alir3z4/html2text, 2011.

[4] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[5] Mohammed K Alzaylaee, Suleiman Y Yerima, and Sakir Sezer. Dl-droid: Deep learning based android malware detection using real devices. *Computers & Security*, 89:101663, 2020.

[6] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. Policylint: investigating internal privacy policy contradictions on google play. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 585–602, 2019.

[7] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words: Entity-Sensitive privacy policy and data flow analysis with PoliCheck. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 985–1002. USENIX Association, August 2020.

[8] Anthropic. Claude: An ai assistant. https://claude.ai/, 2024. Accessed: 2024-06-18.

[9] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, volume 14, pages 23–26, 2014.

[10] Steven Arzt. Static data flow analysis for android applications. 2017.

[11] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[12] KWY. Au, YF. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228, 2012.

[13] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.

[14] Susanne Barth, Dan Ionita, and Pieter Hartel. Understanding online privacy—a systematic review of privacy visualizations and privacy by design guidelines. *ACM Computing Surveys (CSUR)*, 55(3):1–37, 2022.

[15] Harel Berger, Chen Hajaj, and Amit Dvir. When the guard failed the droid: A case study of android malware. *arXiv preprint arXiv:2003.14123*, 2020.

[16] Tom Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

[17] Liang Cai and Hao Chen. On the practicality of motion based keystroke inference attack. In *International Conference on Trust and Trustworthy Computing*, pages 273–290. Springer, 2012.

[18] Minghui Cai, Yuan Jiang, Cuiying Gao, Heng Li, and Wei Yuan. Learning features from enhanced function call graphs for android malware detection. *Neurocomputing*, 423:301–307, 2021.

[19] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[20] Juan Miguel Cejuela, Peter McQuilton, Laura Ponting, Steven J Marygold, Raymund Stefancsik, Gillian H Millburn, Burkhard Rost, FlyBase Consortium, et al. tagtog: interactive and text-mining-assisted annotation of gene mentions in plos full-text articles. *Database*, 2014, 2014.

[21] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, et al. Legal-bert: The muppets straight out of law school. In *Findings of the Association for Computational Linguistics*, 2020.

[22] Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. Booookscore: A systematic exploration of book-length summarization in the era of llms. In *The Twelfth International Conference on Learning Representations*, 2023.

[23] Chaoran Chen, Daodao Zhou, Yanfang Ye, Toby Jia-jun Li, and Yaxing Yao. Clear: Towards contextual llm-empowered privacy policy analysis and risk generation for large language model applications. *arXiv preprint arXiv:2410.13387*, 2024.

[24] Kai Chen, Peng Liu, and Yingjun Zhang. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *Proceedings of the 36th International Conference on Software Engineering*, pages 175–186, 2014.

[25] Kai Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Nan Zhang, Heqing Huang, Wei Zou, and Peng Liu. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 659–674, 2015.

[26] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. Android hiv: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, 2019.

[27] CLPRC. Cybersecurity law of the people's republic of china. http://www.gov.cn/xinwen/2016-11/07/content_5129723.htm, 2016.

[28] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[29] National Information Security Standardization Technical Committee. Information security technology — basic specification for collecting personal informa-

tion in mobile internet applications. http://www.cac.gov.cn/1124853418_15652571749671n.pdf, 2022.

[30] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[31] Zhenjin Dai, Xutao Wang, Pin Ni, Yuming Li, Gangmin Li, and Xuming Bai. Named entity recognition using bert bilstm crf for chinese electronic health records. In *2019 12th international congress on image and signal processing, biomedical engineering and informatics (cisp-bmei)*, pages 1–5. IEEE, 2019.

[32] Google DeepMind. Gemini: A multimodal ai. https://gemini.google.com/, 2024. Accessed: 2024-06-18.

[33] Anthony Desnos and Patrik Lantz. Droidbox: An android application sandbox for dynamic analysis. https://code.google.com/archive/p/droidbox.

[34] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019.

[36] Keyang Ding, Jing Li, and Yuji Zhang. Hashtags, emotions, and comments: a large-scale dataset to understand fine-grained social emotions to online topics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1376–1382, 2020.

[37] Shuaike Dong, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, Xiaofeng Wang, and Kehuan Zhang. Understanding

android obfuscation techniques: A large-scale investigation in the wild. In *Security and Privacy in Communication Networks: 14th International Conference, SecureComm 2018, Singapore, Singapore, August 8-10, 2018, Proceedings, Part I*, pages 172–192. Springer, 2018.

[38] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):1–42, 2008.

[39] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

[40] William Enck, Damien Octeau, Patrick D McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, 2011.

[41] String Encryption. Dexguard. https://www.guardsquare.com, 2017.

[42] Ming Fan, Xiapu Luo, Jun Liu, Meng Wang, Chunyin Nong, Qinghua Zheng, and Ting Liu. Graph embedding based familial analysis of android malware using unsupervised learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 771–782. IEEE, 2019.

[43] LC. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

[44] Dayne Freitag and Andrew McCallum. Information extraction with hmm structures learned by stochastic optimization. *AAAI/IAAI*, 2000:584–589, 2000.

[45] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[46] Andrew Gao. Prompt engineering for large language models. *Available at SSRN 4504303*, 2023.

[47] GDPR. General data protection regulation. https://gdpr-info.eu, 2016.

[48] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[49] Google. Google play. https://play.google.com, 2022.

[50] Google. Google play policies. https://developer.android.com/distribute/play-policies, 2022.

[51] Google-Monkey. Google Monkey. https://developer.android.com/studio/test/monkey, 2021.

[52] New Zealand Government. International covenant on civil and political rights, 2020. https://www.justice.govt.nz/justice-sector-policy/constitutional-issues-and-human-rights/human-rights/international-human-rights/international-covenant-on-civil-and-political-r

[53] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 101–112, 2012.

[54] Michael C Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Systematic detection of capability leaks in stock android smartphones. In *NDSS*, volume 14, page 19, 2012.

[55] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[56] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*, pages 62–79. Springer, 2017.

[57] Yu Gu, Robert Tinn, Hao Cheng, et al. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare*, 2021.

[58] Hamza Harkous, Kassem Fawaz, Rémi Lebret, Florian Schaub, Kang G Shin, and Karl Aberer. Polisis: Automated analysis and presentation of privacy policies using deep learning. In *27th USENIX Security Symposium*, 2018.

[59] Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. Does prompt formatting have any impact on llm performance?, 2024.

[60] Shifu Hou, Yujie Fan, Yiming Zhang, Yanfang Ye, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, and Fudong Shao. αcyber: Enhancing robustness of android malware detection system against adversarial attacks on heterogeneous graph based model. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 609–618, 2019.

[61] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1507–1515, 2017.

[62] Huawei. App gallery. https://appgallery.huawei.com/Featured, 2022.

[63] Huawei. Appgallery review guidelines. https://developer.huawei.com/consumer/en/doc/30202, 2022.

[64] Jesusfreke. smali. https://code.google.com/p/smali/.

[65] Teenu S John, Tony Thomas, and Sabu Emmanuel. Graph convolutional networks for android malware detection with system call graphs. In *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, pages 162–170. IEEE, 2020.

[66] Mohammad Karami, Mohamed Elsabagh, Parnian Najafiborazjani, and Angelos Stavrou. Behavioral analysis of android applications using automated instrumentation. In *2013 IEEE Seventh International Conference on Software Security and Reliability Companion*, pages 182–187. IEEE, 2013.

[67] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[68] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. A multimodal deep learning method for android malware detection using various features. *IEEE Trans. Inf. Forensics Secur.*, 14(3):773–788, 2018.

[69] Ron Kohavi and George H John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

[70] Taku Kudo. Crf++: Yet another crf toolkit. *http://crfpp. sourceforge. net/*, 2005.

[71] S Kulkarni. 11. future of technology chatgpt: Optimizing language models for dialogue.". *ICP Monogram on Digital Technology in Clinical Medicine*, page 62, 2023.

[72] PJM. Van Laarhoven and EHL. Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[73] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[74] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of NAACL-HLT*, pages 260–270, 2016.

[75] Alexander LeClair, Sakib Haque, Lingfei Wu, and Collin McMillan. Improved code summarization via a graph neural network. In *Proceedings of the 28th international conference on program comprehension*, pages 184–195, 2020.

[76] E. Levin, R. Pieraccini, and W. Eckert. Using markov decision process for learning dialogue strategies. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, volume 1, pages 201–204. IEEE, 1998.

[77] Deqiang Li and Qianmu Li. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. *IEEE Transactions on Information Forensics and Security*, 15:3886–3900, 2020.

[78] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. Arms race in adversarial malware detection: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–35, 2021.

[79] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-An, and Heng Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Trans Industr Inform*, 14(7):3216–3225, 2018.

[80] Qinbin Li, Junyuan Hong, Chulin Xie, Jeffrey Tan, Rachel Xin, Junyi Hou, Xavier Yin, Zhun Wang, Dan Hendrycks, Zhangyang Wang, Bo Li, Bingsheng

He, and Dawn Song. Llm-pbe: Assessing data privacy in large language models. *Proceedings of the VLDB Endowment*, 17(11):3201–3214, 2024.

[81] Zongjie Li, Chaozheng Wang, Zhibo Liu, Haoxuan Wang, Dong Chen, Shuai Wang, and Cuiyun Gao. Cctest: Testing and repairing code completion systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1238–1250. IEEE, 2023.

[82] Sue Lim and Ralf Schmälzle. Artificial intelligence for health message generation: an empirical study using a large language model (llm) and prompt engineering. *Frontiers in Communication*, 8:1129082, 2023.

[83] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[84] Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. Data-efficient fine-tuning for llm-based recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 365–374, 2024.

[85] Bing Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[86] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971*, 2024.

[87] Michael Xieyang Liu, Frederick Liu, Alexander J Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J Cai. " we need structured output": Towards user-centered constraints on large language model output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–9, 2024.

[88] Di Lu, Leonardo Neves, Vitor Carvalho, Ning Zhang, and Heng Ji. Visual attention model for name tagging in multimodal social media. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1990–1999, 2018.

[89] Yao Ma, Suhang Wang, Tyler Derr, Lingfei Wu, and Jiliang Tang. Graph adversarial attack via rewiring. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1161–1169, 2021.

[90] Bhavani Malisetty and Alfredo J Perez. Evaluating quantized llama 2 models for iot privacy policy language generation. *Future Internet*, 16(7):224, 2024.

[91] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2017.

[92] Aleecia M McDonald and Lorrie Faith Cranor. The cost of reading privacy policies. *Isjlp*, 4:543, 2008.

[93] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[94] Sudha Morwal, Nusrat Jahan, and Deepti Chopra. Named entity recognition using hidden markov model (hmm). *International Journal on Natural Language Computing (IJNLC) Vol*, 1, 2012.

[95] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[96] Preksha Nema, Pauline Anthonysamy, Nina Taft, and Sai Teja Peddinti. Analyzing user perspectives on mobile app privacy at scale. In *International Conference on Software Engineering (ICSE)*, 2022.

[97] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share first, ask later (or never?) studying violations of gdpr's explicit consent in android apps. In *30th USENIX Security Symposium*, 2021.

[98] NISSTC. Cybersecurity practices guidelines – security guidelines for using software development kit (sdk) for mobile internet applications (app) (tc260-pg-20205a). https://www.tc260.org.cn/front/postDetail.html?id=20201126161240, 2020.

[99] Debora Nozza, Federico Bianchi, Anne Lauscher, Dirk Hovy, et al. Measuring harmful sentence completion in language models for lgbtqia+ individuals. In *Proceedings of the Second Workshop on Language Technology for Equality, Diversity and Inclusion*. Association for Computational Linguistics, 2022.

[100] Cyberspace Administration of China, Ministry of Industry, Information Technology, Ministry of Public Security, and State Administration for Market. Measures for determining the illegal collection and use of personal information by apps. http://m.legaldaily.com.cn/zt/content/2021-11/16/content_8628724.htm, 2019.

[101] California Attorney General office. California privacy rights act. https://www.weil.com/-/media/the-california-privacy-rights-act-of-2020-may-2021.pdf, 2020.

[102] OpenAI. Chatgpt: A large language model. https://www.openai.com/chatgpt, 2024. Accessed: 2024-06-18.

[103] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray,

et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[104] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. Llm is like a box of chocolates: the non-determinism of chatgpt in code generation. *arXiv preprint arXiv:2308.02828*, 2023.

[105] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[106] Nanyun Peng and Mark Dredze. Improving named entity recognition for chinese social media with word segmentation representation learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 149–155, 2016.

[107] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1332–1349. IEEE, 2020.

[108] PISS. Information security technology – personal information security specification. https://www.tc260.org.cn/front/postDetail.html?id=20200918200432, 2020.

[109] Haohao Qu, Wenqi Fan, Zihuai Zhao, and Qing Li. Tokenrec: Learning to tokenize id for llm-based generative recommendation, 2024.

[110] Erwin Quiring, Alwin Maier, Konrad Rieck, et al. Misleading authorship attribution of source code using adversarial learning. In *USENIX Security Symposium*, pages 479–496, 2019.

[111] 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. *Journal of machine learning research*, 21(140):1–67, 2020.

[112] Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh. Appinsight: Mobile app performance monitoring in the wild. In *Presented as part of the 10th {USENIX} symposium on operating systems design and implementation ({OSDI} 12)*, pages 107–120, 2012.

[113] Federal Register. Code of federal regulations. https://www.ecfr.gov/reader-aids/using-ecfr/getting-started, 2017.

[114] Renaming. Proguard. https://www.preemptive.com/dotfuscator/pro/userguide/en/protection_obfuscation_renaming.html, 2017.

[115] Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pages 1–7, 2021.

[116] David Rodriguez, Ian Yang, Jose M Del Alamo, and Norman Sadeh. Large language models: a new approach for privacy policy analysis at scale. *Computing*, 106(12):3879–3903, 2024.

[117] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21*, pages 490–510. Springer, 2018.

[118] Timo Schick and Hinrich Schütze. Exploiting cloze questions for few-shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2021.

[119] M. Shahpasand, L. Hamey, D. Vatsalan, and M Xue. Adversarial attacks on mobile malware detection. In *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*, pages 17–20. IEEE, 2019.

[120] Luman Shi, Jiang Ming, Jianming Fu, Guojun Peng, Dongpeng Xu, Kun Gao, and Xuanchen Pan. Vahunt: Warding off new repackaged android malware in app-virtualization's clothing. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 535–549, 2020.

[121] Taylor Shin, Yasaman Razeghi, Robert Logan IV, et al. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *Advances in Neural Information Processing Systems*, 2020.

[122] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[123] Gulshan Shrivastava and Prabhat Kumar. Android application behavioural analysis for data leakage. *Expert Systems*, 38(1):e12468, 2021.

[124] Aaditya K Singh and DJ Strouse. Tokenization counts: the impact of tokenization on arithmetic in frontier llms. *arXiv preprint arXiv:2402.14903*, 2024.

[125] Chawin Sitawarin and David Wagner. On the robustness of deep k-nearest neighbors. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2019.

[126] Nir Sivan, Ron Bitton, and Asaf Shabtai. Analysis of location data leakage in the internet traffic of android-based mobile devices. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, pages 243–260, 2019.

[127] David B Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning Proceedings 1994*, pages 293–301. Elsevier, 1994.

[128] FTC Staff. Protecting consumer privacy in an era of rapid change–a proposed framework for businesses and policymakers. *Journal of Privacy and Confidentiality*, 3(1), 2011.

[129] statcounter. Mobile operating system market share worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide, 2022.

[130] statista. Development of new android malware worldwide from june 2016 to march 2020. https://www.statista.com/statistics/680705/global-android-malware-volume/, 2021.

[131] statista. Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/, 2022.

[132] Statista. Number of mobile app downloads worldwide from 2019 to 2021, by country. https://www.statista.com/statistics/1287159/app-downloads-by-country/, 2022.

[133] Thilo Strauss, Markus Hanselmann, Andrej Junginger, and Holger Ulmer. Ensemble methods as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1709.03423*, 2017.

[134] Dianbo Sui, Zhengkun Tian, Yubo Chen, Kang Liu, and Jun Zhao. A large-scale chinese multimodal ner dataset with speech clues. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2807–2818, 2021.

[135] Mingshen Sun, Tao Wei, and John CS Lui. Taintart: A practical multi-level information-flow tracking system for android runtime. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 331–342, 2016.

[136] Yi Sun, Abel Valente, Sijia Liu, and Dakuo Wang. Preserve, promote, or attack? gnn explanation via topology perturbation. *arXiv preprint arXiv:2103.13944*, 2021.

[137] RS. Sutton and AG. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[138] Kimberly Tam, Aristide Fattori, Salahuddin Khan, and Lorenzo Cavallaro. Copperdroid: Automatic reconstruction of android malware behaviors. In *NDSS Symposium 2015*, pages 1–15, 2015.

[139] Chenhao Tang, Zhengliang Liu, Chong Ma, Zihao Wu, Yiwei Li, Wei Liu, Dajiang Zhu, Quanzheng Li, Xiang Li, Tianming Liu, and Lei Fan. Policygpt: Automated analysis of privacy policies with large language models. *arXiv e-prints*, Sep 2023.

[140] Kushal Tirumala, Daniel Simig, Armen Aghajanyan, and Ari Morcos. D4: Improving llm pretraining via document de-duplication and diversification. *Advances in Neural Information Processing Systems*, 36:53983–53995, 2023.

[141] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[142] Raja Vallée-Rai, Etienne Gagnon, Laurie Hendren, Patrick Lam, Patrice Pominville, and Vijay Sundaresan. Optimizing java bytecode using the soot framework: Is it feasible? In *Compiler Construction: 9th International Confer-*

ence, CC 2000 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2000 Berlin, Germany, March 25–April 2, 2000 Proceedings 9, pages 18–34. Springer, 2000.

[143] VirusTotal. Virustotal - free online virus, malware and url scanner, 2021.

[144] California voters. California consumer privacy act regulations. https://govt.westlaw.com/calregs, 2016.

[145] Binghui Wang and Neil Zhenqiang Gong. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2023–2040, 2019.

[146] Jice Wang, Yue Xiao, Xueqiang Wang, Yuhong Nan, Luyi Xing, Xiaojing Liao, JinWei Dong, Nicolas Serrano, Haoran Lu, XiaoFeng Wang, and Yuqing Zhang. Understanding malicious cross-library data harvesting on android. In *30th USENIX Security Symposium*, 2021.

[147] Xiaoyun Wang, Minhao Cheng, Joe Eaton, Cho-Jui Hsieh, and Felix Wu. Attack graph convolutional networks by adding fake nodes. *arXiv preprint arXiv:1810.10751*, 2018.

[148] Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[149] Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, et al. Ontonotes release 4.0. *LDC2011T03, Philadelphia, Penn.: Linguistic Data Consortium*, 2011.

[150] Zhiyuan Wen, Jiannong Cao, Ruosong Yang, Shuaiqi Liu, and Jiaxing Shen. Automatically select emotion for response via personality-affected emotion tran-

sition. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 5010–5020, 2021.

[151] Shomir Wilson, Florian Schaub, Aswarth Abhilash Dara, Frederick Liu, Sushain Cherivirala, Pedro Giovanni Leon, Mads Schaarup Andersen, Sebastian Zimmeck, Kanthashree Mysore Sathyendra, N Cameron Russell, et al. The creation and analysis of a website privacy policy corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1330–1340, 2016.

[152] Yueming Wu, XiaoDi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 139–150. IEEE, 2019.

[153] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph backdoor. In *USENIX Security Symposium*, pages 1523–1540, 2021.

[154] Liang Xu, Qianqian Dong, Cong Yu, Yin Tian, Weitang Liu, Lu Li, and Xuanwei Zhang. Cluener2020: Fine-grained name entity recognition for chinese. *arXiv preprint arXiv:2001.04351*, 2020.

[155] Lei Xue, Yuxiao Yan, Luyi Yan, Muhui Jiang, Xiapu Luo, Dinghao Wu, and Yajin Zhou. Parema: an unpacking framework for demystifying vm-based android packers. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 152–164, 2021.

[156] Lei Xue, Hao Zhou, Xiapu Luo, Le Yu, Dinghao Wu, Yajin Zhou, and Xiaobo Ma. Packergrind: An adaptive unpacking system for android apps. *IEEE Transactions on Software Engineering*, 48(2):551–570, 2020.

[157] Lei Xue, Hao Zhou, Xiapu Luo, Yajin Zhou, Yang Shi, Guofei Gu, Fengwei Zhang, and Man Ho Au. Happer: Unpacking android apps via a hardware-

assisted approach. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1641–1658. IEEE, 2021.

[158] Lei Xue, Yajin Zhou, Ting Chen, Xiapu Luo, and Guofei Gu. Malton: Towards on-device non-invasive mobile malware analysis for art. In *USENIX Security Symposium*, pages 289–306, 2017.

[159] Lok Kwong Yan and Heng Yin. {DroidScope}: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis. In *21st USENIX security symposium (USENIX security 12)*, pages 569–584, 2012.

[160] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[161] Yu Yang, Jiannong Cao, Milos Stojmenovic, Senzhang Wang, Yiran Cheng, Chun Lum, and Zhetao Li. Time-capturing dynamic graph embedding for temporal linkage evolution. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[162] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2):82–102, 1999.

[163] Yanfang Ye, Shifu Hou, Lingwei Chen, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, and Fudong Shao. Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection. In *28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[164] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX Annual Technical Conference*, volume 12, pages 1–14, 2012.

[165] Le Yu, Xiapu Luo, Jiachi Chen, Hao Zhou, Tao Zhang, Henry Chang, and Hareton K. N. Leung. Ppchecker: Towards accessing the trustworthiness of android apps' privacy policies. *IEEE Transactions on Software Engineering*, 2018.

[166] Le Yu, Xiapu Luo, Xule Liu, and Tao Zhang. Can we trust the privacy policies of android apps? In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 538–549. IEEE, 2016.

[167] Le Yu, Xiapu Luo, Chenxiong Qian, Shuai Wang, and Hareton KN Leung. Enhancing the description-to-behavior fidelity in android apps with privacy policy. *IEEE Transactions on Software Engineering*, 44(9):834–854, 2017.

[168] Le Yu, Tao Zhang, Xiapu Luo, and Lei Xue. Autoppg: Towards automatic generation of privacy policy for android applications. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015.

[169] Xian Zhan, Lingling Fan, Sen Chen, Feng We, Tianming Liu, Xiapu Luo, and Yang Liu. Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications. In *2021 IEEE/ACM 43rd*

International Conference on Software Engineering (ICSE), pages 1695–1707. IEEE, 2021.

[170] Jingyuan Zhang and Mingjie Chen. People dairy. https://github.com/zjy-ucas/ChineseNER/, 2017.

[171] L. Zhang, P. Liu, and YH. Choi. Semantic-preserving reinforcement learning attack against graph neural networks for malware detection. *arXiv preprint arXiv:2009.05602*, 2020.

[172] Qi Zhang, Jinlan Fu, Xiaoyu Liu, and Xuanjing Huang. Adaptive co-attention network for named entity recognition in tweets. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[173] Suxiang Zhang, Ying Qin, Wen-Juan Hou, and Xiaojie Wang. Word segmentation and named entity recognition for sighan bakeoff3. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, pages 158–161, 2006.

[174] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

[175] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024.

[176] Xiang Zhang, Juntai Cao, and Chenyu You. Counting ability of large language models and impact of tokenization. *arXiv preprint arXiv:2410.19730*, 2024.

[177] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. Enhancing state-of-the-art classifiers with api

semantics to detect evolved android malware. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 757–770, 2020.

[178] Xueling Zhang, Xiaoyin Wang, Rocky Slavin, Travis Breaux, and Jianwei Niu. How does misconfiguration of analytic services compromise mobile privacy? In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1572–1583. IEEE, 2020.

[179] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 611–622, 2013.

[180] Yue Zhang and Jie Yang. Chinese ner using lattice lstm. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1554–1564, 2018.

[181] Yueqian Zhang, Xiapu Luo, and Haoyang Yin. Dexhunter: toward extracting hidden code from packed android applications. In *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II 20*, pages 293–311. Springer, 2015.

[182] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 15–26, 2021.

[183] Kaifa Zhao, Le Yu, Shiyao Zhou, Jing Li, Xiapu Luo, Yat Fei Aemon Chiu, and Yutong Liu. A fine-grained chinese software privacy policy dataset for sequence labeling and regulation compliant identification. In *Proceedings of the 2021*

Conference on Empirical Methods in Natural Language Processing, EMNLP. Association for Computational Linguistics, December 2022.

[184] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. Structural attack against graph based android malware detection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[185] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[186] Hao Zhou, Ting Chen, Haoyu Wang, Le Yu, Xiapu Luo, Ting Wang, and Wei Zhang. Ui obfuscation and its effects on automated ui analysis for android apps. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 199–210, 2020.

[187] Hao Zhou, Xiapu Luo, Haoyu Wang, and Haipeng Cai. Uncovering intent based leak of sensitive data in Android framework. In *ACM Conference on Computer and Communications Security (CCS)*, 2022.

[188] Hao Zhou, Haoyu Wang, Xiapu Luo, Ting Chen, Yajin Zhou, and Ting Wang. Uncovering cross-context inconsistent access control enforcement in android. In *The 2022 Network and Distributed System Security Symposium (NDSS'22)*, 2022.

[189] Hao Zhou, Haoyu Wang, Yajin Zhou, Xiapu Luo, Yutian Tang, Lei Xue, and Ting Wang. Demystifying diehard android apps. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 187–198. IEEE, 2020.

[190] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326, 2012.

[191] Y. Zhu, Y. Lai, K. Zhao, X. Luo, M. Yuan, J. Ren, and K. Zhou. Binarizedattack: Structural poisoning attacks to graph-based anomaly detection. *arXiv preprint arXiv:2106.09989*, 2021.

[192] Yulin Zhu, Yuni Lai, Kaifa Zhao, Xiapu Luo, Mingquan Yuan, Jian Ren, and Kai Zhou. Binarizedattack: Structural poisoning attacks to graph-based anomaly detection. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 14–26. IEEE, 2022.

[193] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel R Reidenberg, N Cameron Russell, and Norman Sadeh. Maps: Scaling privacy compliance analysis to a million apps. *Proc. Priv. Enhancing Tech.*, 2019:66, 2019.