Toward Practical Machine Learning Solutions for IoT Malware Analysis

Yefei Zhang

A Thesis
In the Department
Of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Computer Science and Software Engineering)
at Concordia University
Montréal, Québec, Canada

December 2025

© Yefei Zhang, 2025

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Yefei Zhang**

Entitled: **Toward Practical Machine Learning Solutions for IoT Malware Analysis**

and submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY (Computer Science and Software Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Mohsen Ghafouri

_____ External Examiner
Dr. Rongxing Lu

_____ Arm's Length Examiner
Dr. Lingyu Wang

_____ Examiner
Dr. Abdelhak Bentaleb

_____ Examiner
Dr. Sandra Cespedes

_____ Thesis Supervisor
Dr. Chadi Assi

_____ Co-supervisor
Dr. Jun Yan

Approved by _____
Dr. Andrea Schiffauerova, Graduate Program Director

_____ _____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Toward Practical Machine Learning Solutions for IoT Malware Analysis

Yefei Zhang, Ph.D.

Concordia University, 2025

The Internet of Things (IoT) is rapidly expanding across personal, commercial, and critical infrastructure sectors. However, the low-cost requirements, limited resources, and heterogeneous ecosystems of IoT devices leave them highly vulnerable to malware attacks. Once compromised, attackers can exploit these devices for distributed denial-of-service (DDoS) attacks, data theft, or monetization. The evolving malware landscape—driven by obfuscation, packing, and code reuse—further complicates defenses, as adversaries continuously release new variants and families.

Machine Learning (ML) has become a cornerstone of malware analysis (MA), providing the ability to process large volumes of data and outperform traditional detection methods. Despite this promise, translating ML-based defenses into practical, real-world solutions remains a critical challenge. The difficulty stems from the conflict between ML's assumption of independent and identically distributed (i.i.d.) data and the intrinsic characteristics of the malware domain, where distribution shifts and adversarial threats are the norm: drift leads to inevitable model aging without adaptation, while AML exploits model vulnerabilities to manipulate predictions.

The practicality of solutions is shaped not only by these challenges but by the entire research process. If domain-specific characteristics are overlooked during this process, the resulting solutions risk limited applicability. Malware research can be categorized into three types: observational studies that analyze malware behavior and landscape features,

offensive research that exposes security gaps, and engineering solutions that propose defensive mechanisms. Although observational and offensive works might be expected to dominate, evidence shows that engineering solutions have been disproportionately prevalent, often at the expense of a deeper understanding of domain-specific realities.

To bridge the translation gap of current ML-based Malware analysis research, this research emphasizes the practicality of ML-based IoT malware defense solutions by aligning them with domain-specific characteristics and requirements. The contributions made include: (1) analyzing the vulnerabilities of existing ML-based malware detection models against AML attacks, (2) examining the impact of fundamental malware landscape attributes, including temporal, spatial, and architectural inconsistencies, on solution robustness, and (3) designing new IoT malware classification frameworks that integrate drift-handling mechanisms. By combining observational studies, offensive analysis, and engineering solutions, the research advances both theoretical understanding and practical deployment of ML-based malware defense.

Acknowledgments

This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the National Science Foundation (NSF).

Contents

List of Figures **x**

List of Tables **xii**

1 Introduction **1**

2 Background **6**

2.1 IoT Malware 6

2.1.1 IoT Malware Threats 7

2.1.2 Malware History and Evolution 9

2.1.3 IoT Malware Family and Classification 12

2.1.4 IoT CPU Architecture Diversity 14

2.2 ML-based Solutions 15

2.2.1 Malware Defense Technique Evolution 15

2.2.2 Stage Methods in ML-based Solutions 17

2.3 Practicality Factors for ML-based Solutions in IoT Malware Defense 19

2.3.1 Concept Drift 19

2.3.2 Adversarial Machine Learning Attacks 20

2.3.3 Resource Constraint 21

2.4 Malware Research Categories 22

2.5 Conclusions 23

3 Explore Vulnerability of IoT Malware Family Classifier against Backdoor Attack 26

3.1 Introduction . . . 27

3.2 Preliminaries . . . 29

3.2.1 IoT Malware Binary Format . . . 29

3.2.2 IoT Malware Classification Methods . . . 29

3.2.3 Backdoor Attacks: Research Gap . . . 31

3.2.4 Attack Vectors and Input-model Co-optimization . . . 32

3.3 Related Work . . . 34

3.4 Problem Statement and Threat Model . . . 36

3.5 Solution Formulation . . . 39

3.5.1 Model Selection - Q1 . . . 39

3.5.2 Bi-role Class-agnostic Input-aware Trigger Generator (B-CTG) - Q2 40

3.6 Experimental Evaluation . . . 47

3.6.1 IoT Malware Dataset and Clean Model Performance . . . 47

3.6.2 Attack Evaluation Metrics . . . 49

3.6.3 Attack Performance . . . 49

3.6.4 Mitigation . . . 57

3.7 Conclusion . . . 60

4 A Data-driven Study of IoT Malware Classification Models: Insights into Temporal, Architectural, and Spatial Inconsistency Challenges 61

4.1 Introduction . . . 62

4.2 Preliminaries . . . 63

4.2.1 ML in the IoT Domain . . . 63

4.2.2 IoT Malware Research Dataset Source . . . 66

4.3 Related Work . . . 67

4.4 Challenges and Evaluation Framework Design 68
4.4.1 Challenge Definition 69
4.4.2 Dataset Selection 69
4.4.3 Model Selection 70
4.5 Methodology 71
4.5.1 Dataset and Processing 71
4.5.2 Model Selection 73
4.5.3 Evaluation Configuration and Metrics 75
4.6 Experimental Evaluation 76
4.6.1 L0: Cross Validation (CV) 76
4.6.2 L1: RQ1 78
4.6.3 L2: RQ2, RQ3 and RQ4 80
4.7 Model Analysis 84
4.7.1 RQ2: Family Distinguishability and Temporal Stability 85
4.7.2 RQ3: Family Distinguishability and Architecture Generalizability . 87
4.8 Model Performance Explanation 88
4.9 Lessons Learned 91
4.10 Threats to Validity 92
4.11 Conclusion 93

5 Optimal Transport-Based Adaptation and Evolution Interpretation for IoT Malware under Concept Drift 95

5.1 Intorduction 96
5.2 Preliminaries 98
5.2.1 ML-based Malware Classification and Concept Drift 98
5.2.2 Domain Adaptation and Optimal Transport 98
5.3 Related Works 100

5.4 ELF-OT . . . 102
5.4.1 Feature Engineering . . . 102
5.4.2 Optimal Transport-based Adaptation . . . 104
5.4.3 Framework . . . 106
5.5 Rejection and Explainability Mechanisms . . . 110
5.5.1 Prediction Rejection . . . 110
5.5.2 Explainability . . . 114
5.6 Experiment Setup . . . 119
5.6.1 Dataset . . . 119
5.6.2 Experiment Focus . . . 120
5.7 Framework Evaluation . . . 122
5.8 Rejection Impacts . . . 125
5.8.1 Performance Comparison . . . 125
5.8.2 Rejection Performance *vs* Rejection Ratio . . . 128
5.9 Evolution Interpretation . . . 129
5.9.1 ELFMiner . . . 130
5.9.2 MalConv . . . 132
5.10 Conclusion . . . 133

6 Conclusion and Future Work **134**

Bibliography **138**

List of Figures

1 Mirai operation [16]. 8

2 Malware family relationship via efficiency improvement features [156]. . . 10

3 Feature Statistic 18

4 Training pipeline and poison results evaluation. 38

5 Attacker knowledge scenario 41

6 Trigger generator 42

7 Stages of dataset processing 48

8 Attack performance of main evaluation ($N = 400$) 50

9 Attack performance of main evaluation ($N = 1200$) 50

10 Detectable ability 52

11 Transferred setting experiment description 53

12 Comparison of different classifiers. 54

13 Performance under the TF-IDF representation. 56

14 Isolation forest detection rate 58

15 Clean samples and their tampered versions. 59

16 The impact of RQ1 challenge on models. The index mapping between the Figures and Table 7 is: [36]-[132], [45]-[155], [46]-[86], [48]-[95], [50]-[146], [51]-[129]. 79

17 The impact of RQ2 challenge on models. 80

18 Model Performance $F1_{\text{macro}}$ under RQ3 challenge. 81

19 The impact of RQ4 challenge on models. 82
20 FD&TS Features. 86
21 FD&TS Features extracted by [135] from one dataset. 86
22 FD&AG Features. 87
23 FD&AG Features extracted by [97] from one dataset. 88
24 Testing performance comparison. 89
25 Random Gafgyt sample. 90
26 Feature impacts on model output as Gafgyt across DT1. 91
27 MalConv [127]. 103
28 Intuition of Adaptation Evolution 114
29 Adaptation lineage under OT 115
30 Adaptation lineage under JDOT/DeepJDOT 117
31 Temporal direction study. 121
32 Performance comparison across year-pairs. 122
33 Rejection impacts on performance across year-pairs. 126
34 Experiment ratio. 126
35 F1 variation. 126
36 Group-wise similarity 129
37 Pairwise Pearson correlation of ELF structural-region similarity across adaptation frameworks. 133

List of Tables

1 Comparison Among Methods of Backdoor Attack Against Malware Analysis Models 36

2 Detection performance of the clean model [55] and its sub-components. . . 49

3 Hyperparameters for FFN, SVM, and RF models 55

4 Research Questions, Attributes, and Evaluation Configuration 70

5 Malware Sample Distribution 72

6 Hyperparameter Grids 72

7 Overview of studies IoT malware family classification methods 73

8 Studied models cross-validation performance ($F1_{\text{macro}}$) 77

9 The best performing models utilize [86] 78

10 Top 1/3 least models impacted under RQ2 and RQ3 (ordered by incremental impact) 84

11 Sample distribution over years 120

Chapter 1

Introduction

The Internet of Things (IoT) is rapidly expanding across personal, commercial, and critical infrastructure sectors, connecting billions of devices worldwide and enabling unprecedented functionalities and services. However, this growth is underpinned by low-cost requirements, limited hardware resources, and highly heterogeneous ecosystems, which collectively make IoT devices far more vulnerable than traditional computing systems. Attackers exploit these weaknesses to compromise devices and repurpose them for malicious activities such as launching large-scale distributed denial-of-service (DDoS) attacks, stealing sensitive data, or generating illicit revenue. The problem is compounded by the evolving malware landscape, where adversaries continuously develop new variants and families using obfuscation, packing, and code reuse, which complicates detection and analysis. As IoT devices become increasingly integral to daily life and critical operations, securing them against these persistent and adaptive threats has become an urgent challenge for both research and practice.

Machine Learning (ML) tools, on the other hand, have recently emerged as the cornerstone for IoT malware analysis (MA), offering the ability to process massive datasets and potentially outperform traditional detection methods. This promise has driven a surge in ML-based malware research, as reflected in the growing body of published work, and

ML-generated insights also support analysts in enhancing a variety of defense mechanisms. Yet, translating ML-based solutions into practical, real-world applications remains a critical challenge. As highlighted by Wagstaff's advocacy for "Machine Learning that Matters," much of the research struggles with insufficient pragmatic grounding, which hinders its applicability. As identified by Apruzzese et al. [19], the translation difficulty arises from the underlying conflict between ML's fundamental assumptions, namely, the principle of independent, identically distributed random variables (iid) [57], and the intrinsic characteristics of the cybersecurity domain. To address the translation gap under this conflict, they highlighted two critical issues the research communities must tackle: the lack of pragmatic results and the limited consideration of real-world scenarios. This aligns with the advocacy of [158], which calls for some fundamental changes in the formulation and evaluation of ML solutions. Many recent studies analyzed the challenges and pitfalls in ML-based MA solutions, including [39, 30, 67, 36, 139, 72, 96, 128, 168, 63, 37, 71]. The high-frequency mentioned ones include concept drift, label delay, sample bias, class imbalance, against AML attacks, and explainable results, with concept drift and adversarial ML attacks being the most commonly cited. Concept drift, which reflects the evolving nature of the malware landscape, poses a challenge for ML solutions built on i.i.d. assumptions, as they risk failure without adaptation. Similarly, AML attacks, designed to manipulate model behavior, compromise the reliability of ML solutions that do not account for these threats.

The practicality of a solution is determined by the entire research process. Botacin et al. [30] break down the research pipeline into phases: research project definition, solution design, experimental design, and evaluation. If the domain-specific characteristics are not reflected during the process, the practicality of the solutions would become problematic. The above-mentioned challenges reflect some characteristics in the malware field, and these challenges are revealed from some research studies, such as the label delay issue observed by [175]. Botacin et al. [30] conducted the first comprehensive analysis on malware

research in nearly two decades, examining the difficulties of translating research into practical solutions based on scientific [123] and engineering principles [25]. They categorize the research studies in MA as: observational studies, which focus on understanding and analyzing malware; offensive research, which identifies security gaps and vulnerabilities to inform the development of future security solutions; and engineering solutions, which propose mechanisms to counter malware threats. Based on the emphasis of these three types of studies, observational studies and offensive research would be expected to be the most common. However, an analysis of their distribution over 14 years reveals that engineering solutions have been the most prevalent.

Given the challenges and research directions in IoT ML-based malware analysis, this PhD thesis aims to explore key aspects, identify prevailing trends, contribute new insights, and integrate knowledge to advance the field. Specifically, the research focuses on advancing ML solutions' practicality in the IoT malware domain. This means avoiding excessive optimization on isolated datasets, refraining from integrating complex techniques that enhance specific metrics at the cost of other necessary practical considerations, and ensuring that solutions are built with a comprehensive understanding of malware landscape features. To implement this, the research follows a structured approach: (1) analyzing the vulnerabilities of existing practical IoT ML-based MA solutions, (2) examining the impact of fundamental malware landscape attributes on existing IoT ML-based solutions, and (3) leveraging insights from practical demands, malware characteristics, and existing approaches to develop practical IoT ML-based malware analysis solutions.

Following the above, the thesis makes three contributions to the body of knowledge, which can be classified as follows:

(1) To Explore Vulnerability of IoT Malware Family Classifier against Backdoor Attack. This research aims to investigate the vulnerability of existing practical ML solutions developed for MA under AML attacks. To ensure that the exploration findings reflect

real-world attack conclusions, the target ML solutions would be carefully selected based on their consideration of the practical challenges listed above. Given the diversity of attack types and procedures summarized by [147], we focus on those that are highly stealthy and compatible with the selected ML solutions. With these considerations, this work aims to design a new attack framework and reveal phenomena that benefit the model defenders or attackers in designing new practical solutions.

(2) A Comprehensive Study of IoT Malware Classification Models: Insights into Temporal, Architectural, and Spatial Distribution Challenges. Recognizing the importance of observational studies and offensive solutions, this work conducts a practical investigation into how fundamental malware landscape challenges affect existing approaches. Specifically, temporal, architectural, and spatial distributions are fundamental attributes or variables to characterize datasets. When inconsistencies in these variable values arise between training and testing datasets, model performance potentially has degradation risks. Through the analysis of current solutions, this study aims to draw general insights into the impacts of these challenges, enhance the utilization of existing solutions, and derive insights to inform future solution development.

(3) A Domain Adaptation-based IoT Malware Classification Frameworks. The conflict between concept drift and the fundamental assumptions of machine learning potentially results in performance degradation. Domain adaptation techniques [91] are designed to address the distribution shift issue and have been extensively studied, but their application to malware analysis has been limited. To fill this gap, this work explores their suitability for the malware domain by selecting techniques with the consideration of domain-specific requirements, constructing evaluation frameworks that combine domain-relevant features, and designing rejection mechanisms tailored to each framework's principle to improve their reliability. Through this systematic approach, we aim to thoroughly assess and enhance these frameworks' performance

while generating insights tailored to the malware field.

The thesis is organized as follows: Chapter 2 presents the fundamental research background, including IoT malware threats, ML–based defense approaches, and the key challenges affecting their practical applicability. Chapter 3 introduces the first contribution, which investigates the vulnerability of IoT malware family classification models to backdoor attacks. Chapter 4 discusses the second contribution, providing a comprehensive analysis of challenges in the IoT malware landscape and evaluating their impact on existing solutions, with summarized findings to guide future research. Chapter 5 presents the third contribution, which constructs IoT malware classification frameworks with adaptation mechanisms and explores the evolutionary relationships revealed through sample adaptation lineages. Finally, Chapter 6 concludes the thesis by summarizing the research findings, reflecting on the overall study, and outlining directions for future work.

Chapter 2

Background

2.1 IoT Malware

The Internet of Things (IoT) has emerged as one of the most transformative technologies of the past decade, interconnecting a vast array of devices ranging from consumer electronics, smart home appliances, and wearables to industrial control systems and critical infrastructure sensors [14]. The expansion of IoT has not only enabled new functionalities and services but has also introduced substantial cybersecurity challenges. Many IoT devices are deployed with minimal built-in security, often relying on default credentials or lacking secure update mechanisms. Their heterogeneous hardware and software ecosystems further complicate the implementation of standardized security measures. These vulnerabilities have made IoT devices a prime target for malware attacks, which exploit these weaknesses to compromise devices, exfiltrate data, or disrupt operations [42, 141].

According to IoT hacking statistics 2025 [109], the ecosystem weathers an average of 820,000 hacking attempts every day, a 46% increase from the previous year. This translates to a constant, automated hum of malicious activity that creates a state of continuous compromise for any internet-facing device. The financial consequences of an IoT security failure are severe and escalating. Historical data from Forrester shows that 34% of breaches

involving IoT devices result in cumulative costs between $ 5 million and $ 10 million, significantly higher than traditional IT incidents. The latest data from 2025 provides an even sharper picture. The breach in the industrial sector and healthcare contributes to a global cybercrime economy projected to inflict $ 10.5 trillion in damages in 2025.

2.1.1 IoT Malware Threats

IoT devices are targeted through various attack vectors, as detailed by Sutheekshan et al. [146]. Firmware exploitation is a prevalent avenue where unpatched vulnerabilities or firmware backdoors allow unauthorized access or misuse of the device's functionalities. Weak authentication mechanisms, such as users failing to change default credentials, enable attackers to launch brute-force attacks or gain control through default logins. Based on Nozomi Networks' July 2025 analysis of real-world OT environments, brute forcing default credentials remains a top technique used by botnets to gain initial access [109], and 7.36% of all detected attacks are brute force attempts, while another 5.27% directly exploit default credentials to achieve lateral movement within a network. Insecure communication protocols are another attack vectors, which allow malicious actors to intercept or manipulate data transmitted between IoT devices, applications, and servers. Furthermore, the resource constraints of IoT devices and their relatively low system complexity enable attackers to design malware capable of compromising these devices with minimal effort. These are also the differences between IoT systems and traditional networks, summarized by Makhdoom et al. [106]. Therefore, the conventional perimeter defense mechanism cannot protect the IoT devices from insider attacks and physical compromise by unauthorized employees/personnel.

Makhdoom et al. [106] classifies the attack procedure targeting IoT into five phases: 1) preparatory phase, when attackers carry out reconnaissance and collect information about the potential target; 2) initial exploitation and infiltration phase, when the attackers decide

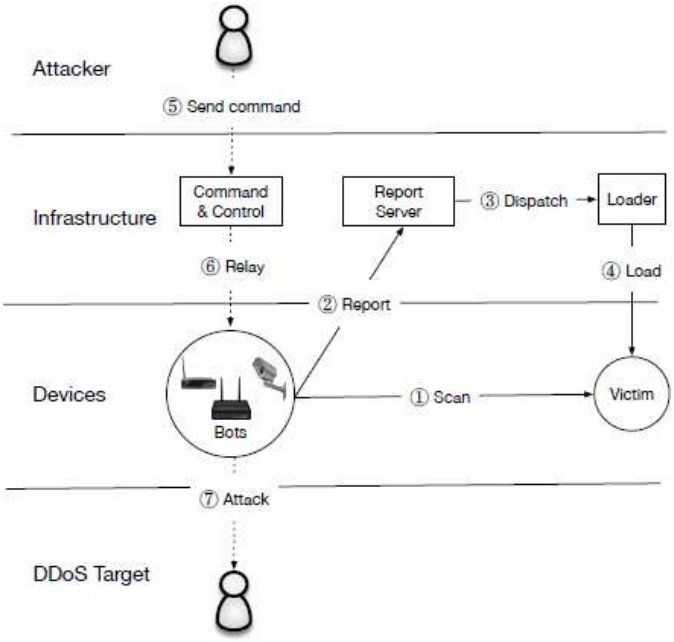


Figure 1: Mirai operation [16].

on the type of exploit, its functionalities, and the attack vectors to deliver the exploits to the target systems; 3) execution phase, when attackers can steal information or perform a malicious action either by remote access or through a sophisticated malware that installs a Trojan on the host system; 4) propagation phase, when the malware reproduce or move from the infected system to a new host; 5) hideout and clean-up phase, when malware use multiple techniques to keep themselves invisible, while operating on a victim system. With the compromised devices, attackers can achieve multiple broader cybercrime activities, including Distributed Denial of Service (DDoS) attacks, cryptocurrency mining, industrial spying and general spying, Botnet as a Service (BaaS), etc. The Mirai botnet, first identified in 2016, is among the most prominent examples [146]. It amassed an unprecedented botnet by compromising numerous devices, ultimately transforming them into a powerful weapon for Distributed Denial of Service (DDoS) attacks. Based on [16], this botnet infected nearly 65,000 IoT devices in its first 20 hours before reaching a steady state population of 200,000–300,000 infections. Figure 1 describes the Mirai procedure. It propagates

by scanning the internet for vulnerable IoT devices with open Telnet ports (TCP/23 and TCP/2323). Once a target is found, it attempts to log in using a brute-force attack with a hardcoded list of common username-password pairs. If successful, the credentials are sent to a report server, and a loader then installs architecture-specific malware on the compromised device. Mirai hides its presence by deleting the installer and obfuscating its process name, and it disables competing malware by killing related processes. After infection, the device connects to a command-and-control (C2) server to receive DDoS instructions while continuing to scan for new victims.

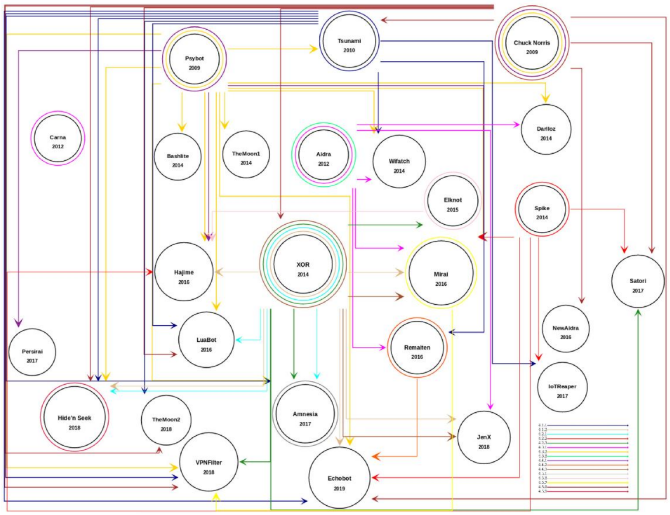
2.1.2 Malware History and Evolution

Alenezi et al. [9] split malware history and evolution across five phases based on the time frame it emerged. In the early phase (1949-1991), the malware's intention was not to harm, steal, or manipulate systems. For example, Brain Boot Sector Virus, the first virus designed in 1986 by two Pakistani brothers to infect the MS-DOS system, was initially to test their company's software and to prove that it was not a secure platform. During the second phase (1992-1999), hackers and attackers became interested in the Windows Operating System since it attracted many users for its simplicity and powerful user interface, and most of the malware in this phase was targeting the Windows Operating System. In the third phase (2000-2008), with the widespread use of the network, malware evolution mainly referred to network worms and viruses, and was transmitted through email attachments, free downloads from compromised websites, or open network shares. The fourth phase (2005-2016) focused on financial gain or illegal control. The most common ways for malware of this phase to infect the victims are phishing emails, remote desktop protocols, downloads from compromised websites, and USB or other removable media. Previous malware types were created by cybercriminals to target businesses or personal PCs. The current/fifth phase (2010-present) of malware evolution is mainly for virtual espionage and sabotage because

it causes severe damage without affecting human lives.

Intelligence (4.1)	Persistence (4.2)	Prevention (4.3)	Discretion (4.4)	AntiAnalyze (4.5)
Code modularity (4.1.1)	Daemonisation (4.2.1)	Other botnet removal (4.3.1)	Binary removal (4.4.1)	Detection of virtualized environment (4.5.1)
Victim's architecture detection (4.1.2)	Anti reboot (4.2.2)	Closing ports (4.3.2)	Process name stealing (4.4.2)	Bad ELF information (4.5.2)
	CRON execution (4.2.3)	Password modification (4.3.3)	Random name (4.4.3)	Corruption UPX header (4.5.3)
	Program infection (4.2.4)	PATCH (4.3.4)		Anti debug (4.5.4)
	File infection (4.2.5)			Anti execution (4.5.5)
				Malicious code execution delayed (4.5.6)
				DGA Algorithm (4.5.7)
				Obfuscation of important information (4.5.8)

(a) Efficiency improvement features.



(b) Efficiency feature propagation multigraph.

Figure 2: Malware family relationship via efficiency improvement features [156].

Sutheekshan et al. [146] reviewed the historical evolution of IoT malware, emphasizing the early 2000s, a period marked by the rapid proliferation of networked devices. During this time, security professionals increasingly encountered attacks on home gadgets, such as networked routers and simple embedded systems. Vignau et al. [156] analyzed the evolution of 28 IoT malware families, spanning from Hydra in 2008 to EchoBot in 2019. By categorizing these families according to four dimensions—ultimate objectives, infection

processes, botnet control methods, and efficiency improvements—they highlighted their relationships and evolution features over time. For example, Figure 2 shows their summarized efficiency improvement features and family relationship. Each malware sample is depicted as a black circle, where the radius corresponds to the number of efficiency-related features it incorporates. A colored circle is added to indicate the first malware sample to introduce a given feature, with each feature represented by a distinct color. Edges sharing the same color are drawn from the originating malware to all subsequent samples that adopt the same feature, illustrating the propagation of that feature across malware families. Their conclusions include: 1) IoT botnets tend to become more complex and more efficient over the years since it is easy to add new features; 2) existing features can be easily reused, especially in the case of open source botnets; 3) botnets moved away from dictionary attacks and tended to exploit vulnerabilities instead, and this phenomenon will likely become more prevalent in coming years as vendors move away from devices with weak default credentials; 4) The methods used to infect new victims are evolving faster than the kinds and goals of the attacks (DDoS, spying etc).

Evolution occurs not only across different malware families but also within individual families. For example, Mirai notably targeted the French firm OVH, generating over one terabyte per second of bandwidth, and later attacked the DYN DNS service, disrupting network access for millions of users on the U.S. East Coast. Following the release of its source code, numerous variants quickly emerged. For instance, roughly a month after Mirai's appearance, a new version exploited vulnerabilities in approximately 900,000 German routers. As Antonakakis et al. [16] observed, several key developments occurred within two months after the Mirai code release: the underlying control and command (C2) infrastructure upgrade from an IP-based C2 to a domain-based C2; the malware began to delete its executing binary, as well as obfuscate its process ID; a number of features added to make the malware more virulent, including the addition of more passwords to infect

additional devices, the closing of infection ports TCP/23 and TCP/2323, and the aggressive killing of competitive malware. A notable recent variant of Mirai is Aquabot, which has been actively targeting Mitel SIP phones [95]. Initially observed in November 2023, its first version closely mirrored Mirai's original framework. The second iteration introduced concealment and persistence features, such as blocking shutdown and reboot operations to maintain device control. The third version introduces a distinctive capability for a Mirai-derived botnet: it enables the bot to communicate with its command-and-control (C2) server upon receiving specific system signals. This behavioral shift, along with other functional enhancements, marks a significant departure from previous versions and supports the classification of this variant as a distinct third stage in Aquabot's evolution.

2.1.3 IoT Malware Family and Classification

IoT malware families have their own traits and approaches, like unique propagation methods, target devices, and attack strategies. For example, Mirai primarily exploits weak or default credentials on IoT devices to form vast botnets to implement largest DDoS attacks recorded to date. Another significant family, Gafgyt targets Linux-based IoT devices and similarly leverages brute-force attacks to co-opt devices into botnets for volumetric network attacks [77]. Unlike Mirai and Gafgyt, Hajime represents a more stealth-oriented approach, propagating through peer-to-peer networks and aiming for persistence rather than immediate disruption, illustrating how IoT malware can vary widely in operational strategy [15]. Other families, such as VPNFilter, focus on compromising network-attached devices like routers to intercept sensitive data or disrupt network communications, highlighting the diverse goals that IoT malware families can pursue [79]. The variation among IoT malware families—from infection techniques to attack objectives—highlights their differences. Given that multiple versions of malware samples employ similar attack techniques,

malware analysts group malware into families to avoid analyzing known malware samples in depth [168]. Meanwhile, tailoring defense mechanisms to each family's distinct behaviors/purposes significantly enhances effectiveness, particularly given the constrained computational resources of IoT devices. For example, defenses against Mirai often focus on enforcing strong credential policies and monitoring network behavior for scanning activities, whereas countermeasures for Hajime may require detection mechanisms capable of identifying stealthy peer-to-peer communication patterns [78]. Meanwhile, according to [70], most of the new malware variants come from known malware, with less than 2% code differences between variants, which indicates that most malware variants of the same family are similar to some extent, and it is necessary to classify malware [169].

Based on 2023 Enterprise IoT malware report [130], Mirai and Gafgyt remain the most dominant IoT malware families, responsible for the majority of observed attacks. From a network security perspective, particularly regarding blocked transactions—including payload delivery, command and control (C2) communications, and malicious URLs—Mirai alone accounted for 91% of detected blocks. In terms of overall blocked payloads, Mirai and Gafgyt contributed to 46% and 20% of incidents, respectively. These trends coincide with a surge in IoT botnet-powered distributed denial-of-service (DDoS) attacks. Notably, such activity resulted in an estimated global financial loss of $ 2.5 billion in just the first half of 2023. Additionally, multiple new IoT malware families emerge consistently. For instance, BrickerBot, active around 2017, distinguished itself by deliberately destroying IoT devices rather than repurposing them. It logged into insecure devices and issued commands that corrupted storage or disabled functionality, aiming to make devices permanently inoperable [3]; Mantis, which was first detected conducting major attacks in 2022, illustrates a notable strategic shift. Unlike Mirai, which relied on a vast network of low-power IoT devices, Mantis operates with a much smaller fleet—approximately 5,000 hijacked

virtual machines and servers. This concentrated approach enables it to launch more sophisticated and resource-intensive HTTPS-based DDoS attacks, including incidents that reached a peak of 26 million requests per second [81, 109].

2.1.4 IoT CPU Architecture Diversity

To enable large-scale IoT deployments—often involving millions of interconnected devices operating under strict cost and energy constraints while still addressing a broad range of security and privacy requirements—IoT hardware is typically designed with limited computational and storage resources when compared to traditional computing platforms like personal computers and laptops. This is also another difference summarized by Makhdoom et al. [106] that makes it difficult to develop a standard security protocol that fits all types of IoT devices and physical security. These constraints necessitate the use of lightweight operating systems (OSs) and specialized CPU architectures that are tailored to the specific demands of IoT environments [129]. For example, ARM architectures, based on reduced instruction set computing (RISC), consume less power and require simpler hardware than complex instruction set computing (CISC) architectures such as x86. This makes ARM a more suitable choice for energy-sensitive and compact devices like wearables, whereas x86 remains dominant in high-performance systems like laptops and desktops. Correspondingly, each architecture differs in instruction sets, opcodes, and assembly language, memory layout, supported operating systems, and lead malware to exhibit architecture-dependent behaviors, such as the instruction sets for ARM [2] and MIPS [41], where MIPS instructions are usually 32 bits long, and have a fixed format that consists of an opcode, a source register, a destination register and an immediate value or an offset, while ARM some versions also support 16-bit or 64-bit instructions and has a variable format that can have different fields depending on the instruction type, such as conditional codes, shift amounts, rotate amounts or flags. Given this diversity, features extracted from malware

on one architecture may not generalize to others, which can degrade the performance of detection systems if architectural variations are not explicitly [10]. Effective malware classification thus requires careful consideration of cross-architecture characteristics, often involving architecture-agnostic feature extraction, normalization, and alignment techniques to ensure detection models remain robust across heterogeneous devices.

2.2 ML-based Solutions

Machine learning presents a viable approach to defending against malware [61]. Its capacity to analyze vast amounts of data enables the timely detection and adaptation to emerging and evolving malware threats. Additionally, ML-generated insights, predictions, and decisions, coupled with extensive datasets, empower analysts to bolster the performance of various defense mechanisms. This technology also addresses the cybersecurity skills gap by minimizing the need for a large workforce to research, analyze, and share information on malware detection [61]. For example, training a system with examples of desired input-output behavior proves simpler than traditional system development methods [82]. As stated by Gibert et al. [67], cybersecurity experts generally believe that AI-powered antimalware tools will enhance malware detection and improve scanning engines. This sentiment is reinforced by the growing published research searched from Google Scholar.

2.2.1 Malware Defense Technique Evolution

Along with malware becoming more sophisticated, detection methods have evolved to keep up with the ideas of malware developers. Works [37, 58] summarize the evolution of defense techniques against malware.

- **Signature-based**: Early malware detection focused on limiting viral spread by restricting suspicious programs' access and analyzing them before execution. These

approaches relied on signature-based detection, which matched binary patterns—such as code fragments, network protocols, packet payloads, and suspicious behaviors—to known malware signatures. While signatures enabled fast identification of existing threats, they proved ineffective against new malware variants.

- **Behavior-based**: These approaches identify malicious behavior during runtime—such as system slowdown, disabled security functions, frequent pop-ups, etc.—to recognize malware patterns and previously unseen threats. While effective at detecting unknown malware, they remain vulnerable to mimicry attacks, where malicious code is hidden within otherwise benign-looking programs.

- **Heuristic and Specification-based**: Heuristic-based methods combine signature and behavior analysis to detect malware based on file characteristics learned from data mining and ML techniques. While effective at identifying known threats, they struggle with obfuscated code and suffer from high false positive rates, computational overhead, and require additional manual analysis. Specification-based methods rely on expert-defined behavior rules and can detect both known and unknown malware with fewer false positives, but they tend to be time-consuming and exhibit high false negative rates due to limited rule coverage.

- **Energy-based**: These methods leverage the energy consumption patterns of hardware and software to identify potential security threats, including emerging forms like stegomalware. By treating power usage as a high-level behavioral indicator, they abstract away from specific attack details. However, their effectiveness depends on accurately modeling a device's normal power profile, and results can be affected by measurement inaccuracies or external hardware influences.

- **Machine Learning-based**: These methods use machine learning, deep learning, and data analysis techniques to identify complex patterns and correlations to detect

known and unknown malware threats. Since the 1990s, the adoption of ML-based detection has surged, now dominating modern defense strategies. These approaches excel at handling obfuscated code, minimizing false positives, and outperforming traditional techniques in identifying previously unseen malware.

- **Others**: Blockchain-based methods ensure secure and reliable storing and sharing of signatures, as well as a framework for using them for detection duties, but these methods are still in early stages, facing practical deployment challenges due to an immature ecosystem and a lack of robust proof-of-concept implementations. Similarly, bio-inspired approaches—such as those using Genetic Algorithms (GA) [144] or Particle Swarm Optimization (PSO) [88] for feature selection and optimization—show promise when integrated with machine learning. Yet, to become viable in real-world applications, they must overcome key limitations, including handling dataset imbalances and ensuring computational efficiency.

2.2.2 Stage Methods in ML-based Solutions

The development pipeline for ML-based solutions typically involves several stages, including data collection, feature extraction, feature reduction, model selection, and model evaluation. In this context, Kumar et al. [94] conducted the first Systematic Literature Review (SLR) on IoT malware defense, examining 50 studies published between 2018 and 2023. Their analysis focused on four dimensions: the features employed, the ML algorithms applied, the detection objectives addressed, and feature reduction techniques.

In the feature dimension, they categorized these solutions into static and dynamic approaches. Static analysis inspects malicious binaries without execution, relying on structural and code-level features such as API call–based methods, which examine the sequence and frequency of suspicious function calls; flow graph–based methods, which represent execution and data flows as directed graphs and apply graph mining for anomaly detection;

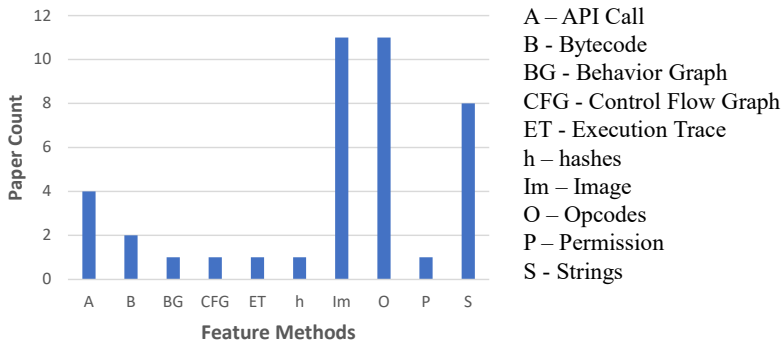


Figure 3: Feature Statistic

opcode–based methods, which treat opcode sequences as textual data and apply natural language processing algorithms; and image processing–based methods, which convert binaries into images and use machine learning techniques designed for image recognition. In contrast, dynamic analysis executes malware in a controlled environment to observe runtime behaviors that static methods may miss, including process activity monitoring to trace process creation and modifications, file system monitoring to log changes such as dropped or altered files, network activity monitoring to analyze inbound and outbound traffic for anomalies and Indicators of Compromise (IoCs), and system or function call monitoring to track low-level interactions with the operating system, files, and memory. Figure 3 summarizes the features and distribution among their examined papers.

In the ML algorithm dimension, they summarized that 56% of the research works examined in this study have adopted Neural Network models (CNN, RNN, LSTM), 36% of studies have adopted Support Vector Machines (SVM) model for classification, and the rest 8% of studies adopted supervised ML models, which include NB, RF, DT, KNN & LR, etc.

In the feature reduction dimension, It was observed that 24% of the studies employed standard feature reduction techniques, such as Information Gain (IG), Opcode Dictionary, Recursive Feature Elimination (RFE), Gray Wolves Optimization (GWO), and Principal Component Analysis (PCA). An additional 12% of the studies refined or customized these

methods, proposing novel feature reduction approaches. About 17% of the studies leveraged Convolutional Neural Networks (CNNs) as feature extractors, most frequently incorporating word embedding techniques. In contrast, approximately 47% of the studies did not adopt any feature reduction method, relying instead on the original attributes.

2.3 Practicality Factors for ML-based Solutions in IoT Malware Defense

The ultimate objective for ML-based solution design is its deployment in real-world scenarios. However, the factors that lack consideration during solution formulation and evaluation hinder their practical translation. Many studies analyze such challenges and pitfalls in current research, including [39, 30, 67, 36, 139, 72, 96, 128, 168, 63, 37, 71].

2.3.1 Concept Drift

Concept drift is a notable issue highlighted by multiple studies [19, 39, 67, 139, 72, 168, 71, 128], describing the problem of changing underlying relationships in the data [67]. Ceschine et al.[39] highlight the challenge of concept drift in the ML solutions for malware analysis, distinguishing it from other domains like image analysis. This characteristic arises due to the perpetual arms race between attackers and defenders, leading to the continual evolution of the malware landscape [19]. Gibert et al.[67] attribute this to the nature of software development in general, not just malware. For example, software naturally evolves due to adding features, fixing bugs, and porting to new environments and platforms; successive versions of software gradually differ from older versions with significant refactoring and changes to libraries. The drift degree is also leveraged by the malware analysts to prioritize the sample analysis [168], demonstrating the presence of concept drift in real-world scenarios. In the image field, “a cat will always be a cat,” allowing ML models

trained on the same dataset over decades to remain effective [19]. However, in the context of malware, continuous changes in the landscape mean that models trained or tuned today may not effectively detect tomorrow's malware, rendering them vulnerable to zero-day attacks. This model decay phenomenon primarily stems from the conflict between the iid assumption of ML and the malware landscape feature, where ML assumed that the mapping learned from historical data will remain valid for future data and that the relationship between input and output does not change over time, but the evolving nature of the malware landscape causes model prediction performance to decay, failing to recognize new samples.

2.3.2 Adversarial Machine Learning Attacks

The concept drift inherent in the malware landscape exposes ML analysis models to vulnerabilities against 0-day attacks, leading to ineffectiveness. However, the vulnerabilities that result in their ineffectiveness can also stem from the ML systems themselves, as attackers target ML models to create adversarial samples capable of evading model detection. This challenge has been highlighted in several studies [19, 39, 30, 67, 36, 20, 128, 37, 71]. According to the summary by Tabassi et al., [147], the various stages of ML model training and evaluation present multiple attack surfaces for exploitation, including poisoning and evasion methods. Various countermeasures have been proposed to improve it, but as noted by [19], a universal solution remains elusive, and some mechanisms may even degrade baseline performance. These challenges highlight the necessity for researchers to focus their efforts on mitigating adversarial attacks [39]. The robustness of an ML system directly determines its successful deployment, and its integrity relies on the robustness of the ML models and samples [39, 20]. Adversarial machine learning (AML) attacks can generally be classified based on the ML solution phases: training and testing. Training phase attacks aim to acquire or influence the training data or the model itself. According to [147], these attacks are classified as data access attacks, which assess some or all the

training data to create substitute models for attack testing; poisoning attacks, which can alter data or models either directly or indirectly. Attacks in the testing phase do not tamper with the target model or the data used in training. This stage's attacks are classified into two categories. Evasion attacks generate adversarial examples to evade proper output classification by the model, and oracle attacks collect and infer information about the model or training data.

2.3.3 Resource Constraint

Driven by the proliferation of the Internet of Things (IoT) and the need for real-time data processing, Wang et al. [162] conducted a recent survey, which comprehensively explored the current state, technical challenges, and future trends of on-device AI models. Compared with cloud-based AI models, on-device AI models have the following benefits: offering low latency, which makes them suitable for real-time applications, and enhancing privacy by processing data locally, thereby reducing the risks of data breaches. By contrast, they are constrained by computational resources, limited scalability due to hardware constraints, and require complex updates and maintenance, particularly in large deployments. In detail, they summarized the following resource constraints that challenge on-device AI models: 1) The limited performance of CPUs and GPUs in these devices may not suffice to meet the real-time processing demands of complex models. Complex models generally demand more computational resources, resulting in increased latency during execution on edge devices, which can adversely affect user experience. 2) Many AI models, particularly state-of-the-art deep learning models, require substantial storage space to accommodate model parameters and intermediate results. This limitation may hinder the ability to load all necessary data during inference, adversely affecting performance and response speed. 3) Many edge devices rely on battery power, and the high energy demands of AI models can lead to rapid battery depletion, adversely affecting user experience. 4) Edge devices

typically face significant communication bandwidth limitations compared to servers, making it challenging to transfer large volumes of data between the edge and the cloud, which poses obstacles for transmitting the substantial data required by many AI models.

2.4 Malware Research Categories

Botacin et al.[30] conducted the first comprehensive analysis of malware research in nearly two decades (2000–2018), examining both the challenges faced by the field and the difficulties of translating research into practical solutions, guided by scientific principles[123] and engineering principles [25]. Following the PRISMA methodology [124], their review synthesized results from 491 papers. Based on their summary, the research under the malware "umbrella" can be generally categorized into four main areas: 1) engineering solutions — research proposing mechanisms to counter malware, including signature-based and behavioral detectors, as well as tools and frameworks supporting malware analysis (e.g., static analyzers, sandboxes, reverse engineering frameworks); 2) offensive techniques — research uncovering gaps or vulnerabilities to guide the development of stronger defenses. This includes presenting exploits and proof-of-concept (PoC) attacks against specific targets; 3) observational studies — research analyzing malware samples from specific families, platforms (e.g., desktop, mobile), or ecosystems (e.g., a country, corporation) to inform future solution development, tools, and the understanding of malware trends; 4) network traffic analysis — research focusing on detecting malicious payloads in network traffic. Such work may or may not involve executing malware samples, since malicious traffic can be collected from various sources (e.g., honeypots, corporations). Although not always based on direct malware analysis, this category contributes by advancing detection through characterizing malicious network behavior. These categories are technology-agnostic, i.e., their goals might be accomplished using distinct techniques.

Because understanding malware should precede the development of defenses—since

such insights inform the impact of vulnerabilities, the evaluation of countermeasures, and the interpretation of real-world contexts—observational studies would be expected to dominate malware research. Likewise, one might anticipate a greater number of offensive studies, as these help anticipate threats, identify gaps left by observational research, and support the design of robust defenses against novel attacks. However, Botacin et al.'s investigation revealed that between 2000 and 2004, only engineering solutions were published, and this category has remained the dominant research type ever since. This significant disparity between the prevalence of papers on Engineering Solutions vs. the remaining types (e.g., observational and offensive research work combined account for less than 50% of all published Engineering Solutions papers) raises concerns about the effectiveness of such approaches—they might address less relevant aspects, factors, and problems due to a lack of insights about real-world malware trends.

2.5 Conclusions

The wide deployment requirements and low cost expectations of IoT devices make them more vulnerable than traditional computing systems. These vulnerabilities arise from resource constraints, weak security mechanisms, less robust communication protocols, heterogeneous CPU architectures, and inconsistent data formats. Combined with their broad application and high business value, IoT devices have become prime targets for attackers. Once adversaries establish access—whether through communication channels or direct physical interfaces—they can manipulate devices for malicious purposes such as launching DDoS attacks, stealing private information, or monetizing compromised systems. To hinder deeper analysis and improve defense efficiency, malware samples are generally grouped into families, where those in the same family typically share a core codebase, architecture,

behavior, and objectives. Due to the ongoing arms race between attackers and defenders, adversaries continuously generate new variants through techniques such as obfuscation, packing, and encryption. Meanwhile, publicly released code from specific malware families serves as a resource for developing new families, resulting in strong lineage connections across different families. Given the strong capabilities of ML, and its advantages over traditional detection methods, the research increasingly leverages ML to develop malware defense solutions. Designing ML-based solutions involves a multi-step development pipeline, so even when pursuing the same design objectives, diverse solutions can be proposed across the pipeline stages.

The ultimate objective of designing ML-based solutions for malware defense is to deploy them in real-world scenarios while effectively implementing their initial design purpose. To achieve this, three major concerns must be addressed during the design process. Concept drift, which reflects the evolving nature of the malware landscape, can cause models to degrade over time if they are unable to adapt to new threats, leading to model aging issues. Adversarial machine learning (AML) attacks pose direct risks to the security and reliability of ML-based solutions, as carefully crafted perturbations can mislead models and render them ineffective. Finally, resource constraints inherent to many IoT and embedded environments limit model complexity and computational capability, restricting the use of complex defenses and requiring lightweight yet effective alternatives.

The advancement of malware research is sustained through the contributions of diverse study types and the dependencies that connect them. Engineering-focused studies develop defensive mechanisms, detection frameworks, and analytic tools, which serve as the frontline against emerging threats. Offensive research, in turn, exposes weaknesses in existing systems, highlights overlooked attack vectors, and drives the need for more resilient defenses. Observational studies, by analyzing malware samples, ecosystems, or attack campaigns in the wild, provide empirical evidence of real-world trends. These insights feed

back into both engineering and offensive work, ensuring solutions remain grounded in practice rather than theory alone. Without this interplay, malware research risks stagnation, either by developing defenses detached from real-world attacks or by conducting analyses without actionable outcomes. Thus, progress in the field is not the result of isolated contributions, but rather the integration of these complementary perspectives.

Chapter 3

Explore Vulnerability of IoT Malware Family Classifier against Backdoor Attack

The ultimate goal of ML-based solutions in the malware domain is their reliable deployment in real-world environments. However, adversarial machine learning attacks pose a significant threat to the reliability of these models, raising several critical questions: Which types of attacks warrant deeper exploration? What defensive strategies should be investigated in response? How should such attacks be designed to accurately reflect practical conditions? And, most importantly, how do these attacks undermine the effectiveness of practical solutions? And what indications can be obtained for the defenders? The research presented in this chapter is primarily guided by these questions.

3.1 Introduction

Multiple research studies [55, 169, 150, 154] developed ML-based solutions for IoT malware classification and gained good performance. However, the inherent unpredictability in the behavior of the developed ML models raises critical concerns, which allows attackers to introduce and conceal sophisticated threats within IoT systems [174, 160]. For instance, recent studies highlight backdoor attacks against ML classification systems as a targeted and deceptive emerging threat, where adversaries embed specific triggers or patterns within a targeted ML model to enable the manipulation of its behavior under certain conditions [147, 114]. In other words, a successful backdoor attack can lead to the misclassification of certain malware samples into different/incorrect families, resulting in potentially dangerous outcomes such as evading detection and providing misleading information for security practitioners and analysts. Note that identifying misclassified malware samples is typically a labor-intensive process that demands deep domain knowledge [66]. The complexity and potential impact of such misclassifications underscore the urgent need for robust countermeasures across the malware classification systems.

Indeed, recent surveys offer comprehensive insights into backdoor attacks and their defenses in various domains such as image processing and classification [100, 66, 74]. Furthermore, a relatively smaller number of studies have investigated backdoor attacks on malware detection/classification models [134, 166, 99]. Despite the effort, a number of limitations exist. For instance, the studies introduce Adversarial ML models that operate within the feature space derived from manually crafted rules (i.e., rule-based extracted features), which is expensive to generate while not necessarily generalizable and effective in different contexts. Moreover, previous work did not focus on investigating an attacker-centric approach, which considers the practical objectives/viewpoints of an attacker when targeting specific domains such as IoT malware.

Motivated by the high-security demands for IoT systems and the threats associated

with backdoor attacks on ML models, in this work, we assume the attackers' viewpoint to explore the vulnerabilities of state-of-the-art IoT malware family classifiers. It is worth mentioning that crowd-sourced threat feeds are often utilized as valuable resources for building comprehensive and diverse training datasets for ML-based classifiers. This is also applicable to the malware domain, where there is a lack of empirical data to support malware labeling and family attribution. Thus, relying on labeled data provided by multiple independent malware detection engines. However, this reliance on external data sources introduces potential attack surfaces, since malicious actors may submit tampered samples to contaminate the training data/labels. Moreover, a practical backdoor attack to contaminate the dataset is by using the "clean-label" approach, where an attacker poisons the samples of the target class to embed specific patterns within the classifier, leading to the desired misclassification of non-target classes during the model inference. As a result, the characteristics of the target domain constrain the backdoor attack strategies and attack vector to adversarial samples and/or poisoned models [66].

Given these motivations and identified gaps, the target model explored in this paper aims to reflect the latest trends in the IoT malware classification field, and its effectiveness is verified through real-world scenarios to ensure that the attack observations are both practical and convincing. The target IoT malware family classification model utilizes the model-based feature extractors. Given the inherent interpretability challenges of these feature extractors, we developed a backdoor attack framework that includes a trigger generator called the bi-role source class-agnostic input-aware trigger generator (B-CTG) and an associated training procedure. The trigger generator is designed to manipulate samples directly within their problem space (i.e., the malware binary file), while the training procedure ensures that this manipulation effectively achieves the backdoor attack objectives. From the attackers' perspective, the design of B-CTG and its training procedure leverages knowledge of the target model, with the initial evaluation conducted under the white-box assumption as

the main evaluation. Subsequently, assuming the attacker has gray-box knowledge—where the attacker's selected proxy model differs from the target model but is trained on the same dataset—we test B-CTG's performance on other proven models in the transferred setting.

3.2 Preliminaries

3.2.1 IoT Malware Binary Format

To ensure that malware manipulation from the problem space does not impact its functionality, we introduce the format of IoT malware. IoT malware generally is a binary object in Executable and Linkable Format (ELF) for Linux-based systems [150]. The file includes the ELF header, program header, section header, segments, and sections, where segments and sections refer to the same body contents but are used for different purposes. If the file is examined from an executable perspective, the section header is optional, and the body content is mainly used as segments; by contrast, if the file is intended for the link, the program header is optional [142]. Segments can encompass multiple sections with potential overlap(s), while sections are typically independent. During ELF file execution, the ELF header, program header, and segments are sequentially loaded into memory. Additionally, interpreter segments are responsible for loading the necessary libraries into memory for further linking and the overall execution. This format and the roles of each component suggest that the malware binary file manipulation is feasible as long as the linkage between each component during execution remains intact and unaffected.

3.2.2 IoT Malware Classification Methods

IoT malware classification methods can be categorized into two main types: dynamic and static. Compared with dynamic ones, static methods primarily focus on analyzing malware files rather than behaviors, which makes them less reliant on complex dependencies

and enables faster processing. The classification result is mainly influenced by the feature extraction step. Many works proposed rule-based feature extraction methods. Ahmadi et al. [8] extract Haralick features and Local Binary Pattern features from malware grayscale images to achieve high accuracy by using the XGBoost classifier. Tian et al. [149] extract and select printable strings from Trojan and virus families and assess the classification result across various algorithms, including Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), and Naive Bayes. Similar research, such as [10, 122], introduces diverse rules for feature extraction and evaluates their effectiveness with various simple classifiers like SVM.

Apart from the classifiers trained with manually engineered features, many studies increasingly favor utilizing models such as Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Graph Neural Network (GNN) to automatically extract features due to low expertise knowledge demands and robust classification performance. This trend is particularly evident in the enhanced performance of ensemble learning and transfer learning. Dib et al. [55] introduced a multi-level approach that utilizes image and string submodels to extract features from raw files for the final classifier. Tien et al. [150] developed a malware analysis framework with combined feature pre-processing from ELF features and operation codes. Vasan et al. [154] leverage transfer learning to extract features from the raw malware files with fine-tuned VGG-16 [138] and achieve promising results on malware image classification. Their results demonstrated the effective classification performance of the lightweight classifiers when coupled with robust feature extractors, but it can remain challenging to pinpoint the specific binary features that impact the classification outcome.

3.2.3 Backdoor Attacks: Research Gap

Malware classification models typically consist of feature extractors and classifiers. The feature extractors, which obtain features from binary files, can be either rule-based or model-based. Furthermore, to have a meaningful attack, any modifications made to the features during backdoor attacks must be mapped back to the original binary while preserving the malware's functionality. Indeed, recent work that investigates backdoor attacks against ML malware classification models [134, 166, 99], primarily focuses on malware detection models that are operating within the feature space derived from manually crafted rules. Correspondingly, their attack methods are based on manipulating the sample in the feature space (i.e., rule-based extracted features), then leveraging reverse engineering to tamper the corresponding areas of the problem space (i.e., the malware binary files). Note that while the mapping process is more straightforward in rule-based methods, obtaining the rule-based features is expensive, requires domain expertise, and cannot be easily scaled/generalized to other domains. However, while model-based methods are more challenging to implement due to the inherent difficulty in interpretation, they are known for their robustness and lower demand for high-level domain expertise. Particularly, the effectiveness from ensemble learning [55, 153] and transfer learning [154], makes them a prominent focus in malware classification research. Therefore, in this paper, we focus on exploring the model-based methods to address the existing research gap in terms of backdoor attacks against ML models.

Additionally, we identified the lack of consideration from the attackers' viewpoint when analyzing recent work. Indeed, the discussed attack methods can broadly apply to different domains once the mapping step is removed. Correspondingly, their methods may not fully leverage the intricacies and validated insights specific to target models in certain domains, such as IoT malware, leading to insufficient exploration of attacks tailored to those fields. However, from the attacker's viewpoint, targeting a specific domain model requires

corresponding knowledge to select practical and effective models. Subsequently, attackers may leverage the insights derived from the model to craft their strategies, as the knowledge is crucial for executing effective attacks. To evaluate the effectiveness of their methods in real-world scenarios, attackers typically test their approaches under different knowledge assumptions during experiments. This study adopts this attacker-centric perspective, rigorously selecting target models, integrating model-validated knowledge into the attack method design, and thoroughly testing performance under both white-box and gray-box knowledge assumptions.

3.2.4 Attack Vectors and Input-model Co-optimization

Adversarial ML attacks generally involve two attack vectors: adversarial samples and the poisoned model. While they both share similar objectives, the adversarial samples are manipulated by the attacker to deceive the target model and evade detection. On the other hand, the poisoned model attacks rely on training ML models with tampered samples in order to influence the outcomes of the classification in the detection phase when presented with pre-defined inputs. Pang et al. [119] conducted the first systematic study about the correlation between them across various attack approaches, such as evasion attack, model poison attack, and backdoor attack. In the context of backdoor attacks, the attacker leverages some techniques to render the model sensitive to pre-defined trigger patterns, leading to the misclassification of trigger-embedded inputs during inference.

Drawing from this observation, the objective of these two attack vectors is defined as Eq. (1), where x denotes the adversarial version of the target sample x_0, and θ represents the parameters of the poisoned model updated from θ_0. Given a sample set Γ, the objective of the two attack vectors is to minimize the loss $\ell()$ of the poisoned model concerning the prediction for input x and the target label t_{x_0}. However, in practical situations, the perturbation of the target sample x_0 is often limited to certain conditions, such as avoiding

inappropriate feature perturbation that may disrupt the original functionality of the malware sample. Therefore, the selection of its adversarial version should be limited to a feasible set $\mathcal{F}_\epsilon(x_0)$. Similarly, model parameter perturbation is also constrained to a feasible set $\mathcal{F}_\delta(\theta_0)$, ensuring the minimum impact on the non-target samples.

$$min_{x\in\mathcal{F}_\epsilon(x_0),\theta\in\mathcal{F}_\delta(\theta_0)}\mathbb{E}_{x_0\in\Gamma}[\ell(x,t_{x_0};\theta)] \tag{1}$$

Based on [31], Eq. (1) is further transformed as Eq. (2), where $\lambda\ell_f(x)$ and $v\ell_s(\theta)$ represent the impact of the constraints $\mathcal{F}_\epsilon(x_0)$ and $\mathcal{F}_\delta(\theta_0)$, respectively.

$$min_{x,\theta}\ell(x;\theta)+\lambda\ell_f(x)+v\ell_s(\theta) \tag{2}$$

However, due to the mutual reliability between x and θ, Eq. (2) is still difficult to optimize. Pang et al. introduced further transformations and derived the final bi-optimization formulation, Eq. (3).

$$x_* = \mathrm{argmin}_x\,\ell(x;\theta_*)+\lambda\ell_f(x) \tag{3a}$$

$$\theta_* = \mathrm{argmin}_\theta\,\ell(x_*;\theta)+v\ell_s(\theta) \tag{3b}$$

Based on this formula, the attack objective can be optimized by alternatively updating the adversarial sample x and the poisoned model θ in each iteration. Through empirical and theoretical analyses, Pang et al. finally demonstrate the effectiveness of the input-model co-optimization (IMC) framework.

The scenario outlined in [119] assumes that the attacker has direct control over both attack vectors, enabling the evaluation of final attack performance using the resulting poisoned model and adversarial samples derived from the given formula. However, in the malware model training pipeline, attackers typically have limited and indirect control over

the model. This fundamental difference in the attacker's capabilities means that the poisoned model generated by the aforementioned training procedure cannot be regarded as the final poisoned model. Consequently, the training procedure described above is not directly applicable to malware model attacks. Despite this, the construction of both attack vectors remains interdependent. To address this limitation, we have redefined the training procedure to obtain the two attack vectors.

3.3 Related Work

A number of recent studies have focused on the backdoor attack aimed at malware classifiers [134, 166, 99]. In this section, we will discuss the details of these works in greater depth and provide a detailed comparison with our work. It is worth noting that their target classifiers are all trained on the datasets with rule-based extracted features, e.g., the EMBER dataset [13] and the Drebin dataset [21]. The EMBER dataset encompasses eight features comprising both parsed values and format-agnostic histograms. Drebin dataset extracts eight groups of distinct features from the Android manifest file and disassembled code. The well-defined feature extraction rules facilitate the assessment of each feature's impact on the classification result. This, in turn, simplifies the manipulation of the corresponding binary regions.

Severi et al. [134] are among the first to study such work. They utilize the SHAP [103] value calculated from the optimal proxy model to identify each feature-value pair's importance and design multiple rules to select certain pairs as watermarks to poison samples. The paper proves the method is model-agnostic by testing the attack's performance under the transferred setting, where the proxy model differed from the target model. In addition to the EMBER dataset, this work also performed extra evaluations on PDF files and the Drebin dataset. Yang et al. [166] proposed a jigsaw puzzle method that focuses on learning

a trigger that complements the latent patterns present in the malware samples, so the backdoor is activated only when the trigger and latent pattern are combined in a sample, akin to completing a jigsaw puzzle. Their primary evaluation is conducted on the AndroZoo dataset [11] extracted by Drebin rules, particularly analyzing their method's stealthiness due to unique objectives. The triggers generated by these methods are all fixed. This is partly due to the constraints posed by the feature representation that their target model utilized. In addition, they all assume the practical way to poison the malware classifiers is through the clean-labels.

By contrast, Li et al. [99] designed a corrupted-label approach. Due to the attacker's inability to control the labeling step during the training phase, they introduce a label-reversing step apart from trigger selection and injection steps similar to other works. This step iteratively perturbs the feature vectors of the poisoned samples based on the result from a substitute model until the desired conditions are met. Their experiments evaluated the method on multiple models, including Drebin, DroidCat, MaMaDroid, and DroidAPIMiner. Notably, although the features used in each model from the Drebin dataset differed, their extraction methods were all rule-based, such as API calls, API parameters, package-level information, etc. After perturbing each sample, they identify manipulable features and modify the binary file using reverse engineering.

D'Onghia et al. [56] proposed a dynamic trigger generation method, but their objective falls under the category of targeted poisoning attacks [114], with distinct objectives compared to ours. The key distinction lies in the fact that the study [56] anticipates the effective performance of their poisoned model on non-target samples, only aiming for misclassification in target samples. The target and non-target samples are presented in their original, unperturbed forms. In contrast, this study aims to have the compromised model correctly classify samples without perturbations but misclassify their manipulated versions. Furthermore, these methods target the model classifying malware and benign samples instead of

Table 1: Comparison Among Methods of Backdoor Attack Against Malware Analysis Models

Related Work	Objectives / Poison Manner	Poison Scenario (Dataset / Target Model)	Methodology / Key Notes
Severi *et al.* [134]	Backdoor Attack (Clean Label)	Malware Detection Ember [13], Drebin [21] Rule-based: EmberNN; LightGBM [13]	Static trigger; SHAP-based trigger selection; Reverse-engineering mapping
Yang *et al.* [166]	Targeted Backdoor Attack (Clean Label)	Malware Detection AndroZoo [11] Rule-based: MLP	Static trigger; Binary trigger optimization; Reverse-engineering mapping
Li *et al.* [99]	Backdoor Attack (Corrupted Label)	Malware Detection Drebin [21] (malware) + PlayDrone [155] (goodware) Rule-based: Drebin [21], DroidAPIMiner [7], etc.	Static trigger; Genetic algorithm + binary search to find trigger; Label reversing algorithm; Reverse-engineering mapping
D'Onghia *et al.* [56]	Targeted Poisoning (Corrupted + Clean Label)	Malware Detection VirusShare [157] (malware) + Chocolatey (goodware) Model-based: MalConv [127], AvastNet [93]	Dynamic trigger; Max-pool-based latent space trigger generation; Greedy algorithm to minimize loss and generate bytes
This work	Backdoor Attack (Clean Label)	Malware Family Classification IoTPoT and X-PoT [118] Model-based: Multi-level Model [55], SVM, RF	Dynamic trigger; Alternative optimization to train trigger generator; Input-aware trigger generation applied to raw samples

malware family classification.

Table 1 illustrates the distinctions between the above-related studies and this work, especially the objectives, poison manner, and poison scenario, which limited the rationality of the experimental comparison [38] due to their regulations to the attack technique design. Therefore, we mainly discussed recent related works and their distinctions qualitatively.

3.4 Problem Statement and Threat Model

The entire training process of the IoT malware classification model encompasses data collection, labeling, and training steps [38]. Given the attackers' limited capacity, they are typically involved in the data collection step. The capability of model designers and attackers can be explained as follows. From the model designer's perspective, the goal is to develop a model that performs well during testing after training on a broad range of collected samples. From the attacker's perspective, the goal is to embed a trigger pattern into the designer's model via poisoned samples, enabling them to control the compromised

model. Additionally, to minimize disruption to the model's performance and avoid raising the designer's suspicion, attackers also aim for the poisoned model to perform well on clean datasets. Therefore, as outlined in previous work [100, 74], the process and objectives of the backdoor attack targeting malware classifiers can be elaborated below.

When the whole training pipeline is clean, the model designer will obtain a clean model F through training on the clean dataset X. When the pipeline is subject to poisoning, the attackers will implement their hidden control purpose by tampering with some samples X_{SP}^t through their designed method, where $X_{SP}^t \subseteq X$, and release their poisoned version X_P^t. To be practical, X_{SP}^t are selected from target class y^t (clean-label restriction). When the model designer trains their models on the poisoned dataset, the generated poisoned model F_b will not only implement the **Main task** as F does but the **Attacker-chosen sub-task**, i.e., the attacked samples X_A tampered from their clean version X_{SA} of classes y will be misclassified into y^t. Additionally, to avoid attackers' suspicions, attackers expect lower **Detectable ability** of X_P^t.

Considering the IoT malware classification research trend, the model we target leverages model-based feature extractors. Therefore, we assume F, F_b exclusively represent classifier components for better clarity. Let $\mathcal{F}$ denote the set of feature extraction models, where $\mathcal{F} = \{F_0^e, F_1^e, \ldots, F_n^e\}$ and each F_i^e, with $i = 0, \ldots, n$, is a distinct feature extractor. Additionally, inputs represent the raw malware files. Then, the main task and sub-task can be verified as follows:

- **Main task**: measure if the poisoned model behaves normally as its clean counterpart model for clean inputs, making it impossible to distinguish the poisoned model from the clean model by checking testing performance:

$$F(\mathcal{F}(X)) = F_b(\mathcal{F}(X))$$

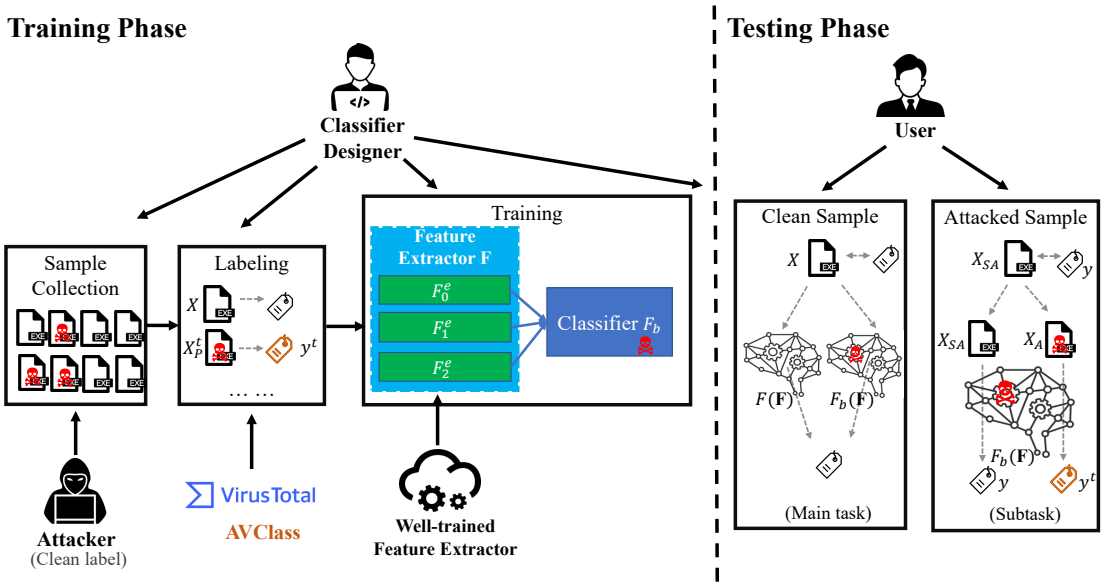


Figure 4: Training pipeline and poison results evaluation.

- **Attacker-chosen sub-task**: indicates whether backdoor attackers can successfully achieve their malicious purposes when the hidden trigger is present in the target input:

$$F_b(\mathcal{F}(X_{SA})) = y; F_b(\mathcal{F}(X_A)) = y^t; y^t \neq y$$

- **Detectable ability**: evaluate whether the poisoned sample is detectable:

$$F_b(\mathcal{F}(X_P^t)) = y^t$$

The entire training pipeline and attack objectives described above are depicted in Figure 4.

Based on the attack surface and objectives discussed above, the problems can be summarized as follows:

- *Q1) Given the attack scenario, which model could be better explored, considering that the approach proposed in previous works [134, 166, 99] relied on specific proxy*

models to generate triggers?

- *Q2) How to design a trigger generation method in a clean-label manner, ensuring that the poisoned model achieves these objectives?*

3.5 Solution Formulation

Given the issues outlined in the attack scenario and the objectives elucidated in Section 3.4, we design a backdoor attack framework targeting IoT malware classification models. To align with the attackers' viewpoint, selecting the target model within the IoT malware classification domain will be our first objective. Subsequently, drawing inspiration from the selected model, we craft the attack framework.

3.5.1 Model Selection - Q1

The target model (i.e., IoT malware classification models) should represent the state-of-the-art in the field and be a proven solution in practical scenarios. Ideally, the model should utilize model-based feature extractors to derive sample features and incorporate feature enhancement techniques. The feature representation should be widely studied, such as those listed in Ngo et al.[112], which reviews recent literature and highlights commonly used feature representations such as operation codes[76, 22], strings [151, 10], file headers [135], and grayscale images [111, 143]. During the experiments, the model effectiveness should be validated under the setting considering the malware landscape challenges [20], including class imbalance and sample bias. With these considerations, we select the model designed in [55]. The selection reasons are summarized as follows: 1) the feature extractors are model-based and leverage the typically explored feature representation [112]: image and string; 2) the feature extractors are enhanced by incorporating techniques including ensemble learning and transfer learning; 3) the experiments use realistic and recent dataset; 4)

The dataset is imbalance and includes four common malware families, i.e., Mirai, Gafgyt, Tsunami, and Dofloo; and 5) The model is tested on unseen malware samples and demonstrates strong generalization across variants and new samples. These characters also prompt the attack observations to reflect real-world conditions better.

3.5.2 Bi-role Class-agnostic Input-aware Trigger Generator (B-CTG) - Q2

We design a bi-role source class-agnostic input-aware trigger generator (B-CTG) and, inspired by the IMC framework [119], we design a procedure for training B-CTG to implement poison and attack purposes. If the input is intended to be a poisoned sample, its trigger-embedded version will be utilized to poison the model. If it is meant to be an attacked sample (i.e., adversarial sample mentioned in Section 3.2.4), its trigger-embedded version will be employed to deceive the target model (Bi-role). The target model introduced in Section 3.5.1 is evaluated on an imbalanced dataset with four families. In this work, we designate Mirai, the majority class, as the target label, aiming for B-CTG to misclassify trigger-embedded samples from the other three classes—Gafgyt, Tsunami, and Dofloo—into the Mirai class (class-agnostic).

To effectively assess the attack performance, the assumption of the attacker's knowledge is gradually restricted during the method design and evaluation. In similar works, [134, 166] initially operate under the assumption with white knowledge, i.e., the target model is equivalent to the surrogate model, and the designers' dataset is known to attackers, which corresponds to the case of Figure 5 Main Evaluation. Then, the attacker's knowledge is confined to gray knowledge, limiting their ability to thoroughly evaluate the poisoned method's effectiveness. This knowledge limitation can usually be categorized into two main aspects: Dataset Restriction, restricting access to dataset knowledge, and Transferred Setting, where the target model is assumed to differ from the proxy model.

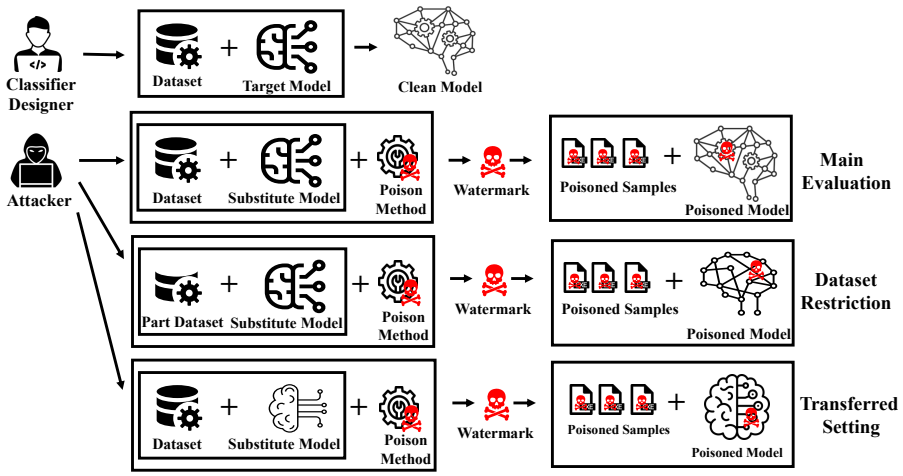


Figure 5: Attacker knowledge scenario

The inconsistency between the dataset and model will cause the attacker's predictions to differ from the actual attack performance, as the resulting models might vary, as illustrated in Figure 5. Similarly, we would first design (Section 3.5.2, Section 3.5.2) and evaluate under the white knowledge assumption (Section 3.6.3). Then, we test under the transferred setting by observing the model and feature knowledge importance to the attackers (Section 3.6.3).

B-CTG Design

The samples (X) are raw malware binary files in ELF format. To preserve their functionality, we analyze their execution process to explore appropriate methods $A(.)$ for manipulating the samples. The malware execution relies on the program header, which loads the segments into the memory during execution. These segments contain malware functions and necessary data; their invocation depends on their specific addresses. If we insert contents at improper locations, it may alter the addresses of certain execution elements, thus disrupting their functionality. Therefore, we need to handle the process with caution to

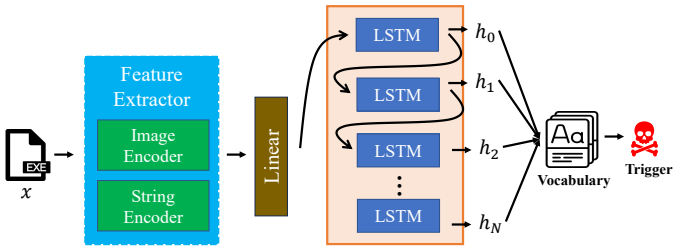


Figure 6: Trigger generator

maintain the intended behavior of the malware. For this purpose, we chose a straightforward method to manipulate the samples: appending the trigger at the end of the file. This way, we can minimize the chances of disrupting the malware's execution while achieving our objectives.

The sample manipulation space requires the appended trigger to be a binary sequence. To ensure the proxy model gets influenced by the trigger-embedded sample, the content expressed by the binary sequence should be relevant to the feature representation utilized by the feature extractors. The target model introduced in Section 3.5.1 employs ensemble methods, including image and string representation, to extract features. The image representation converts each file byte into an image pixel value, while the string method extracts printable strings from the binary. The string-based method demands a higher specificity in terms of the content expressed by the binary. In other words, if the binary can be converted to a string, it will also be part of the image representation. However, the reverse is not true: not all content that can be represented as an image pixel value can be expressed as a printable string. Due to this distinction, B-CTG utilizes the string as the trigger unit to ensure compatibility with the model's feature representation. To leverage the knowledge from the target model, the generator is set to use the same feature extractors.

Inspired by the automatic image captioning [80], we designed the trigger generator as Figure 6. N is the number of string units in the trigger generated for sample x. Each string unit maintains a contextual relationship with the sample's features and the strings

generated before. Consequently, the information retained by hidden states in early LSTM units serves as input for the subsequent LSTM units, enabling the generation of the string at its corresponding position. Each LSTM output h_i can be transformed into a position indexing vector. Then, the matrix constructed by all these outputs is used to index the vocabulary matrix and extract the corresponding strings to form the final trigger for x.

Training Procedure of B-CTG

The objective of Algorithm 1 is to train the B-CTG to implement its poisoning and attacking purposes. The entire training dataset is divided into two parts: the majority class Mirai (X_m, Y_m) and the rest (X_{rest}, Y_{rest}) containing the three minority classes. The proxy model is denoted as $c : f_c(f_{cf}(x; \theta_{cf}); \theta_c)$, where f_{cf} represents feature extractors with well-trained parameter θ_{cf}, and f_c is the classifier part with parameter θ_c. The trigger generator is defined as $g : f_g(f_{gf}(x; \theta_{gf}); \theta_g)$, where f_{gf} is the feature extractor with parameter θ_{gf}, and f_g is the trigger generator with parameter θ_g . The features extracted by f_{gf} serve as input to f_g, enabling the implementation of input-aware trigger generation (input-aware).

Given the distinctions explained in Section 3.2.4, the direct application of the IMC framework is not feasible for our purposes. In our scenario, the attacker achieves the objectives of model poisoning and backdoor attacks through the created poisoned samples. This raises the question: *how can one ensure that the model perturbed by the poisoned sample misclassifies the attacked samples?* This question highlights the need for a link between the construction of poisoned and attacked samples, and the objective presented in this question emphasizes the model's critical role in achieving this connection. Since the sample construction is carried out through B-CTG, the training of B-CTG is inseparable from the model. B-CTG's connection to the model varies with the purpose of the constructed sample. The poisoned sample generated by B-CTG is used for model perturbation, while the

Algorithm 1 Procedure of Training B-CTG

Input: (X_{train}, Y_{train}): entire training dataset;
(X_m, Y_m): majority class dataset;
(X_{rest}, Y_{rest}): rest classes dataset;
$f_c(f_{cf}(x;\theta_{cf});\theta_c)$: proxy model;
$f_g(f_{gf}(x;\theta_{gf});\theta_g)$: trigger generator;
α: poisoning ratio
Output: $f_g(f_{gf}(x;\theta_{gf});\theta_g)$
$X^{SP} \leftarrow$ random(X_m, α)
Initialize parameters $\theta_{cf}, \theta_c, \theta_{gf}, \theta_g$
while not converged **do**
  Split X_{train} into batches $\{X_{tr}^0, X_{tr}^1, \ldots, X_{tr}^N\}$
  for $n \leftarrow 0$ to N **do**
    Extract poison and attack samples:
    $X_{SP}^n \leftarrow X^{SP} \cap X_{tr}^n$, POISON_SET$^n = (X_{SP}^n, Y_{SP}^n)$
    $X_{rest}^n \leftarrow X_{tr}^n \setminus X_{SP}^n$, REST_SET$^n = (X_{rest}^n, Y_{rest}^n)$
    $X_{SA}^n \leftarrow X_{tr}^n \setminus (X_{tr}^n \cap X_m)$, ATTACK_SET$^n = (X_{SA}^n, Y_{SA}^n)$
    Update trigger generator:
    $X_P^n \leftarrow A(X_{SP}^n, f_g(f_{gf}(X_{SP}^n;\theta_{gf});\theta_g))$
    $X_A^n \leftarrow A(X_{SA}^n, f_g(f_{gf}(X_{SA}^n;\theta_{gf});\theta_g))$
    $(X_{AP}^n, Y_{SAP}^n) \leftarrow (X_A^n \cup X_P^n, Y_{SA}^n \cup Y_{SP}^n)$
    $Y_{AP}^n \leftarrow f_c(f_{cf}(X_{AP}^n;\theta_{cf});\theta_c)$
    $(\theta_{gf}^*, \theta_g^*) \leftarrow \arg\min_{\theta_{gf},\theta_g} \ell(Y_{AP}^n, Y_{SAP}^n;\theta_{cf},\theta_c) + \lambda\ell(\|f_{cf}(X_{SP}^n) - f_{cf}(X_P^n)\|)$
    $\theta_{gf} \leftarrow \theta_{gf}^*,\ \theta_g \leftarrow \theta_g^*$
    Update classifier:
    $X_P^n \leftarrow A(X_{SP}^n, f_g(f_{gf}(X_{SP}^n;\theta_{gf});\theta_g))$
    $(X_{PRest}^n, Y_{SPRest}^n) \leftarrow (X_P^n \cup X_{rest}^n, Y_{SP}^n \cup Y_{rest}^n)$
    $Y_{PRest}^n \leftarrow f_c(f_{cf}(X_{PRest}^n;\theta_{cf});\theta_c)$
    $\theta_c^* \leftarrow \arg\min_{\theta_c} \ell(Y_{PRest}^n, Y_{SPRest}^n;\theta_{gf},\theta_g)$
    $\theta_c \leftarrow \theta_c^*$
  end for
end while
return $f_g(f_{gf}(x;\theta_{gf});\theta_g)$

attacked sample is for misclassification. In other words, the model manipulation is connected with the training of B-CTG through the poisoned sample, and B-CTG also needs to learn how to construct attacked samples based on the poisoned model.

The poisoned sample generated by B-CTG is used to poison the designer's model. The randomly constructed sample may also perturb the model's decision boundary, but poor

perturbation may lead to degraded performance on clean samples and raise suspicion about poisoned samples. This raises the question: *how can one construct the poisoned sample effectively to achieve the desired objectives without adversely affecting the model's performance on clean data and ensure the poisoned sample's stealthiness?* To avoid the above poor performance, the current poisoned sample needs an indicator to show their poisoning influence, and this indicator can be a guide for their reconstruction. The indicator is from the poisoned model, which shows the reliance between the model and poisoned samples. To implement this, B-CTG gets optimized by the loss[1] ℓ_f calculated from the sample perturbation degree to improve sample stealth, while the model uses the loss[2] ℓ_s evaluated on the clean sample to update its parameters to show current poisoning performance. The above two losses can be regarded as restrictions to limit B-CTG and model training direction. Then, combined with the same objective: classify the poisoned sample correctly by the poisoned model, i.e., loss[2] ℓ, B-CTG, and model can be ensured to be properly trained. Due to the above reasons, we redesign Eq (3) as Eq (4), where x^{SP} represents the selected sample to be poisoned. $A(.)$ is the concatenation method for one sample and its trigger. θ_g, θ_c represent the related updated parameter in g and c respectively.

$$\theta_g^* = \text{argmin}_{\theta_g} \ell(A(x^{SP}, g_{\theta_g}(x^{SP})); \theta_c^*) + \lambda \ell_f(A(x^{SP}, g_{\theta_g}(x^{SP}))) \tag{4a}$$

$$\theta_c^* = \text{argmin}_{\theta_c} \ell(A(x^{SP}, g_{\theta_g^*}(x^{SP})); \theta_c) + v\ell_s(\theta_c) \tag{4b}$$

The second role of B-CTG is to generate attacked samples, so it requires updates based on the trained poisoned model. This process can be implemented by Eq (5), where x^{SA} represents the sample to be attacked. This updating process ensures that B-CTG generates

[1]Mean Square loss, measure the perturbation degree

[2]Cross-Entropy loss, the loss concerning the predicted and target label

triggers that effectively deceive the model for the specific attack objective.

$$\theta_g^* = \text{argmin}_{\theta_g}\, \ell(A(x^{SA}, g_{\theta_g}(x^{SA})); \theta_c^*) \tag{5}$$

The updates from Eq (5) will influence the value of θ_g^* as shown in (Eq 4a), which in turn affects θ_c according to Eq (4b). Hence, the entire process can be summarized as Eq (6).

$$\begin{aligned} \theta_g^* &= \text{argmin}_{\theta_g}\, \ell(A(x^{SP}, g_{\theta_g}(x^{SP})); \theta_c^*) && \text{(6a)} \\ &\quad + \beta\ell(A(x^{SA}, g_{\theta_g}(x^{SA})); \theta_c^*) + \lambda\ell_f(A(x^{SP}, g_{\theta_g}(x^{SP}))) \\ \theta_c^* &= \text{argmin}_{\theta_c}\, \ell(A(x^{SP}, g_{\theta_g^*}(x^{SP})); \theta_c) + v\ell_s(\theta_c) && \text{(6b)} \end{aligned}$$

The poisoned sample set is decided by the sample selection and poisoned sample number, which subsequently influences the poisoned model performance (line 1). In Algorithm 1, we use X^{SP} and α to denote the selected poisoned sample set and poisoning ratio. For the proxy model parameters, the feature extractor f_{cf}'s parameter θ_{cf} is initialized and fixed with the pre-trained model parameter values. The parameters θ_c, θ_{gf} and θ_g are initialized using the "Xavier" approach [69].

Based on Eq (6), the updated objective (i.e., Eq (6a)) of the generator g is twofold: first, to deceive the model into misclassifying the attack samples X_A^n, and second, to perturb the model through the poison sample X_P^n. The corresponding implementation is shown in lines 12-17, and these objectives are captured by the loss functions $\ell(Y_{AP}^n, Y_{SAP}^n; \theta_{cf}, \theta_c)$, and $\ell(\|f_{cf}(X_{SP}^n) - f_{cf}(X_P^n)\|)$ serves for ℓ_f. Compared with the raw file, the representation from the feature extractors exhibits significantly reduced noise and reveals more underlying information. Therefore, instead of evaluating the modification degree in input space, we force the sample poisoned version to lie near itself in the extracted feature space. For the

model c (i.e., Eq (6b)), its update procedure is shown in lines 18-22, and the objective is achieved by minimizing the loss $\ell(Y^n_{PRest}, Y^n_{SPRest}; \theta_{gf}, \theta_g)$.

3.6 Experimental Evaluation

Combined with the model designers' training process illustrated in Figure 4, the entire poisoning procedure is as follows:

1. Train B-CTG using Algorithm 1 and the proxy model with the obtained dataset.
2. Generate the poisoned sample by B-CTG.
3. Mimic the designer's process by training their model using the dataset consisting of poisoned samples.
4. Generate the attacked sample using B-CTG to attack the designer's model.

The poisoning experiments also follow the steps above. Section 3.6.1 shows the clean model performance. Section 3.6.2 shows the metrics to evaluate attack performance. Section 3.6.3 shows the attack observations by following the poisoning procedure. Section 3.6.4 shows the mitigation effectiveness against the attack methods.

3.6.1 IoT Malware Dataset and Clean Model Performance

To ensure the experiment observations closely reflect real-world scenarios, yield practical conclusions, and repeat the target model performance of [55], we utilize the same and recent dataset resources: IoTPoT and X-PoT [118]. Then, we follow the steps shown in Figure 7 to process the dataset. The details are as follows. This dataset comprises 142,089 IoT malware samples, with 93% of them being in ELF format. However, since the type of these samples is not explicitly indicated in the dataset, we employ LIEF [148] to examine

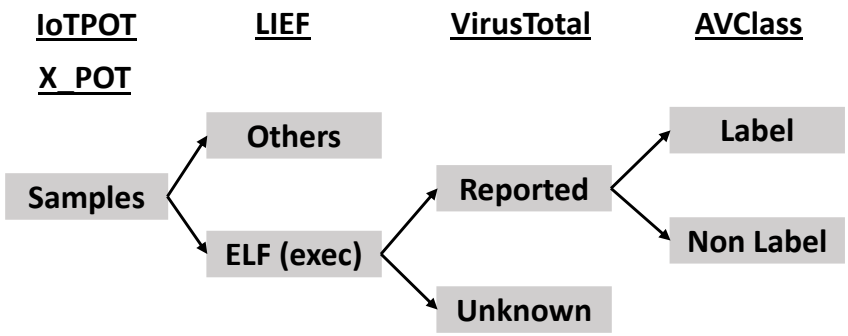


Figure 7: Stages of dataset processing

the samples with ELF format. Subsequently, we use VirusTotal to extract the samples with reported documents and then employ AVClass to label the samples based on their reports [133].

The resulting dataset consists of 57,471 samples representing 33 different malware families. Among them, four families, namely Mirai, Gafgyt, Tsunami, and Dofloo, make up 99.7% of the dataset, which are the only families with sample sizes exceeding 50. Specifically, the number of samples from each of the four malware families is 51,781 (Mirai), 5,226 (Gafgyt), 202 (Tsunami), and 59 (Dofloo), respectively, which represents a significant imbalance, especially with the majority being the Mirai class. To match the adopted target model, the final dataset only consists of these four classes.

The multi-model architecture in the explored model [55] shows its usage of the ensemble learning paradigm, while its experiments separating the training of the feature extractors, i.e., image and string components, and classifier show its leverage of the transfer learning paradigm. Following the same experiment steps, we train their classifier components with the entire dataset to get robust feature extractors first. Then, to test its effectiveness, we use 20% randomly selected samples from each malware class to train the classifier part. To comprehensively assess the final model's performance, we evaluate it using two datasets: 1) the selected 20% subset of the data and 2) the entire dataset. Table 2 presents the testing results. The test accuracy (ACC) and F1 score reflect the performance

metrics achieved on the 20% selected dataset, whereas the entire-test F1 score represents the model's performance across the complete dataset. The results demonstrate that despite training the classifier on a small dataset, it exhibits improved performance when tested on the entire dataset.

Table 2: Detection performance of the clean model [55] and its sub-components.

Classifier Model	test ACC	F1 score	entire-test F1 score
Image extractor	0.983	0.945	0.945
String extractor	0.964	0.963	0.924
Multi-ModalClassifier	0.987	0.972	0.958

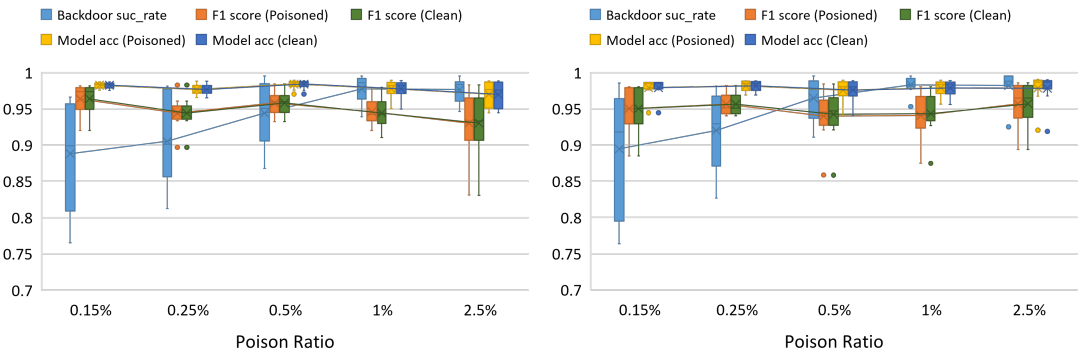
3.6.2 Attack Evaluation Metrics

The evaluation metrics for the attack approach primarily stem from the objective outlined in Section 3.4. Combined with the benchmark mentioned in [100, 66, 74, 134, 166, 99], the metrics can be summarized as follows: 1) model testing performance on the clean and poisoned dataset (main task): F1 score and model accuracy; 2) backdoor attack success rate (subtask); 3) detectable ability to the poisoned sample.

3.6.3 Attack Performance

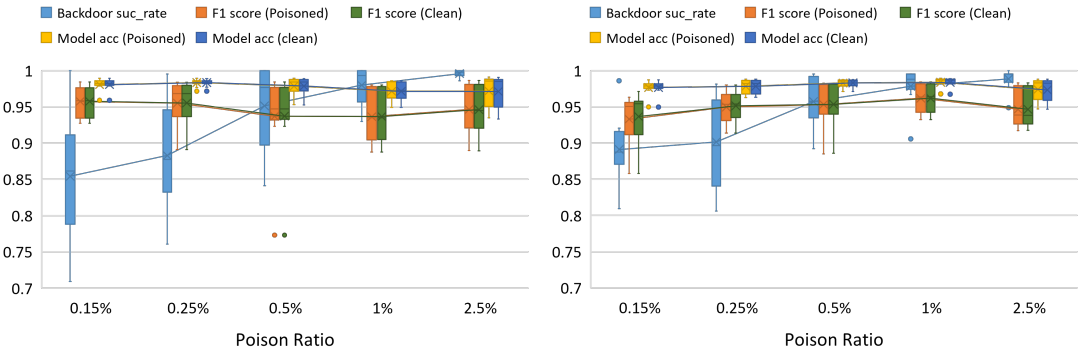
Main Evaluation

This section shows the attack performance evaluation under the white-box knowledge assumption. The entire poisoning procedure encompasses several variables that can potentially impact the efficacy of the poisoning strategy. During step 1, N and α play significant roles in the design and training of B-CTG. In step 2, various poison ratios can be employed to assess the model and attack performance. For a fixed poison ratio, the influence of variations in the composition of the poisoned sample set can also be explored. Each chosen set represents a potential dataset that a model designer might encounter. To explore the impact



(a) $N = 400, \alpha = 5\%$ (b) $N = 400, \alpha = 30\%$

Figure 8: Attack performance of main evaluation ($N = 400$)



(a) $N = 1200, \alpha = 5\%$ (b) $N = 1200, \alpha = 30\%$

Figure 9: Attack performance of main evaluation ($N = 1200$)

of these variables on the poisoning performance, we test the entire threat procedure with two sets for $\alpha = 5\%, 30\%$ and $N = 400, 1200$ and five poison ratios. We plot the attack performance distribution for each poison ratio over 10 potential datasets that designers may get. The attack evaluation results of these different settings are shown in Figures 8 and 9.

Every graph depicts how the poisoning performance changes with varying poison ratios under consistent values of α and N. As the poison ratio increases, the rate of successful attacks also increases, and their distribution becomes more concentrated. These distribution patterns suggest that attack performance stabilizes with higher poison ratios. When assessing the poisoned model's performance on the primary task, it is noticeable that these

models are not significantly different from the clean model, regardless of whether the testing dataset is poisoned or clean, which also indicates that the designers are unlikely to raise suspicion whether they conduct tests solely on the poisoned dataset or new samples. The trends of these main task metrics are not as clear as the attack success rate. Still, from observation, to maintain stability in model performance, the poison ratio is better to be neither excessively high nor excessively low. This is because when the poison ratio falls within the middle range, the model evaluation results tend to exhibit higher concentration than those from very low or high ratios.

Different configurations of α and N lead to distinct generators. When training B-CTG using the same trigger size N but different α values, as demonstrated in Figure 8a vs 8b, or Figure 9a vs 9b, it becomes apparent that a higher α leads to an improved generator. This enhancement can be observed in the comparison results at a fixed poison ratio. For instance, when $\alpha = 30\%$, the attack performance becomes more concentrated at nearly any given poison ratio. The model's performance is predominantly influenced by the poisoned sample. However, drawing a clear conclusion is challenging due to the unclear trend. Still, by comparing the outcomes of Figure 8a with 8b, or Figure 9a with 9b, when attack performance becomes stable, we observe that the model's behavior becomes more predictable, particularly the accuracy, when poisoned with the sample generated by B-CTG trained with a high α.

Unlike the previously mentioned α-based comparison, the observable distribution trends of attack performance are not as evident when contrasting generators trained with the same α but varying N values. Nonetheless, the attack performance does exhibit an increase across various poison ratios. Additionally, with a larger N value, the attack performance demonstrates heightened sensitivity to changes in the poison ratio. Notably, these insights derived from both α-based, and N-based comparisons are particularly prominent when

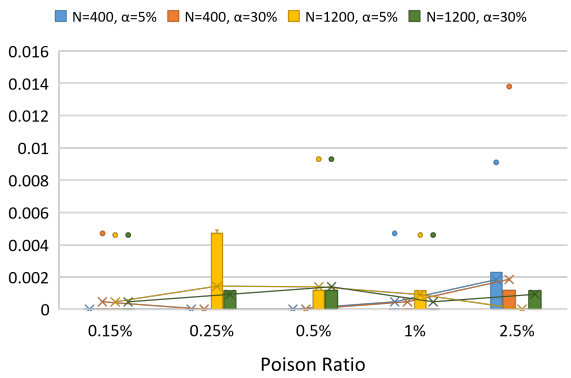


Figure 10: Detectable ability

poison ratios exceed 0.15%, suggesting that lower poison ratios significantly influence performance consistency. Furthermore, when contrasting attack performance shifts across different α values within varying N contexts, more pronounced distribution alterations manifest under $N = 1200$ compared to $N = 400$. Here, N denotes the trigger size, representing the number of strings appended to the samples. This distinction highlights that higher N values render the generator more responsive to changes in α. Moreover, the average attack performance of the α-based comparison shows a consistent trend, which is not the case in the N-based comparison. For instance, when considering $N = 1200$, the attack performance trend for $\alpha = 5\%$ closely resembles that of $\alpha = 30\%$.

In summary, to balance the model and poison performance, the poison ratio used by the attacker should be neither too high nor too low. The trigger size affects the performance, particularly in terms of elevating attack performance, although it does not significantly address the dispersion in the distribution. Based on the comparison related to α, this dispersion issue could be attainable by increasing α values. Nonetheless, as the trigger size increases, the sensitivity of the distribution issue to higher α values becomes more pronounced.

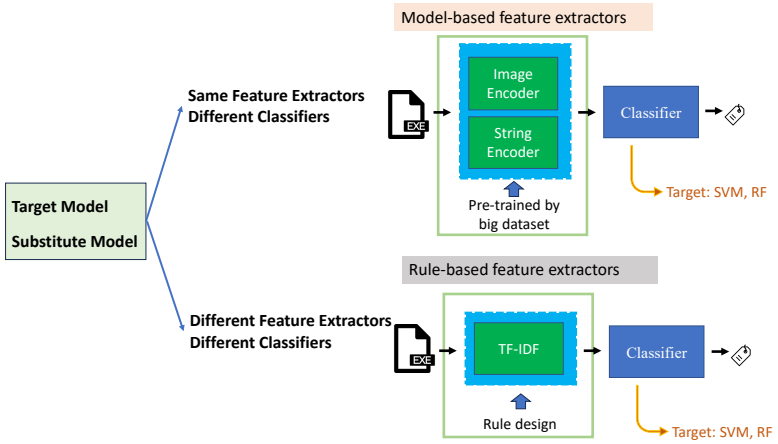


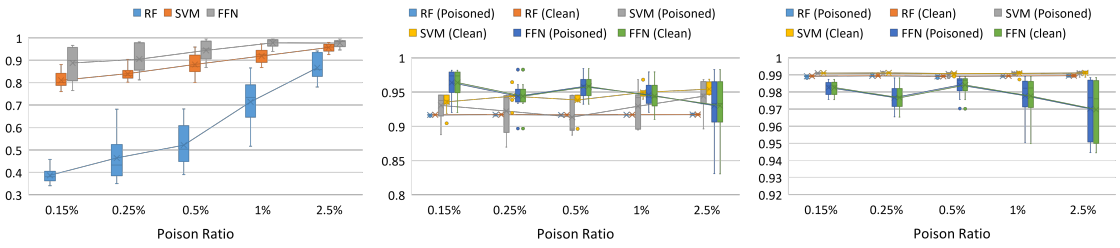
Figure 11: Transferred setting experiment description

Figure 10 illustrates the detectable ability associated with these four $N - \alpha$ pair configurations across five different poison ratios. The average detection rate across all configurations remains below an acceptable rate, i.e., 0.002; however, with an increase in the poison ratio, the likelihood of high detectable ability occurring becomes higher. This observation aligns with the intuition that a higher poison ratio will increase exposure to the poisoned sample.

Transferred Setting

This section aims to assess the effectiveness of the attack situation where the substitute model differs from the designer's model. First, we investigate models that share identical feature extractors but use distinct classifier components. Second, we evaluate the performance when the model is trained using distinct feature representations. These two experiment processes are illustrated by Figure 11. In the first setting, features are extracted using the same feature extractors as in the main evaluation, but different target classifiers, namely SVM and RF, are employed. In the second setting, features are extracted using Term Frequency-Inverse Document Frequency (TF-IDF), where samples are represented as string sequences only, and the target classifiers include SVM and RF classifiers.

This evaluation aims to examine how effectively the attack performance can be transferred across various types of target models. Rather than selecting models based on the same theory category, we opt for models derived from different theoretical foundations. In Section 3.6.3, the target model utilizes a Feedforward Neural Network (FFN) as its classifier. In this context, we evaluate the attack performance of the trained B-CTG on a model with similar feature extractors but different types of classifiers, namely SVM and RF. These three techniques, FFN, SVM, and RF, are rooted in diverse theoretical frameworks [115]. FFN represents a neural network architecture capable of learning intricate hierarchical features and capturing complex patterns, albeit prone to overfitting. SVM operates as a kernel method, striving to identify the hyperplane with the maximum margin between classes. RF, on the other hand, is an ensemble learning technique that emphasizes feature importance. Based on the observations from Figures 8 and 9, it is evident that the B-CTG trained under the $N = 400, \alpha = 5\%$ setting exhibits the least favorable performance among the four settings. We focus on evaluating B-CTG's attack performance on models with SVM and RF as the classifiers, which achieves an F1 score of 0.935 and an accuracy of 0.9895 (SVM-based), and an F1 score of 0.9171 and an accuracy of 0.9895 (RF-based), respectively. The FNN, SVM, and RF hyperparameters are shown in Tabel 3. Figure 12 provides an overview of attack performance on these models.



(a) Attack success rate (b) F1 score (c) Model accuracy

Figure 12: Comparison of different classifiers.

In Figure 12a, the upward trend of attack success rate along with the increasing poison ratio remains evident, but changes in distribution are less pronounced for SVM-based and

Model	Hyperparameters	Values
FNN	hidden layers	256, 128, 16
	dropout rate	0.2, 0.58
	optimizer	Adam
	lr	0.0005
	batch size	32
	maximum epochs	100
SVM	kernel	poly
	d	3
	γ	scale
	r	0
	C	1
RF	number of trees	100
	criterion	gini
	min sample split	2

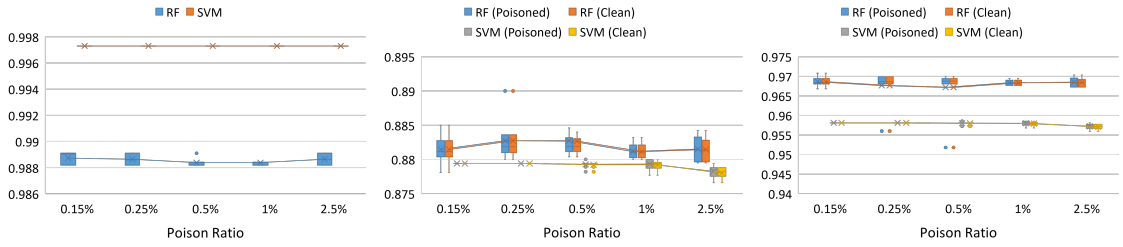
Table 3: Hyperparameters for FFN, SVM, and RF models

RF-based models. Furthermore, it is notable that the magnitude of the increase is notably more pronounced for the RF-based model compared to the other two, indicating the RF-based model's heightened sensitivity to changes in the poison ratio. While a noticeable distinction exists between RF and the other two classifiers at the poison ratio 0.15%, this disparity gradually diminishes as the poison ratio increases, and it becomes exceedingly marginal as the poison ratio reaches 2.5%. Figure 12b shows the F1 score of these three tampered models when tested on the poisoned and clean testing dataset. Compared with the FNN-based and SVM-based models, the RF-based model manifests consistent performance across different datasets and poison ratios.

Compared with the FNN-based and RF-based models, the model with the SVM classifier is much more sensitive to the poisoning state of the testing dataset. Specifically, while the F1 score of the SVM-based model maintains stability across diverse datasets in the clean scenario, its distribution becomes more dispersed when the dataset is subjected to poisoning. Figure 12c shows the model accuracy on the poisoned and clean testing dataset, and we can observe a slight influence of the poisoning state on these three poisoned models.

Moreover, upon comparing Figure 12b and Figure 12c, it appears highly probable to train an FFN-based model that yields good F1 scores compared to the other two. Conversely, it is notably likely to train SVM-based and RF-based models that achieve higher accuracy. That indicates the proper of the FFN-based model in dealing with imbalanced datasets.

The transferred setting test above only considers the target model's variations in classifier components. To extend our examination of attack generalization, we undertook further assessments encompassing cases where the target model underwent training with diverse feature extractors. Here, we show the scenario where the designer adopts rule-based techniques for their classifiers. Dib et al. [55] introduced five rule-based extracted features. We tested and found that the TF-IDF approach yielded the best performance when trained with both SVM (F1 score: 0.8794; Accuracy: 0.9592) and RF models (F1 score: 0.8802; Accuracy: 0.9681) on the clean dataset. The SVM and RF hyperparameters are shown in Tabel 3. Here, we used the same hyperparameters to compare with the first setting. We use these two models as the target model and evaluate their poisoned versions in terms of attack performance, as depicted in Figure 13.



(a) Attack success rate (b) F1 score (c) Model accuracy

Figure 13: Performance under the TF-IDF representation.

Figure 13a shows the result of attack performance, showcasing that while attack performance remains prominent, the ascent and distribution patterns observed in Figure 12a are not seen. That emphasizes the importance of feature representation adopted by the model designer. The insights garnered from Figure 13a indicate that despite substantial poisoning applied to the model designer's dataset, differences in feature representation can

render such efforts futile. Figure 13b shows the poisoned model F1 score on the poisoned and clean datasets. It shows that the SVM poisoned model's performance degrades and becomes dispersed increasingly with rising poison ratios. However, this trend is not obvious for the poisoned RF model, which is trained with the same feature representation. Furthermore, despite their consistent model types, this phenomenon is not as apparent in Figure 12b for the SVM-based model. Conversely, the impact of the dataset poison state for the SVM-based model shown in Figure 12b does not emerge here, and the consistent performance for the RF-based model does not appear either. The observations illustrated in Figure 13c about the model accuracy align with those derived from Figure 13b. This reinforces the notion that the influence of poison ratios on the target models varies based on their types and adopted feature representations.

When contrasting the insights obtained from Figure 12a and Figure 13a, it becomes apparent that the inconsistency in feature representation carries a more pronounced impact on attack prediction than the model's architecture. While the SVM-based and RF-based models continue to exhibit analogous escalating trends as the substitute model in Figure 12a, this consistency does not extend to Figure 13a. However, upon examining these two visualizations, it is evident that when the representation is uniform, their attack patterns closely align. In addition, the variance in feature representation also influences the testing performance of poisoned models, a trend discernible from the range within which their F1 scores and accuracies lie.

3.6.4 Mitigation

This section discusses the mitigation of the proposed backdoor attack method. Section 3.6.3 tests different types of model performance under one certain and multiple feature representations. The outcomes illustrated in Figure 12 and Figure 13 underscore the significance of feature representation knowledge to the attacker and defender. Therefore, instead of

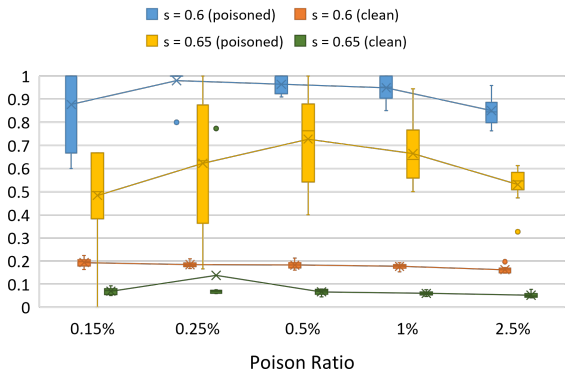
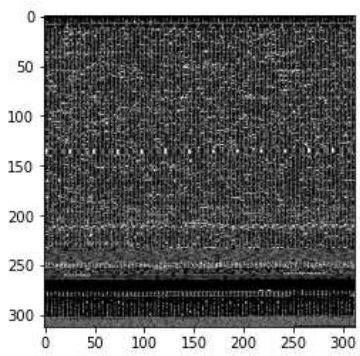


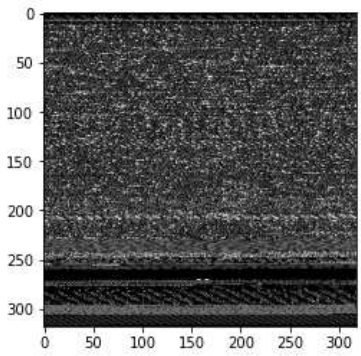
Figure 14: Isolation forest detection rate

assessing mitigation techniques targeting specific models like [160, 165], we evaluate the approaches that detect the anomalous sample by exploring the dataset features. The isolation forest [101] and HDBSCAN [34] belong to this category. In this context, we mainly investigate the isolation forest due to its superior mitigation performance as demonstrated in [134].

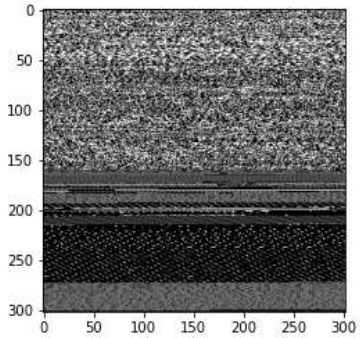
The isolation forest is devised on the premise that anomalies are both rare and distinct, aligning well with the objectives of poisoning in backdoor attacks. Capitalizing on these traits, the approach posits that these samples can be swiftly isolated through iterative partitioning. Specifically, fewer samples would result in a small number of portions, and the sample with distinguished attributes would be separated in early partitioning. Key parameters associated with the isolation forest are the number of trees, denoted as t, and the sub-sampling size, denoted as ψ. This study also delves into these parameters, establishing that the model converges effectively and provides sufficient detail for anomaly detection when $t = 100$ and $\psi = 256$. Accordingly, this configuration is adopted throughout the analysis. Ultimately, an instance x anomaly score s is calculated based on $E(h(x))$. Here, $h(x)$ is determined from the path length from the root node to a terminating node and unbuilt tree size, while $E(h(x))$ represents the average $h(x)$ over t trees. The score s is inversely proportional to $E(h(x))$. This research investigates s values of 0.5, 0.6, and 0.7,



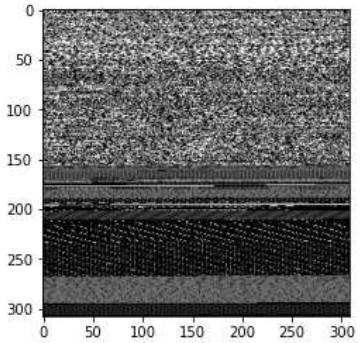
(a) Sample 1 (Mirai clean)



(b) Sample 1 (Mirai poisoned)



(c) Sample 2 (Gafgyt clean)



(d) Sample 2 (Gafgyt attacked)

Figure 15: Clean samples and their tampered versions.

concluding that instances with $s \geq 0.6$ can be considered potential anomalies.

Based on the above configuration, we evaluate its detection performance to the poisoned sample across diverse datasets and poison ratios. Figure 14 illustrates the outcomes, showcasing the detection rates for $s = 0.6$ and $s = 0.65$ concerning both poisoned and clean samples. Notably, when $s = 0.6$, a significant portion of the poisoned samples can be identified, yet there is an approximately 20% false positive rate where non-poisoned samples are flagged as anomalies. Conversely, by adjusting the detection sensitivity to $s = 0.65$ to align more closely with the clean samples, the detection rate for the poisoned samples notably declines. Higher s will cause the instances with lower $E(h(x))$ to be identified as anomalies. Based on the detection reduction at a high anomaly score s, the $E(h(x))$ calculated for many poisoned samples is higher than the threshold $E(h(x))$ for $s = 0.65$, resulting in these samples are flagged as normal. Furthermore, the analysis of distribution

and value-based outcomes reveals that the lowest and highest poison ratios result in lower detection rates for the isolation forest. Specifically, when the poison ratio is set at 2.5%, the detection rate becomes consistently low and stable across different datasets.

Apart from the mitigation techniques, the strategy of modifying samples $A(.)$ also imposes certain constraints on their adaptability. In this study, we chose to solely append the trigger at the end of the binary file to minimize the execution disruption. Figure 15 depicts the grayscale images of two samples with clean and tampered versions. Figure 15a and Figure 15b are clean and poisoned versions of one Mirai sample, respectively. Figure 15c and Figure 15d are one clean Gafgyt sample and its attacked version. To effectively identify the poisoned samples designed by this approach, a comprehensive binary analysis grounded in the ELF format would be considerably advantageous. However, in efforts to enhance sample stealthiness, attackers commonly eliminate crucial information, such as the section header, that would otherwise reveal the structure of the malware sample.

3.7 Conclusion

This study investigated the backdoor attack scenario, wherein the model designer utilizes a model-based feature extractor to train classifiers. We assessed the designed framework through the main evaluation and transferred setting to expose its most significant vulnerability. The primary evaluation aims to transparently showcase the effectiveness of the proposed framework, conducted under the assumption of white knowledge. To evaluate the attack performance in practical scenarios where the attacker's knowledge may not align with that of the model designer, we extended our assessment to include the transferred setting scenario. The results and conclusions derived from the experiments highlight the efficacy of the backdoor attack approach, reveal weaknesses in model performance regarding poison indication, and the importance of feature representation in securing model design.

Chapter 4

A Data-driven Study of IoT Malware Classification Models: Insights into Temporal, Architectural, and Spatial Inconsistency Challenges

Inconsistencies in architecture, temporal, and spatial values between the datasets used for developing ML-based solutions and those encountered during deployment are inherent challenges in the malware domain. Given the wide diversity of datasets and existing approaches, several key questions emerge: Are these three challenges universally relevant across all solutions? Among the many proposed methods, which should be prioritized for evaluation to best guide future research? What evaluation frameworks are most appropriate? Which solutions demonstrate greater robustness to these challenges? And, ultimately, what insights can be drawn from systematically analyzing both the challenges and the solutions? The research in this chapter is primarily motivated by these questions.

4.1 Introduction

The diversity in feature representations, extractions, and ML techniques of the solution development pipeline has led to various proposed approaches, it presents challenges to researchers and users in assessing performance and building new solutions due to the lack of comprehensive evaluations under consistent metrics. To solve this, some studies evaluated existing ML solutions against real-world challenges. For instance, Ma et al. [104] conducted the first empirical study on ML-based PE malware classification under concept drift, while Gao et al. [65] assessed Android malware detectors facing obfuscation, concept drift, and adversarial attacks. These studies aim to assess how well existing solutions adapt to evolving malware threats, namely, the inconsistency of the sample distribution between model deployment and development, which is a critical issue in the ML domain due to its fundamental assumption: independent and identically distributed (iid) [19].

Unlike the previous studies, this work aims to systematically evaluate ML-based IoT malware family classification solutions under three fundamental inconsistency challenges: *temporal*, *architectural*, and *spatial*. *Temporal inconsistency*, related to concept drift, arises as malware evolves, potentially causing model aging issues. *Spatial inconsistency* refers to performance degradation when the class ratio differs between the training and testing phases. These two challenges were identified in [121], but their impacts and the resistance of existing solutions were not fully explored. While temporal and spatial inconsistencies are common across malware domains such as PE and Android, architectural inconsistency is particularly critical for IoT. This stems from the widespread deployment of IoT devices across diverse application scenarios, where different CPU architectures are chosen to balance cost, power consumption, and performance. The landscape becomes even more complex as modern IoT devices are increasingly expected to support AI and ML tasks, which demand significantly higher computational performance, efficiency, and low latency. For example, ARM processors—based on a reduced instruction set computing (RISC) [6]

architecture—offer low power consumption and hardware simplicity, making them well-suited for wearable devices. In contrast, x86 architectures, based on complex instruction set computing (CISC) [4], are more appropriate for devices like laptops and desktops that require higher computational capabilities. These practical requirements drive the research for developing ML solutions capable of generalizing across diverse IoT platforms [129].

The evaluation aims to generate insights and findings that are valuable for future model design. Key evaluation results from this study show that the temporal and architectural inconsistencies present significant challenges, with all evaluated solutions experiencing performance degradation. Notably, under temporal inconsistency, the degree of model decline is greatly impacted by the temporal values of the model development and deployment. In contrast, the model decline extent varies less with the architectural values of these two stages under architectural inconsistency. Additionally, some solutions show greater resistance to these challenges. To gain insights from them, we conducted a one-way ANOVA analysis and used the error bar to identify their extracted features that most significantly contribute to family distinguishability, temporal stability, and architecture generalizability. The findings during the evaluation, including the challenges, impacts, the high-resistance solutions, and key features, can benefit future solution development.

4.2 Preliminaries

4.2.1 ML in the IoT Domain

On-device AI model challenge

With the proliferation of the IoT, edge computing, and mobile devices, an increasing number of AI models are being deployed on-device. Unlike traditional approaches that rely on transmitting data to centralized cloud infrastructures—often incurring high latency and system overhead—on-device AI enables real-time data processing, improves efficiency,

conserves network bandwidth, and enhances data privacy [68]. In a recent comprehensive survey, Wang et al. [162] examined the current landscape, technical challenges, and future directions of on-device AI. Their analysis underscores that on-device AI is a promising trend; however, its practical implementation faces significant challenges. These challenges arise from the inherent mismatch between the computational demands of modern AI models — often requiring intensive processing and large parameter sets — and the resource-constrained nature of edge environments, which typically offer limited computing power, memory, and energy capacity. To solve these challenges, the on-device AI models need to be implemented with optimizations in terms of data, model, and hardware. Feature engineering is one critical step of data processing, which could reduce data dimensionality while retaining only the essential information for analysis to minimize model complexity and improve interpretability [152]. At the model level, except for the special techniques like pruning, compression, and parameter sharing, they suggest designing models with a lightweight architecture.

Best "universal" ML solution and its misconception

The best "universal" ML solution is not conclusive, and DL is not necessarily the best "universal" ML solution for cybersecurity. This statement is supported by recent surveys analyzing the roles of ML in cybersecurity (e.g.,[19]) and the works comparing the performance of shallow models and deep models in the cybersecurity and malware domain (e.g., [18, 35]). Deep learning refers to ML methods based on neural networks, which typically require more computational power and larger training datasets compared to shallow ML methods. Apruzzese et al. [19] did the first effort to provide a comprehensive analysis of the role of ML in cybersecurity and clarified many misconceptions related to ML in the context of cybersecurity. One misconception they clarified is that deep learning is not necessarily better than shallow ML. When the data to analyze a small number of

features, shallow ML can attain similar performance as deep learning, but the latter still requires more resources and the results are more difficult to interpret [12]. Because of on-device AI model challenges, Cannarile et al. [35] conducted a comparative study of deep and shallow models for API call-based malware detection. Their objective was to identify solutions that could effectively operate on edge computing devices, achieving high detection accuracy while maintaining low computational complexity and minimal execution time. Their findings, based on evaluations across two distinct datasets, revealed that shallow and deep models delivered comparable predictive performance. However, deep models required substantially more training and inference time. As noted in their analysis, deep learning models rely on backpropagation across multiple epochs to reach convergence, and the use of nonlinear activation functions adds significant computational overhead. These observations highlight the potential advantages of shallow models on edge devices, where resource efficiency is critical.

Feature engineering for cross-architecture generalization

The cross-architecture challenge is special in the IoT platform compared with others. To tackle this problem, the research in this area focuses on identifying features that are consistent across different architectures, enabling models trained on one architecture to perform well when applied to others. Raju et al. [129] reviewed recent studies that address the cross-architecture challenge, and their analysis primarily focuses on how each solution's selected and processed features could ensure the generalizability across different architectures. Darwish et al.[53] conducted a recent survey and mainly studied the deep models for IoT analysis. The models they proposed for cross-architecture are also grounded in this theory, namely, feature selection/processing to generalize well on different architectures, such as [40, 153]. In summary, feature engineering is the main study focus of this research field to ensure model generalization over different architectures.

4.2.2 IoT Malware Research Dataset Source

Recent surveys [112, 129, 105, 53] have investigated the use of machine learning (ML) to enhance IoT malware defense. As noted by Madan et al.[105], academic studies typically obtain IoT malware samples either through honeypot deployments (e.g., IoTPot) or third-party repositories such as VirusTotal and VirusShare. Given the cross-architecture challenges inherent to IoT malware domain, Raju et al.[129] conducted a comprehensive review of recent developments in cross-architecture IoT malware detection and classification approaches. Their analysis highlighted dataset availability as a key challenge, noting a significant shortage of representative ELF malware datasets in comparison to the abundance of Windows PE malware datasets. This disparity is attributed to the relatively late emergence of Linux-based malware as a focus of cybersecurity research, gaining attention only around 2014. To date, IoTPot remains the only widely acknowledged benchmark dataset for IoT malware. Raju et al. also pointed out limitations associated with repositories such as VirusTotal and VirusShare. Although these platforms have made considerable efforts to collect malware samples, they exhibit severe imbalances in the distribution of samples across different CPU architectures. In a more recent survey, Darwish et al. [53] provided an updated summary and also examined the datasets used in existing studies. As they summarized from their investigation, in addition to widely used sources such as IoTPot, VirusTotal, and VirusShare, some studies also rely on datasets hosted on platforms like GitHub (e.g., github.com/azmoodeh/IoTMalwareDetection), Kaggle (e.g., kaggle.com/dgawlik/3s19-iot-malware), and competition websites (e.g., CyberAICup2024). However, many of these datasets are either no longer publicly available or provide only preprocessed features without access to raw malware binaries, making them unsuitable for alternative feature engineering approaches. This limitation is largely due to the risks involved in distributing raw malware samples.

4.3 Related Work

Recent studies [104] and [65] evaluate existing solutions across various challenges to get findings and derive insights.

Ma et al. [104] conducted the first large-scale, systematic empirical study on learning-based PE malware family classification methods. Building on previous research that typically classifies methods based on feature representations, they selected solutions from three categories: image-, binary-, and disassembly-based. The study was guided by research questions examining the effects of different datasets and concept drift on these models. By evaluating these methods under the same conditions, they aimed to identify the category with the best adaptability across various challenges and summarized the findings, such as the image-based PE malware classification models improved by leveraging transfer learning.

Gao et al. [65] evaluated learning-based Android malware detectors under the challenges of obfuscation, concept drift, and adversarial attacks. Like study [104], they selected solutions from three feature representation categories: string-, image-, and graph-based, and evaluated them under the same challenge settings, but incorporated more detailed model selection criteria to enhance the evaluation credibility: diversity, popularity, relevance, recency, and reproducibility. In addition to concept drift, the challenges of obfuscation and adversarial attacks were evaluated under specific tools: Obfuscapk [17] to obfuscate samples, and the AE-crafting algorithms HIV_JSMA and HIV_CW from [43] to generate adversarial attacks.

Apart from the target platform and evaluated challenges difference, this work does not impose category restrictions on model selection since an effective model may incorporate various feature representations (e.g., ensemble learning [55]), and the ML solution development pipeline often includes multiple stages [39]. Comparing and concluding the observations and findings (OF) from the feature representation-based model categorization

is insufficient. The conclusion is also demonstrated by related work [104]. They indicate that the data format of malware should not be a critical factor that impacts the classification performance, which is more likely decided by the model itself. Therefore, instead of concluding the best feature representation categories for specific challenges, this work treats each model individually and primarily investigates the impact of challenges on all identified models, aiming to derive general conclusions applicable to all.

4.4 Challenges and Evaluation Framework Design

To ensure that the conclusions drawn from the practical study are comprehensive, effective, and generalizable, the evaluation framework should be designed based on this principle: minimizing biases arising from specific testing conditions. To achieve this, our evaluation methodology follows these guiding principles. From the challenges perspective, datasets should accurately reflect real-world constraints by incorporating multiple sources to account for sample variability and attribute diversity, ensuring comprehensiveness. Effectiveness is maintained by selecting datasets that highlight the impact of specific challenges, with conclusions drawn through comparative analysis. Additionally, to enhance generalizability, datasets should be generated randomly rather than through selective inclusion or exclusion of data to prevent artificial biases. From the models perspective, comprehensiveness is achieved by selecting a diverse range of published solutions to capture the full spectrum of existing approaches. To ensure effectiveness, models should be generated following the original methodologies used by their authors, rather than relying on pre-determined parameters. Finally, generalizability is reinforced by assessing model performance across multiple test scenarios to ensure that conclusions are not confined to specific experimental conditions.

4.4.1 Challenge Definition

- **Temporal inconsistency**: The testing samples originate from different time periods than the training samples. The potential unfamiliar features and inconsistent feature patterns arising from data shifts over time might lead to the model performance decline.

- **Architectural inconsistency**: The testing samples belong to the same family categories as the training data, but their target architectures were not encountered during the model training.

- **Spatial inconsistency**: The malware family ratios in the deployed environments differ from that of the model development, leading to a mismatch in the model's performance between these two stages.

Given these challenges' interactive influences, the research questions are structured into 4 cases and two levels as Table 4.

4.4.2 Dataset Selection

Dataset selection criteria is critical for ensuring practical relevance. The selection criteria are:

- **Relevance**: The dataset should be related to the IoT malware domain, reflect real-world scenarios, and be widely studied.

- **Recency**: It should be up-to-date, capture current trends, and threats, and incorporate the challenges evaluated.

- **Accessibility**: It must be publicly available or easily accessible to enable reproducibility and comparison.

Table 4: Research Questions, Attributes, and Evaluation Configuration

	Research Questions		Challenges		Real Applications Scenario	Evaluation Configuration
			Consistency	Inconsistency		
L1	**RQ1**	Given the consistency of temporal, architectural, and spatial distribution, how do these models perform?	temporal architectural spatial		The deployed model is tested on the dataset with the same architecture, from the same period, and with family ratios as the training dataset; however, these samples differ from the training samples.	Train: $dt^0(y^0, a^0, s^0)$; Test: $dt^1(y^0, a^0, s^0)$; dt^1 samples differ dt^0, but from the same y^0, a^0, and s^0 as dt^0.
L2	**RQ2**	Given the architectural and spatial consistency, how does the temporal inconsistency influence models?	architectural spatial	temporal	The deployed model is tested on new or old samples.	Train: $dt^0(y^0, a^0, s^0)$; Test: $dt^2(y^1, a^0, s^0)$; $y^0 \neq y^1$.
	RQ3	With temporal and spatial consistency, how does the architectural inconsistency challenge these models?	temporal spatial	architectural	Tested sample belongs to the trained families but with an unseen architecture	Train: $dt^0(y^0, a^0, s^0)$; Test: $dt^3(y^0, a^1, s^0)$; $a^0 \neq a^1$.
	RQ4	Given the temporal and architectural consistency, how does the spatial inconsistency influence these models?	temporal architectural	spatial	Deployed models face different family ratios than the training dataset	Train: $dt^0(y^0, a^0, s^0)$; Test: $dt^4(y^0, a^0, s^1)$; $s^0 \neq s^1$.

dt: dataset; y: year, a: architecture; s: spatial; dt^0/dt^4 from the Training dataset/TDD2; dt^i, $i \in \{1, 2, 3\}$ from TDD1
$y^j \in \{2020, 2021, 2022\}$, $j \in 0, 1$; $a^0 \in \{ARM, MIPS\}$; $a^1 \in \{m68k, SH\}$;
$s^0 = 120 : 11 : 1$(Mirai: Gafgyt: Tsunami); $s^1 = 25 : 5 : 1$;

4.4.3 Model Selection

Given the background (Section 4.2.1) and the unique challenge of cross-architecture in the IoT domain, the model selection was narrowed to those designed for cross-architecture scenarios, and for the selection criteria, in addition to **diversity**, **popularity**, **relevance**, **recency**, and **reproducibility** mentioned in [65], we include:

- **Shallow models**: The models adopt lightweight architectures, allowing them to operate efficiently on resource-constrained devices while delivering timely responses.
- **Feature engineering and classifier generalizability**: The models primarily explore feature engineering and have been validated across various ML techniques like Random Forest (RF) and K-Nearest Neighbors (KNN).

This selection also aligns with the overarching goal of evaluation, namely, facilitating future research within the IoT malware domain. Compared with the solutions built around

designing machine learning models—which tend to generalize across domains and may offer limited relevance to IoT malware—the approaches grounded in exploring feature engineering are more related to the IoT malware, which aim to identify effective ways of representing and characterizing malware samples through feature design to achieve strong performance regardless of the underlying model. Correspondingly, by centering the evaluation on these solutions, the resulting insights are more specific to IoT malware and more readily applicable to future model development. Furthermore, since these feature-based methods are compatible with a variety of ML algorithms, they offer broad applicability and ease of reuse. This focus also aligns with the principle of "garbage in, garbage out," emphasizing that high-quality features are essential for robust model performance. Meanwhile, the features with reduced dimension not only help simplify model complexity but also enhance interpretability [152], making it easier to understand and analyze the behavior of malware.

4.5 Methodology

4.5.1 Dataset and Processing

Given the dataset background (Section 4.2.2) and the selection criteria (Section 4.4.2), this study adopts the IoTPot dataset [118]. Considering the recency and label reliability issue, we utilized the IoTPot dataset collected from 2020 to 2022[1]. This dataset originally contained 142,089 malware samples. After performing file type cleaning with LIEF [148], eliminating duplicate samples, removing packer-obfuscated files by radare2, collecting reports via and removing unknown samples of VirusTotal, and identifying labels by AVClass [133], the final dataset was refined to 57,471 samples. We retained the Mirai, Gafgyt,

[1]The most recent dataset collected by IoTPot dates to 2023. However, as discussed in [39], there are reliability concerns regarding labels assigned by VirusTotal, which can take anywhere from 10 days to over two years, so here we utilize the second most recent IoTPot dataset, covering the period from 2020 to 2022.

and Tsunami families, which make up 98.5% of the dataset. The architectures analyzed include 8, such as ARM, MIPS, etc.

Table 5: Malware Sample Distribution

Training			**Testing**			
Spatial Ratio: 120:11:1			*TDD1, Spatial Ratio: 120:11:1*			
	Architecture			*Temporal*		
Temporal	ARM	MIPS	*Architecture*	2020	2021	2022
2020	1056	792	ARM	1056	4752	1188
2021	4752	3564	MIPS	792	3564	1056
2022	1188	1056	m68k	132	1188	264
			SH	264	1584	264
			TDD2, Spatial Ratio: 25:5:1			
				Temporal		
			Architecture	2020	2021	2022
			ARM	248	1116	279
			MIPS	186	837	248

Table 6: Hyperparameter Grids

Model	**Hyperparameters and Values**
RF	n_estimators: [50, 100, 200] max_depth: [None, 10, 20] min_samples_split: [2, 5] min_samples_leaf: [1, 2] class_weight: ['balanced']
KNN	n_neighbors: [3, 5, 7, 9] weights: ['uniform', 'distance'] algorithm: ['auto', 'ball_tree', 'kd_tree', 'brute']
LSVC	C: [0.001, 0.01, 0.1, 1, 10, 100] penalty: ['l1', 'l2'] loss: ['squared_hinge'] max_iter: [2000, 5000, 8000] class_weight: ['balanced']
MLP	hidden_layer_sizes: [(50,), (100,), (50, 50), (100, 50)] activation: ['tanh', 'relu'] solver: ['adam', 'sgd'] alpha: [0.0001, 0.001, 0.01] learning_rate: ['constant', 'adaptive'] learning_rate_init: [0.001, 0.01, 0.1] max_iter: [500, 1000, 2000]

To study the challenge of architectural inconsistency, we separate the samples by architecture. To explore temporal and spatial consistency, we process the dataset as [121], which proposed temporal and spatial biases elimination methods and has been widely adopted by leading studies such as [24, 171, 172, 44]. Table 5 shows the attributes of the training and testing datasets. The testing dataset TDD1 maintains the same family distribution as the training dataset, and TDD2 exhibits spatial inconsistency. The 120:11:1 distribution reflects the retained dataset's imbalance ratio, while the 25:5:1 distribution creates a more balanced dataset. These ratios are maintained across all the architectures and three years in each dataset. Since some architectures do not have enough samples to maintain the distribution across years, we ultimately retained samples only for ARM, MIPS, m68k, and SH. The training dataset, depending on the training year and architecture, has six training scenarios. Considering the evaluation bias due to reliance on a single dataset, the process generates 10 dataset variants for each attribute combination of Table 5, each sampled from the full set of available samples with matching attributes.

4.5.2 Model Selection

Table 7: Overview of studies IoT malware family classification methods

	Feature Representation	Feature Selection	Selection Reason	Tested ML Methods	Hyper Eval
Shahzad et al. [135], 2012	ELF header, Section header, Program header, Symbol table, Dynamic section, Dynamic symbol section, Global offset table, Hash table	Information Gain	Cross-platform properties of ELF file	RIPPER [46], PART [62], C4.5 [125] Rules, and J48 [126]. cAntMiner [116], XCS [163], UCS [28], and GAssist ADI [23]	10-fold cross validation
Wan et al. [159], 2020	L byte sequence from ELF entry point, N-gram vector		Promote cross-architecture by considering CPU specific information learning.	SVM, KNN, NB, and MLP	10-fold cross validation
Karanja et al. [86], 2020	Grayscale images, texture features calculated from Gray Level Co-occurrence Matric(GLCM) [108]	Angular Second Moment(ASM), Inverse Differential Moment (IDM), Contrast, Entropy, Correlation calculated from four directions: 0°, 45°, 90°, 135° wit distance offset 1	Platform independent feature via selected Haralick textural features	RF, NB and KNN	
Lee et al. [97], 2020	String length frequency (SLF), Printable string information (PSI)	Document frequency (DF), co-occurrence based elimination, recursive feature elimination (RFE) [75], information entropy measure string uncertainty and selection	Printable strings captrues common traits of malware from the same family across different CPU architectures	RF, KNN and SVM	10-fold cross validation
Tien et al. [150], 2020	ELF features: instruction set architectures (ISAs), sample size external library, packer, function number, networking and other potential behavior via Yara rules; Opcode feature	12 types opcode features for different ISAs: logic, control and status, memory, stack, procedure, prefixed, system input and output, arithmetic, system, branch, execution timing, and the others	Adapted opcodes to improve cross-architectural detection and classification	FFN, CNN and SVM	10-fold cross validation
Samantray et al. [132], 2021	2 KB byte sequence from ELF entry point, base64 converter, 2-gram, and TF-IDF		Byte sequence from entry points are independent of underlying architecture	RF, DT, SVC, multinomial NB and LR	4-fold cross validation

Given the background (Section 4.2.1) and the selection criteria (Section 4.4.3), we mainly refer to surveys [112, 129] to select solutions. Table 7 presents the chosen works, detailing their feature representation, feature selection, tested ML methods, and hyperparameter evaluation techniques, which are summarized from the studies and the analysis of

surveys. The feature representations used in these studies are diverse, including ELF structure information, byte sequences, string sequences, images, and opcodes, with some studies employing multiple feature representations. Moreover, even within the same feature representation, different processing methods are applied. For instance, while both [159] and [132] extract byte sequences from ELF entry points, work [159] uses frequency vectors, whereas work [132] applies TF-IDF. These varieties encompassed in feature engineering complicate direct comparisons and conclusions based purely on the step of feature representation. Additionally, these studies focus on feature engineering and assess the effectiveness of their methods by testing with various types of ML classifiers. The strong performance across different ML techniques highlights the effectiveness of their feature engineering approaches. Additionally, k-fold cross-validation (CV) is commonly used in the hyperparameter evaluation.

The datasets used in each work differ, but to facilitate meaningful comparisons, a unified dataset should be selected. This difference may result in the best hyperparameter values differing from those identified in the original studies. Additionally, many studies report performance on different ML classifiers without specifying the exact hyperparameter values. Therefore, rather than directly adopting the hyperparameter values from these studies, we opted to follow their hyperparameter evaluation processes - k-fold cross-validation - to determine the best ones for each classifier. Moreover, the strong performance of their methods across various ML classifiers demonstrates these methods' robust generalization capabilities, but their tested specific ML classifiers are not entirely identical. Given their methods' good generalization, and the purpose of fair comparison to these works, we selected commonly-used shallow ML algorithms, namely, RF, KNN, Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP) [18]. Table 6 details each algorithm's hyperparameter setting for grid search, and the best combination is identified by cross-validation.

4.5.3 Evaluation Configuration and Metrics

Table 5 describes the sample distribution. Based on the RQs, this section describes the relationship between the training and testing dataset utilized in each RQ. As Table 4 shows, the sample year y, architecture a, and spatial distribution s are regarded as variables that characterize dataset dt, and each value combination of (y, a, s) is one dt setting. The three challenges arise from the inconsistency in these variable values between the training and testing datasets. In our evaluation, except for RQ2 and RQ3, each RQ has a one-to-one corresponding relationship between the training and testing setting. In RQ2 and RQ3, the relationship is one-to-two and one-to-three, respectively. To answer each RQ, we need to derive insights from the testing performance of the trained models[2]. However, the conclusions may be constrained by the specific value combination of (y, a, s) used in the training datasets of that RQ. Therefore, to improve the generalization of the conclusions, we increase the value combination diversity of the training dataset, so instead of concluding the testing performance of the models trained on a single dataset setting, we analyzed the results from 6 different training settings for each RQ, i.e., $dt^0(y^0, a^0, s^0)$, where $y^0 \in \{2020, 2021, 2022\}$, and $a^0 \in \{ARM, MIPS\}$. Meanwhile, 10 best-performing models are derived for each training setting.

The primary objective of each RQ is to assess the impact of the corresponding challenge on the studied models. To observe the impacts, based on the RQs, the experiments are divided into three levels. In Level 0 (L0), we replicate the model training and evaluation processes to show their performance on the selected dataset. Specifically, all samples from both the training and testing datasets are included in the model training and validation to identify the best hyperparameters. In Level 1 (L1), the best models generated from L0 are evaluated on datasets with the same settings but not used in the models' training and validation (Table 4 RQ1). In Level 2 (L2), the setting of the evaluated dataset differs from

[2]Each studied model is composed of one studied work, i.e., its feature engineering method, and one type of ML classifiers. Here, we use method→classifier to represent each model.

the training dataset setting (Table 4 RQ2-RQ4). Considering the increasing variation in the samples and settings between the training and testing datasets at each level, the impact of the challenges on the models in RQ1-RQ4 is primarily observed through their performance differences between their levels: L1 *vs* L0 and L2 *vs* L1.

Due to the sample imbalance in the malware domain, the main evaluation metric for the testing dataset with N classes is the macro F1 score $F1_{\text{macro}}$, which assigns equal importance to each class. Additionally, to understand the changes in the macro F1 score, some evaluation exhibits precision P_i, recall R_i, and F1 scores $F1_i$ for each class i. To answer each RQ, the evaluated model is derived from one dataset with this RQ's training setting by cross-validation and grid search, and the evaluation is on a testing dataset that maintains this RQ's setting relationship with the training dataset as specified in Table 4. The performance metric for each training-testing setting for each model is determined as the average of 10 evaluation results.

$$P_i = \frac{TP_i}{TP_i + FP_i}, R_i = \frac{TP_i}{TP_i + FN_i}, F1_i = 2 \cdot \frac{P_i \cdot R_i}{P_i + R_i} \tag{7}$$

$$F1_{\text{macro}} = \frac{1}{N} \sum_{i=1}^{N} F1_i \tag{8}$$

4.6 Experimental Evaluation

4.6.1 L0: Cross Validation (CV)

Table 8 shows cross-validation performance $F1_{\text{macro}}$ of each studied model under each training setting. For each value pair (y^0, a^0), the models are represented by the intersection of the method row and the classifier column, and the best performance is highlighted in bold. In the setting (y^0, a^0)=(2020, ARM), the best performing models of 7 datasets utilize [86], and Table 9 shows the classifier (C) used by it to achieve the best performance in three

datasets, along with $F1_{\text{macro}}$ (Fm) and classifier hyperparameters (HP).

Table 8: Studied models cross-validation performance ($F1_{\text{macro}}$)

y^0	a^0	ARM				MIPS			
		RF	KNN	SVC	MLP	RF	KNN	SVC	MLP
2020	[135]	0.973	0.992	0.851	0.421	0.984	0.999	0.893	0.674
	[159]	0.851	0.631	0.869	0.68	0.85	0.345	0.834	0.731
	[86]	**0.995**	0.974	0.769	0.649	0.963	0.764	0.733	0.452
	[97]	0.991	0.982	0.856	0.826	0.995	**1.0**	0.887	0.85
	[150]	0.888	0.899	0.811	0.584	0.94	0.969	0.861	0.615
	[132]	0.851	0.821	0.848	0.661	0.844	0.809	0.849	0.473
2021	[135]	0.984	0.992	0.774	0.781	0.962	0.996	0.78	0.72
	[159]	0.849	0.594	0.87	0.77	0.845	0.654	0.85	0.799
	[86]	0.997	**1.0**	0.601	0.593	0.966	**1.0**	0.56	0.389
	[97]	0.977	0.998	0.824	0.828	0.979	0.999	0.748	0.841
	[150]	0.846	0.906	0.632	0.761	0.846	0.912	0.549	0.649
	[132]	0.842	0.836	0.843	0.76	0.843	0.852	0.842	0.8
2022	[135]	0.974	0.998	0.828	0.654	0.99	0.999	0.883	0.776
	[159]	0.924	0.468	0.974	0.711	0.932	0.462	0.978	0.743
	[86]	0.981	0.925	0.559	0.545	0.993	**1.0**	0.725	0.564
	[97]	0.988	**0.998**	0.867	0.901	0.983	**1.0**	0.89	0.787
	[150]	0.961	0.981	0.542	0.737	0.943	0.975	0.614	0.8
	[132]	0.922	0.957	0.981	0.926	0.932	0.975	0.975	0.818

OF^3: Table 9 shows that while [86] consistently outperforms other methods, the best classifiers used vary across different datasets such as dt0 and dt1/dt2. Even when the same classifier type is employed, such as dt1 and dt2, the optimal hyperparameters differ. This highlights the importance of testing different classifiers and applying hyperparameter tuning techniques to select the best model and hyperparameter values for each dataset rather than relying on the classifiers and hyperparameter values identified in these works.

Despite the variety in training settings and datasets, some consistent patterns can be observed in Table 8. For instance, the model [86]→KNN achieves an $F1_{\text{macro}}$ score of 1.0 in three training settings, and [97] consistently performs best among MLP-based models in four training settings. These observations suggest the potential of drawing generalizable conclusions, even under the diversity of training settings and datasets.

Not all optimal models derived from the hyperparameter search in Table 6 exhibit acceptable performance. For instance, some models like [159]→KNN show low $F1_{\text{macro}}$ in setting (2020, MIPS), i.e., 0.345. However, the goal of this study is to examine the impact of these challenges, so we will consider different performing models.

Table 9: The best performing models utilize [86]

	dt0	dt1	dt2
C	KNN	RF	RF
Fm	1	1	0.998
HP	n_neighbors: 5 weights: 'distance' algorithm: 'auto'	n_estimators: 50 max_depth: None min_samples_split: 2 min_samples_leaf: 1	n_estimators: 50 max_depth: 20 min_samples_split: 5 min_samples_leaf: 1

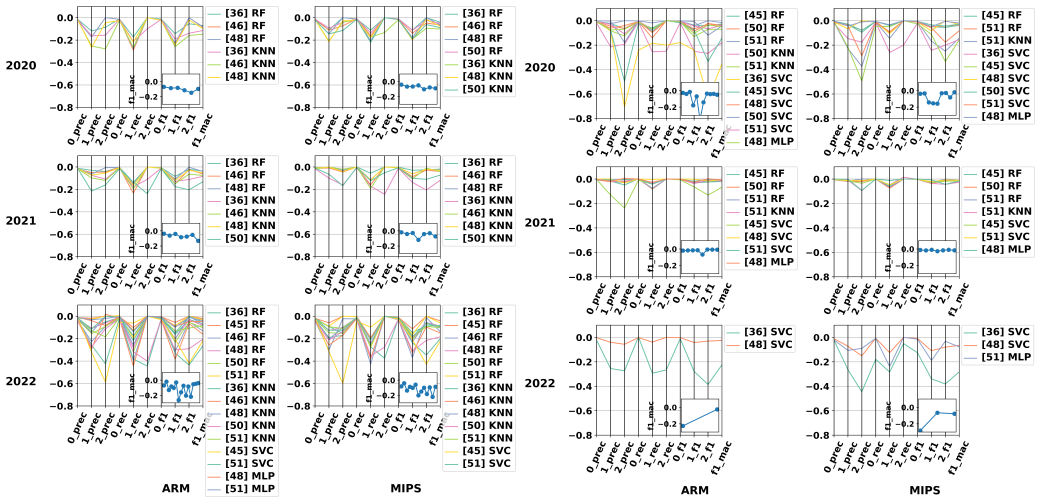
4.6.2 L1: RQ1

RQ1: Given the consistency of temporal, architectural, and spatial distribution, how do these models perform?

Figure 16 illustrates the impact of the RQ1 challenge on the optimal models. Each model shows its performance delta Δ between its performance on the new testing dataset and its cross-validation results. $\Delta > 0/\Delta < 0$ represents performance increase/decrease. Included metrics are presented in Section 4.5.3. The prefix number on each x tick represents the class, where Mirai (0), Gafgyt (1), and Tsunami (2); for instance, *0_prec* indicates the precision of Mirai. These models are categorized into three groups by cross-validation $F1_{\text{macro}}$: $F1_{\text{macro}} \geq 0.9$, $0.8 \leq F1_{\text{macro}} < 0.9$ and $F1_{\text{macro}} < 0.8$.

When comparing the performance Δ among the models in Figure 16a, Figure 16b, and Figure 16c, it is evident that larger performance differences are observed among models as model group $F1_{\text{macro}}$ declines. For example, in Figure 16a setting (2022, ARM), 15 models are shown, but their Δ patterns over different metrics are quite similar. In contrast, Figure 16c displays much more variability in metrics Δ among the models. For instance, in the setting (2022, MIPS), while only 8 models are included, 3 models show $\uparrow$ in *2_prec*, whereas 4 models show $\downarrow$. 4 models show $\uparrow$ in $F1_{\text{macro}}$ in Figure 16c, but all the models show $\downarrow$ in Figure 16b and Figure 16a. The average $F1_{\text{macro}}$ decline/standard deviation for model group $F1_{\text{macro}} \geq 0.9$ and $0.8 \leq F1_{\text{macro}} < 0.9$ is 0.092/0.055 and 0.066/0.079.

OF: While some models show $\uparrow$ in $F1_{\text{macro}}$, the maximum increase is just 0.013. 97% of the models experience $\downarrow$, and all models with $F1_{\text{macro}} \geq 0.8$ exhibit a decline. This



(a) $F1_{\text{macro}} \geq 0.9$ (b) $0.8 \leq F1_{\text{macro}} < 0.9$

(c) $F1_{\text{macro}} < 0.8$

Figure 16: The impact of RQ1 challenge on models. The index mapping between the Figures and Table 7 is: [36]-[132], [45]-[155], [46]-[86], [48]-[95], [50]-[146], [51]-[129].

suggests that testing on new datasets poses a challenge to these models, especially to those well-trained, but the small indicates the impact is insignificant.

The high similarity in the Δ patterns among well-trained models ($F1_{\text{macro}} \geq 0.9$) contrasted by great variability on poorly trained models ($F1_{\text{macro}} < 0.8$) indicate that not all models experience the same impacts from RQ1 challenge and suggest the potential capability to predict the challenge's effects on the well-trained.

4.6.3 L2: RQ2, RQ3 and RQ4

Due to the consistent Δ patterns among models of $F1_{\text{macro}} \geq 0.9$ under L1, the analysis of the impact of the L2 challenge focuses on them, specifically those in Figure 16a.

RQ2: Given the architectural and spatial consistency, how does the temporal inconsistency influence models?

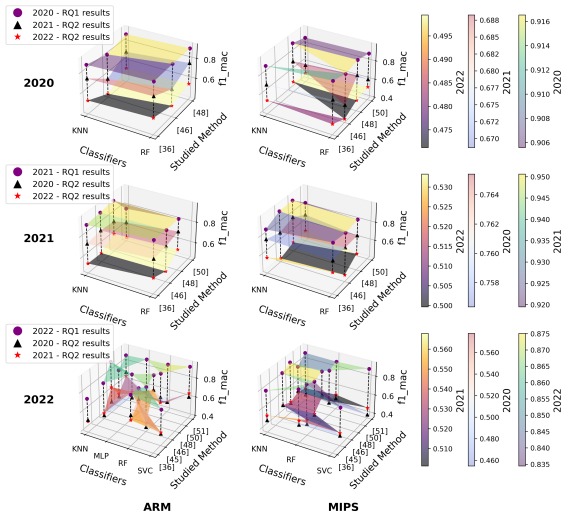
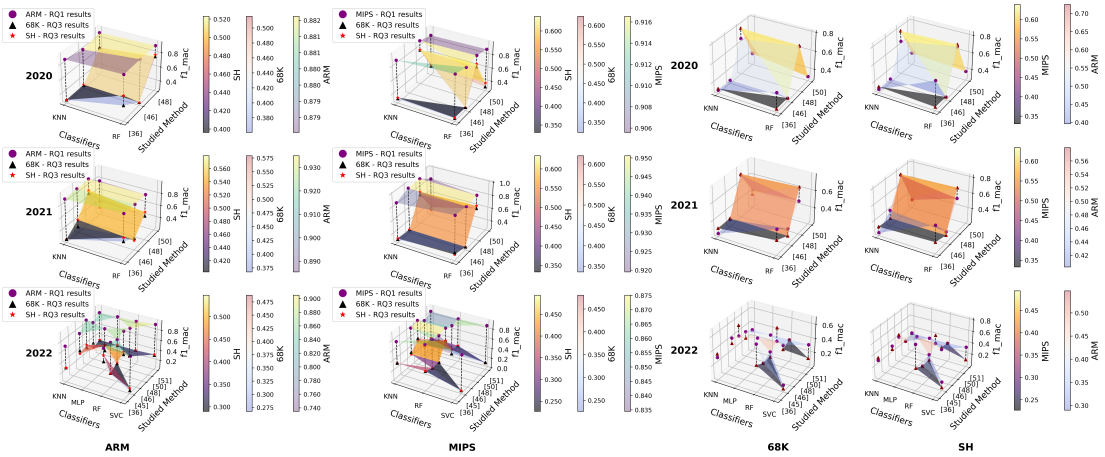


Figure 17: The impact of RQ2 challenge on models.

Figure 17 shows the impact of the RQ2 challenge on optimal models; each figure represents one training setting. It also shows the difference between testing on a dataset with different y^1 values. For instance, the training setting (y^0, a^0)=(2020, ARM) includes three surfaces: one with (purple o) represents the model $F1_{\text{macro}}$ under RQ1, one with (black ^) represents the models $F1_{\text{macro}}$ under RQ2 with y^1 =2021, and the other one (red *) is for RQ2 with y^1 =2022.

All models experience $\downarrow$ in $F1_{\text{macro}}$, despite the comparison is with RQ1, not CV. $F1_1$ and $F1_2$ are the main reason for the decline. The decline range of $F1_{\text{macro}}$ is 0.296$\pm$0.1, with a maximum 0.518 from the model [86]$\rightarrow$KNN trained on (2021, MIPS) and tested on (2022, MIPS), and the minimum is 0.105 from the model [135]$\rightarrow$KNN trained on (2021, MIPS) and tested on (2020, MIPS). Additionally, most models trained in y^0=2020



(a) Tesing with different architectures

(b) Training from different architectures

Figure 18: Model Performance $F1_{\text{macro}}$ under RQ3 challenge.

(2021) show significant differences between testing over y^1=2021 (2020) and y^1=2022, while the model trained in y^0=2022 displays only slight variations when tested on y^1=2020 and y^1=2021. Furthermore, models trained in y^0=2020 and 2021 performed poorly on the y^1=2022 compared to the other year.

OF: The noticeable $F1_{\text{macro}}$ Δ between the results of RQ1 and RQ2 underscores the significance of the temporal inconsistency challenge. This trend is consistent across various training settings, indicating that models generally perform worse when the testing dataset year differs from the training dataset year.

The similarity in testing performance between models trained in 2020 and 2021, and the similarity in the testing performance of these years' models compared to the model trained in 2022, suggests that the features of 2020 are more closely aligned with those of 2021, indicating that the 2022 data may have undergone significant changes or concept drift. This highlights the varying degrees of data shift over time, with some years showing more pronounced changes in distribution or features. Furthermore, it indicates that the values of y^0 and y^1 can significantly affect the model's performance degradation.

RQ3: With temporal and spatial consistency, how does the architectural inconsistency challenge these models?

Figure 18a illustrates the impact of the RQ3 challenge on the optimal models. Each figure represents one training setting and shows three results: one for RQ1 results, and the other two depict the testing results under RQ3 with different a^1 values. Similar to RQ2, the $F1_{\text{macro}} \downarrow$ is observed across all training settings. However, the decline is more significant, with the range 0.471±0.153. The maximum is 0.771 from the model [132]→SVC trained on (2022, MIPS) and tested on (2022, SH), while the minimum is 0.121 from the model [97]→RF trained on (2020, MIPS) and tested on (2020, SH).

It is evident that most models exhibit minor differences when tested on m68k and SH (range 0.032±0.036), and 12% of models show zero differences. Furthermore, Figure 18b compares the models, which derived from different a^0 values but tested on the same a^1 value, and still, the models remain close (range 0.085±0.087), and 9% show zero difference.

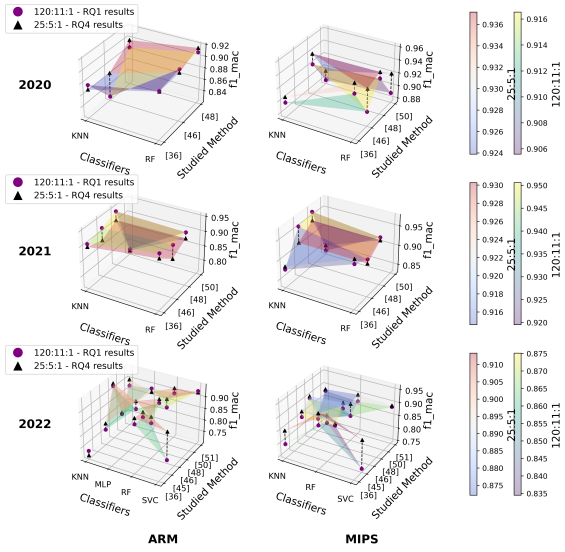


Figure 19: The impact of RQ4 challenge on models.

OF: Compared to the $F1_{\text{macro}}$ Δ in RQ2, models show a significantly greater decline under the RQ3 challenge, with the average $F1_{\text{macro}}$ Δ nearly double that of the temporal

inconsistency challenge. While these models are designed for cross-architecture scenarios, architectural inconsistency remains a major challenge.

ARM, MIPS, and SH architectures follow the RISC philosophy, whereas the m68k architecture uses CISC. Despite this, models trained on ARM/MIPS yield similar testing $F1_{\text{macro}}$ on SH and m68k datasets (Figure 18a), indicating that the testing instruction approach and architecture do not significantly affect model $F1_{\text{macro}}$ decline under architectural inconsistency. Similarly, as shown in Figure 18b, models trained from different architectures show close $F1_{\text{macro}}$ on the same testing architecture. This suggests that the training architecture has minimal impact on performance over datasets from distinct architectures. In summary, under architectural inconsistency, the $F1_{\text{macro}}$ decline of models is insignificantly affected or even unaffected by the specific a^0 and a^1 values.

RQ4: Given the temporal and architectural consistency, how does the spatial inconsistency influence these models?

Figure 19 illustrates the impact of the RQ4 challenge on the models. In contrast to RQ2 and RQ3, where all models experience a decline in $F1_{\text{macro}}$, RQ4 shows that only 30.9% models experience a decrease, and the remaining 69.1% exhibit an increase. The decline range is $F1_{\text{macro}}$ is 0.018$\pm$0.014, and the increase range is 0.027$\pm$0.024. The $F1_{\text{macro}}$ decline of around 0.1 to 0.15 reported in [121] with malware as the minority class is not observed in our results. The reason might be the dual impacts of temporal and spatial inconsistency in [121]. However, the decline from benign as the minority class is close to what we observed.

OF: The metric Δ is calculated from the model's performance on the testing dataset of RQ1 s^0 and RQ4 s^1. The negligible difference in $F1_{\text{macro}}$ suggests that, despite the substantial variation in the spatial distribution, the testing performance is not significantly affected.

4.7 Model Analysis

This section analyzes the models least affected by each challenge, with the primary objective of identifying the feature engineering solutions that enhance the models' resistance. The insights derived from them can be further utilized to develop solutions against similar challenges.

Table 10: Top 1/3 least models impacted under RQ2 and RQ3 (ordered by incremental impact)

	RQ2		**RQ3**		
y^0	a^0 =ARM				
	2021	2022	m68K	SH	MIPS
2020	[135]→KNN	[135]→RF	[97]→RF	[97]→RF	[97]→KNN
	[135]→RF	[135]→KNN	[97]→KNN	[97]→KNN	[97]→RF
	2020	2022	m68K	SH	MIPS
2021	[97]→RF	[97]→RF	[97]→KNN	[97]→KNN	[97]→KNN
	[150]→KNN	[150]→KNN	[97]→RF	[97]→RF	[97]→RF
	2020	2021	m68K	SH	MIPS
	[159]→RF	[159]→RF	[135]→RF	[135]→RF	[97]→MLP
	[97]→RF	[97]→RF	[86]→KNN	[97]→MLP	[97]→KNN
2022	[132]→KNN	[97]→MLP	[150]→KNN	[86]→KNN	[97]→RF
	[135]→RF	[132]→KNN	[135]→KNN	[97]→RF	[135]→RF
	[135]→KNN	[159]→SVC	[97]→MLP	[135]→KNN	[135]→KNN
y^0	a^0 =MIPS				
	2021	2022	m68K	SH	ARM
2020	[97]→RF	[150]→RF	[97]→KNN	[97]→RF	[97]→RF
	[135]→RF	[135]→KNN	[97]→RF	[97]→KNN	[97]→KNN
	2020	2022	m68K	SH	ARM
2021	[135]→KNN	[135]→KNN	[97]→RF	[97]→RF	[97]→RF
	[150]→KNN	[135]→RF	[97]→KNN	[97]→KNN	[97]→KNN
	2020	2021	m68K	SH	ARM
	[97]→RF	[135]→KNN	[97]→RF	[97]→RF	[97]→RF
2022	[150]→KNN	[159]→SVC	[97]→KNN	[97]→KNN	[135]→KNN
	[159]→SVC	[150]→KNN	[135]→KNN	[135]→KNN	[97]→KNN
	[135]→KNN	[97]→RF	[135]→KNN	[150]→RF	[86]→RF

Given that temporal and architectural inconsistencies are major challenges evaluated from Section 4.6, this section focuses on analyzing their top third least affected models, especially the studied methods. Table 10 displays the top third least affected models by these two challenges under each training and testing setting. It is important to note that the models achieving the highest $F1_{\text{macro}}$ during cross-validation do not necessarily represent the least impacted models.

Among the models with $F1_{\text{macro}} \geq 0.9$, the number of models using methods [135, 86, 97, 150] is nearly balanced across each training setting. However, two methods stand

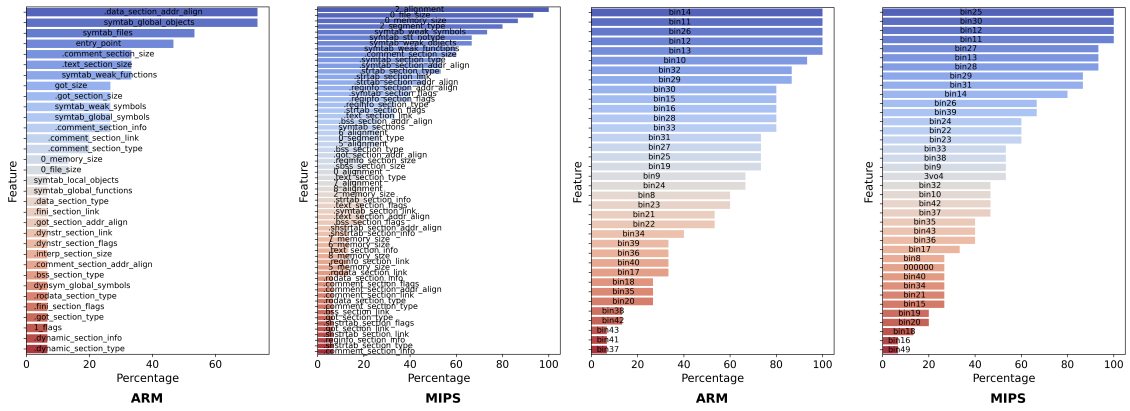
out: [97] and [135], which are most frequently found in the top third of listed models. In the RQ2 challenge, models using [135] constitute 38.24%, while those using [97] make up 23.53%. In the RQ3 challenge, models employing [97] account for 70.59%, and those with [135] represent 19.61%.

4.7.1 RQ2: Family Distinguishability and Temporal Stability

The methods frequently appearing in the RQ2 lists indicate that their selected features are capable of consistently distinguishing families over time. We referred to them as features with family distinguishability and temporal stability (FD&TS Features). This section focuses on analyzing such features extracted by [135] and [97].

Here, we use one-way ANOVA [1] to test the effect of each feature on differentiating families. As outlined in Section 4.5.3, each training setting includes 10 testing datasets, with each dataset comprising samples from three years. For each testing dataset by year, we first apply one-way ANOVA to identify the top 30 features of [135] and [97] with the highest F-statistic and lowest p-value. We then filter for features consistently among the top 30 across all three years, thereby identifying features with consistently high distinguishability over time. Next, we calculate the percentage of testing datasets that include each of these filtered features to show their generalizability across different datasets. Figure 20a and Figure 20b show [135] and [97] filtered features and their percentage.

In [135], the data section address alignment, which describes how the data section is aligned in memory, is the most generalized FD&TS Feature for the ARM architecture, while the second program alignment, which specifies alignment constraints for segments in memory and in the file, is the most generalized for the MIPS architecture. In [97], bin14 (number of strings within length range between $[e^{(j-1)\ln(L)/50}, e^{(j)\ln(L)/50}]$, where L is the maximum string length of the dataset; j =14) is most generalized for ARM, and bin25 is most generalized for MIPS.



(a) [135] (b) [97]

Figure 20: FD&TS Features.

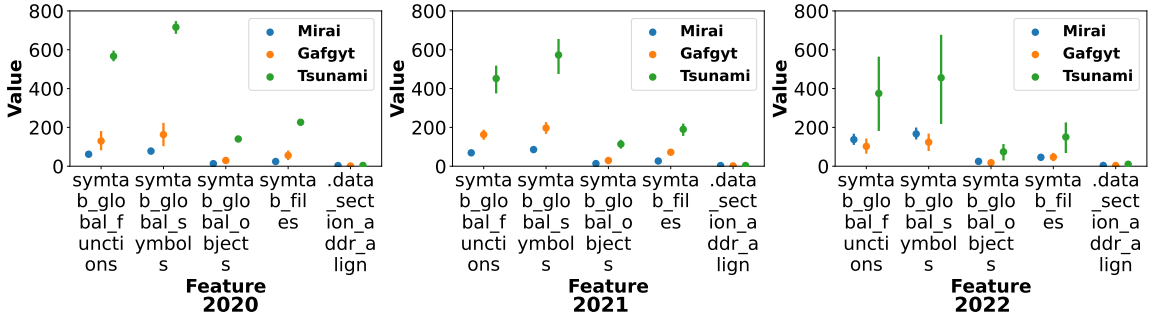
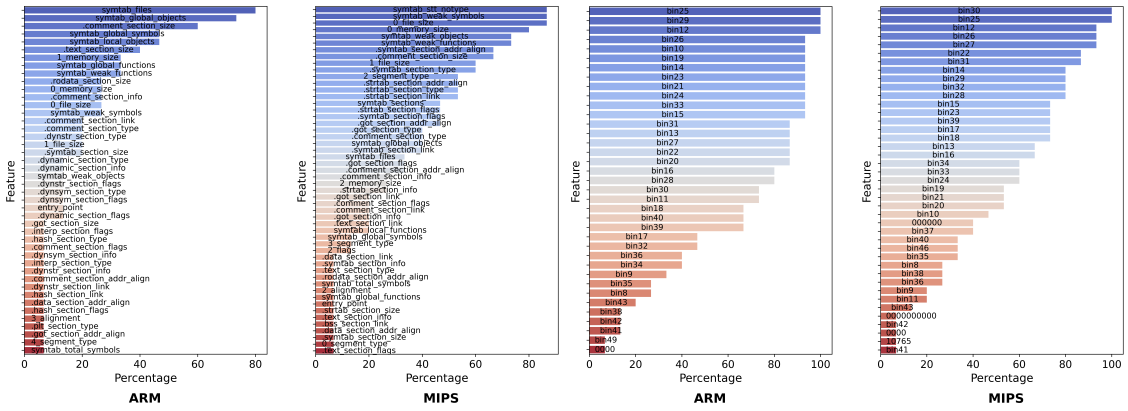


Figure 21: FD&TS Features extracted by [135] from one dataset.

Following [32], we utilize error bars (the samples mean and 0.95 confidence interval) in Figure 21 to illustrate the value distribution of the FD&TS features extracted by [135] from one dataset. The features, including symtab_global_functions and symtab_global_symbols, exhibit stronger distinguishability in families than others, and the distinguishability is consistent across different years. This suggests that these two features have the potential to enhance model classification performance and maintain their effectiveness over time.



(a) [135] (b) [97]

Figure 22: FD&AG Features.

4.7.2 RQ3: Family Distinguishability and Architecture Generalizability

The methods that frequently appear in the RQ3 lists suggest that their selected features effectively distinguish families across different architectures. We refer to these as features with family distinguishability and architecture generalizability (FD&AG features). We extract the FD&AG features following a similar approach presented in [135] and [97] and display their generalization in Figure 22a and Figure 22b. In [135], symtab_files (number of symbols with file type) is the most generalized FD&AG feature for the ARM architecture, whereas symtab_stt_notype (count of symbols without a specific type) is the most generalized for MIPS. In [97], bin25 is the most generalized FD&AG feature for ARM, while bin30 is the most generalized for MIPS.

Similarly, we employ error bars to depict the value distribution of the FD&AG features extracted by [97] from one dataset. As Figure 23 shows, the features, including bin14, bin16, and bin10, demonstrate greater distinguishability among families compared to other features, which holds true across various architectures. This indicates that these features could improve model classification performance and ensure effectiveness across different architectures.

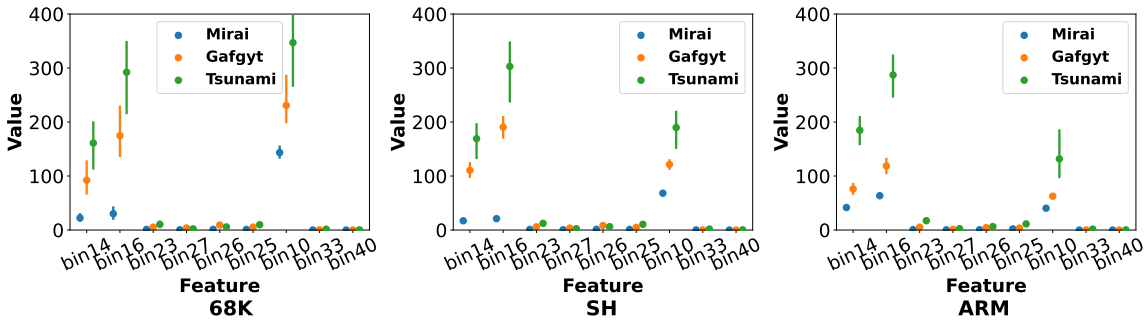


Figure 23: FD&AG Features extracted by [97] from one dataset.

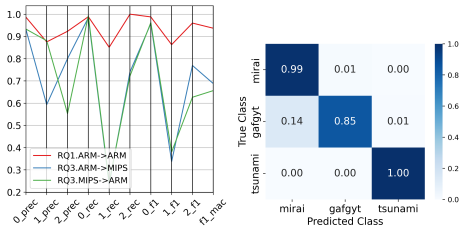
OF: As Table 7 shows, the extracted feature of [135] is from structure of each ELF file. Based on the results shown in Figure 20a and Figure 22a, we can observe that the symbol table, section header, and program header have many more features of family distinguishability, temporal stability, and architecture generalizability.

Work [97] extracts two feature vectors from each ELF file: string length frequency (SLF) vector and printable string information (PSI) vector. Figure 20b and Figure 22b show that SLF contributes most to the family distinguishability, temporal stability, and architecture generalizability. Especially in RQ3, most top third-ranked models use the method [97], which indicates the SLF feature extraction methods in this work enable the models to have good architecture generalizability, namely, least impacted by the architectural inconsistency challenge.

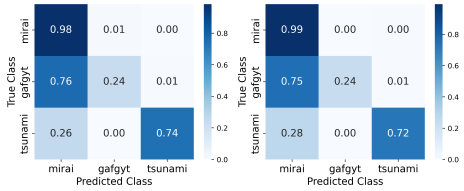
4.8 Model Performance Explanation

Given the significant impact of architecture inconsistency, we further investigate one specific case to identify the factors contributing to model performance degradation. The model selected for this analysis is [97]→RF, chosen based on its strong performance reported in Table 10 under RQ3. Figure 24a presents the model's testing metric values across three scenarios: (1) trained on dataset DT0 with a^0 =ARM to get model m_{arm} and tested this model on dataset DT1 with a^1 =ARM (RQ1 testing); (2) same model m_{arm} but tested on

dataset DT2 with a^1 =MIPS (RQ3 testing); and (3) trained on dataset DT3 with a^0 =MIPS (m_{mips}) and tested on dataset DT1 (RQ3 testing). The corresponding confusion matrices are shown in Figures 24b,24c, and24d. Two major comparisons are drawn: (a) scenario (1) *vs* (3), which share the same testing dataset; and (b) scenario (1) *vs* (2), which share the same model. Based on the testing metrics (Figure 24a) and the confusion matrices (Figures 24b,24c, and24d), it can be concluded that the primary cause of performance degradation is the misclassification of Gafgyt and Tsunami samples into the Mirai class, regardless of the comparison considered.



(a) Metric Value (b) ARM→ARM

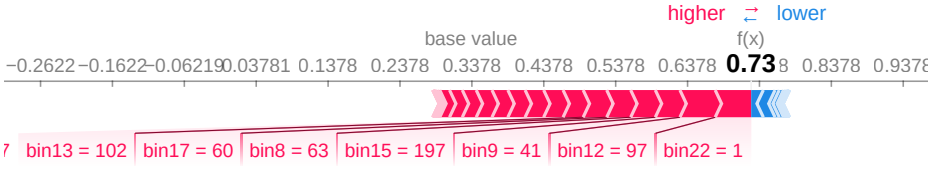


(c) ARM→MIPS (d) MIPS→ARM

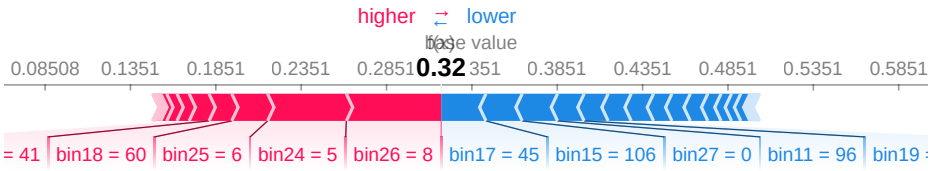
Figure 24: Testing performance comparison.

We then apply SHAP [103] to interpret feature contributions for a representative sample selected from comparison (b), specifically one that is correctly classified in scenario (1) but misclassified in scenario (3). Figure 25 presents an example of such a sample with the true label Gafgyt. As shown in Figure 25a, when the sample is tested using model m_{arm}, multiple supportive features (e.g., bin22) contribute positively to the correct classification as Gafgyt. In contrast, Figure 25b illustrates the model's behavior under m_{mips}, where although some features still contribute positively, their influence is outweighed by opposing features (e.g., bin17), resulting in a prediction score below 0.5 and misclassification. Figure 25c further

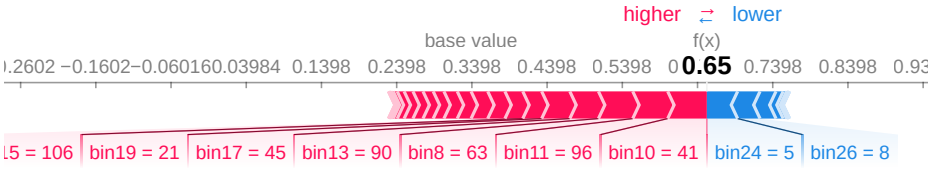
shows the SHAP values of m_{mips} leading the prediction as Mirai. Notably, feature bin17 plays a dominant role in predicting the sample as Gafgyt under m_{arm}, but contributes most to its misclassification as Mirai under m_{mips}.



(a) m_{arm}, explanation for Gafgyt prediciton



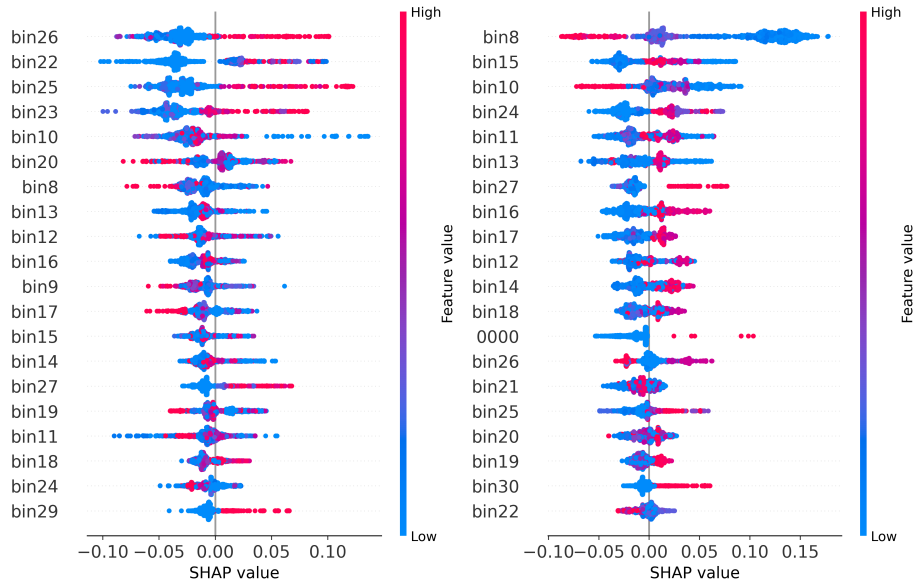
(b) m_{mips}, explanation for Gafgyt prediciton



(c) m_{mips}, explanation for misclassification into Mirai

Figure 25: Random Gafgyt sample.

Figure 26 shows feature impacts on the output of models m_{arm} and m_{mips} for predicting Gafgyt from the whole dataset perspective. A key observation is the divergence in feature importance rankings between the two models: m_{arm} relies heavily on features such as bin26 and bin22, whereas m_{mips} places greater emphasis on bin8, bin15, and bin10. Furthermore, certain features exhibit opposite contribution patterns, such as bin24, where low values contribute positively to the Gafgyt prediction in Figure 26a, while high values do so in Figure 26b. The variation in feature contributions in the above comparisons strongly underscores the impact of architectural inconsistency and emphasizes the importance of evaluation by the models trained from the same architecture.



(a) m_{arm} (b) m_{mips}

Figure 26: Feature impacts on model output as Gafgyt across DT1.

4.9 Lessons Learned

Here, we summarize and list some directions for future research based on the above OFs.

Lessons from challenge impacts: 1) The solutions' performance relies on the evaluation metrics utilized. Considering the temporal and spatial biases, Pendlebury et al. [121] established evaluation constraints, which many studies in leading venues adopted to assess their solutions. Our evaluations also reflect the impacts of these two biases, but the spatial bias is notably less significant than the temporal bias during independent evaluation. This finding indicates the varying degrees of these constraints and introduces flexibility into the evaluation process. Additionally, it emphasizes the necessity of designing appropriate evaluation metrics and specific details, such as the constraint strictness levels.2) The effect of temporal inconsistency on model performance is closely tied to the similarity between datasets used in model development and deployment. This conclusion is demonstrated across all evaluated models in Section 4.6.3 (RQ2). Accordingly, model update strategies

should be adapted dynamically based on the level of similarity between training and deployment environments. Relevant adaptation strategies from the image domain, such as those proposed in [113, 173, 131], may offer valuable insights. 3) The similarity is significantly diminished under architectural inconsistency, challenging the cross-architecture solutions design. However, some feature engineering methods such as [97] show good resistance. This indicates the potential to find good feature engineering solutions to solve this issue. Meanwhile, given its significant impact, the evaluation frameworks for the IoT domain should incorporate architectural bias alongside temporal and spatial biases.

Lessons from the model analysis: 1) The FD&TS and FD&AG features identified from the methods that exhibit the greatest resistance to challenges, specifically [135] and [97], can be further utilized or developed in future solutions aimed at enhancing resistance against such challenges. 2) The studied methods proposed different feature engineering methods. These methods not only show different ways of characterizing/understanding the malware samples but also assign models with different abilities. For example, the features extracted from [135] are based on the IoT malware file format. These features are much more specialized to the IoT malware than the string length frequency extracted by [97]. Thus, users might gain more indications about IoT malware by analyzing the behaviors of the models using such features. Additionally, since the features extracted from [135] are more limited to tamper with, the models with such features are more resistant to adversarial attacks [49].

4.10 Threats to Validity

Threats to internal validity primarily arise from the implementation of different solutions. The solutions evaluated in this study mainly focus on feature engineering techniques. Given that variations in the utilized datasets can lead to different optimal hyperparameter values and ML classifiers, the evaluation does not depend on the hyperparameter values

and classifiers identified in the studied solutions. Instead, we utilized their model validation approaches and assessed performance across multiple classifiers. Additionally, to minimize bias from the datasets, the performance reported for each model reflects the average of its 10 best-tuned models, with each derived from a single dataset and achieved through cross-validation and grid search.

Threats to external validity primarily relate to the selection of methods and datasets that reflect the challenges being studied. Acknowledging the significance of interpretability in the security domain, we concentrate on solutions that highlight feature engineering techniques. To facilitate future solution development, we incorporate approaches that provide different perspectives for characterizing malware samples. Thus, the solutions that exhibit greater resistance can be easily interpreted and applied in future work. Moreover, the diverse insights of these solutions provide wide application needs. Regarding the studied dataset, in addition to the dataset selection criteria, we evaluate various training and testing settings to mitigate bias from specific challenge scenarios. To improve the conclusion's generalizability, we summarize the findings for each challenge applicable across most settings.

4.11 Conclusion

This work comprehensively evaluates the existing IoT malware classification models under three fundamental challenges: temporal, architectural, and spatial distribution inconsistency. By categorizing the evaluation into levels and increasing the evaluation scenario varieties, we aim to summarize clear and broad conclusions about the challenge impacts. Key conclusions include that the architectural inconsistency challenges models most and performance decline nearly unaffected by the architecture of the training and testing datasets. By contrast, spatial inconsistency nearly has no impact on models despite the substantial difference between these two datasets. Based on the evaluation results, we analyzed the

solutions and their features most resistant to the challenges. Finally, we summarized the lessons and insights from the observations and analysis to benefit future work design and solution application.

Chapter 5

Optimal Transport-Based Adaptation and Evolution Interpretation for IoT Malware under Concept Drift

The evolving nature of the malware landscape inherently creates distributional discrepancies between training and testing datasets, making performance degradation of models without adaptation mechanisms inevitable. This raises several critical questions: How can models be designed with effective adaptation capabilities? Given that the primary objective of these models is to predict labels for unseen testing data by learning from labeled training data, how can both datasets be leveraged to address prediction failures on the testing set? What kind of learning procedure can be formalized to bridge this gap? How can the reliability of the predication of these models be enhanced? And, ultimately, can such investigations also yield insights that improve our broader understanding of the malware landscape? The research presented in this chapter is primarily guided by these questions.

5.1 Intorduction

Machine learning (ML)-based malware defense solutions are increasingly adopted due to their effectiveness in handling large-scale, dynamic datasets with reduced reliance on expert-driven feature engineering [61]. However, recent studies [39, 67, 19] highlight several persisting challenges that hinder the practical application of ML in this domain. Among them, *concept drift* remains a fundamental concern. It characterizes the malware landscape's inherent dynamic features–distributional shifts over time, which is driven by the ongoing arms race between attackers and defenders. These shifts violate the core ML assumption that training and testing data originate from the same distribution, ultimately leading to model performance degradation or aging.

To address this, some works [39] advocate for stream or continual learning methods, which enable models to adapt incrementally and maintain predictive accuracy in dynamic environments. Several approaches have been proposed along these lines, including online, and continual learning systems [110, 164, 59, 44, 145]. Complementary efforts focus on designing time-resilient feature representations [32, 172], employing rejection mechanisms to filter unreliable predictions [24, 167], or leveraging specialized model architectures such as one-class classifiers [137, 89] and ensemble learning frameworks [107, 117]. Despite these advancements, many of these models remain fundamentally dependent on the training distribution, limiting their robustness under real-world distribution shifts.

Domain adaptation (DA) techniques [92] explicitly address this limitation by enabling models to bridge the distributional gap between training (source) and testing (target) datasets. Given their objective, DA methods are naturally well-suited to mitigate concept drift in the malware domain. Li et al.[98] represent an early effort to apply DA in this context by integrating Domain Adversarial Neural Networks (DANN)[64], which leverage adversarial training to learn domain-invariant feature representations. Broader reviews such as [92]

highlight the diversity of DA approaches, yet their application within the malware domain remains limited. Furthermore, the selection of DA techniques for a specific field should extend beyond conventional performance metrics and account for the field's other requirements. *Malware lineage analysis*—understanding how malware evolves over time to better inform defense strategies—is critical for the malware ecosystem, particularly in the IoT domain, where rapid evolution and short development cycles exacerbate concept drift [50, 161].

Optimal transport (OT) principle-based DA methods present unique advantages. These techniques not only align source and target distributions in the feature space but also provide an explicit transport plan that reveals sample-level correspondences between domains. Such relationships inherently support lineage interpretation by capturing how individual source-target sample pairs relate under distributional shifts, offering insights through global distribution alignment. Motivated by these strengths, this work presents a first attempt to apply OT principle-based methods to the concept drift challenge in the malware domain. By integrating these methods with ELF-based feature extraction techniques—capable of characterizing malware in a high-level and uniform manner—we construct the ELF-OT frameworks. These frameworks are diverse in exact feature extractions and OT principle-based DA methods. Given their potential values in addressing several critical concerns in the malware domain, including model aging, malware evolution interpretation, and generalization across diverse IoT malware, this study first thoroughly evaluates and compares their performance under realistic scenarios of the IoT malware domain. Next, to enhance these frameworks' prediction reliability, we propose rejection mechanisms customized for each framework, which also indicates the direction for future works to improve the frameworks. Lastly, leveraging the relationships of the source-target sample pair inherently revealed by OT-based adaptation, we interpreted the malware evolution from the perspective of changes in ELF structure region features.

5.2 Preliminaries

5.2.1 ML-based Malware Classification and Concept Drift

The process of generating ML-based malware classification models can be formally described as follows. Let $\mathcal{X} \times \mathcal{Y}$ denote the sample feature-label space. The training dataset $\mathcal{D}_{\mathcal{S}}$ consists of n labeled samples drawn from the source distribution $p_{\mathcal{S}}(x, y)$, while the testing dataset $\mathcal{D}_{\mathcal{T}}$ contains m unlabeled samples drawn from the target distribution $p_{\mathcal{T}}(x, y)$. The objective of classification is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ from $\mathcal{D}_{\mathcal{S}}$ that generalizes well to $\mathcal{D}_{\mathcal{T}}$, where $f \in \mathcal{H}$, and $\mathcal{H}$ denotes the hypothesis space. The function f is trained by minimizing a loss function $\ell : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$, which quantifies the discrepancy between predicted outputs and true labels. The model's expected performance under a distribution is captured by the risk function $R(f) = \mathbb{E}[\ell(f(x), y)]$, which measures the expected classification error of f over the input distribution.

The performance of ML-based models is fundamentally based on the independent and identically distributed (IID) assumption, expressed as $p_{\mathcal{S}}(x, y) = p_{\mathcal{T}}(x, y)$ [5]. However, the dynamic nature of concept drift in the malware landscape violates this assumption, leading to model performance degradation. The joint distribution can be decomposed as either $p(y|x)p(x)$ or $p(x|y)p(y)$. Concept drift is typically classified into three types: covariate shift, where $p_T(x) \neq p_S(x)$ while $p_T(y|x) = p_S(y|x)$; prior shift, where $p_T(y) \neq p_S(y)$ while $p_T(x|y) = p_S(x|y)$; and conditional shift, where $p_T(y|x) \neq p_S(y|x)$ [24]. In the absence of domain labels, it becomes challenging to determine the specific shift type and quantify the contribution of each shift to model errors [24, 73].

5.2.2 Domain Adaptation and Optimal Transport

Standard learning assumes that the source and target domains share the same joint distribution, i.e. $p_{\mathcal{S}}(x, y) = p_{\mathcal{T}}(x, y)$. In contrast, domain adaptation considers the more realistic

scenario where $p_{\mathcal{S}}(x, y) \neq p_{\mathcal{T}}(x, y)$; that is, the source and target domains follow different joint distributions. Correspondingly, the goal of domain adaptation is to predict labels for samples drawn from the target domain, using labeled samples from the source domain and unlabeled samples from the target domain itself [92, 170]. Kouw et al. [92] categorize the adaptation approaches into sample-based, feature-based, and inference-based methods. Among them, the feature-based methods aim to map, project, and represent features such that a source classifier performs well on the target domain.

Optimal transport (OT) principle-based methods fall under the feature-based adaptation category. Their central idea assumes that distribution shift between domains can be addressed by identifying a transformation $T : \mathcal{X}_S \to \mathcal{X}_T$ that pushes the source distribution μ_S toward the target distribution μ_T, such that $T\#\mu_S = \mu_T$. This implies that the target samples X_T are drawn from the same probability density function as the pushforward $T\#\mu_S$. Under the assumption that the transformation preserves the conditional distribution—i.e., $p_S(y \mid x^s) = p_T(y \mid T(x^s))$—a principled approach to domain adaptation is to train a classifier on the transformed source samples, which resemble the target data, would generalize well on the target samples. However, directly solving for the optimal transport map T is generally intractable. Courty et al. [48] propose instead minimizing the transportation cost, defined as: $C(T) = \int_{\mathcal{X}_S} c(x, T(x))\, d\mu(x)$, where $c(x, T(x))$ is a distance function over the space $\mathcal{X}$, and $\mu(x)$ represents the probability mass transported from x to $T(x)$. This problem can be formulated using Monge's formulation (Eq. 9) and further relaxed by Kantorovich [85] into a convex optimization problem seeking the optimal transport plan γ_0, as in Eq. 10, where Π denotes the set of all joint probability distributions over $\mathcal{X}_S \times \mathcal{X}_T$.

$$T_0 = \arg\min_T \int_{\mathcal{X}_S} c(x, T(x))\, d\mu(x), \quad \text{s.t.} \quad T\#\mu_S = \mu_T. \tag{9}$$

$$\gamma_0 = \arg\min_{\gamma \in \Pi} \int_{\mathcal{X}_S \times \mathcal{X}_T} c(x^s, x^t)\, d\gamma(x^s, x^t). \tag{10}$$

In practice, μ_S and μ_T are estimated by discrete samples, namely, $\mu_S = \sum_{i=1}^{n_s} p_i^s \delta_{x_i^s}$ and $\mu_T = \sum_{j=1}^{n_t} p_i^t \delta_{x_i^t}$, where δ_{x_i} is the Dirac function at location $x_i \in \mathbb{R}^d$; p_i^s and p_i^t are probability mass associated to the ith sample; $\sum_{i=1}^{n_s} p_i^s = 1$ and $\sum_{i=1}^{n_t} p_i^t = 1$. Then each γ of Π is an $n_s \times n_t$ matrix, representing a probability coupling between the two empirical distributions, and $\gamma(i, j)$ tells how much probability mass of x_i^s is transferred to x_j^t. Meanwhile, $\gamma \mathbf{1}_{n_t} = \mu_s$, $\gamma^\top \mathbf{1}_{n_s} = \mu_t$, which means the sum over columns of γ corresponds to the empirical source distribution μ_s, and the sum over row corresponds to the empirical target distribution μ_t. The optimal transport plan γ_0 calculated from Eq. 10 could be used to map the source samples in the target domain [48], propagate source labels to infer target labels, or benefit the classifier's training [47, 52].

5.3 Related Works

Evolution-resilient features: Cai et al. [32] investigated the temporal evolution of Android applications [33] and proposed Sensitive Access Distribution (SAD), a behavioral profile that captures runtime access patterns of sensitive data sources and operations (sinks). Given that malware variants originate from different code but with the same semantics, Zhang et al. [172] developed APIGraph to capture the semantic relationships between Android APIs to slow down model aging. **Online/continuous learning:** Xu et al. [164] proposed DroidEvolver, an Android malware detection system built upon five online learning models. Its performance is sustained by adaptively updating the models, whose *Juvenilization Indicator* (JI) values surpass a defined threshold, with the new samples and pseudo-labels assigned from younger models. Chen et al. [44] proposed a continuous learning system

with an encoder-classifier architecture. To incrementally adapt to evolving data distributions, the system is periodically updated by the informative samples selected by the joint predictions from both subnetworks. **Rejection mechanisims:** Barbero et al. [24] introduced the TRANSCENDENT framework, which extends Transcend [83] through conformal prediction-based evaluators. These evaluators quantify the nonconformity of incoming samples relative to the training distribution, rejecting those likely to originate from shifted or unfamiliar regions of the input space. Similarly, Yang et al. [167] introduced CADE, which enhances model reliability by discarding test samples whose distances—measured in the contrastive learning embedding space—exceed the Median Absolute Deviation (MAD) threshold relative to the known training data.

Domain adaptation: Li et al. [98] are likely the first to explore the domain adaptation techniques for addressing concept drift in the malware domain. They leveraged DANN [64], a framework motivated by theoretical foundations of domain adaptation [27, 26], which suggests that a good representation for cross-domain transfer is one where an algorithm cannot learn to identify the domain of origin of the input observation, into their Control Flow Graphs (CAG) based malware detection models. DANN consists of two components: a label predictor and a domain classifier, and the domain classifier is trained adversarially to encourage the model to learn domain-invariant features. Additionally, Li et al. [98] evaluated Deep Adaptation Networks (DAN) [102] in the malware context, which also aims to learn domain-invariant feature spaces by minimizing the Maximum Mean Discrepancy (MMD) loss between source and target distributions.

Given the suitability of domain adaptation techniques for addressing concept drift, this work also proposes frameworks that incorporate such techniques into the model learning process. However, the selection of adaptation techniques in this study goes beyond predictive performance, with an additional focus on satisfying malware domain-specific requirements, particularly evolution interpretation. Motivated by this, we adopt optimal

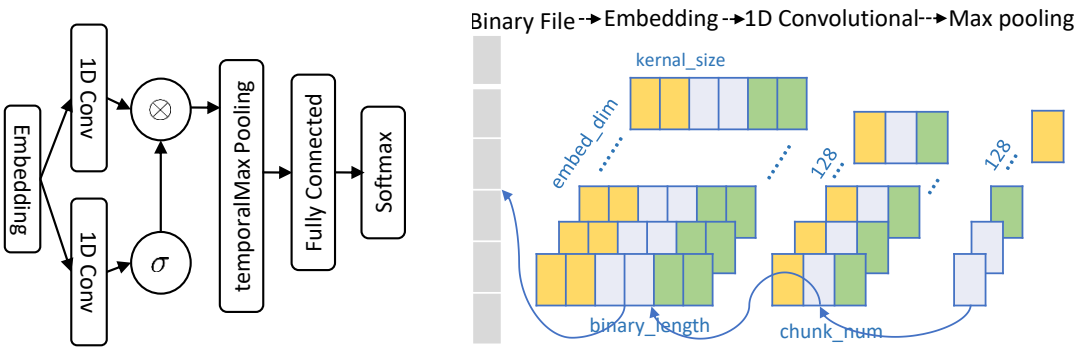
transport-based adaptation methods, a class of approaches that align source and target distributions within the feature space. A key advantage of these methods lies in the explicit transport plan, which inherently reveals relationships between source and target samples. Given these methods' benefits and expecting their further expansion in the malware domain, we first integrate optimal transport-based adaptation with malware-specific features and conduct a comprehensive performance evaluation. Subsequently, we design rejection mechanisms informed by the adaptation principles of these methods, aiming to improve their robustness and guide future advancements. Finally, we exploit the sample relationships revealed by the transport plan to construct adaptation lineages and interpret malware evolution characteristics.

5.4 ELF-OT

This section shows the joint frameworks ELF-OT, which incorporates the ELF feature extraction and optimal transport adaptation.

5.4.1 Feature Engineering

In Linux-based IoT systems, executable binary files typically follow the Executable and Linkable Format (ELF) standard [150]. Leveraging this format, Shahzad et al. [135] developed a feature engineering framework—ELF-Miner—that extracts critical structural information from ELF files for sample characterization. They initially derive 383 features from various ELF components, including the ELF header, section headers, program headers, symbol tables, dynamic sections, relocation entries, global offset tables, and hash tables. Afterward, these features are ranked using information gain and filtered by their discriminative power. The remaining features are subsequently used to train classification models.



(a) Model architecture. (b) Sparse-CAM.

Figure 27: MalConv [127].

Raff et al. [127] introduced MalConv (Figure 27a), a malware detection model specifically designed to process raw byte sequences. Its design considers the substantial positional variability inherent in executable files. Using an embedding layer to reduce the raw bytes dependence and combining the convolutional activations with a global max-pooling to regardless of the detected features' locations, Malconv has the potential to capture both local and global information of the binary file. The convolutional component employs a gated convolutional mechanism, as proposed in [54], with 128 filters of size 500 and a stride of 500. A key strength of MalConv is its architecture, enabling the analysis of the structural regions that are most informative for a file classification result, as shown in Figure 27b. MalConv was originally developed for malware detection and evaluated on PE-format samples [127]. It was subsequently examined for malware classification tasks [84] and adapted for use on other platforms [29].

The model utilizing learned feature extraction (MalConv) may surpass the manually engineered approach (ELF-Miner) in performance; however, their resource-intensive training and slow execution speed may hinder their applicability on IoT devices, which often have limited resources and need to respond in a timely manner. Sharma et al.[136] proposed a classification model based on a fine-tuned Xception architecture[45], where the final fully connected layer is replaced. This fine-tuned model demonstrated strong performance on

IoT datasets, regardless of the specific type of malware image representation used.

5.4.2 Optimal Transport-based Adaptation

Once the transport plan γ_0 is calculated, each source sample $\mathbf{x}_i^s$ can be transformed to the target domain by barycentric mapping (Eq. 11). The calculation of γ_0 from Eq. 10, expressed as $\gamma_0 = \arg\min_{\gamma \in \Pi} \langle \gamma, \mathbf{C} \rangle_F$, where $\langle ., . \rangle_F$ is the Frobenius dot product, $\mathbf{C}$ is the cost function, can be solved by combinatorial algorithms such as the simplex methods. However, due to its high computational complexity, Cuturi et al. [51] proposed the regularized optimal transport as $\gamma_0^\lambda = \arg\min_{\gamma \in \Pi} \langle \gamma, \mathbf{C} \rangle_F + \lambda \Omega_s(\gamma)$, where $\Omega_s(\gamma) = \sum_{i,j} \gamma(i,j) log \gamma(i,j)$ computes the negentropy of γ. Unlike the linear programming approach that yields a sparse γ_0, this regularization promotes a smoother solution, encouraging source points to spread their probability mass across a broader set of target points.

$$\hat{\mathbf{x}}_i^s = \arg\min_{\mathbf{x} \in \mathbb{R}^d} \sum_j \gamma_0(i,j) c(\mathbf{x}, \mathbf{x}_j^t) \tag{11}$$

The linear programming solution and regularization [51] do not use any class information. Given the available label information from the source domain, Courty et al. [48] leverage it and propose two regularizations by adding a new term to the regularized optimal transport plan (Eq. 12), where $\Omega_c(\gamma)$ is a class-based regularization term: 1) regularization with group sparsity, which expects the target sample only receives masses from sources samples sharing the same label. $\Omega_c(\gamma) = \sum_j \sum_{cl} \|\gamma(\mathcal{I}_{cl}, j)\|_2$, where $\mathcal{I}_{cl}$ contains the indices of rows in γ with corresponding source domain samples of class cl and $\gamma(\mathcal{I}_{cl}, j)$ contains the coefficients from $\mathcal{I}_{cl}$ associated to the jth target sample; 2) laplacian regularization, which aims to preserves the source sample similarity and class structures after transformation. $\Omega_c(\gamma) = \frac{1}{N_s^2} \sum_{i,j} S_s(i,j) \left\| \hat{x}_i^s - \hat{x}_j^s \right\|_2^2$, where $S_s \in \mathbb{R}^{N_s * N_s}$ is a positive symmetric similarity matrix encoding similarity between pairs of source sample, and $S_s(i,j) = 0$ if $y_i^s \neq y_j^s$.

$$\gamma_0^{\lambda} = \arg\min_{\gamma \in \Pi} \langle \gamma, \mathbf{C} \rangle_F + \lambda \Omega_s(\gamma) + \eta \Omega_c(\gamma) \tag{12}$$

The above approaches focus on identifying a transport plan γ_0 between the marginal distributions μ_S and μ_T. To handle both marginal and conditional alignment, Courty et al. [47] introduced JDOT, which seeks a transformation that directly aligns the joint distributions $p_{\mathcal{S}}(x, y)$ and $p_{\mathcal{T}}(x, y)$. The optimization problem Eq 10 then is transformed as Eq. 13, where $D(x^s, y^s; x^t, y^t) = \alpha c(x^s, x^t) + \mathcal{L}(y^s, y^t)$ is a joint cost measure combining both the distances between the samples and a loss function $\mathcal{L}$ measuring the discrepancy between y^s and y^t. Given the target labels are unknown, they aim to find a function on the target domain $f : \mathcal{X} \to \mathcal{Y}$ which enables the produced predictions to match optimally source labels to the aligned target instances in the transport plan. Thus, y^t and p_T is replaced by $f(x^t)$ and $p_T^f(x^t, f(x^t))$ respectively. Under the empirical distribution, f is to be learned by Eq 14, where $\Omega(f)$ is a regularization term on f, and the learning procedure is block coordinate descent, which optimizes γ and f alternatively.

$$\gamma_0 = \arg\min_{\gamma \in \Pi(p_S, p_T)} \int_{(\mathcal{X} \times \mathcal{Y})^2} D(x^s, y^s; x^t, y^t)\, d\gamma(x^s, y^s; x^t, y^t) \tag{13}$$

$$\min_{f \in \mathcal{H}, \gamma \in \Delta} \sum_{i,j} \gamma_{i,j} \left(\alpha d(x_i^s, x_j^t) + \mathcal{L}(y_i^s, f(x_j^t)) \right) + \lambda \Omega(f) \tag{14}$$

These approaches have primarily focused on calculating γ between the input spaces. Given the influence of the fixed and potential poor representation from the input space, Damodaran et al. [52] proposed DeepJDOT, which enables the alignment on the model extracted feature space. Assuming the feature embedder function as $g : \mathcal{X} \to \mathcal{Z}$, which maps the input to the latent space, and the classifier function as $f : \mathcal{Z} \to \mathcal{Y}$, which maps the latent space to the label space, the learning objective of DeepJDOT is as Eq 15, where L_s and L_t are the loss calculation functions of source and target domain. Except for the

similar optimization part as in JDOT, DeepJDOT incorporates the loss on the source domain to avoid the catastrophic forgetting issue of the classifier when disregarding the source samples. Given the large-scale dataset issue, the learning procedure of DeepJDOT is the stochastic approximation with minibatches from both domains.

$$\begin{aligned}\min_{\gamma,f,g} \quad & \frac{1}{n^s}\sum_i L_s\left(y_i^s, f\left(g\left(x_i^s\right)\right)\right) \\ & + \sum_{i,j}\gamma_{ij}\left(\alpha\|g(x_i^s) - g(x_j^t)\|^2 + \lambda_t L_t\left(y_i^s, f\left(g(x_j^t)\right)\right)\right)\end{aligned} \tag{15}$$

5.4.3 Framework

This subsection outlines the three frameworks and their training procedures. These frameworks are constructed to reflect the developmental progression of OT principle–based adaptation techniques (Section 5.4.2) and the possibility of application in the IoT malware domain, enabling a clearer observation of how each strategy refinement improves adaptation performance and considering the resource constraints of IoT devices.

Algorithm 2 ELFminer with OT

Input: $\mathcal{D}_S = (\mathcal{X}_S, \mathcal{Y}_S)$, $\mathcal{D}_T = (\mathcal{X}_T, \mathcal{Y}_T)$, $f_{\Lambda,w}$, $\mathcal{L}$, Φ_{ELF}
Output: $f_{\Lambda,w'}$

$\tilde{\mathcal{X}}_S \leftarrow \Phi_{\text{ELF}}$`.fit_transform(`$\mathcal{X}_S$`)`
$\tilde{\mathcal{X}}_T \leftarrow \Phi_{\text{ELF}}$`.transform(`$\mathcal{X}_S$`)`
`ot_emd` ← `ot.da.EMDTransport()`
Fit transport: `ot_emd.fit(`$\tilde{\mathcal{X}}_S, \tilde{\mathcal{X}}_T$`)`
Map source samples: $\mathcal{X'}_S \leftarrow$ `ot_emd.transform(`$\tilde{\mathcal{X}}_S$`)`
while not converged **do**
    $w' \leftarrow \arg\min_{\boldsymbol{w}} \mathcal{L}(\mathcal{Y}_S, f(\mathcal{X'}_S))$
end while
return $f_{\Lambda,w'}$

Algorithm 3 ELFminer with JDOT

Input: $\mathcal{D}_S = (\mathcal{X}_S, \mathcal{Y}_S)$, $\mathcal{D}_T = (\mathcal{X}_T, \mathcal{Y}_T)$, $f_{\Lambda,w}$, $\mathcal{L}$, Φ_{ELF}, α, I_b, I_n
Output: $f_{\Lambda,w'}$

$\tilde{\mathcal{X}}_S \leftarrow \Phi_{\text{ELF}}$.fit_transform($\mathcal{X}_S$)
$\tilde{\mathcal{X}}_T \leftarrow \Phi_{\text{ELF}}$.transform($\mathcal{X}_S$)
ns, nt = len($\mathcal{X}_S$), len($\tilde{\mathcal{X}}_T$)
ws, wt = ot.unif(ns), ot.unif(nt)
C0 = ot.cdist($\tilde{\mathcal{X}}_S, \tilde{\mathcal{X}}_T$)
while $i < I_b$ **do**
    fcost = $\mathcal{L}(\mathcal{Y}_S, f_{\mathbf{\Lambda},\hat{\mathbf{w}}}(\tilde{\mathcal{X}}_T))$
    C = α * C0 + $fcost$
    G = ot.emd(ws, wt, C)
    Yst = nt * G.T.dot($\mathcal{Y}_S$)
    $f_{\Lambda,w'} \leftarrow f_{\Lambda,w}$.fit($\tilde{\mathcal{X}}_T, Yst, I_n$)
end while
return $f_{\Lambda,w'}$

Algorithm 4 DeepJDOT

Input: $\mathcal{D}_S = (\mathcal{X}_S, \mathcal{Y}_S)$, $\mathcal{D}_T = (\mathcal{X}_T, \mathcal{Y}_T)$, $f_{\Lambda,w}(g_{\Lambda_g,w_g})$, $\mathcal{L}_f$, $\mathcal{L}_\sqcup$, α, λ, I
Output: $f_{\Lambda,w'}(g_{\Lambda_g,w'_g})$

ns, nt = len($\mathcal{X}_S$), len($\mathcal{X}_T$)
ws, wt = ot.unif(ns), ot.unif(nt)
while $i < I$ **do**
    for each batch $(\mathcal{X}_S^b, \mathcal{Y}_S^b)$, $(\mathcal{X}_T^b)$ **do**
        C0 = ot.cdist($g_{\mathbf{\Lambda_g},\hat{\boldsymbol{w}}_{\boldsymbol{g}}}(\mathcal{X}_S^b), g_{\mathbf{\Lambda_g},\hat{\boldsymbol{w}}_{\boldsymbol{g}}}(\mathcal{X}_T^b)$)
        fcost = ot.cdist($\mathcal{Y}_S, f_{\mathbf{\Lambda},\hat{\mathbf{w}}}(g_{\mathbf{\Lambda_g},\hat{\boldsymbol{w}}_{\boldsymbol{g}}}(\mathcal{X}_T^b))$)
        C = α * C0 + fcost
        G = ot.emd(ws, wt, C)
        $w'_g, w' = min_{w_g,w}\mathcal{L}_s(\mathcal{Y}_S^b, f_{\Lambda,w}(g_{\Lambda_g,w_g}(\mathcal{X}_S^b))) + G * (\alpha * \|g_{\Lambda_g,w_g}(\mathcal{X}_S^b) - g_{\Lambda_g,w_g}(\mathcal{X}_T^b)\|^2 + \lambda * \mathcal{L}_t(\mathcal{Y}_S^b, f_{\Lambda,w}(g_{\Lambda_g,w_g}(\mathcal{X}_T^b)))$
    end for
end while
return $f_{\Lambda,w'}(g_{\Lambda_g,w'_g})$

ELFMiner with OT

The first framework integrates ELFMiner with OT-based adaptation techniques (Eq. 12) to construct the *ELF-OT* classifier, considering ELFMiner's benefits in improving efficiency and saving resources compared to model-based approaches. The training and integration procedure is outlined in Algorithm 2. As both JDOT and DeepJDOT require an initial model trained in a non-adaptation phase—where training and validation are conducted exclusively on the source dataset—this framework similarly fixes the hyperparameters λ for the input model f, allowing only the model parameters w to be updated during training. Algorithm 2 presents an example utilizing `ot.da.EMDTransport` (EMD) from the POT library [60] to project source samples into the target domain, based on the formulation in Eq. 10 (i.e., the unregularized form of Eq. 12) (lines 3–5). In alignment with the OT-based methods discussed in Section 5.4.2 and considering those supported by the POT library, the implemented methods include `ot.da.SinkhornTransport` (Sinkhorn), `ot.da.SinkhornLpl1Transport` (SinkhornLpl1), and `ot.da.EMDLaplaceTransport` (EMDLaplace). Since this framework does not incorporate target domain samples in the model training process (line 6), as its objective is to align source representations toward the target domain, the training proceeds with early stopping, and hyperparameters (λ, η in Eq. 12) are selected via grid search. $\mathcal{L}$ is the cross-entrophy loss function. Φ_{ELF} represents the ELFMiner feature engineering.

ELFMiner with JDOT

Given that the main advancement of JDOT over OT is joint distribution alignment, this framework integrates ELFMiner with JDOT. The training process, illustrated in Algorithm 3, follows the optimization objective defined in Eq. 14. $d(x_i^s, x_j^t)$ and $\mathcal{L}(y_i^s, f(x_j^t))$ are implemented on line 5 and 7. Based on the principle of block coordinate descent, the framework alternately optimizes γ and f. In detail, when f is fixed, the optimization boils

down to a classical OT problem (lines 7–9); when γ is fixed, the optimization relies on the propagated labels to the target samples from the transport plan to update f (lines 10–11). $\mathcal{L}$ [1] could be the mean square or hinge loss function. As JDOT relies on a fixed number of iterations to regulate model training, I_b and I_n denote the maximum iteration counts for block coordinate descent and network training, respectively.

MalConv/Xception with DeepJDOT

Given that the primary improvement of DeepJDOT over JDOT is its use of a learnable feature extractor, this framework integrates DeepJDOT with MalConv/Xception. The overall training procedure is illustrated in Algorithm 4. In this setup, the last fully connected layer of MalConv/Xception is designated as f, while the preceding layers are treated as g. For Xception, an additional fully connected layer is inserted into g to facilitate feature space updates. Similar to JDOT, γ and $f(g(.))$ are optimized alternatively on lines 5–8 and line 9, respectively. ${L_s}^1$ and ${L_t}^1$ utilize the cross-entrophy loss function. Similarly, I denotes the maximum iteration counts for training epochs.

Except for Xception with DeepJDOT, all frameworks support γ_0-based interpretation of source–target sample relationships in ELF structural regions during adaptation. This focus on ELF structure features is driven by the fact that IoT malware is typically collected as ELF-format binaries, produced through the compilation and linking of source code and libraries using a variety of toolchains and build configurations. This process embeds both program content (e.g., code sections, data regions, symbol tables) and metadata (e.g., section types, flags, alignment) into the binary, reflecting variations introduced by compiler types, optimization settings, linker scripts, and target architectures [140, 87, 90], and offering insights into both the implementation and the cumulative changes induced throughout the software build pipeline. Moreover, ELF structural analysis is a foundational step in

[1] The loss function is chosen based on prior work and available code.

malware reverse engineering [49], indicating its interpretation importance before deeper analysis. This focus also yields complementary insights without overlapping with previous evolutionary analysis [50, 161].

5.5 Rejection and Explainability Mechanisms

This section presents the proposed target sample prediction rejection and explainability mechanisms, building upon the features and theory of ELF-OT frameworks introduced in Section 5.4.

5.5.1 Prediction Rejection

OT

Algorithm 5 outlines the proposed OT-based target sample rejection mechanism, with implementation details varying depending on the specific OT method. This strategy is driven by two main intuitions: (1) In OT-based domain adaptation, the classifier is trained on source samples that have been barycentrically mapped to the target domain. A fundamental assumption in classification is that a target sample is more likely to share the label of its nearest neighbors in feature space [44]. However, noisy or ambiguous samples may violate this assumption. To assess class-wise proximity, the algorithm calculates the Euclidean distance from each target sample to the centroid of its k nearest mapped source samples per class (lines 2–10), where k is adjusted based on the size of each class. These distances are transformed into similarity scores and normalized to yield a class-wise similarity distribution. High entropy in this distribution indicates that a target sample is comparably close to multiple classes, increasing its risk of misclassification. Thus, target samples with entropy exceeding the threshold α_1 are filtered out (lines 11–14). (2) The barycentric mapping formulation (Eq. 11) expresses each mapped source sample $\mathbf{x}'^{s}_{i}$ as a convex combination of

Algorithm 5 Barycentric Entropic Rejection

Input: ${\mathcal{X}'}_S$, $\hat{\mathcal{X}}_T$, $\mathcal{Y}_S$, γ_0, k, α_1, α_2, α_3
Output: $\mathbf{X}_T^{\text{keep}}$

$\mathcal{C} \leftarrow \texttt{unique}(\mathcal{Y}_S)$
for each $c \in \mathcal{C}$ **do**
  ${\mathcal{X}'}_S^{(c)} \leftarrow {\mathcal{X}'}_S[\mathcal{Y}_S = c]$
  $\texttt{NN} \leftarrow \texttt{NearestNeighbors}(k).fit({\mathcal{X}'}_S^{(c)})$
  for each $\mathbf{x}_t \in \mathcal{X}_T$ **do**
    $\mathcal{N}_k \leftarrow \texttt{NN}.\text{kneighbors}(\mathbf{x}_t)$
    $\boldsymbol{\mu}_t^{(c)} \leftarrow \frac{1}{k} \sum_{\mathbf{x}_s \in \mathcal{N}_k} \mathbf{x}_s$
    $d_c(\mathbf{x}_t) \leftarrow \|\mathbf{x}_t - \boldsymbol{\mu}_t^{(c)}\|_2$
  end for
end for
$\mathbf{D} \leftarrow [d_c(\mathbf{x}_t)]_{\forall c, \mathbf{x}_t}$
$\mathbf{S} \leftarrow \exp(-\mathbf{D}); \quad \mathbf{S} \leftarrow \mathbf{S} / \sum_c \mathbf{S}$
$\mathcal{H}_c(\mathbf{x}_t) \leftarrow -\sum_c \mathbf{S}_c(\mathbf{x}_t) \log \mathbf{S}_c(\mathbf{x}_t)$
$\mathbf{m}_1 \leftarrow \mathcal{H}_c < \texttt{percentile}(\mathcal{H}_c, \alpha_1)$
for each $c \in \mathcal{C}$ **do**
  $\gamma_0^{(c)} \leftarrow \gamma_0[\mathcal{Y}_S = c, :]$
  $\mathbf{P}_r \leftarrow \gamma_0^{(c)} / \sum_{\text{col}} \gamma_0^{(c)}$
  $\mathcal{H}_r \leftarrow -\sum_j \mathbf{P}_r(i,j) \log \mathbf{P}_r(i,j)$
  $\mathbf{m}_s \leftarrow \mathcal{H}_r < \texttt{percentile}(\mathcal{H}_r, \alpha_2)$
  $\gamma' \leftarrow \gamma_0^{(c)}[\mathbf{m}_s, :]$
  $\mathbf{P}_t \leftarrow \gamma' / \sum_{\text{row}} \gamma'$
  $\mathcal{H}_t \leftarrow -\sum_i \mathbf{P}_t(i,j) \log \mathbf{P}_t(i,j)$
  append $\mathcal{H}_t$ to $\mathcal{H}_{\text{total}}$
end for
$\mathbf{P}_c \leftarrow \mathcal{H}_{\text{total}} / \sum_{\text{row}} \mathcal{H}_{\text{total}}$
$\mathcal{H}_c \leftarrow -\sum_i \mathbf{P}_c(i,j) \log \mathbf{P}_c(i,j)$
$\mathbf{m}_2 \leftarrow \mathcal{H}_c < \texttt{percentile}(\mathcal{H}_c, \alpha_3)$
$\mathbf{m} \leftarrow \mathbf{m}_1 \wedge \mathbf{m}_2$
return $\hat{\mathcal{X}}_T^{\text{keep}} \leftarrow \hat{\mathcal{X}}_T[\mathbf{m}]$

Algorithm 6 Consistency-Based Rejection (JDOT)

Input: $\hat{\mathcal{X}}_S$, $\mathcal{Y}_S$, $\hat{\mathcal{X}}_T$, $f_{\Lambda,w'}$, k, α
Output: $\mathbf{X}_T^{\text{keep}}$

$\mathcal{C} \leftarrow \texttt{unique}(\mathcal{Y}_S)$
$\texttt{fcost} = \mathcal{L}(\mathcal{Y}_S, f_{\mathbf{\Lambda},\hat{\boldsymbol{w}}}(\tilde{\mathcal{X}}_T))$
for each $\mathbf{x}_t \in \mathcal{X}_T$ **do**
  $\hat{y}_t \leftarrow \arg\max f(\mathbf{x}_t)$
  $\mathbf{c}_{\text{tot}} \leftarrow [\,]$
  for each $c \in \mathcal{C}$ **do**
    $\mathcal{X}_S^{(c)} \leftarrow \mathcal{X}_S[\mathcal{Y}_S = c], \quad \texttt{fcost}^{(c)} \leftarrow \texttt{fcost}[\mathcal{Y}_S = c]$
    $\mathcal{N}_k \leftarrow \texttt{NN}_k(\mathbf{x}_t, \mathcal{X}_S^{(c)})$
    $c_f \leftarrow \frac{1}{k_c} \sum_{\mathbf{x}_s \in \mathcal{N}_k} \|\mathbf{x}_t - \mathbf{x}_s\|_2$
    $c_\ell \leftarrow \frac{1}{k_c} \sum_{\mathbf{x}_s \in \mathcal{N}_k} \texttt{fcost}^{(c)}(\mathbf{x}_s)$
    $\mathbf{c}_{\text{tot}}[c] \leftarrow c_f + \alpha \cdot c_\ell$
  end for
  $c^* \leftarrow \arg\min \mathbf{c}_{\text{tot}}$
  $\mathbf{m}_t \leftarrow 1[c^* = \hat{y}_t]$
end for
return $\hat{\mathcal{X}}_T^{\text{keep}} \leftarrow \hat{\mathcal{X}}_T[\mathbf{m}]$

Algorithm 7 Consistency-Based Rejection (DeepJDOT)

Input: $\mathcal{X}_S$, $\mathcal{Y}_S$, $\mathcal{X}_T$, $f_{\Lambda,w'}(g_{\Lambda_g,w'_g})$, k
Output: $\mathbf{X}_T^{\text{keep}}$

$\mathcal{C} \leftarrow \texttt{unique}(\mathcal{Y}_S)$
$\texttt{C0} = \texttt{ot.cdist}(g_{\mathbf{\Lambda_g},\boldsymbol{w'_g}}(\mathcal{X}_S), g_{\mathbf{\Lambda_g},\boldsymbol{w'_g}}(\mathcal{X}_T))$
$\texttt{fcost} = \texttt{ot.cdist}(\mathcal{Y}_S, f_{\mathbf{\Lambda},\boldsymbol{w'}}(g_{\mathbf{\Lambda_g},\boldsymbol{w'_g}}(\mathcal{X}_T)))$
$\texttt{C} = \alpha * \texttt{C0} + \texttt{fcost}$
for each $\mathbf{x}_t \in \mathcal{X}_T$ **do**
  $\hat{y}_t \leftarrow \arg\max f_{\mathbf{\Lambda},\boldsymbol{w'}}(g_{\mathbf{\Lambda_g},\boldsymbol{w'_g}}(x_t))$
  $\mathbf{c}_t \leftarrow \texttt{C}_{:,t}; \quad \mathbf{c}_{\text{tot}} \leftarrow [\,]$
  for each $c \in \mathcal{C}$ **do**
    $\mathbf{c}_t^{(c)} \leftarrow \{\mathbf{c}_t[i] \mid \mathcal{Y}_S[i] = c\}$
    $\mathcal{N}_k \leftarrow \texttt{argsort}(\mathbf{c}_t^{(c)})[:k]$
    $\mathbf{c}_{\text{tot}}[c] \leftarrow \frac{1}{k_c} \sum_{i \in \mathcal{N}_k} \mathbf{c}_t^{(c)}[i]$
  end for
  $c^* \leftarrow \arg\min \mathbf{c}_{\text{tot}}$
  $\mathbf{m}_t \leftarrow 1[c^* = \hat{y}_j]$
end for
return $\mathcal{X}_T^{\text{keep}} \leftarrow \mathcal{X}_T[\mathbf{m}]$

target samples, weighted by the optimal transport plan γ_0. Under the constraint $\gamma_0 \mathbf{1}_{n_t} = \mu_s$, if a source sample $\hat{\mathbf{x}}_i^s$ allocates most of its mass to a single target sample $\hat{\mathbf{x}}_j^t$, then $\mathbf{x}'^s_i$ becomes strongly aligned with $\hat{\mathbf{x}}_j^t$. Additionally, under the dual constraint $\gamma_0^\top \mathbf{1}_{n_s} = \mu_t$, if $\hat{\mathbf{x}}_j^t$ predominantly receives mass from $\hat{\mathbf{x}}_i^s$, it becomes less influenced by other target samples. Consequently, a classifier trained on $\mathbf{x}'^s_i$ is more likely to correctly predict $\hat{\mathbf{x}}_j^t$. To implement this idea, the algorithm first identifies confident source samples—those with the lowest entropy in their transport distributions (lines 17–20)—which typically concentrate their mass on a few target samples and are thus more representative. It then computes the entropy of the target-side normalized transport contributions from these confident sources (lines 21–22). Lower entropy suggests that a target sample is primarily supported by one or a few confident sources, making its classification more reliable. Since methods such as SinkhornLpl1 and EMDLaplace maintain class-wise structure, entropy filtering is performed on a per-class basis (lines 15–24). A target sample is retained if it is well-supported by confident source samples within a specific class (lines 25–27). The final retained target set $\hat{\mathcal{X}}_T^{\text{keep}}$ includes those that satisfy both filtering conditions (lines 28–29).

JDOT and DeepJDOT

Algorithm 6 and Algorithm 7 present the rejection strategies designed for JDOT and DeepJDOT, respectively. In contrast to OT-based methods that rely on geometric transport mappings, JDOT and DeepJDOT are grounded in the assumption that similar samples from the same class should reside close to each other. The underlying optimization objective seeks to learn a model whose predicted labels naturally align with this principle. Guided by this assumption, the rejection mechanism for these methods follows a class-consistency criterion: for a given target sample, its average cost (distance) to the k nearest source samples of its predicted class should be lower than that to samples from other classes in the feature-label space. If this condition is violated, the prediction is considered unreliable,

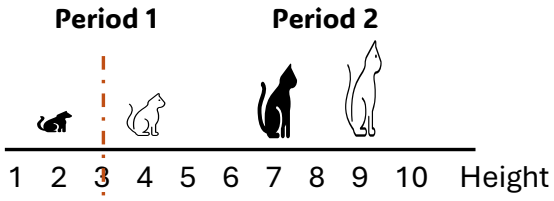


Figure 28: Intuition of Adaptation Evolution

and the corresponding target sample is rejected. The cost is calculated from the feature and label space (Eq. 14 and Eq.15). Compared with JDOT, DeepJDOT has a learnable feature representation, which makes it adjustable towards the optimization objective but also more sensitive to model quality. By contrast, JDOT has a fixed feature representation, which cost will not be dominated by the trained model. Given this and the assumption that samples with similar features are more likely to be from the same class, in the Algorithm 6 designed for JDOT, we select the cloest samples by the cost from the feature space first and then calculating the average cost to the target sample for each class (lines 7-11). Comparatively, for the DeepJDOT (Algorithm 7), we treat the cost from the feature and label space equally, and the nearest samples are decided by the whole cost from these two spaces (lines 9-11).

5.5.2 Explainability

Figure 28 illustrates the intuition behind examining malware evolution through the lens of domain adaptation. Assuming there are two groups of cats at period 1 (p1): black (b) and white (w), without seeing their colors, height is the only feature to distinguish their category. The boundary (orange dashed line) learned by a model from their height at this period correctly classifies them. With environmental changes, in period 2 (p2), the heights of all categories increase. Apparently, the model learned from p1 does not generalize to p2 anymore. Meanwhile, since all categries' height increases, relying on height similarity solely to study the relationship of two cats from different periods p1 and p2 is not sufficient. For example, the similarity result would make the evolution study focus on

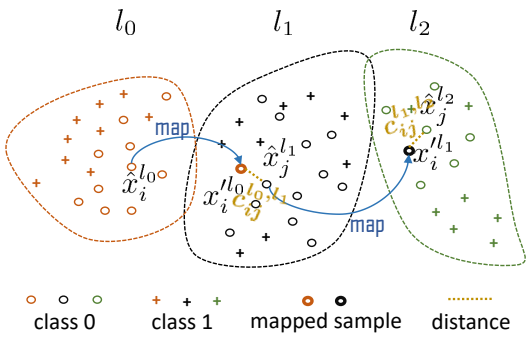


Figure 29: Adaptation lineage under OT

the comparison between both p1 b and w with p2 b. This indicates that before studying this relationship, the group evolution should be considered. The relationship study is expected to derive from the similarity results between p1 cat b/w and p2 cat b/w. The adaptation distribution alignment considers the group distribution difference, thus enabling the model learned from p1 samples to classify p2 samples correctly. Correspondingly, the relationship study between samples from different periods under distribution alignment would be more meaningful, since it considers the group evolution.

ELFMiner with OT

Malware evolution primarily analyzes the relationships between samples over a temporal sequence of years l_v, where $v \in \{0, \ldots, V\}$, and V denotes the total number of years under investigation. This temporal analysis can be decomposed into a series of consecutive year-pair studies: $(l_0, l_1), \ldots, (l_v, l_{v+1}), (l_{v+1}, l_{v+2}), \ldots, (l_{V-1}, l_V)$. In the context of adaptation mechanisms, we concentrate on identifying strongly related sample pairs within each yearly transition (l_v, l_{v+1}). Specifically, for a given adaptation step (l_v, l_{v+1}), we examine the relationship between a source sample $\hat{x}_i^{l_v} \in l_v$ and a target sample $\hat{x}_j^{l_{v+1}} \in l_{v+1}$. The location of the mapped source sample $x_i'^{l_v}$ depends on the particular OT adaptation mechanism. To ensure meaningful interpretation, we select the target sample $\hat{x}_j^{l_{v+1}}$ that (i) is

correctly classified under the adaptation, (ii) shares the same class as $\hat{x}_i^{l_v}$, and (iii) is closest to the mapped source $x_i'^{l_v}$ in the feature space. This criterion ensures that the selected target sample is positively influenced by the mapped source $x_i'^{l_v}$ under the adaptation. The process is repeated for each subsequent year pair (l_{v+1}, l_{v+2}), with the new source samples restricted to the previously selected target samples from l_{v+1}. This iterative selection results in a continuous chain of highly related samples from l_0 to l_V, referred to as a *adaptation lineage*. The primary focus of our analysis lies in examining the node/sample relationships along this lineage. An illustration of this lineage is provided in Figure 29. Each lineage can be evaluated based on two criteria: the cumulation of the distance $c_{ij}^{l_v,l_{v+1}}$ between each pair $(x_i'^{l_v}, \hat{x}_j^{l_{v+1}})$, and the prediction entropy $\mathcal{E}_j^{l_{v+1}}$ associated with $\hat{x}_j^{l_{v+1}}$. The lineage that minimizes both metrics is considered the most representative *adaptation lineage* of the given OT adaptation mechanism. We mainly study the top R most representative *adaptation lineage*.

The features extracted by ELFMiner are grouped into eight categories (Section 5.4), each representing a distinct structural component of ELF files. Along each selected *adaptation lineage*, for every pair (l_v, l_{v+1}), we examine both the feature groups retained by ELFMiner and the group-wise similarities between paired samples. Unlike JDOT and DeepJDOT, OT-based adaptation consists of two distinct stages: barycentric mapping and classifier training. Thus, the relationship between sample pairs $(\hat{x}_i^{l_v}, \hat{x}_j^{l_{v+1}})$ can be analyzed from multiple perspectives: (1) the group-wise similarity between the mapped source $x_i'^{l_v}$ and the target $\hat{x}_j^{l_{v+1}}$, which reveals the contribution of each feature group to the adaptation performance; (2) the group-wise similarity between the original source $\hat{x}_i^{l_v}$ and the target $\hat{x}_j^{l_{v+1}}$, which reveals the structural similarities or divergences that characterize the evolutionary transitions between the most closely related source-target samples along the lineage; and (3) the association between group-wise similarities of $(\hat{x}_i^{l_v}, \hat{x}_t^{l_{v+1}}), t \in \{1, m\}$ and the transport mass contributions from $\hat{x}_t^{l_{v+1}}$ to $\hat{x}_i^{l_v}$ based on γ_0, which helps assess

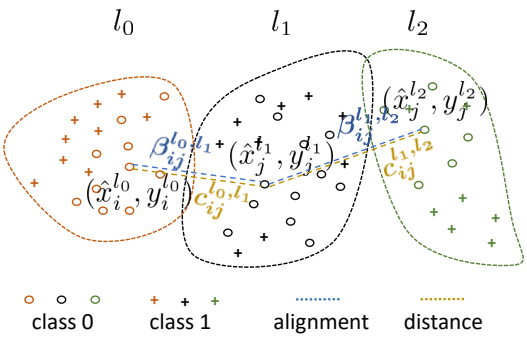


Figure 30: Adaptation lineage under JDOT/DeepJDOT

the structural similarity influences on the mapping process under specific OT adaptation settings.

ELFMiner with JDOT

JDOT aims to align the joint distribution directly, and incorporates the generation of f into the alignment objective (Eq. 14). With the predicted/known target labels $\hat{\mathcal{Y}}_T$, we could derive the transport plan γ_0, which tells the exact measure to align the distribution $p_l(x, y)$ and $p_{l+1}(x, y)$. Each γ_{ij} reflects how much responsibility or contribution $x_i^{l_v}$ has in representing $x_j^{l_{v+1}}$ to make the source and target distributions align. Given this, different from OT, the representative adaptation lineage for JDOT will be mainly ranked and selected by the contribution ratio, i.e., $\sum_{v=1}^{V-1} \beta_{ij}^{l_v,l_{v+1}}$, where $\beta_{ij}^{l_v,l_{v+1}} = \gamma_{0,ij}^{l_v,l_{v+1}} / \sum_{j=1}^{m} \gamma_{0,ij}^{l_v,l_{v+1}}$, and secondary sorted by the distance and entropy metrics. This aligns with the adaptation lineage's purpose. Additionally, unlike the OT, the nodes constructing the lineage are their original samples, rather than the mapped version. Figure 30 illustrates an example of an adaptation lineage identified under JDOT. Since the feature extraction process remains consistent with that described in Section 5.5.2, the feature analysis follows the same methodology but is restricted to the perspective outlined in point (2).

MalConv with DeepJDOT

DeepJDOT adopts the same adaptation principle as JDOT, with the primary distinction being the flexibility of its feature space. Accordingly, the strategy used to trace adaptation lineages remains consistent with the methodology described in Section 5.5.2. The key difference in this framework lies in the feature extraction process. Specifically, given the learned model $f_{\Lambda,w'}(g_{\Lambda_g,w'_g})$, the output of the feature extractor g_{Λ_g,w'_g} enables identification of structural regions in the byte file that serve as input to the classifier $f_{\Lambda,w'}$. As shown in Figure 27b, each convolutional filter can be traced back to a specific region in the binary, with each region covering 500 bytes, and up to 128 regions being marked in total. For a sample pair $(x_i^{l_v}, x_j^{l_{v+1}})$, the highlighted regions identified by the filters for $x_i^{l_v}$ may differ from those of $x_j^{l_{v+1}}$. Similar to ELFMiner, these regions can be categorized based on structural components; however, unlike ELFMiner, the structural parts containing the highlighted regions of $x_i^{l_v}$ and $x_j^{l_{v+1}}$ may not necessarily correspond. Moreover, while the filter-mapped regions can theoretically cover all structural parts of the binary, to ensure consistency with ELFMiner, our analysis is restricted to the structural components that ELFMiner targets. The explanation in MalConv focuses on the structure part frequency calculated from the filter activations. Therefore, the relationship between sample pairs is analyzed using *structure importance similarity*, which compares the structural frequency vectors of the two samples. Each vector quantifies the number of filter activations associated with each structural component, offering insight into shifts in structure-level importance over time. This aligns with the objective of analysis point (2) introduced in Section 5.5.2 for ELFMiner.

5.6 Experiment Setup

5.6.1 Dataset

The evaluation is carried out on the IoTPot dataset [118], which is publicly available, regularly updated, and specifically curated for the IoT domain [129, 105]. The dataset is preprocessed to align with the objectives of this study, minimize potential confounding factors, and meet the requirements of the ELF-OT framework. Specifically, 1) this study targets the challenge of malware concept drift. To capture this temporal dynamic, samples are partitioned by year. To ensure the evaluation reflects the most recent malware landscape, only samples collected from 2020 to 2024 are included; 2) to mitigate other confounding variables—such as architectural bias [129] and spatial bias [121]—only samples from a single architecture are used, and class distributions are maintained across different years; 3) since ELFMiner extracts features from ELF files, the analysis excludes packed samples identified by VirusTotal. Additional preprocessing includes filtering out non-ELF files via LIEF [148], removing duplicate and unknown samples, retrieving metadata from VirusTotal, and assigning labels using AVClass [133]. Meanwhile, due to input size limitations of MalConv, samples larger than 300KB are excluded, in accordance with the file size distribution observed from the dataset and the statistics from [49]. After processing, three malware families—*Mirai*, *Gafgyt*, and *Tsunami*—remain, with a total of 15,236 samples, each has samples distributed over the selected years. To ensure spatial consistency, a sampling ratio of 120:18:1 is applied, which is the available ratio across each year. It also reflects the class imbalance inherent to the malware landscape. To address potential sampling bias, five distinct datasets are generated, with an average sample overlap rate of 0.64. The sample distribution across these datasets is summarized in Table 11.

Table 11: Sample distribution over years

Year	Mirai	Gafgyt	Tsunami
2020	720	108	6
2021	3000	450	25
2022	720	108	6
2023	480	72	4
2024	240	36	2

5.6.2 Experiment Focus

The experiments are designed to address three core objectives: evaluating the overall performance of each ELF-OT framework, assessing the impact of the proposed rejection mechanism, and interpreting malware evolution through the lens of the frameworks. These correspond directly to the three primary goals of this study, and are detailed in the following Section 5.7, Section 5.8, and Section 5.9 respectively.

Although ELFMiner and MalConv are tailored for the malware domain, prior evaluations have predominantly assumed a standard learning scenario ($p_{\mathcal{S}}(x, y) = p_{\mathcal{T}}(x, y)$). In contrast, optimal transport (OT) methods are specifically designed to handle domain shifts, where the joint distributions differ ($p_{\mathcal{S}}(x, y) \neq p_{\mathcal{T}}(x, y)$), a situation that aligns with the concept drift problem frequently encountered in real-world malware defense. However, such OT-based methods have not yet been systematically examined within the malware domain under concept drift settings. To bridge this gap, the first set of experiments evaluates the ELF-OT framework's ability to address concept drift in IoT malware scenarios. Building on the theoretical foundations of each OT adaptation strategy, we propose rejection mechanisms in Section 5.5.1 to enhance the reliability of target predictions. Accordingly, the second set of experiments investigates the influence of these mechanisms on the framework performance, which also provides empirical support for their utility and offers guidance for future research in improving ELF-OT adaptation. The evolution tracking is important in the malware domain to understand the changes of the malware landscape over

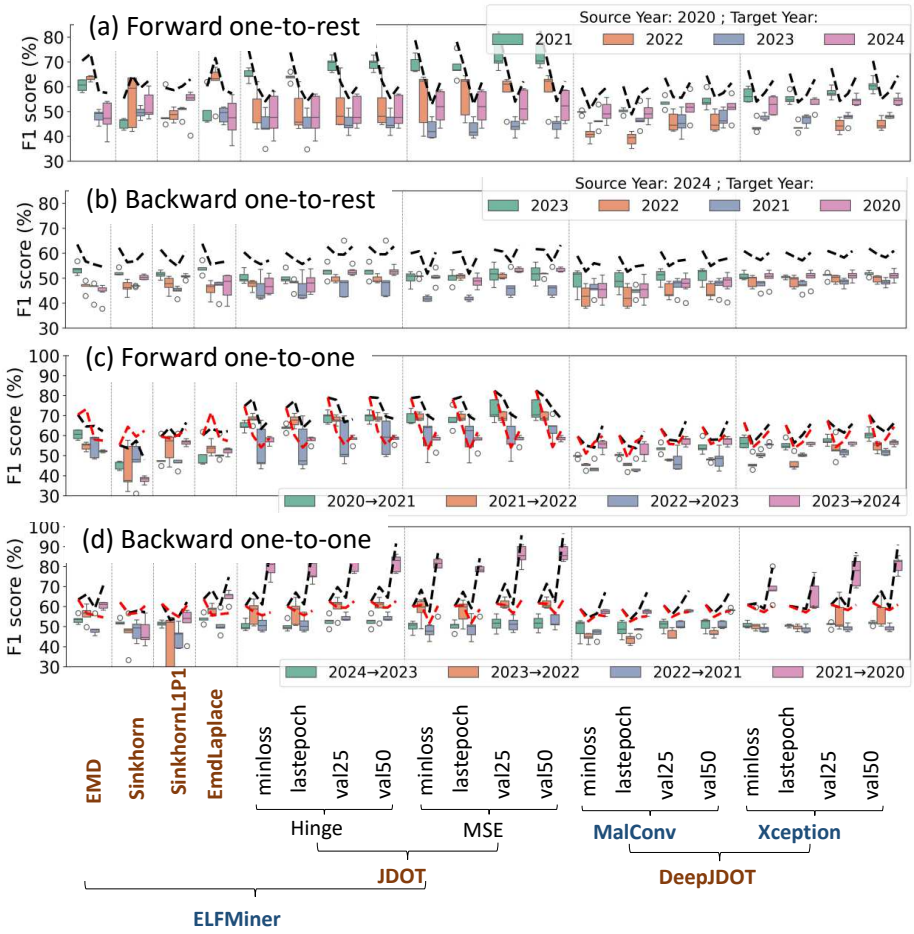
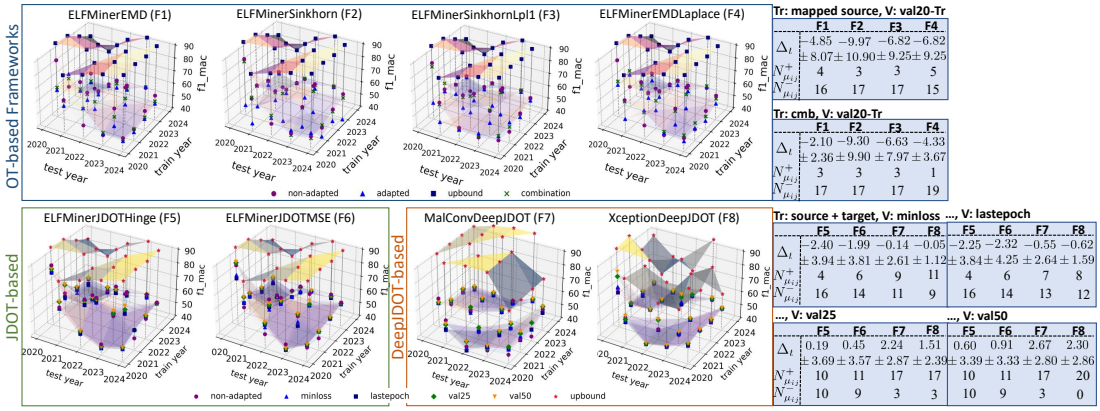


Figure 31: Temporal direction study.

time. Different adaptations yield distinct transport plans [48], which, in turn, affect the inferred relationships among samples. Thus, we designed the interpretation mechanism tailored to the adaptation lineage of each ELF-OT variant in Section 5.5.2. Section 5.9 uses these lineage insights to study the evolution patterns of malware. Given the class imbalance inherent in the malware landscape, we adopt the macro-averaged F1 score as the primary performance metric of each experiment.



Tr: mapped source, V: val20-Tr

	F1	F2	F3	F4
Δ_t	−4.85 ± 8.07	−9.97 ± 10.90	−6.82 ± 9.25	−6.82 ± 9.25
$N^+_{\mu_{ij}}$	4	3	3	5
$N^-_{\mu_{ij}}$	16	17	17	15

Tr: cmb, V: val20-Tr

	F1	F2	F3	F4
Δ_t	−2.10 ± 2.36	−9.30 ± 9.90	−6.63 ± 7.97	−4.33 ± 3.67
$N^+_{\mu_{ij}}$	3	3	3	1
$N^-_{\mu_{ij}}$	17	17	17	19

Tr: source + target, V: minloss ..., V: lastepoch

	F5	F6	F7	F8	F5	F6	F7	F8
Δ_t	−2.40 ± 3.94	−1.99 ± 3.81	−0.14 ± 2.61	−0.05 ± 1.12	−2.25 ± 3.84	−2.32 ± 4.25	−0.55 ± 2.64	−0.62 ± 1.59
$N^+_{\mu_{ij}}$	4	6	9	11	4	6	7	8
$N^-_{\mu_{ij}}$	16	14	11	9	16	14	13	12

..., V: val25 ..., V: val50

	F5	F6	F7	F8	F5	F6	F7	F8
Δ_t	0.19 ± 3.69	0.45 ± 3.57	2.24 ± 2.87	1.51 ± 2.39	0.60 ± 3.39	0.91 ± 3.33	2.67 ± 2.80	2.30 ± 2.86
$N^+_{\mu_{ij}}$	10	11	17	17	10	11	17	20
$N^-_{\mu_{ij}}$	10	9	3	3	10	9	3	0

Figure 32: Performance comparison across year-pairs.

5.7 Framework Evaluation

Unlike other adaptation scenarios (e.g., image adaptation from Amazon to Webcam), concept drift emphasizes temporal distribution shifts, where the primary goal is for a model trained on earlier data to generalize effectively to future data. This temporal progression reflects realistic deployment conditions and is central to evaluation constraints such as those proposed in [121]. Meanwhile, other studies, including [65], have also considered the reverse scenario, where models trained on more recent data are evaluated on earlier datasets. Based on these perspectives, our first performance evaluation examines the ELF-OT framework's adaptation effectiveness across time, in both *forward* and *backward* directions. Additionally, we investigate two adaptation configurations: *one-to-rest*, where a model trained on samples from a single year l_{v_i} adapts to all other years $l_{v_j}, j \neq i$, and *one-to-one*, where adaptation is performed between specific year pairs $(l_{v_i}, l_{v_{i+1}})$. Figure 31 illustrates the performance of the ELF-OT frameworks under these timeline-based adaptation settings. To provide a comprehensive comparison, we also report the performance of JDOT and DeepJDOT under different model selection strategies. Each boxplot shows the distribution of adaptation performance across five datasets for each adaptation scenario. The black dashed line represents the average adaptation performance over scenarios. For visual clarity, their presented values are offset by 10.

Intuitively, in the *one-to-rest* setting, one might expect better performance when adapting to the closer following year, given that sample characteristics evolve gradually due to software updates, enhanced obfuscation techniques, and other temporal factors. However, as shown in Figure 31 (a) and (b), such monotonic patterns of increasing or decreasing performance with respect to time gaps are rarely observed. This suggests that the complexity of distribution shifts arises not only from feature evolution but also from the diverse composition of samples within each dataset, making adaptation dynamics more intricate than purely temporal distance. Figure 31 (c) and (d) show the adaptation performance of the *one-to-one* setting. The red dashed line of these two figures represents the average adaptation performance trend of *one-to-rest* setting for comparison. In pairwise adaptation, the key interest lies in whether certain year pairs consistently outperform others, or whether these pairs' performance follows the same order. These comparisons offer insights into the similarity between specific year pairs. However, the observed adaptation performance trends shown in these two figures vary across different ELF-OT frameworks, indicating that the effectiveness of adaptation is strongly tied to the underlying adaptation mechanism. One promising observation is that, in most scenarios, the adaptation performance under the *one-to-one* setting surpasses the average performance observed in the *one-to-rest* setting. This suggests that targeted pairwise adaptation is generally more effective, and supports the notion that adapting by temporally closer years yields better results than adaptation by ones with larger temporal gaps.

The second experiment aims to compare the performance of these adaptation frameworks against two baselines: (1) non-adapted models trained solely on the source domain, which serve to quantify the improvements achieved through adaptation (non-adapted); and (2) models trained directly on the target (testing) data, which represent an upper bound on performance and reflect the learning capacity of the model (upbound). To provide a more comprehensive view of adaptation performance, the evaluation includes scenarios beyond

timeline-based settings. Figure 32 shows the comparison results of each framework. Similarly, we compared different model selection strategies for JDOT and DeepJDOT. Meanwhile, given that OT-based frameworks produce both original and mapped source samples, we also evaluate the performance of models trained on the combined source dataset (cmb). The statistical values displayed beside mean: Tr indicates the samples for training the adapted model; V represents the adapted model selection strategy; $\Delta_t = \mu \pm \sigma$ where $\mu = \frac{1}{N_{V(V-1)}} \sum_{i=0}^{V} \sum_{j=0, j\neq i}^{V} \mu_{ij}$, and $\mu_{ij} = \frac{1}{N_D} \sum_{n_d=1}^{5} (F1^{i\to j}_{n_d,\texttt{adapted}} - F1^{i\to j}_{n_d,\texttt{non-adapted}})$; $F1^{i\to j}_{n_d,\texttt{adapted}}$ is the adaptation performance from scenario l_{v_i} to l_{v_j} within dataset n_d ; $F1^{i\to j}_{n_d,\texttt{non-adapted}}$ is the non-adapted model performance; $N^{+}_{\mu_{ij}} = \sum_{i=0}^{V} \sum_{j=0, j\neq i}^{V} I_{\mu_{ij}>0}$, $N^{-}_{\mu_{ij}} = \sum_{i=0}^{V} \sum_{j=0, j\neq i}^{V} I_{\mu_{ij}<0}$ where $I_{\mu_{ij}>0}$, $I_{\mu_{ij}<0}$ as 1-0 indicator of $\mu_{ij} > 0$ and $\mu_{ij} < 0$ respectively.

Some consistent observations across these frameworks: 1) the upbound performance confirms that all models possess the capacity to learn effectively from the testing dataset, and all other learning scenarios consistently perform worse, and 2) except for Xception-DeepJDOT with the `val50` model selection strategy, based on the results of $N_{\mu_{ij}}$, no framework demonstrates consistently superior adaptation performance across all year-pair scenarios. When comparing the three adaptation methods using the metric μ, frameworks based on DeepJDOT generally achieve higher adaptation performance, while OT-based frameworks yield the lowest. A similar trend is observed in metric σ, highlighting that OT-based adaptations suffer from greater performance variability across year-pairs. In terms of $N_{\mu_{ij}}$, OT-based adaptation has fewer than one-fourth of the scenarios with better adaptation performance, regardless of exact frameworks. This slightly gets improved under JDOT-based frameworks, especially with `val`-based selection strategy, and it is further improved in DeepJDOT-based frameworks, where less than one quarter non-adaptation performance is better than the adaptation in the `val`-based selection strategy. It is worth noting that DeepJDOT uses a different feature extraction process compared to JDOT and

OT-based frameworks, so conclusions about adaptation effectiveness should be drawn cautiously. However, JDOT and OT-based methods share the same feature extractors, making the observed differences in μ and σ a strong indicator of JDOT's relative effectiveness and stability. Additional observations include: 1) incorporating both original and mapped source samples slightly improves adaptation performance, and 2) `val`-based model selection consistently yields better performance than `minloss` and `lastepoch` in terms of μ and $N_{\mu_{ij}}$ for both JDOT and DeepJDOT frameworks.

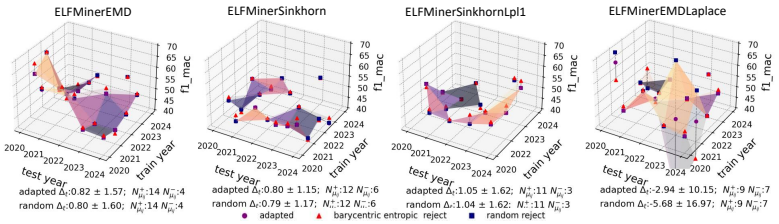
5.8 Rejection Impacts

This section evaluates the impacts of the proposed rejection mechanisms. The evaluation is from three perspectives: the influence of rejection on overall performance, the relationship between the ratio of rejected samples and the resulting performance change, and the distribution of rejected samples.

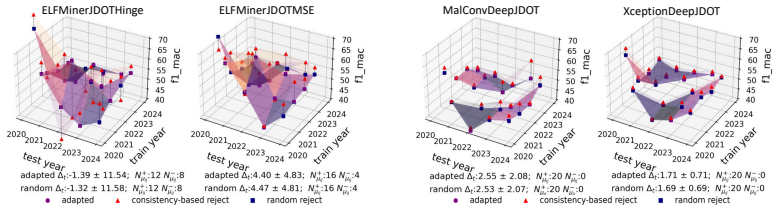
5.8.1 Performance Comparison

To ensure a fair comparison with OT-based adaptation, where no labeled target samples are used for model generation, the evaluation of the JDOT/DeepJDOT-based frameworks in this section focuses on models selected by the `minloss` strategy. Since the temporal analysis would be similar to that in Section 5.7, this section emphasizes performance comparisons across year pairs. The evaluation focuses on the performance of each framework on datasets that exclude the samples filtered by the proposed rejection mechanisms. Two perspectives are considered: (1) the performance change induced by applying the rejection mechanisms, and (2) a comparison against randomly rejected samples, where the number of rejected samples is equal to that filtered by the rejection strategy. To mitigate selection bias, random rejection is repeated ten times, and the average performance

across these runs is utilized. Figure 33 presents the rejection impacts on framework performance. The statistical values shown below each figure are defined as: $\Delta_t = \mu \pm \sigma$, where $\mu = \frac{1}{N_{V(V-1)}} \sum_{i=0}^{V} \sum_{\substack{j=0 \\ j \neq i}}^{V} \mu_{ij}$ and $\mu_{ij} = \frac{1}{N_D} \sum_{n_d=1}^{5} \left(F1^{i \to j}_{n_d,\texttt{reject}} - F1^{i \to j}_{n_d,\texttt{adapted/random}} \right)$. Here, $F1^{i \to j}_{n_d,\texttt{reject}}$ denotes the performance after applying the rejection mechanism, while $F1^{i \to j}_{n_d,\texttt{random}}$ corresponds to the performance under random rejection. $N^{+}_{\mu_{ij}} = \sum_{i=0}^{V} \sum_{\substack{j=0 \\ j \neq i}}^{V} I_{\mu_{ij}>0}$, $N^{-}_{\mu_{ij}} = \sum_{i=0}^{V} \sum_{\substack{j=0 \\ j \neq i}}^{V} I_{\mu_{ij}<0}$, where $I_{\mu_{ij}>0}$ and $I_{\mu_{ij}<0}$ are their indicator functions, respectively. The rejection mechanisms introduced in Section 5.5.1 include several parameters. Meanwhile, Figure 33 only includes the experiments with rejection rato $\alpha_r < 0.9$, where $(\alpha_r = 1 - |\mathcal{X}_T^{\text{keep}}|/|\mathcal{X}_T|)$.



(a) OT-based frameworks



(b) JDOT-based frameworks (c) DeepJDOT-based frameworks

Figure 33: Rejection impacts on performance across year-pairs.

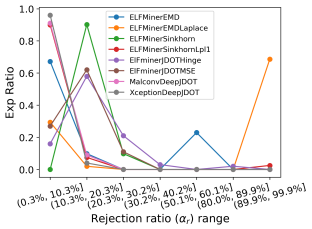


Figure 34: Experiment ratio.

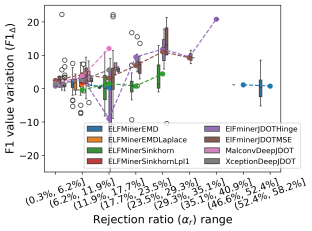


Figure 35: F1 variation.

Figure 33a presents the results of the Barycentric Entropic Rejection method (Algorithm 5) designed for the OT-based adaptation frameworks. From the reported values of `adapted` $N^{+}_{\mu_{ij}}$ and $N^{-}_{\mu_{ij}}$, we observe that framework behaviors vary, and no single framework consistently improves across all year-pair scenarios after the application of the rejection mechanism. However, the consistent inequality $N^{+}_{\mu_{ij}} > N^{-}_{\mu_{ij}}$ across all frameworks—especially the pronounced gap observed in all but ELFMinerEMDLaplace—supports the overall effectiveness of the proposed mechanism. Furthermore, the values of `adapted` μ

indicate stable performance gains in all frameworks except ELFMinerEMDLaplace. This pattern also holds when comparing against random rejection. Since the random rejection outcomes are averaged over multiple trials to eliminate selection bias, they serve as a robust baseline, further validating the utility of the proposed rejection strategy.

Figure 33b presents the results of the Consistency-Based Rejection method (Algorithm 6) designed for the JDOT-based adaptation frameworks. Consistent with the observations in Figure 33a, no framework demonstrates uniform performance gains across all year-pair adaptations after applying the rejection mechanism. However, the notable gap between `adapted` $N^+_{\mu_{ij}}$ and $N^-_{\mu_{ij}}$ confirms the general effectiveness of the mechanism in enhancing adaptation performance. Additionally, the comparison with the `random` rejection baseline—reflected by the corresponding values of $N^+_{\mu_{ij}}$ and $N^-_{\mu_{ij}}$—further validates the utility of the proposed sample selection strategy. Particularly noteworthy is the ELFMinerJDOT framework using the MSE loss, which achieves substantial improvements, as evidenced by the large margin between $N^+_{\mu_{ij}}$ and $N^-_{\mu_{ij}}$ and the high μ values, regardless of whether compared to adapted or random rejection. In contrast, performance under the Hinge loss function in ELFMinerJDOT is considerably less stable, as indicated by the large σ values.

Compared to the previous results, the Consistency-Based Rejection method (Algorithm 7) designed for DeepJDOT-based adaptation frameworks demonstrates the most favorable performance, as shown in Figure 33c. Specifically, the comparison of $N^+_{\mu_{ij}}$ and $N^-_{\mu_{ij}}$ reveals that all evaluated frameworks consistently achieve performance improvements across all year-pair adaptation scenarios. Furthermore, the relatively low values of σ indicate stable performance across these scenarios. These findings support the effectiveness and reliability of the proposed rejection mechanism for the DeepJDOT-based frameworks and its potential to generalize well to other similar adaptation settings.

5.8.2 Rejection Performance *vs* Rejection Ratio

In the above rejection experiments, the number of samples rejected by each mechanism varies with the specific parameter settings, and no parameter in each mechanism to control the rejection count/ratio as the Algorithms shown in Seciton 5.5.1. Therefore, this section focuses on analyzing the implicit relationship between the rejection ratio (α_r) and the variation in F1 score ($F1_{\Delta} = F1_{\texttt{reject}} - F1_{\texttt{adapted}}$). Here, we ignore the specific adaptation scenarios and datasets, and category all the experiments of each framwork by α_r range.

Based on the rejection performance results in Section 5.8.1, we first analyze the distribution of experiments across different α_r ranges for each framework, aiming to understand framework-based rejection behavior. As shown in Figure 34, with the exception of ELFMinerEMD and ELFMinerEMDLaplace, nearly all experiments in the remaining framework-rejetion pairs result in α_r values below 40%. For the DeepJDOT-based frameworks in particular, nearly 90% of experiments fall within the α_r range of $(0.3\%, 10.3\%]$. Combined with the performance improvements observed in Section 5.8.1, this confirms the effectiveness of the proposed rejection mechanisms with the selected parameter settings—demonstrating that performance can be improved while retaining the majority of samples across nearly all experiments. JDOT-based frameworks rank second, with approximately 80% of experiments rejecting fewer than 20.3% of samples. Notably, ELFMinerJDOTMSE achieves both substantial performance gains and low average rejection ratios. Among the OT-based rejections, ELFMinerSinkhornLpl1 achieves the best across both metrics. In contrast, `ELFMinerEMDLaplace` shows the weakest performance, with around 70% of its experiments rejecting nearly all samples and overall weak performance. The strong alignment between rejection performance and the average rejection ratio across frameworks with integrated rejection mechanisms indicates a close relationship between these two aspects. Specifically, when a framework-rejection pair discards a larger number of samples, its rejection performance tends to be lower.

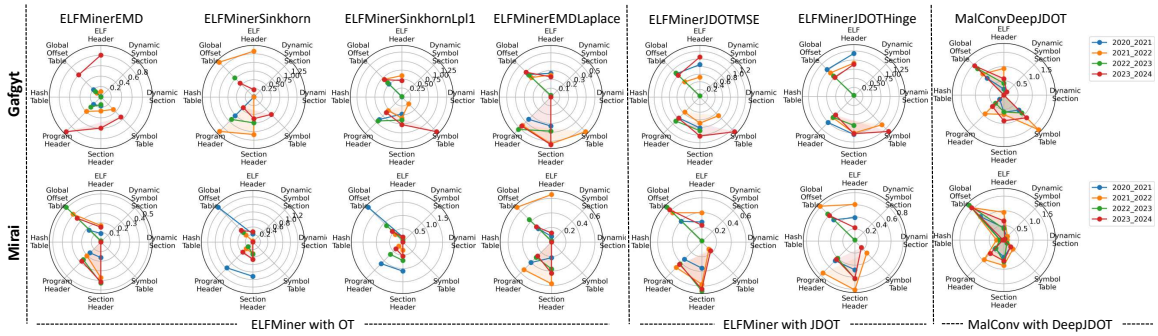


Figure 36: Group-wise similarity

Next, we analyze the variation of $F1_\Delta$ with respect to α_r for experiments with $\alpha_r < 90\%$ under each framework. Figure 35 illustrates this relationship. The boxplots depict the distribution of $F1_\Delta$, while the line plots show the mean $F1_\Delta$ within each α_r range for each framework–rejection mechanism pair. Based on the trends observed in the mean value curves, we conclude that for a given framework–rejection pair, rejection performance tends to improve as the rejection ratio increases. Additionally, the boxplots suggest that the variance of $F1_\Delta$ becomes larger with higher α_r values. Although, as indicated in Figure 34, the number of experiments in higher α_r bins decreases, the overall trend suggests that $F1_\Delta$ becomes less stable with increasing rejection, but it generally trends higher in magnitude.

5.9 Evolution Interpretation

Due to the limited number of Tsunami samples, the evolution interpretation primarily focuses on the Gafgyt and Mirai families. Following the adaptation lineage construction methods described in Section 5.5.2, the number of possible lineages for these two families reaches approximately the trillion scale. To make the analysis tractable, a primary evaluation metric is first applied to identify the top 1000 adaptation lineages. Subsequently, secondary metrics are used to rank these lineages and to identify those consistently appearing within the top 90% of the selected set, which regarded as the top R most representative

adaptation lineages. The specific metrics employed for each framework are outlined in their corresponding sections.

5.9.1 ELFMiner

OT-based

To maintain consistency with the analysis of JDOT and DeepJDOT, the relationship of framework ELFMinerOT focuses on the the perspective (2) introduced in Section 5.5.2, specifically, the group-wise similarity between the original source $\hat{x}_i^{l_v}$ and the target sample $\hat{x}_j^{l_{v+1}}$. The primary and secondary evaluation metrics employed are the accumulated distance and the prediction entropy, respectively. Figure 36 presents the average group-wise similarity for each year pair (l_v, l_{v+1}) of each OT-based framework, calculated along their most representative adaptation lineages. In ELFMiner, features are filtered based on information gain after extraction, with the retained features depending on the sample distribution from year l_v. According to the results shown in Figure 36, the retained feature composition exhibits minimal variation across different years l_v. In particular, structural components such as HashTable and DynamicSection consistently have no remaining features across all l_v. Conversely, structural parts including ELFHeader, GlobalOffsetTable, ProgramHeader, and SectionHeader retain features almost consistently across all years, irrespective of the malware class. Additionally, features from the SymbolTable are found to be informative for $l_v = 2021$ in the Gafgyt family and for $l_v = 2023$ across both families.

Next, we analyze the group-wise similarity along the adaptation lineages. To ensure generalizable evolution conclusions, we primarily summarize patterns consistent across most frameworks. The analysis focuses on the structural components consistently preserved across year pairs, specifically ELFHeader, GlobalOffsetTable, ProgramHeader, and SectionHeader. The similarity of each feature group under each year pair is evaluated using the average evolutionary gap between corresponding sample pairs along the representative

adaptation lineages. For the Gafgyt results, ELFHeader consistently exhibits the lowest average gap values, while ProgramHeader shows the highest. This observation is consistent across all OT-based adaptation methods, suggesting that ELFHeader features remain relatively similar, whereas ProgramHeader features undergo significant changes along the adaptation lineages. In the Mirai results, ELFHeader similarly displays the highest similarity, while GlobalOffsetTable exhibits the lowest. This trend also remains consistent across all OT-based adaptation approaches. Additionally, the spread or concentration of gap values within each group reflects the degree of evolutionary variability across adaptation lineages. A highly concentrated distribution indicates minimal variability in group similarity over time, while greater diversity implies significant fluctuation across years. For both malware families, the group with the minimum variability differs across OT-based adaptation methods, but the group with the maximum variability remains consistent—ELFHeader for Gafgyt and approximately GlobalOffsetTable for Mirai.

JDOT-based

As introduced in Section 5.5.2, the primary evaluation metric of adaptation lineage for JDOT-based frameworks is the contribution ratio calculated from the γ_0, while the secondary is the accumulated distance and the prediction entropy. Figure 36 shows the results for JDOT-based frameworks. The JDOT-based frameworks share the same features as OT-based frameworks. Here, we mainly analyze the group-wise similarity results along their selected representative adaptation lineage. Although the lineage construction methods of JDOT differs from OT, but their results in terms of the average gap values and diversity have some common results. In detail, the group with minimum gap values for both families and the maximum for the Mirai family keep consistent as in the OT-based framework. In the Gafgyt results, the group showing consistent maximum gap values changes to SectionHeader. For the gap diversity, the ELFHeader is also the group exhibiting the highest

diversity for Mirai.

5.9.2 MalConv

The interpretation for MalConvDeepJDOT focuses on the structure importance similarity as explained in Section 5.5.2. Figure 36 shows the results. The structural regions mapped by MalConv potentially cover the entire ELF file, to maintain consistency with the previous analyses, we focus on the key structural parts, including ELFHeader, GlobalOffsetTable, ProgramHeader, and SectionHeader. The meaning of the group-wise similarity metric differs from the previous frameworks, as the MalConv interpretation is based on region importance derived from filter activations. Consequently, the average gap and diversity values exhibit distinct interpretations. In this context, the group with the lowest gap value reflects the structural component that maintains a highly similar importance across samples from l_v and l_{v+1}. A low average gap indicates that this structural group consistently makes closer contributions to model decisions across time along the adaptation lineage. Conversely, groups with higher average gap values indicate components where structural importance diverges across the timeline. Furthermore, the diversity of gap values provides insight into the consistency of a group's influence. A highly concentrated distribution (low diversity) suggests stable importance patterns for that structural component along adaptation lineages, while greater diversity implies variability in its influence over time. Based on the results from Figure 36, the structure part with the minimum and maximum average gap value is ELFHeader and GlobalOffsetTable, while the one with the minimum and maximum diversity is SectionHeader and ELFHeader. This conclusion also applies to both classes.

To further observe common patterns, we compute the Pearson [120] rank correlation coefficient between every pair of frameworks with respect to their ELF structural regions similarity rank–Figure 37. The predominantly positive correlation values and consistent ranking of Mirai indicate the findings' generalization and independence from specific frameworks.

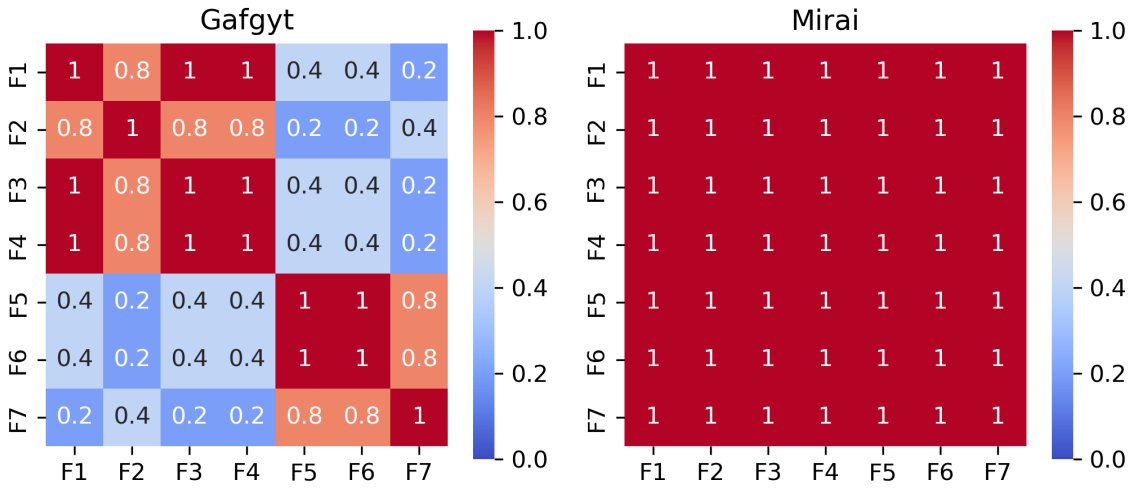


Figure 37: Pairwise Pearson correlation of ELF structural-region similarity across adaptation frameworks.

5.10 Conclusion

This work integrates optimal transport (OT)-based domain adaptation into IoT malware classification to address concept drift, a critical challenge in maintaining reliable machine learning-based defenses. Beyond aligning source and target distributions, OT-based methods provide explicit transport plans that reveal relationships between samples, offering unique insights into malware evolution. We design multiple OT-based frameworks incorporating diverse ELF-derived features and evaluate them under realistic conditions. Guided by adaptation theory, we propose rejection mechanisms that effectively filter unreliable predictions and improve performance. Furthermore, by leveraging the transport plan, we construct adaptation lineages to interpret malware evolution over time. The proposed frameworks and rejection mechanisms enhance both adaptability and interpretability, providing a foundation for advancing malware defense in dynamic environments.

Chapter 6

Conclusion and Future Work

The whole research aims to enhance the practicality of ML-based solutions in the IoT malware domain. To align with ongoing trends and ensure focus on the most pressing factors influencing real-world applicability, the study begins by analyzing current challenges and pitfalls that hinder solutions' applicability. Meanwhile, the research type is not restricted to engineering solutions. Instead, it also considers other categories, seeking to reveal insights from the malware landscape and existing solutions to benefit future research.

The combination of diverse feature-processing techniques for IoT malware samples and the wide variety of methods applied across stages of the ML pipeline has resulted in many proposed solutions. This diversity results in solutions' diversity, but it also creates an urgent need to evaluate these solutions against the stringent reliability requirements of the security field. In response to this urgency, the research starts by examining the vulnerabilities of existing solutions to adversarial machine learning (AML) attacks. To ensure that the exploration reflects realistic attack conditions, the first study was designed according to several principles: selecting target models developed and evaluated under practical scenarios; choosing and designing attack strategies reflecting real-world adversaries; and evaluating attack performance under different knowledge assumptions, ranging from white-box to gray-box settings. This stepwise evaluation approach allows a gradual demonstration of

the effectiveness of the attack strategy, the revealing of the vulnerabilities of the models, and the insights relevant to both attackers and defenders. Through this research process, the goal is not only to propose a new solution but also to expose the threats posed by AML attacks to ML-based malware analysis models in the real-world scenario and to provide actionable insights that help future defenders better understand and mitigate potential risks.

The ultimate objective of solution development is successful deployment in real-world scenarios. The breadth of proposed solutions highlights many possible ways to combine IoT malware features with ML techniques. However, deployment remains constrained by factors such as resource limitations, malware landscape variability, and domain-specific requirements. These constraints raise critical questions: which solutions are best suited for deployment, and which requirements or factors may undermine performance? Driven by these questions, the second study analyzes the impact of fundamental malware landscape factors on existing solutions and assesses their resistance to these challenges, with the goal of generating insights to inform future development. The evaluation follows a hierarchical structure: factor analysis is performed using ablation principles, isolating individual factors to reduce bias and assess their effects independently. In parallel, solutions are examined individually rather than grouped by specific pipeline stages (e.g., feature representation). This ensures a fairer assessment, since solution design spans multiple stages with diverse methodological choices, making it unconvincing to categorize solutions solely by one stage and then conclude which category is superior. Unlike engineering-focused contributions, this research aligns more closely with observational studies and offensive analyses. Its purpose is to uncover characteristics of the malware landscape and highlight the strengths of existing solutions, thereby generating insights that can guide both the design of future defenses and the definition of more reliable evaluation frameworks.

The ongoing arms race between attackers and defenders drives the continuous evolution

of the malware landscape. This evolution gives rise to concept drift, a phenomenon that directly conflicts with the fundamental assumption of machine learning and inevitably leads to model degradation over time. This challenge is particularly critical given the common expectation that models trained on historical datasets will later be deployed in real-world settings to defend against unseen samples. Concept drift, in essence, refers to the distributional shift between training and testing data. In this context, domain adaptation techniques, which aim to bridge the gap between source and target distributions, hold strong potential. While these techniques have matured considerably—supported by solid theoretical foundations and a wide range of methods—their adoption in the IoT malware domain remains limited. This gap highlights the need for early attempts to adapt domain adaptation methods by tailoring them to the IoT malware features, comprehensively evaluating the performance of constructed solutions, and designing mechanisms to improve their predictive reliability. Moreover, the diversity of available techniques provides many options, but in the IoT malware domain, method selection must go beyond raw performance to also account for domain-specific requirements. The third study builds on this background and intuition, selecting techniques that consider other specific needs of the IoT malware domain, evaluating their effectiveness across multiple adaptation scenarios, and incorporating rejection mechanisms to enhance reliability. The results demonstrate the applicability of these techniques, highlight the potential of proposed rejection mechanisms in performance improvement, and provide new insights into the IoT malware analysis.

These three works are interconnected, with each study building on the insights and motivations of the previous one. The first study focused on practical target models that capture recent trends, providing a strong starting point for the overall research. By designing and evaluating adversarial attacks against these models, it demonstrated the threat posed by adversarial machine learning—showing, for example, that high attack success rates could be achieved without noticeably degrading overall model performance. This

finding also raised questions about whether existing models perform as well as assumed, even in the absence of deliberate tampering. Building on this insight, the second study examined challenges intrinsic to the malware landscape itself, highlighting how fundamental factors can degrade model performance. Among these, concept drift emerged as a critical challenge, one that must be carefully addressed before advancing to deeper concerns such as adversarial machine learning. Recognizing its importance, the third study investigated how adaptation techniques and rejection mechanisms can mitigate the effects of concept drift and strengthen the reliability of ML-based malware defenses. Meanwhile, the studies adopt different perspectives: the first approaches the problem from an attacker's viewpoint to uncover vulnerabilities, whereas the second and third take the perspective of classifier designers to benefit solution design to implement their basic objectives. Given the resource constraints inherent in IoT environments, the model design and evaluation primarily focus on shallow models, whose lower computational overhead and faster runtime enable them to have more potential for deployment in real-world scenarios.

The whole research is driven by the purpose of advancing ML-based solutions' practicality in the IoT malware domain. To achieve this, the research aims to capture the major concerns, adopts a broad and integrative perspective, and examines the problem from multiple angles. This broader scope allows a clearer understanding of how different challenges, factors, and solution strategies interact, and how each contributes to the overarching goal of developing practical solutions. Given the breadth of this research undertaken during the valuable PhD period, the subsequent work proceeds along two directions: deeper and wider. On the one hand, the topics already explored address major concerns in the field; therefore, part of the follow-up research will focus on these issues more deeply, like the Interaction between concept drift and AML attacks. On the other hand, in light of the related works studied so far, further investigations are also needed to expand into other relevant topics like optimized deep models, thereby broadening the overall scope.

Bibliography

[1] Analysis of variance. https://en.wikipedia.org/wiki/Analysis_of_variance. [Online; accessed Oct. 2024].

[2] Arm instruction set. https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf.

[3] Brickerbot. https://en.wikipedia.org/wiki/BrickerBot, 2023.

[4] Complex Instruction Set Computer. https://en.wikipedia.org/wiki/Complex_instruction_set_computer, 2024. [Online; accessed September-2024].

[5] Domain Adaptation – Wikipedia, The Free Encyclopedia, 2024. [Online; accessed March 3, 2025].

[6] Reduced Instruction Set Computer. https://en.wikipedia.org/wiki/Reduced_instruction_set_computer, 2024. [Online; accessed September-2024].

[7] Y. Aafer, W. Du, and H. Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *Security and Privacy in Communication Networks: 9th International ICST Conference, SecureComm 2013, Sydney, NSW, Australia, September 25-28, 2013, Revised Selected Papers 9*, pages 86–103. Springer, 2013.

[8] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194, 2016.

[9] M. N. Alenezi, H. Alabdulrazzaq, A. A. Alshaher, and M. M. Alkharang. Evolution of malware threats and techniques: A review. *International journal of communication networks and information security*, 12(3):326–337, 2020.

[10] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen. Efficient signature generation for classifying cross-architecture iot malware. In *2018 IEEE conference on communications and network security (CNS)*, pages 1–9. IEEE, 2018.

[11] K. Allix, T. F. Bissyandé, J. Klein, and Y. 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.

[12] K. Amarasinghe, K. Kenney, and M. Manic. Toward explainable deep neural network based anomaly detection. In *2018 11th international conference on human system interaction (HSI)*, pages 311–317. IEEE, 2018.

[13] H. S. Anderson and P. Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[14] I. Andrea, C. Chrysostomou, and G. Hadjichristofi. Internet of things: Security vulnerabilities and challenges. In *2015 IEEE symposium on computers and communication (ISCC)*, pages 180–187. IEEE, 2015.

[15] S. P. Anh and Y. Nakamura. A baseline investigation into the evolution and prevalence of mirai and hajime utilizing a network telescope. *IEEE Access*, 2024.

[16] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[17] S. Aonzo, G. C. Georgiu, L. Verderame, and A. Merlo. Obfuscapk: An open-source black-box obfuscation tool for android apps. *SoftwareX*, 11:100403, 2020.

[18] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti. On the effectiveness of machine and deep learning for cyber security. In *2018 10th international conference on cyber Conflict (CyCon)*, pages 371–390. IEEE, 2018.

[19] G. Apruzzese, P. Laskov, E. Montes de Oca, W. Mallouli, L. Brdalo Rapa, A. V. Grammatopoulos, and F. Di Franco. The role of machine learning in cybersecurity. *Digital Threats: Research and Practice*, 4(1):1–38, 2023.

[20] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, 2022.

[21] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[22] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE transactions on sustainable computing*, 4(1):88–95, 2018.

[23] J. Bacardit and J. M. Garrell. Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In *International Workshop on Learning Classifier Systems*, pages 59–79. Springer, 2003.

[24] F. Barbero, F. Pendlebury, F. Pierazzi, and L. 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.

[25] W. Beitz, G. Pahl, and K. Grote. Engineering design: a systematic approach. *Mrs Bulletin*, 71, 1996.

[26] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine learning*, 79:151–175, 2010.

[27] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. *Advances in neural information processing systems*, 19, 2006.

[28] E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary computation*, 11(3):209–238, 2003.

[29] S. Bose, T. Barao, and X. Liu. Explaining ai for malware detection: Analysis of mechanisms of malconv. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2020.

[30] M. Botacin, F. Ceschin, R. Sun, D. Oliveira, and A. Grégio. Challenges and pitfalls in malware research. *Computers & Security*, 106:102287, 2021.

[31] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[32] H. Cai. Assessing and improving malware detection sustainability through app evolution studies. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(2):1–28, 2020.

[33] H. Cai and B. G. Ryder. Understanding application behaviours for android security: A systematic characterization. Technical report, Department of Computer Science, Virginia Polytechnic Institute & State . . . , 2016.

[34] R. J. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.

[35] A. Cannarile, V. Dentamaro, S. Galantucci, A. Iannacone, D. Impedovo, and G. Pirlo. Comparing deep learning and shallow learning techniques for api calls malware prediction: a study. *Applied Sciences*, 12(3):1645, 2022.

[36] L. Cavallaro, J. Kinder, F. Pendlebury, and F. Pierazzi. Are machine learning models for malware detection ready for prime time? *IEEE Security & Privacy*, 21(2):53–56, 2023.

[37] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska. Tight arms race: Overview of current malware threats and trends in their detection. *IEEE Access*, 9:5371–5396, 2020.

[38] F. Ceschin, M. Botacin, A. Bifet, B. Pfahringer, L. S. Oliveira, H. M. Gomes, and A. Grégio. Machine learning (in) security: A stream of problems. *Digital Threats: Research and Practice*, 2020.

[39] F. Ceschin, M. Botacin, A. Bifet, B. Pfahringer, L. S. Oliveira, H. M. Gomes, and A. Grégio. Machine learning (in) security: A stream of problems. *Digital Threats: Research and Practice*, 5(1):1–32, 2024.

[40] R. Chaganti, V. Ravi, and T. D. Pham. Deep learning based cross architecture internet of things malware detection and classification. *Computers & Security*, 120:102779, 2022.

[41] Charles Price. Mips iv instruction set. `https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf`, September, 1995.

[42] K. Chen, S. Zhang, Z. Li, Y. Zhang, Q. Deng, S. Ray, and Y. Jin. Internet-of-things security and vulnerabilities: Taxonomy, challenges, and practice. *Journal of Hardware and Systems Security*, 2(2):97–110, 2018.

[43] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren. Android hiv: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, 2019.

[44] Y. Chen, Z. Ding, and D. Wagner. Continuous learning for android malware detection. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1127–1144, 2023.

[45] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[46] W. W. Cohen. Fast effective rule induction. In *Machine learning proceedings 1995*, pages 115–123. Elsevier, 1995.

[47] N. Courty, R. Flamary, A. Habrard, and A. Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. *Advances in neural information processing systems*, 30, 2017.

[48] N. Courty, R. Flamary, D. Tuia, and A. Rakotomamonjy. Optimal transport for domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1853–1865, 2016.

[49] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti. Understanding linux malware. In *2018 IEEE symposium on security and privacy (SP)*, pages 161–175. IEEE, 2018.

[50] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti. The tangled genealogy of iot malware. In *Annual Computer Security Applications Conference*, pages 1–16, 2020.

[51] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[52] B. B. Damodaran, B. Kellenberger, R. Flamary, D. Tuia, and N. Courty. Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 447–463, 2018.

[53] R. Darwish, M. Abdelsalam, and S. Khorsandroo. Deep learning based xiot malware analysis: A comprehensive survey, taxonomy, and research challenges. *arXiv preprint arXiv:2410.13894*, 2024.

[54] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[55] M. Dib, S. Torabi, E. Bou-Harb, and C. Assi. A multi-dimensional deep learning framework for iot malware classification and family attribution. *IEEE Transactions on Network and Service Management*, 18(2):1165–1177, 2021.

[56] M. D'Onghia, F. Di Cesare, L. Gallo, M. Carminati, M. Polino, and S. Zanero. Lookin'out my backdoor! investigating backdooring attacks against dl-driven malware detectors. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 209–220, 2023.

[57] M. Dundar, B. Krishnapuram, J. Bi, and R. B. Rao. Learning classifiers when the training data is not iid. In *IJCAI*, volume 2007, pages 756–61. Citeseer, 2007.

[58] J. Ferdous, R. Islam, A. Mahboubi, and M. Z. Islam. A review of state-of-the-art malware attack trends and defense mechanisms. *IEEe Access*, 11:121118–121141, 2023.

[59] I. Finder, E. Sheetrit, and N. Nissim. A time-interval-based active learning framework for enhanced pe malware acquisition and detection. *Computers & Security*, 121:102838, 2022.

[60] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[61] J. B. Fraley and J. Cannady. The promise of machine learning in cybersecurity. In *SoutheastCon 2017*, pages 1–6. IEEE, 2017.

[62] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization, 1998.

[63] L. Galante, M. Botacin, A. Grégio, and P. de Geus. Machine learning for malware detection: Beyond accuracy rates. In *Anais Estendidos do XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 47–56. SBC, 2019.

[64] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35, 2016.

[65] C. Gao, G. Huang, H. Li, B. Wu, Y. Wu, and W. Yuan. A comprehensive study of learning-based android malware detectors under challenging environments. In *Proc. of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024.

[66] Y. Gao, B. G. Doan, Z. Zhang, S. Ma, J. Zhang, A. Fu, S. Nepal, and H. Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv preprint arXiv:2007.10760*, 2020.

[67] D. Gibert, C. Mateu, and J. Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[68] B. Gill and S. Rao. Technology insight: Edge computing in support of the internet of things. *Gartner Research*, 2017.

[69] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[70] S. Greengard. Cybersecurity gets smart. *Communications of the ACM*, 59(5):29–31, 2016.

[71] A. Guerra-Manzanares. Android malware detection: mission accomplished? a review of open challenges and future perspectives. *Computers & Security*, page 103654, 2023.

[72] A. Guerra-Manzanares, M. Luckner, and H. Bahsi. Concept drift and cross-device behavior: Challenges and implications for effective android malware detection. *Computers & Security*, 120:102757, 2022.

[73] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing. Lemna: Explaining deep learning based security applications. In *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 364–379, 2018.

[74] W. Guo, B. Tondi, and M. Barni. An overview of backdoor attacks against deep neural networks and possible defences. *IEEE Open Journal of Signal Processing*, 2022.

[75] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46:389–422, 2002.

[76] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo. A deep recurrent neural network based approach for internet of things malware threat hunting. *Future Generation Computer Systems*, 85:88–96, 2018.

[77] S. Haiba and T. Mazri. Build a malware detection software for iot network using machine learning. In *Proceedings of the 4th International Conference on Networking, Information Systems & Security*, pages 1–8, 2021.

[78] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin. Measurement and analysis of hajime, a peer-to-peer iot botnet. In *Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[79] S. Hilt and F. Merces. Vpnfilter two years later: Routers still compromised, 2021.

[80] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CsUR)*, 51(6):1–36, 2019.

[81] Jason Firch. Mantis botnet: The largest ddos attack ever mitigated. `https://purplesec.us/breach-report/mantis-botnet/`, April 28, 2024.

[82] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[83] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX security symposium (USENIX security 17)*, pages 625–642, 2017.

[84] M. A. Kadri, M. Nassar, and H. Safa. Transfer learning for malware multi-classification. In *Proceedings of the 23rd International Database Applications & Engineering Symposium*, pages 1–7, 2019.

[85] L. V. Kantorovich. On the translocation of masses. In *Dokl. Akad. Nauk. USSR (NS)*, volume 37, pages 199–201, 1942.

[86] E. M. Karanja, S. Masupe, and M. G. Jeffrey. Analysis of internet of things malware using image texture features and machine learning techniques. *Internet of Things*, 9:100153, 2020.

[87] S. Kell, D. P. Mulligan, and P. Sewell. The missing link: explaining elf static linking, semantically. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 607–623, 2016.

[88] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

[89] C. Kim, S.-Y. Chang, J. Kim, D. Lee, and J. Kim. Automated, reliable zero-day malware detection based on autoencoding architecture. *IEEE Transactions on Network and Service Management*, 2023.

[90] J. Kim, D. Genkin, and K. Leach. Revisiting lightweight compiler provenance recovery on arm binaries. In *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*, pages 292–303. IEEE, 2023.

[91] W. M. Kouw and M. Loog. A review of domain adaptation without target labels. *IEEE transactions on pattern analysis and machine intelligence*, 43(3):766–785, 2019.

[92] W. M. Kouw and M. Loog. A review of domain adaptation without target labels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(3):766–785, 2021.

[93] M. Krčál, O. Švec, M. Bálek, and O. Jašek. Deep convolutional malware classifiers can learn from raw executables and labels only. *ICLR*, 2018.

[94] S. Kumar, P. Ahlawat, and J. Sahni. Iot malware detection using static and dynamic analysis techniques: A systematic literature review. *Security and Privacy*, 7(6):e444, 2024.

[95] Kyle Lefton and Larry Cashdollar. Active exploitation: New aquabot variant phones home. `https://www.akamai.com/blog/security-research/new-aquabot-mirai-variant-exploiting-mitel-phones?utm_source=chatgpt.com`, January 28, 2025.

[96] V. Le Pochat and W. Joosen. Analyzing cyber security research practices through a meta-research framework. In *Proceedings of the 16th Cyber Security Experimentation and Test Workshop*, pages 64–74, 2023.

[97] Y.-T. Lee, T. Ban, T.-L. Wan, S.-M. Cheng, R. Isawa, T. Takahashi, and D. Inoue. Cross platform iot-malware family classification based on printable strings. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 775–784. IEEE, 2020.

[98] A. S. Li, A. Iyengar, A. Kundu, and E. Bertino. Revisiting concept drift in windows malware detection: Adaptation to real drifted malware with minimal samples. Network and Distributed System Security (NDSS) Symposium, 2025.

[99] C. Li, X. Chen, D. Wang, S. Wen, M. E. Ahmed, S. Camtepe, and Y. Xiang. Backdoor attack on machine learning based android malware detectors. *IEEE Transactions on Dependable and Secure Computing*, 19(5):3357–3370, 2021.

[100] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[101] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[102] M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015.

[103] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[104] Y. Ma, S. Liu, J. Jiang, G. Chen, and K. Li. A comprehensive study on learning-based pe malware family classification methods. In *Proc. of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1314–1325, 2021.

[105] S. Madan, S. Sofat, and D. Bansal. Tools and techniques for collection and analysis of internet-of-things malware: A systematic state-of-art review. *Journal of King Saud University-Computer and Information Sciences*, 34(10):9867–9888, 2022.

[106] I. Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni. Anatomy of threats to the internet of things. *IEEE communications surveys & tutorials*, 21(2):1636–1675, 2018.

[107] S. Millar, N. McLaughlin, J. M. del Rincon, and P. Miller. Multi-view deep learning for zero-day android malware detection. *Journal of Information Security and Applications*, 58:102718, 2021.

[108] S. Mohammad and D. Morris. Texture analysis for glaucoma classification. In *2015 International Conference on BioSignal Analysis, Processing and Systems (ICBAPS)*, pages 98–103. IEEE, 2015.

[109] Mohammed Khalil. Iot hacking statistics 2025: The definitive report on threats, risks regulations. https://deepstrike.io/blog/iot-hacking-statistics, August 24, 2025.

[110] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):157–175, 2017.

[111] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7, 2011.

[112] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen. A survey of iot malware and detection methods based on static features. *ICT Express*, 6(4):280–286, 2020.

[113] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[114] A. Oprea, A. Singhal, and A. Vassilev. Poisoning attacks against machine learning: Can machine learning be trustworthy? *Computer*, 55(11):94–99, 2022.

[115] F. Osisanwo, J. Akinsola, O. Awodele, J. Hinmikaiye, O. Olakanmi, J. Akinjobi, et al. Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3):128–138, 2017.

[116] F. E. Otero, A. A. Freitas, and C. G. Johnson. c ant-miner: An ant colony classification algorithm to cope with continuous attributes. In *International Conference on Ant Colony Optimization and Swarm Intelligence*, pages 48–59. Springer, 2008.

[117] I. B. A. Ouahab, L. Elaachak, Y. A. Alluhaidan, and M. Bouhorma. A new approach to detect next generation of malware based on machine learning. In *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 230–235. IEEE, 2021.

[118] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow. {IoTPOT}: Analysing the rise of {IoT} compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[119] R. Pang, H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang. A tale of evil twins: Adversarial inputs versus poisoned models. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 85–99, 2020.

[120] K. Pearson. Vii. note on regression and inheritance in the case of two parents. *proceedings of the royal society of London*, 58(347-352):240–242, 1895.

[121] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX security symposium (USENIX Security 19)*, pages 729–746, 2019.

[122] T. N. Phu, L. H. Hoang, N. N. Toan, N. D. Tho, and N. N. Binh. Cfdvex: A novel feature extraction method for detecting cross-architecture iot malware. In *Proceedings of the 10th International Symposium on Information and Communication Technology*, pages 248–254, 2019.

[123] K. Popper. *The logic of scientific discovery*. Routledge, 2005.

[124] P. C. PRISMA. Transparent reporting of systematic reviews and meta-analyses. *Fluxograma Prisma Statement*, pages 41–58, 2015.

[125] J. R. Quinlan. Mdl and categorical theories (continued). In *Machine Learning Proceedings 1995*, pages 464–470. Elsevier, 1995.

[126] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[127] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas. Malware detection by eating a whole exe. In *Workshops at the thirty-second AAAI conference on artificial intelligence*, 2018.

[128] E. Raff and C. Nicholas. A survey of machine learning methods and challenges for windows malware classification. *arXiv preprint arXiv:2006.09271*, 2020.

[129] A. D. Raju, I. Y. Abualhaol, R. S. Giagone, Y. Zhou, and S. Huang. A survey on cross-architectural iot malware threat hunting. *IEEE Access*, 9:91686–91709, 2021.

[130] Saachi Gupta Ghosh. The state of enterprise iot, malware families in 2023. `https://ciosea.economictimes.indiatimes.com/news/security/the-state-of-enterprise-iot-malware-families-in-2023/105121376`, Nov 14, 2023.

[131] K. Saito, K. Watanabe, Y. Ushiku, and T. Harada. Mcd: Maximum classifier discrepancy for unsupervised domain adaptation. In *CVPR*, 2018.

[132] O. P. Samantray and S. N. Tripathy. Iot-malware classification model using byte sequences and supervised learning techniques. In *Next Generation of Internet of Things: Proceedings of ICNGIoT 2021*, pages 51–60. Springer, 2021.

[133] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero. Avclass: A tool for massive malware labeling. In *Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings 19*, pages 230–253. Springer, 2016.

[134] G. Severi, J. Meyer, S. Coull, and A. Oprea. Explanation-Guided backdoor poisoning attacks against malware classifiers. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1487–1504. USENIX Association, Aug. 2021.

[135] F. Shahzad and M. Farooq. Elf-miner: Using structural knowledge and data mining methods to detect new (linux) malicious executables. *Knowledge and information systems*, 30:589–612, 2012.

[136] O. Sharma, A. Sharma, and A. Kalia. Windows and iot malware visualization and classification with deep cnn and xception cnn using markov images. *Journal of Intelligent Information Systems*, 60(2):349–375, 2023.

[137] K. Shaukat, S. Luo, and V. Varadharajan. A novel machine learning approach for detecting first-time-appeared malware. *Engineering Applications of Artificial Intelligence*, 131:107801, 2024.

[138] K. Simonyan and A. Zisserman. Vgg-16. *arXiv preprint*, 2014.

[139] M. Smith, R. Krishnakumar, J. Lubars, S. Verzi, X. Zhou, and A. Goyal. All models are wrong, but some (times) are useful: Evaluating when machine learning models are useful for detecting novel malware in the wild. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2022.

[140] P. F. Specification. Tool interface standard (tis) portable formats specification. 1993.

[141] A. Srivastava, S. Gupta, M. Quamara, P. Chaudhary, and V. J. Aski. Future iot-enabled threats and vulnerabilities: State of the art, challenges, and future prospects. *International Journal of Communication Systems*, 33(12):e4443, 2020.

[142] T. I. Standard. Executable and linking format (elf) specification version 1.2. *TIS Committee*, pages 1–106, 1995.

[143] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai. Lightweight classification of iot malware based on image recognition. In *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)*, volume 2, pages 664–669. IEEE, 2018.

[144] B. H. Sumida, A. Houston, J. McNamara, and W. Hamilton. Genetic algorithms and evolution. *Journal of Theoretical Biology*, 147(1):59–84, 1990.

[145] T. Sun, N. Daoudi, W. Pian, K. Kim, K. Allix, T. F. Bissyandé, and J. Klein. Temporal-incremental learning for android malware detection. *ACM Transactions on Software Engineering and Methodology*, 2024.

[146] B. Sutheekshan, S. Basheer, G. Thangavel, and O. P. Sharma. Evolution of malware targeting iot devices and botnet formation. In *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, volume 5, pages 1415–1422. IEEE, 2024.

[147] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton. A taxonomy and terminology of adversarial machine learning. *NIST IR*, 2019:1–29, 2019.

[148] R. Thomas. Lief - library to instrument executable formats. https://lief.quarkslab.com/, apr 2017.

[149] R. Tian, L. Batten, R. Islam, and S. Versteeg. An automated classification system based on the strings of trojan and virus families. In *2009 4th International conference on malicious and unwanted software (MALWARE)*, pages 23–30. IEEE, 2009.

[150] C.-W. Tien, S.-W. Chen, T. Ban, and S.-Y. Kuo. Machine learning framework to analyze iot malware using elf and opcode features. *Digital Threats: Research and Practice*, 1(1):1–19, 2020.

[151] S. Torabi, M. Dib, E. Bou-Harb, C. Assi, and M. Debbabi. A strings-based similarity analysis approach for characterizing iot malware and inferring their underlying relationships. *IEEE Networking Letters*, 3(3):161–165, 2021.

[152] L. Van Der Maaten, E. Postma, J. Van den Herik, et al. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71), 2009.

[153] D. Vasan, M. Alazab, S. Venkatraman, J. Akram, and Z. Qin. Mthael: Cross-architecture iot malware detection based on neural network advanced ensemble learning. *IEEE Transactions on Computers*, 69(11):1654–1667, 2020.

[154] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng. Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks*, 171:107138, 2020.

[155] N. Viennot, E. Garcia, and J. Nieh. A measurement study of google play. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, pages 221–233, 2014.

[156] B. Vignau, R. Khoury, S. Hallé, and A. Hamou-Lhadj. The evolution of iot malwares, from 2008 to 2019: Survey, taxonomy, process simulator and perspectives. *Journal of Systems Architecture*, 116:102143, 2021.

[157] VirusShare, 2024.

[158] K. Wagstaff. Machine learning that matters. *arXiv preprint arXiv:1206.4656*, 2012.

[159] T.-L. Wan, T. Ban, S.-M. Cheng, Y.-T. Lee, B. Sun, R. Isawa, T. Takahashi, and D. Inoue. Efficient detection and classification of internet-of-things malware based on byte sequences from executable files. *IEEE Open Journal of the Computer Society*, 1:262–275, 2020.

[160] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[161] H. Wang, W. Zhang, H. He, P. Liu, D. X. Luo, Y. Liu, J. Jiang, Y. Li, X. Zhang, W. Liu, et al. An evolutionary study of iot malware. *IEEE Internet of Things Journal*, 8(20):15422–15440, 2021.

[162] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia. Empowering edge intelligence: A comprehensive survey on on-device ai models. *ACM Computing Surveys*, 57(9):1–39, 2025.

[163] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary computation*, 3(2):149–175, 1995.

[164] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu. Droidevolver: Self-evolving android malware detection system. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 47–62. IEEE, 2019.

[165] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li. Detecting ai trojans using meta neural analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 103–120. IEEE, 2021.

[166] L. Yang, Z. Chen, J. Cortellazzi, F. Pendlebury, K. Tu, F. Pierazzi, L. Cavallaro, and G. Wang. Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 719–736. IEEE, 2023.

[167] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2327–2344, 2021.

[168] M. Yong Wong, M. Landen, M. Antonakakis, D. M. Blough, E. M. Redmiles, and M. Ahamad. An inside look into the practice of malware analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3053–3069, 2021.

[169] B. Yuan, J. Wang, P. Wu, and X. Qing. Iot malware classification based on lightweight convolutional neural networks. *IEEE Internet of Things Journal*, 9(5):3770–3783, 2021.

[170] Z. Yuan, X. Hu, Q. Wu, S. Ma, C. H. Leung, X. Shen, and Y. Huang. A unified domain adaptation framework with distinctive divergence analysis. *Transactions on Machine Learning Research*, 2022.

[171] X. Zhang, M. Zhang, Y. Zhang, M. Zhong, X. Zhang, Y. Cao, and M. Yang. Slowing down the aging of learning-based malware detectors with api knowledge. *IEEE Transactions on Dependable and Secure Computing*, 20(2):902–916, 2022.

[172] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. 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.

[173] Y. Zhang, Z. Ding, Y. Fu, Y. Wu, Y. Wang, and Z. Liu. Distributionally robust learning for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[174] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, 2021.

[175] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang. Measuring and modeling the label dynamics of online {Anti-Malware} engines. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2361–2378, 2020.