Hybrid Static-Dynamic Feature-Weighted Analysis for IoT Botnet Malware Detection

Colleen Lemak

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2025

Committee
Geetha Thamilarasu, Chair
Yang Peng
Brent Lagesse

Program Authorized to Offer Degree:
Computer Science & Software Engineering

University of Washington

Abstract

Hybrid Static-Dynamic Feature-Weighted Analysis for IoT Botnet Malware Detection

Colleen Lemak

Chair of the Supervisory Committee:
Geetha Thamilarasu

As the Internet of Things (IoT) domain continues to evolve, IoT devices face escalating security challenges. Recent waves of IoT botnets have exploited device vulnerabilities to launch dangerous large-scale Distributed Denial of Service (DDoS) attacks from compromised, resource-constrained devices. These networks of infected devices pose a unique threat to modern infrastructure, homes, schools, medical facilities, and transportation systems at heightened risk of malicious exploitation.

This paper proposes a novel hybrid framework that combines static and dynamic analysis techniques for IoT botnet malware detection without relying on complex Machine Learning (ML) models. By extracting and weighing the importance of key features from malware binaries based on their relevance to DDoS behavior, the framework maintains statistical adaptability to observed data while avoiding large memory usage and opaque black-box decision processes. Designed for interpretability and efficiency, this malware detection framework bridges code-level structure and runtime behavior, offering a transparent and practical botnet detection strategy for diverse resource-constrained IoT ecosystems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

BOTNET: A network of compromised devices controlled by a malicious actor, often used to perform coordinated attacks such as DDoS.

CONTROL FLOW GRAPH (CFG): A representation of all paths that might be traversed through a program during its execution, used in static analysis.

DDOS (DISTRIBUTED DENIAL OF SERVICE): A type of cyberattack in which multiple compromised devices flood a target system or network with traffic to disrupt its normal operations.

DYNAMIC ANALYSIS: The process of analyzing software by executing it in a controlled environment to observe its behavior at runtime.

FEATURE WEIGHTING (FW): The process of assigning importance to extracted features based on their relevance to malware behavior, such as indicators of DDoS or malicious activity.

MACHINE LEARNING (ML): The process of using and developing computer systems that are able to algorithmically learn and adapt to patterns in data without following explicit instructions. A subset of artificial intelligence which typically involves retraining a model with large datasets to improve performance.

MALWARE: Malicious software designed to disrupt, damage, or gain unauthorized access to systems. Examples include viruses, worms, and botnets.

MEMORY USAGE: The amount of system memory consumed during the execution of a program. Lightweight malware analysis frameworks aim to mitigate memory usage for resource-constrained devices.

MIRAI: A well-known IoT botnet malware family that infects devices using default credentials and uses them to launch large-scale DDoS attacks.

PROGRAM STRUCTURE INTERFACE (PSI): A visual representation of a software system's code, architecture, and interdependencies using a graph data structure.

HARDCODED STRINGS: Static strings embedded in binaries, such as IP addresses or command-and-control (C2) server names, often extracted during static analysis.

HYBRID ANALYSIS: In this context, an approach that combines static and dynamic analysis to gain both structural and behavioral insights into malware. Many modern ML frameworks refer to hybrid analysis as the blend of two dynamic analysis approaches; however, this limits holistic insights for the proposed framework.

IOT (INTERNET OF THINGS): A network of physical devices embedded with sensors, software, and connectivity that enables data exchange and remote control.

OPCODE: Short for operation code; a binary instruction that tells the processor what operation to perform. Used in static analysis to characterize malware behavior.

RULE-BASED CLASSIFIER: A detection mechanism that uses a set of manually defined rules to identify patterns or behaviors indicative of malware, rather than training a model.

STATIC ANALYSIS: The process of identifying potentially malicious characteristics by examining the binary, or source code, of a program without executing it.

SYSTEM CALL: A request made by a program to the operating system to perform privileged operations, such as accessing files or network sockets.

TRANSPARENT DETECTION: A characteristic of systems that provides interpretable and human-understandable results, yielding valuable insights into how a detection decision was made.

Chapter 1

INTRODUCTION

1.1 Background and Motivation

The recent expansion of Internet of Things (IoT) devices has fundamentally transformed modern society, driving automation and enabling seamless data exchange across urban infrastructure, industrial systems, healthcare networks, and everyday environments [1]. Billions of interconnected smart devices now sense, process, and communicate information in real time, affording operational efficiency and user convenience. However, this pervasive interconnectivity has simultaneously expanded the attack surface accessible to cyber adversaries [2]. Many IoT devices lack robust built-in security mechanisms, rely on outdated or unpatched firmware, and operate within fragmented vendor ecosystems—conditions that make them highly susceptible to malware compromise and large-scale botnet infections [3, 4].

IoT botnets—composed of compromised devices under the control of malicious actors—are frequently leveraged to execute large-scale Distributed Denial of Service (DDoS) campaigns, data exfiltration, and other disruptive cyber operations targeting critical infrastructure [5, 6]. Such attacks can severely degrade service availability, inflict substantial economic losses, and undermine public trust in IoT ecosystems. High-profile incidents, such as the *MIRAI* botnet, have vividly demonstrated the destructive potential of these threats in real-world contexts [7]. As IoT technologies continue to be deployed in safety- and mission-critical domains—including healthcare, transportation, and industrial control systems—the development of effective, lightweight, and transparent malware detection mechanisms has become imperative for IoT environments [8].

1.2 Challenges in IoT Malware Detection

Detecting malware within Internet of Things (IoT) environments poses numerous challenges stemming from the inherent diversity, scale, and limitations of IoT systems. Unlike traditional computing environments, IoT networks consist of billions of devices with varying computational capabilities, operating systems, and communication protocols. These characteristics complicate the deployment of robust and scalable security mechanisms capable of addressing the entire IoT spectrum.

A primary challenge lies in the resource-constrained nature of IoT devices. Many nodes operate with minimal processing power, memory, and storage, leaving little room for computationally intensive security algorithms or real-time threat monitoring [9]. Consequently, malware detection techniques must be both lightweight and efficient to remain viable in these constrained environments. However, the architectural heterogeneity of IoT systems further complicates malware detection solutions. Devices are built on diverse hardware platforms, firmware versions, and network stacks, which hinder the design of universal detection frameworks and complicate the integration of standardized defense strategies [10].

Modern adversaries often exploit sophisticated evasion methods to bypass conventional security controls. Techniques such as code obfuscation, encryption, packing, and polymorphic transformations enable malware to conceal malicious behavior and evade signature-based scanners [11, 12]. Furthermore, many IoT devices continue to operate with outdated or unpatched firmware, often due to the absence of secure update mechanisms or limited manufacturer support. These vulnerabilities provide attackers with persistent entry points into critical systems [4]. In addition, IoT deployments frequently lack robust update pipelines or centralized monitoring capabilities, allowing malware to remain undetected for extended periods of time, complicating the production of timely incident responses [13].

Traditional malware detection techniques applied to IoT environments can be broadly categorized as either static or dynamic analysis. Static analysis examines executable code or binaries without running them, relying on characteristics such as signatures, opcode dis-

tributions, or control-flow graphs to detect known threats [14, 7]. While computationally efficient and well-suited for constrained devices, static methods often fail against obfuscated or previously unseen malware variants. In contrast, dynamic analysis observes runtime behavior—such as system calls, resource utilization, and network activity—to capture indicative behavioral patterns [15, 16]. Although more resilient to code obfuscation, dynamic techniques typically demand greater computational and energy resources, making them impractical for direct deployment on lightweight IoT devices.

Overall, these challenges underscore the urgent need for statistically adaptive, lightweight, and hybrid detection approaches that can balance analytical depth with efficiency, accuracy, and scalability within resource-limited IoT ecosystems.

Another challenge often overlooked in IoT malware research is the distribution of model complexity across heterogeneous hardware. Therefore, achieving scalability requires a design philosophy that decouples heavy offline computation from lightweight, deterministic classification. The proposed model explicitly addresses this issue by ensuring that all real-time classification tasks involve only basic arithmetic and threshold operations, independent of dataset size or training history.

1.3 Hybrid Analysis Approaches

Hybrid malware detection seeks to combine the complementary strengths of static and dynamic analysis to enhance detection accuracy and robustness in IoT environments. Static analysis examines the structural and semantic characteristics of a binary without execution, capturing patterns that are intrinsic to the code. Dynamic analysis, in contrast, observes the runtime behavior of programs under controlled conditions, including system calls, memory usage, and network interactions. By integrating both approaches, hybrid frameworks provide a more comprehensive understanding of malware behavior and are better equipped to identify sophisticated, evasive threats.

Recent studies have demonstrated that combining opcode sequences, system call traces, and behavioral metrics improves detection performance in IoT malware scenarios [17, 18, 5].

For example, PSI graph-based static features paired with dynamic string-enriched graphs have shown effectiveness in detecting IoT botnets [15]. However, the majority of these hybrid models rely on machine learning (ML) techniques such as random forests, neural networks, or ensemble methods. While achieving high accuracy, such approaches require extensive labeled datasets, iterative training, and significant computational resources. Furthermore, ML-based models often produce opaque black-box outputs, limiting interpretability and trust in practical IoT deployments [8, 9]. These limitations highlight the need for lightweight, transparent, and resource-efficient alternatives.

1.4 Research Contribution

This research introduces a novel hybrid static–dynamic malware detection framework specifically tailored for lightweight IoT ecosystems. The proposed approach emphasizes interpretability and computational efficiency while maintaining high detection performance. Unlike conventional ML-based systems, the framework relies on feature weighting rather than iterative training, producing a lightweight and transparent classifier suitable for deployment on resource-constrained devices.

Once feature weights are determined, they are deployed to the IoT device, where a lightweight rule-based classifier evaluates the weighted feature vectors to produce binary classification decisions (malicious or benign), alongside supporting metrics such as F1-score, runtime, and memory consumption. By separating heavy feature extraction and weighting from on-device classification, the framework achieves both high detection accuracy and minimal computational cost, without relying on opaque machine learning models.

Overall, this research contributes an interpretable, hybrid static–dynamic malware detection methodology that bridges code-level structures with runtime behavior. By leveraging frequency-based feature weighting, the framework provides efficient, transparent, and feasible malware detection suitable for real-world IoT deployments, including resource-constrained devices in critical and consumer environments. It demonstrates that high detection performance can be achieved without sacrificing explainability or deployability, addressing a key

gap in current IoT malware research.

1.5 Research Questions

The experiments aim to answer these central questions:

1. **Can a purely arithmetic feature-weighted classifier achieve comparable accuracy to machine-learning baselines while running on constrained IoT hardware?**

2. **How do static and dynamic features individually contribute to overall detection performance, and what synergy does the hybrid model achieve?**

Through systematic evaluation across varying feature subsets, architectures, and environments, the experimental setup validates that the hybrid feature-weighted model remains lightweight, reproducible, and interpretable—satisfying the practical demands of real-world IoT malware detection.

1.6 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 presents a comprehensive literature review of existing IoT malware detection methodologies and feature selection strategies. Chapter 3 explains the details of the hybrid malware detection framework, and Chapter 4 describes the experimental setup, datasets, and evaluation metrics of the research. Chapter 5 discusses the significance of the results and compares them with the current state-of-the-art. Chapter 6 addresses limitations and concludes the thesis with practical impacts and future research directions, and Chapter 7 and 8 detail the environment configuration and associated class file location constructed with LaTeX.

Chapter 2

RELATED WORK

2.1 Network and Router-Level IoT Security

Several IoT security approaches focus on preventing insecure or compromised devices from joining a network by enforcing protections at the router or gateway level rather than on the device itself. Ogunnaike and Lagesse [19] propose a consumer-friendly IoT security architecture in which home routers automatically perform vulnerability scanning, access control, and quarantine before admitting devices to the network. Their SDN-assisted design demonstrates that substantial risk reduction can occur at the gateway, particularly for consumer environments where users cannot manually assess device security.

Similarly, Condry and Nelson [20] show that IoT edge gateways can act as security enforcement points by monitoring traffic patterns and device behavior to identify anomalies before they propagate across the network. These router-level and gateway-level approaches improve baseline IoT security posture but do not inspect binary code or runtime execution behaviors. As a result, they cannot detect malware embedded within otherwise functional devices, motivating complementary on-device detection methods such as the hybrid static–dynamic analysis presented in this thesis.

2.2 Static and Dynamic Malware Analysis

Malware detection techniques traditionally fall into two broad categories: static and dynamic analysis. Static analysis inspects the code or binary without execution, extracting features such as opcode sequences, control flow graphs, and embedded strings to identify malicious patterns [7]. It is computationally efficient and scalable but can be bypassed by obfuscation or encrypted payloads.

Dynamic analysis, on the other hand, monitors the behavior of software during execution, capturing system calls, memory usage, CPU activity, and network traffic [16]. This method detects malicious activity based on runtime behavior, including anomalies that static analysis may overlook. However, dynamic analysis is typically more resource-intensive and may require controlled environments such as sandboxes [4].

2.3 Hybrid Approaches in IoT Malware Detection

Hybrid malware detection frameworks combine structural static evidence with dynamic behavioral signals to improve robustness and accuracy in IoT environments. Early hybrid systems, such as Shijo and Salim (2015) [16], extracted opcode-based static features alongside system call–based dynamic features and concatenated them for Random Forest (RF) classification. Although influential, this work predates modern graph-based feature extraction and does not incorporate higher-order behavioral modeling.

More recent efforts employ richer graph representations and advanced machine learning models. For example, Ngo et al. (2021) [7] introduced a hybrid IoT malware framework that fuses PSI graphs extracted using IDA Pro with system-call graphs (SCG) generated by their private V-Sandbox system. These static and dynamic graphs are encoded using graph2vec embeddings and concatenated before classification with RF and SVM models. While methodologically strong, this pipeline depends on proprietary infrastructure (IDA Pro, V-Sandbox, custom graph2vec implementations), making exact replication infeasible without licensed access. This paper reproduces the given methodology [7], as their Random Forest model builds on existing foundational research [16] and performs exceptionally well with 98.52% accuracy.

Other modern hybrid approaches [15, 5] similarly combine static and dynamic features but rely on computationally expensive embedding generation or iterative ML training. Such designs limit deployability on constrained IoT devices and introduce opacity that complicates operational debugging or forensic analysis.

These observations highlight a gap in current research: modern hybrid approaches are

accurate but often non-reproducible, computationally heavy, or unsuitable for embedded IoT deployments. This thesis positions Pi-HSDF as a lightweight, interpretable alternative that aligns conceptually with modern hybrid paradigms while avoiding proprietary dependencies.

2.3.1 Classical Hybrid ML Baselines and Their Role in This Thesis

A foundational hybrid method is the integrated static–dynamic Random Forest model proposed by Shijo and Salim [16], which remains widely cited because its extraction steps—opcode counts and syscall traces—are fully reproducible using open-source tools. For this reason, it serves as the classical hybrid baseline in this thesis.

To achieve comparison with a modern hybrid model, this thesis also draws methodological alignment from Ngo et al. (2021) [7]. Ngo et al. employ:

- IDA Pro for static PSI graph extraction,
- V-Sandbox for dynamic system-call graph (SCG) generation,
- graph2vec embeddings for structural representation,
- early-fusion concatenation of static and dynamic vectors,
- RF and SVM classifiers for final decision-making.

Because IDA Pro, V-Sandbox, and their embedding components are closed-source, exact replication of this pipeline is not feasible. However, the methodological core of their system *is* reproducible:

1. extraction of static structural features,
2. extraction of dynamic behavioral features,
3. early hybrid feature fusion,

4. classification using the Random Forest family.

These principles define the essential hybrid paradigm evaluated in [7].

Accordingly, this thesis implements a "Ngo-inspired" hybrid Random Forest baseline using:

- static PSI-derived structural metrics,
- dynamic 3-gram syscall sequences as an open analogue to SCG embeddings,
- early concatenation of static and dynamic vectors,
- a Random Forest classifier, consistent with [7].

This approach preserves the methodological intent of modern hybrid detectors while remaining fully reproducible on open infrastructure. Combined with the Shijo–Salim baseline, it anchors Pi-HSDF against both classical and contemporary hybrid frameworks, demonstrating its relevance and competitiveness within current research trends.

2.4 Critical Analysis of Existing Approaches

Across the surveyed literature, several consistent limitations emerge. First, nearly all high-performing IoT malware detectors depend on supervised ML models that require extensive retraining when new malware families emerge, reducing long-term maintainability. Second, many "lightweight" ML methods remain unsuitable for deployment at the network edge, as their models exceed available memory or require floating-point operations unavailable on constrained processors. Third, interpretability remains a systemic gap: while methods such as SHAP or LIME provide post-hoc explanations, they do not yield inherent transparency during detection. The proposed research differentiates itself by embedding interpretability directly into the scoring mechanism, thereby ensuring that every classification is traceable to specific static and dynamic feature contributions.

2.5 Research Gap and Motivation

With an exponential increase in deployment of smart devices, sophistication of adapting malware follows, consequently requiring robust IoT malware detection frameworks. A particularly relevant threat is botnet-driven DDoS malware which has sparked a range of research efforts in the IoT domain. Traditional security solutions predominately implement either static or dynamic analysis in standalone methods for detecting malware, although each possesses its own set of limitations.

Static analysis examines the malware binary without execution, offering speed and system safety, but is often limited by obfuscation and polymorphic malware because of a lack of runtime insight [1] [3]. In contrast, dynamic analysis executes malware in a controlled environment, revealing behavior-based insights like network connections or memory usage, but it requires a plethora of computational resources; in an IoT environment, devices are resource-constrained, and standalone dynamic analysis methods face capacity challenges being deployed with computationally expensive solutions [1] [3].

To address these issues with standalone methods, researchers have developed hybrid detection models that combine static and dynamic features for improved performance in general malware environments. Sharma and Devare [21] propose an explainable deep neural network model for malware detection, focusing on interpretability through post-hoc analysis. Similarly, Purkayastha et al. [22] utilize federated learning to improve privacy and efficiency in Android malware detection. Table 2.1 outlines related state-of-the-art malware analysis work performed by researchers from various security-focused domains.

Table 2.1: Summary of Related Work on Malware Detection

Research	**Technique Used**	**Targeted Attacks**	**Targeted Malware**
Garg et al. (2023) [8]	Hybrid ML with MQTT traffic features	Botnets, MQTT-based DDoS	MQTT-based IoT malware

Research	**Technique Used**	**Targeted Attacks**	**Targeted Malware**
Cassel and Majd (2024) [9]	Lightweight ML classification	Obfuscation, multi-class classification	General IoT malware
Sharma and Devare (2024) [21]	Explainable Deep Neural Networks	General malware detection	Android malware
Purkayastha et al. (2024) [22]	Hybrid NN with Federated Learning	Android attacks, privacy-preserving detection	Android malware
Kurniawan et al. (2024) [17]	Filter-based Reduced Feature Selection with ML	IoT DDoS botnet attacks	General IoT malware
Ali et al. (2024) [5]	Hybrid ML (RF + DT) for botnet detection	IoT-based DoS, generic and worm botnet attacks	MIRAI, BASHLITE
Tang et al. (2025) [11]	Adversarial packed dynamic + traffic analysis	Evasive malware, adversarial traffic	Android malware
Deng et al. (2023) [13]	Trust-based hybrid evaluation at user-edge	Resource abuse, trust-level threats	General IoT malware
Jeon et al. (2024) [3]	Multi-feature CNN-based static analysis	Behavioral attacks in smart IoT	IoT-targeted malware
Kumar et al. (2024) [1]	Systematic review of hybrid analysis	N/A (Survey)	IoT malware families
Yunmar et al. (2024) [12]	Heuristic static + dynamic review	Signature evasion	Android malware
Hong et al. (2022) [18]	Hybrid feature selection for DDoS mitigation	Flooding, HTTP/UDP-based DDoS	General IoT malware
Hossain and Islam (2023) [6]	Hybrid feature selection + ensemble ML	Botnets, statistical anomalies	General IoT botnets

Research	**Technique Used**	**Targeted Attacks**	**Targeted Malware**
Saez de Camara et al. (2024) [4]	Gotham: emulated testbed for hybrid analysis	Custom attack simulations	General malware in IoT networks

In the IoT-specific space, Cassel and Majd [9] leverage a lightweight multi-class malware classifier tailed for IoT environments, while comparably, Garg et al. [8], target MQTT-based IoT malware with Machine Learning (ML) and network traffic features. These works, among others, are novel in the IoT malware detection space and are foundational for further hybrid method research.

Despite advancements in hybrid models, the majority of solutions heavily hinge on complex ML architectures; although powerful and increasingly accurate, these frameworks often lack interpretability and demand substantial computational resources. Asam et al. and Ambekar et al. use deep learning (DL) models which achieve high accuracy, but render impractical in IoT edge deployment because of their resource requirements [2] [23] . Similar novel approaches include quantum-based detection [24] and blockchain-enabled federated systems [25], although these render either cost-prohibitive or still in experimental stages. Although effective, these approaches rely on opaque, black-box models, which hinder interpretability and traceability needed for high-stakes decisions [8] [12]. Moreover, many state-of-the-art models require high computational overhead and model training, making them far less suitable for real-time or resource-constrained environments such as smart home IoT deployments [5][26].

As a remedy for IoT and DDoS specific challenges, several recent studies have implemented hybrid feature selection techniques to improve classification accuracy and reduce computational overhead. Ali et al. [5] introduce a hybrid Random Forest (RF) and Decision Tree (DT) ML model for botnet detection in IoT, emphasizing efficiency, while Mutupuri et al. [26] offer a classical ML model to detect anomalous traffic. Hong et al. and Hossain et al. apply hybrid feature selection techniques to optimize performance, but rely on differ-

ing features such as network traffic-based features and statistical and flow-level data rather than deep system-level behavior designed for IoT edge-deployment [18] [6]. Additionally, work proposing a minimal universal feature set for DDoS detection in IoT has been well-researched, though these methods focus on statistical relationships more than behavioral indicators [27]. In a similar fashion, Kurniawan et al. explore the removal of redundant features to improve classification performance, but the research does not involve a functional framework for system-level behaviors [17].

Chapter 3

PROPOSED PI-HSDF

The proposed framework, Pi-Weighted Hybrid Static-Dynamic Analysis Framework (Pi-HSDF) introduces a lightweight and interpretable hybrid malware detection system designed to identify IoT botnet binaries using both static and dynamic characteristics. Unlike prior approaches that rely solely on machine learning classifiers, this system integrates *feature-weighted analysis*—a transparent, equation-based technique that enables efficient classification on resource-constrained devices such as the Raspberry Pi 5. The framework aims to bridge the gap between computationally heavy, black-box hybrid models, and purely static or dynamic systems that capture only partial behavioral evidence.

3.1 Overview and Motivation

Existing IoT malware detectors frequently employ complex ensembles or deep learning architectures that achieve high accuracy on benchmark datasets but suffer from two limitations: lack of interpretability and high resource consumption. In IoT deployments, devices typically operate with limited memory and processing power, making it impractical to store or execute models containing thousands of parameters. Moreover, forensic investigators often require insight into *why* a binary is classified as malicious—something opaque classifiers cannot easily provide.

The proposed framework resolves these issues through a modular, two-phase design. All computationally expensive operations such as feature extraction and weighting are performed *offline* on a workstation or virtual machine, while the lightweight inference stage executes directly on the IoT endpoint. This separation ensures transparency, reproducibility, and adaptability across architectures without sacrificing detection accuracy.

3.2 Architecture Overview

Figure 3.1 illustrates the overall architecture of Pi-HSDF. All computationally heavy stages—including static PSI graph extraction, dynamic syscall tracing, and feature-weight computation—occur offline in a controlled workstation environment. Only the final lightweight scoring logic executes on the Raspberry Pi IoT device. This separation enforces safety during analysis, enables cross-architecture portability, and ensures that on-device inference remains both deterministic and efficient.

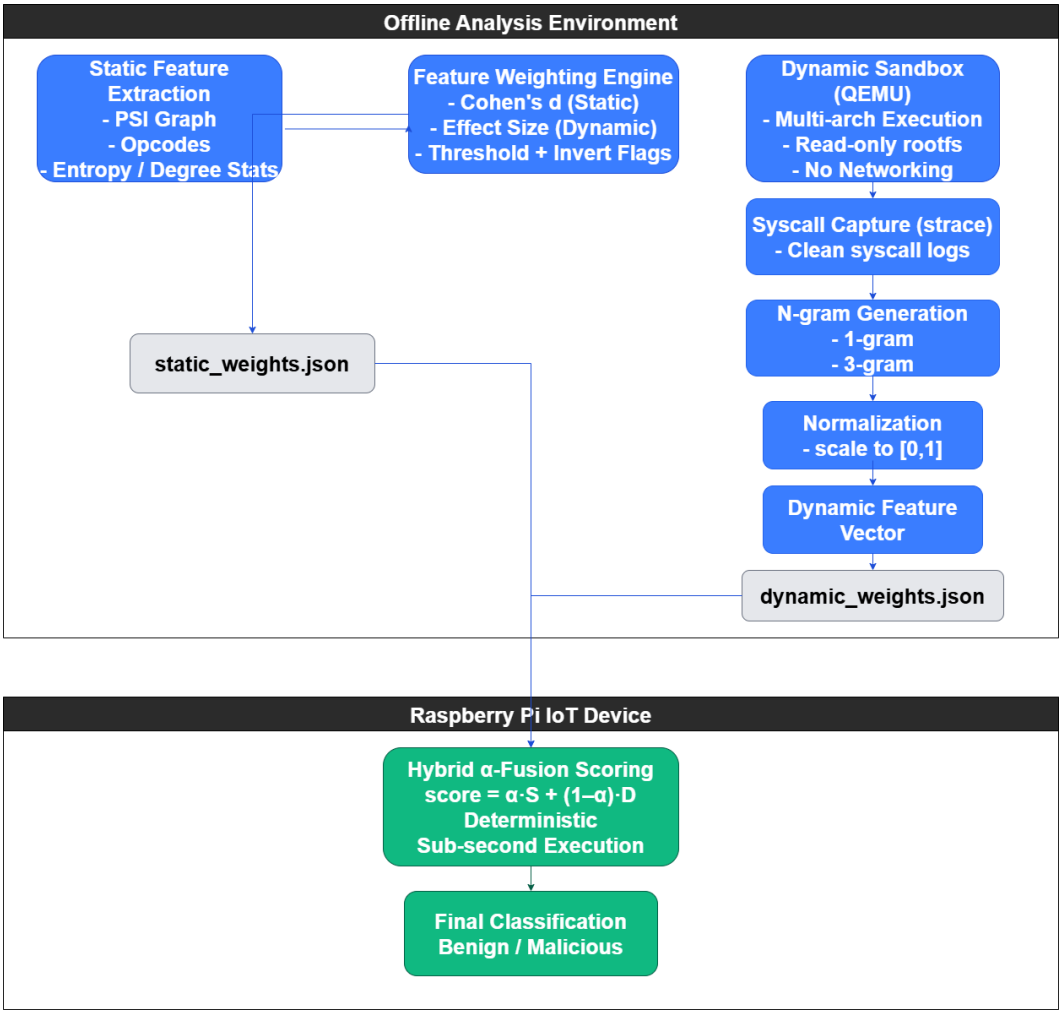


Figure 3.1: Hybrid Malware Detection Architecture

With the overall architecture defined, the following section details the author's specific contributions to the design, implementation, and deployment of the Pi-HSDF framework.

3.3 Author Contributions to Pi-HSDF Design and Implementation

To address the shortcomings of prior IoT malware research and create a solution that could operate realistically on constrained hardware, this thesis develops a fully end-to-end hybrid detection framework that was designed, implemented, evaluated, and deployed by the author. Unlike existing hybrid studies that primarily combine features for training machine-learning classifiers, the Pi-HSDF system proposes and builds a new, non-ML feature-weighted model that performs transparent arithmetic-based inference directly on an IoT device. This contribution required designing both the theoretical structure of the feature-weighting method and the practical software architecture that makes such an approach viable across diverse CPU architectures.

The framework's architecture is the result of several deliberate engineering decisions that distinguish it from the state of the art. First, the author implemented a modular pipeline that separates heavy PSI-graph generation, opcode extraction, syscall trace collection, and effect-size-based weighting into an offline stage that executes on a secure workstation. This design choice is intentional: to the best of the author's knowledge, prior work has not demonstrated a reproducible pipeline that fully decouples feature computation from on-device classification for hybrid IoT malware analysis. Second, the runtime classifier deployed on the Raspberry Pi is specifically authored to execute with constant-time, constant-memory computations, ensuring that classification remains deterministic even under the limited resources typical of IoT environments. This lightweight portion of the framework was also implemented from scratch and includes threshold logic, invert-flag handling, data normalization, and the final hybrid scoring function.

Beyond architectural novelty, the author designed and implemented a multi-architecture sandboxing environment using Docker, QEMU user-mode emulation, strict seccomp profiles, timeouts, and file-system isolation. This environment enables safe execution of ARM, MIPS,

x86, and PowerPC malware on a Raspberry Pi—something not demonstrated in previous hybrid IoT malware research, where datasets are analyzed offline or within homogeneous architectures. The custom tooling developed for this work includes automated binary ingestion, syscall-trace capture, n-gram transformation, and container orchestration. Together, these components form a reproducible and secure dynamic analysis pipeline purpose-built for IoT malware.

Finally, the resulting hybrid weighting algorithm was designed not only as a theoretical contribution but was implemented, tuned, and evaluated entirely by the author using multiple cross-analysis configurations (static-only, dynamic 1-gram, dynamic 3-gram, and mixed hybrid configurations). Through this process, this thesis demonstrates how each design choice—feature selection, thresholding strategy, weighting rationale, and on-device classification—affects practical detection outcomes. In contrast to prior work, which largely treats datasets as material for ML training, this thesis uses them to build a transparent, behaviorally interpretable model that can articulate why a given binary is malicious.

Taken together, these efforts constitute a complete engineering and research contribution that spans theoretical formulation, software implementation, architectural design, security-conscious execution environments, and empirical validation. The proposed Pi-HSDF system is therefore not only conceptually distinct from existing hybrid approaches but also realized as a functional, deployable malware detection mechanism that demonstrates how interpretable hybrid analysis can be executed efficiently on actual IoT hardware.

3.4 Static Analysis

The static component extracts graph- and opcode-level features from each binary without executing it. Using the `angr` framework, control-flow and function-call graphs are generated and serialized as PSI (Program Structure Interface) graphs. From these, eleven metrics are computed, including node and edge counts, density, average degree, and clustering coefficients. Opcode frequency, syscall counts, and ELF entry patterns (e.g., `main`, `exit`, `frame_dummy`) are also collected. These features characterize the internal organization and

complexity of the binary, allowing detection of statically embedded malicious behavior such as code obfuscation.

3.5 Dynamic Analysis

Complementing the static view, the dynamic module captures runtime behavior through system call tracing. Each binary is executed within a Docker-based sandbox equipped with QEMU user-mode emulation to support multiple CPU architectures (ARM, MIPS, x86, PowerPC). During execution, the `strace` utility records all system calls, which are transformed into n-gram sequences representing temporal patterns of API usage. This translation captures behavioral signatures of malware families (e.g., frequent use of `connect`, `write`, and `execve` sequences typical of botnets). To ensure reliability, the sandbox enforces strict timeouts, error handling, and file system isolation, preventing uncontrolled network activity or data corruption during analysis.

3.6 Hybrid Integration

The hybrid layer unifies static and dynamic evidence through an interpretable weighted linear fusion. Static and dynamic features are first normalized and assigned weights according to their discriminative power, quantified using Cohen's d effect size between benign and malicious distributions. For each feature, the model additionally computes a *threshold* (the midpoint between benign and malicious means) and an *invert flag* indicating whether higher or lower values are associated with malicious behavior. These parameters allow each feature to contribute a directionally correct and scale-adjusted signal during inference.

During classification, each feature produces a normalized score $f_i^{(s)}$ or $f_j^{(d)}$, which reflects whether the feature value lies on the benign or malicious side of its threshold. This score is then multiplied by its corresponding weight $w_i^{(s)}$ or $w_j^{(d)}$, ensuring that features with greater discriminative strength exert proportionally stronger influence on the final decision.

Inspired by prior hybrid static-dynamic analysis and feature selection algorithms [7, 16], this thesis proposes the following interpretable hybrid scoring function:

$$S = (1 - \alpha) \sum_{i=1}^{n} w_i^{(s)} f_i^{(s)} + \alpha \sum_{j=1}^{m} w_j^{(d)} f_j^{(d)} \tag{3.1}$$

where S is the final hybrid score, $\alpha \in [0, 1]$ controls the relative influence of static and dynamic components, $f_i^{(s)}$ and $f_j^{(d)}$ denote the normalized and threshold-adjusted feature values, and $w_i^{(s)}$, $w_j^{(d)}$ represent their corresponding effect-size-based weights. A sample is classified as malicious if S exceeds the empirical mean decision threshold computed from the training data.

3.7 Mathematical Rationale for Weight Computation

Feature weights are derived from effect-size analysis rather than purely empirical tuning. Cohen's d statistic [28] captures the standardized difference between benign and malicious feature distributions, expressed as:

$$d = \frac{|\mu_m - \mu_b|}{\sqrt{\frac{\sigma_m^2 + \sigma_b^2}{2}}} \tag{3.2}$$

where μ_m and μ_b denote the malware and benign means, and σ values represent corresponding variances. This measure quantifies the discriminative magnitude of each feature in a scale-invariant manner. Features with higher d values are assigned proportionally higher weights, while low-effect features are suppressed to minimize noise. This strategy provides a formal statistical underpinning for feature selection and ensures that the weighting process is both deterministic and interpretable.

3.8 Feature Weighting and Interpretability

Rather than depending on opaque learned parameters, each feature in this system has a clear, measurable contribution. Weights quantify how strongly a feature separates benign from malicious samples, while thresholds and invert flags encode human-interpretable boundaries. For instance, a frequent `connect-write-read` syscall pattern directly increases the score toward a malicious classification. Because the model performs only arithmetic

operations, investigators can easily trace which features triggered a detection and reproduce the same result independently.

This approach contrasts with the Random Forest baseline [7], which, although accurate, obscure individual feature effects within decision trees. The proposed feature-weighted model therefore offers both quantitative performance and qualitative insight.

3.9 Advantages of the Framework

Together, these attributes establish the proposed hybrid static–dynamic feature-weighted analysis as a practical and transparent solution for IoT botnet malware detection.

Instead of using ML to select statistically strong features, this proposed framework calculates a unique set of features weights based on behavioral semantics and feature frequencies specific to IoT binaries from known DDoS attacks. Operating within the resource-constraint bounds of these environments, this research proposes a hybrid static-dynamic malware analysis framework that prioritizes classification transparency. Feature-weighting logic is designed for interpretability and lightweight execution in IoT contexts. Unlike prior work that uses datasets solely for ML training, this framework treats them as sources of behavioral insight aligned with observed malware patterns.

While relevant studies have explored feature selection techniques for IoT malware detection, the majority have focused on single analysis; using standalone static or dynamic limits the breadth of insights derived from malware components and runtime behaviors. [6, 17, 18] achieve high detection rates using filter-based and wrapper-based selection methods over dynamic traffic features such as flow metrics, packet sizes, and port numbers, but they rely on solely network-level observations. On the other hand Filus et al. [14] employ feature selection with static analysis using code-level metrics for vulnerability prediction, but lack insight from malware runtime behavior.

Notably, many works describe their methods as hybrid approaches, typically combining different subsets of static or dynamic attributes, rather than truly integrating features across the two domains. This characterization may be misleading, implying a cross-domain

synthesis not present in their methodologies.

In contrast, this proposed research introduces a hybrid framework that unifies static and dynamic analysis through an explicit feature selection method. Static features such as op-code patterns, hardcoded strings, and system call signatures are combined with dynamic behavioral indicators such as memory usage and system traces, for a holistic approach on prior successes in research, aiming to select semantically meaningful features within individual domains. By leveraging a feature-weighed selection strategy that prioritizes cross-domain relevance and interpretability, this framework enables lightweight and behaviorally grounded malware detection for resource-constrained IoT environments.

The proposed design offers three practical benefits:

- **Efficiency:** Training and weighting are performed once offline; on-device classification is linear in feature count and requires minimal computation.

- **Interpretability:** Each decision can be traced back to explicit feature contributions and threshold crossings.

- **Adaptability:** The modular architecture supports future integration of additional feature types (e.g., packet-level, memory access) without retraining a complex model.

To the best of the author's knowledge, no prior research has systematically integrated static and dynamic features under a unified feature-weighted selection process for detecting IoT-targeted DDoS malware such as the *MIRAI* botnet, nor proposed an interpretable and lightweight non-ML-based detection model with this integration. Thus, this work fills a critical research gap with a holistic, transparent approach capable of bridging code-level characteristics with runtime behaviors for an impactful malware detection mechanism.

Chapter 4

EXPERIMENTAL SETUP

To evaluate the performance, interpretability, and efficiency of the proposed hybrid static–dynamic framework, a comprehensive experimental environment was constructed that mirrors practical IoT deployment conditions. This section details the datasets, hardware and software configurations, sandbox environment, and evaluation metrics used in all analyses.

The framework consists of three coordinated components: static analysis, dynamic analysis, and hybrid integration. Each stage produces interpretable features that can be combined or independently evaluated.

4.1 Dataset Composition

The evaluation utilized 6,287 unique ELF binaries encompassing both benign and IoT botnet malware samples. All binaries were collected from verified and research-cited repositories, such as *VirusShare* and *IoTPOT*, organized by architecture to ensure representativeness across common IoT platforms such as ARM, MIPS, PowerPC, and x86.

Malware Dataset. A total of 2,891 malicious binaries were included, distributed across five architectures as shown below:

- **Training set (1,427 samples):** 424 ARM, 415 MIPS, 270 x86, 168 PPC, 150 x64
- **Test set (1,464 samples):** 441 ARM, 428 MIPS, 277 x86, 176 PPC, 142 x64

Benign Dataset. The benign dataset contained 3,396 compiled non-malicious Linux executables, similarly divided into training and test subsets:

- **Training set (1,698 samples):** 1,432 x86, 266 ARM
- **Test set (1,698 samples):** 1,435 x86, 263 ARM

Table 4.1: Dataset Composition by Architecture and Label

Set	Label	ARM	MIPS	x86	PPC	x64
Training	Malware	424	415	270	168	150
Test	Malware	441	428	277	176	142
Training	Benign	266	–	1432	–	–
Test	Benign	263	–	1435	–	–

To ensure one-to-one comparability between analysis modes, binaries appearing in both static and dynamic pipelines were matched by filename, omitting those lacking valid traces or PSI graphs. This produced a unified hybrid dataset where each binary contributed both structural (static) and behavioral (dynamic) representations.

4.2 Analysis Environment

All static feature extraction, weighting, and Random Forest training were executed on a Kali Linux virtual machine configured with 20 GB RAM and 2 CPU cores. Dynamic execution occurred within a secure Docker-based sandbox running on the same host, isolated from the network and configured to emulate multiple architectures using QEMU user-mode binaries. Each binary was traced via `strace` with a 60 second timeout to prevent indefinite execution. The sandbox environment enforced read–write isolation to a single output directory, ensuring malware binaries could not access or modify the host file system.

Raspberry Pi Deployment. Network connectivity was disabled during all tests for safety. Each inference script executed directly on the Pi's CPU using a Python 3.13 virtual environment with minimal dependencies (`e.g.` `numpy`, `pandas`, `psutil`, `scikit-learn`). Because

all feature weights, thresholds, and z-score statistics were precomputed offline, the Pi only performed lightweight arithmetic during inference.

4.3 Deployment on the Raspberry Pi 5

To validate real-world feasibility, the lightweight classifier was deployed and tested on a Raspberry Pi 5 with 8 GB RAM and a 128 GB microSD storage medium running Ubuntu 22.04 LTS (64-bit). Offline feature extraction and weighting occurred on a Kali Linux virtual machine; only the resulting CSV feature files and weight dictionaries were transferred to the Pi. At runtime, each Pi script loads the precomputed parameters, normalizes inputs, computes scores, and outputs true-positive, false-positive, and other metrics. No retraining or external dependency is required.

Measured results confirm that the Top-N selecting feature-weighted classifier consumes less than 1 MB of memory for standalone static and dynamic analysis and less than 15 megabytes for hybrid evaluation, with inference times under one second. This validates that the design is lightweight for IoT devices while maintaining detection rates above 98%.

4.4 Safety and Containment Controls

All malware execution was performed within isolated Docker containers using QEMU user-mode emulation. Each container enforced read-only file system mounts and disabled outbound networking; this ensured that no malware process could escape the sandbox or interact with host resources. These precautions ensured ethical and secure handling of live malware samples throughout the experiments.

4.5 Feature Extraction and Weighting Process

Static features were derived from PSI graphs generated by the `angr` binary analysis framework. Each PSI graph was traversed to compute metrics such as number of nodes and edges, average and maximum degree, density, and clustering coefficients. Opcode and

syscall frequency counts were extracted directly from ELF binaries to capture instruction-level patterns.

Dynamic features were obtained from system call traces recorded during sandbox execution. The traces were tokenized into n-gram sequences representing temporal API patterns; both 1-gram and 3-gram variants were analyzed to balance semantic depth and computational cost. Feature weights, thresholds, and invert flags were computed offline using Cohen's d effect size between benign and malware training distributions. These statistics were serialized as text and JSON files and later used by the Raspberry Pi classifiers for deterministic inference.

The proposed feature-weighted model leverages existing feature- extraction selection techniques to blend static and dynamic analysis for a lightweight hybrid approach.

Table 4.2: Comparison of Feature Selection Techniques in IoT Malware Detection

Research	**Feature Selection Method**	**Used Features**	**Focus / Limitation**	**Uses Static + Dynamic Analysis?**
Hossain and Islam (2023) [6]	Filter + wrapper (IG, chi-square)	Network traffic stats, port numbers	ML-based, limited interpretability	Dynamic only
Kurniawan et al. (2024) [17]	Correlation-based + RFE	Flow features, packet sizes, duration	Improves accuracy, ML-reliant	Dynamic only
Hong et al. (2022) [18]	PCA + statistical analysis	DDoS traffic logs, flow metrics	Efficient but not IoT-specific	Dynamic only
Ebrahem et al. (2025) [27]	Statistical reduction to universal set	Packet counts, durations, byte rates	Defines minimal features; no hybrid method	No, feature framework only
Filus et al. (2021) [14]	Feature selection for vulnerability prediction	Vulnerabilities, static code attributes	Static-only; focus on vulnerabilities, not malware	Static only

Continued on next page

Research	**Feature Selection Method**	**Used Features**	**Focus / Limitation**	**Uses Static + Dynamic Analysis?**
Ngo et al. (2021) [7]	Hybrid static + dynamic vector	Static code metadata, network flow behavior	ML-based; Mirai-focused hybrid improves accuracy	Yes
Nguyen et al. (2020) [10]	PSI-Graph from function calls	Static PSI graph features (high-level function-call)	Lightweight, multi-arch support; no dynamic data	Static only
Nguyen et al. (2022) [15]	Hybrid with dynamic string-enriched graphs	PSI graph + dynamic printable strings	Accurate hybrid graph-based ML model; higher complexity	Yes
Shijo and Salim (2015) [16]	Combined static + dynamic extraction	Opcodes, API usage, runtime behavior: file / mem / registry	General malware focus; not IoT-specific	Yes
Pi-HSDF (2025)	**Feature-weighted (semantic relevance)**	**Opcode patterns, strings, memory, system call traces**	**DDoS IoT malware; no ML; interpretable; lightweight**	**Yes**

4.6 Random Forest Baselines: Classical and Modern Comparisons

To contextualize the performance of the proposed Pi-HSDF framework, a Random Forest baseline was implemented. This baseline reproduces the modeling strategy used in classical and modern hybrid malware detection:

The baseline is derived from work by Ngo et al. [7] who combine static PSI graphs and dynamic syscall-graph embeddings using early fusion, then evaluate ML models including Random Forests. Because their pipeline depends on proprietary tools (IDA Pro), private sandbox infrastructure (V-Sandbox), and graph embedding methods (graph2vec), an exact replication is not feasible. Instead, this thesis constructs an equivalent reproducible hybrid RF classifier using the same static PSI and dynamic 3-gram features extracted for Pi-HSDF. This baseline mirrors the methodological intent—hybrid feature fusion with an RF

classifier—while remaining fully open and reproducible.

This baseline was evaluated using the same metrics as Pi-HSDF. This model helps contextualize the efficiency–interpretability tradeoffs of the proposed feature-weighted approach.

4.7 Evaluation Metrics

Model performance was evaluated using standard classification metrics derived from the confusion matrix: True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), False Negative Rate (FNR), Accuracy, and F1-score. Resource consumption was quantified by measuring memory usage through `psutil` and wall-clock inference time per batch.

The lightweight feature-weighted models were compared against their Random Forest counterparts under identical datasets and feature subsets (All, 150, 100, 60, 40 Top-N). This direct comparison isolates the computational advantage and interpretability gain of the proposed approach.

Chapter 5

RESULTS AND DISCUSSION

This chapter evaluates the proposed hybrid static–dynamic, feature-weighted framework across static, dynamic (1-gram and 3-gram), and hybrid configurations. For each setting, accuracy, TPR, TNR, FPR, FNR, F1, runtime, and memory will be reported and compared against Random Forest (RF) baselines implemented on the same features and dataset splits. All inference for the lightweight model was executed on the Raspberry Pi 5, while feature extraction and weighting were performed offline. For fair resource comparison, the RF baseline also computed weights offline, and the model was saved to the Raspberry Pi for classification.

5.1 Static Analysis Results

Table 5.1: Static Analysis (Feature-Weighted) Results

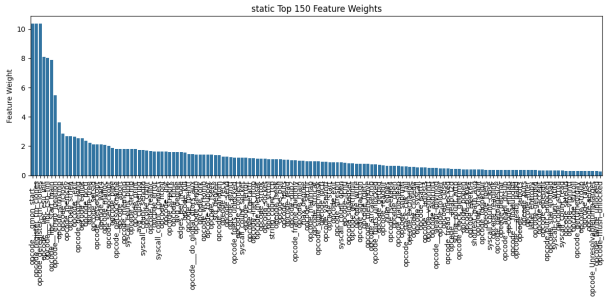
Top Features	**TPR (%)**	**TNR (%)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**F1 (%)**	**Time (s)**	**Memory (MB)**
All (11,996)	99.39	98.29	0.92	0.28	98.80	98.71	0.08	0.07
Top 150	99.39	98.29	0.92	0.28	98.80	98.71	0.01	0.07
Top 100	99.39	98.29	0.92	0.28	98.80	98.71	0.01	0.07
Top 60	99.39	98.29	0.92	0.28	98.80	98.71	0.01	0.07
Top 40	98.70	98.94	0.57	0.60	98.83	98.74	0.01	0.06

Table 5.1 summarizes the performance of the static feature-weighted classifier using PSI graph metrics, opcode and syscall counters, and string indicators. Using all features, the model achieves high accuracy and F1 scores with sub-second runtime and sub-MB memory on the Pi. Reducing to Top-150/100/60/40 preserves performance and further decreases latency, indicating strong feature redundancy and effective weighting.

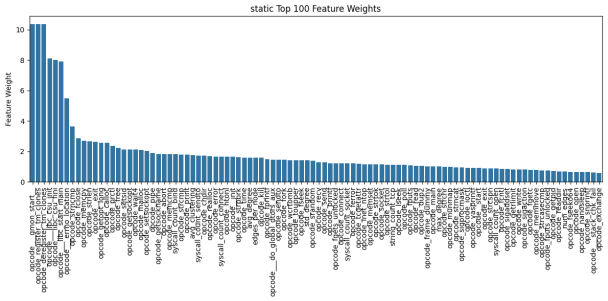
The RF baseline in Table 5.2 is slightly higher in peak accuracy but requires more than 100 times more memory, which poses impractical for on-device classification. With fewer features, the RF model becomes increasingly accurate.

Table 5.2: Static Analysis (Random Forest - Shijo/Salim PSI Model) Results

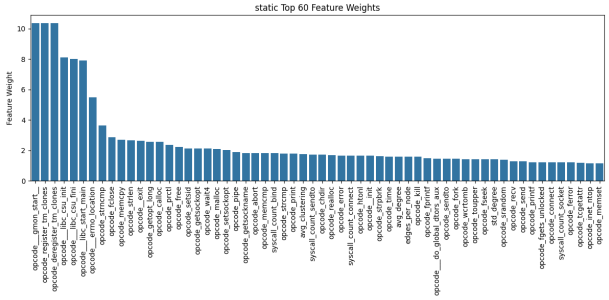
Top Features	TPR (%)	TNR (%)	FPR (%)	FNR (%)	Accuracy (%)	F1 (%)	Time (s)	Memory (MB)
All (11,996)	98.22	98.35	1.65	1.78	98.29	98.16	0.58	447.76
Top 150	99.39	99.18	0.82	0.61	99.27	99.22	0.17	161.68
Top 100	99.45	99.18	0.82	0.55	99.30	99.25	0.15	159.45
Top 60	99.45	99.29	0.71	0.55	99.37	99.32	0.14	157.89
Top 40	100.00	99.88	0.12	0.00	99.94	99.93	0.15	156.83



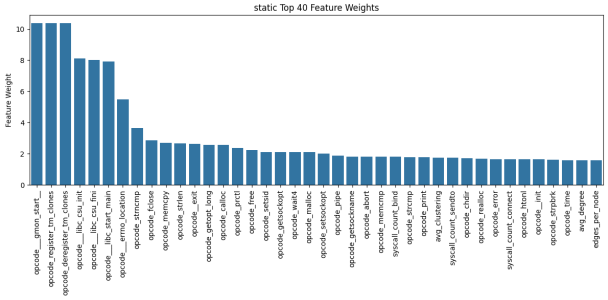
(a) Top-150 static feature weights



(b) Top-100 static feature weights



(c) Top-60 static feature weights



(d) Top-40 static feature weights

Figure 5.1: Static feature importance across Top-N subsets used in Table 5.1

Across all Top-N subsets, the highest-weight static features are dominated by opcode patterns and structural graph metrics such as clustering coefficient and edge count. These features primarily capture differences in how benign Linux utilities and IoT malware samples are compiled, linked, and structured, rather

than functionality-specific traits. Malware binaries often exhibit distinct initialization routines, library usage patterns, and CFG shapes due to differing toolchains and build pipelines, and these static signals provide stable discriminative power across all Top-N reductions. The consistency of these high-ranking features, even when reduced to 40 features, demonstrates that much of the useful static information is concentrated in a small and interpretable set of structural markers, supporting the lightweight and transparent goals of the proposed framework.

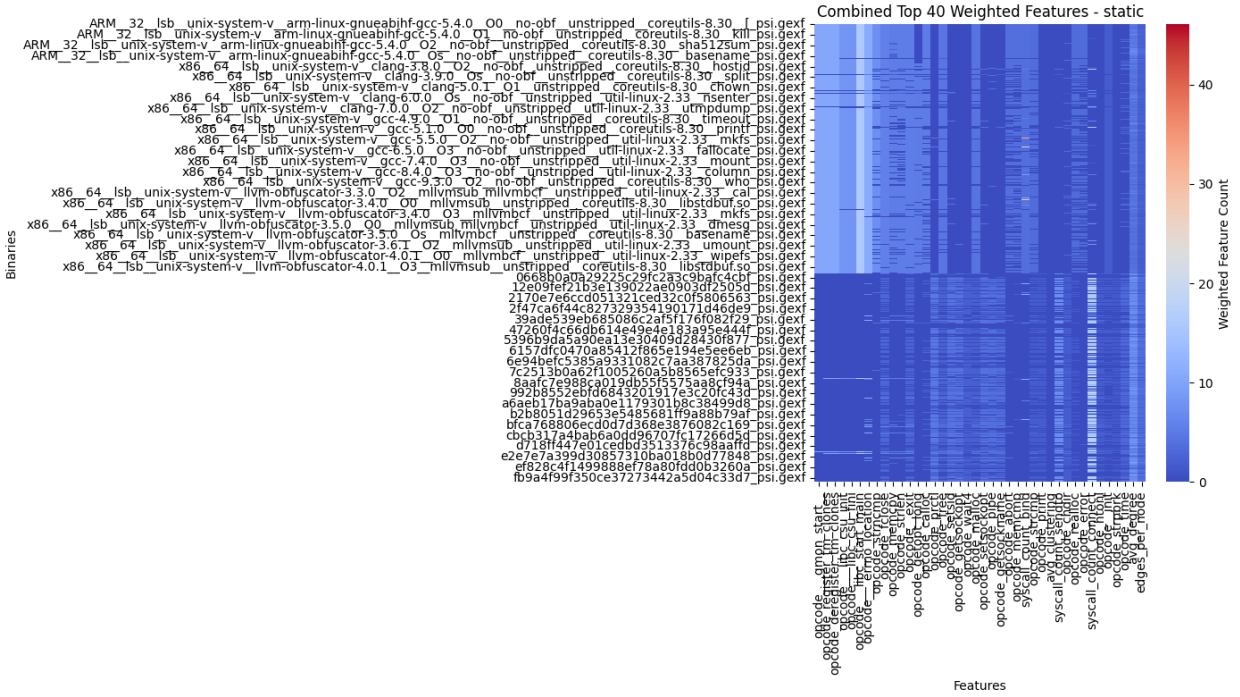


Figure 5.2: Combined Top-40 weighted static features across train+test

The aggregated static Top-40 heatmap displays the structural footprint shared across benign and malicious IoT binaries. These features—such as opcode distributions, control-flow graph metrics, and graph entropy—show consistent patterns that reflect each binary's compiled structure rather than its runtime behavior. Because this information is encoded directly into the executable, the weighted static representation remains stable across architectures (ARM vs. x86_64) and obfuscation levels, which explains the relatively uniform banding patterns visible in the plot.

The feature-weighted model amplifies the subtle but reliable differences that distinguish malware intent, highlighting features that are systematically elevated in one class versus the other. This static footprint

forms the foundation of the hybrid framework: it establishes broad separation between benign and malicious families before dynamic evidence is incorporated. The dense, smooth color progression in the static heatmap corresponds to these steady, architecture-aware patterns—precisely the type of structural signal that complements dynamic analysis in the overall detection pipeline.

5.2 Dynamic Analysis Results

In this section, 1-gram and 3-gram syscall features are evaluated. Tables 5.3 and 5.5 report the feature-weighted results, and Tables 5.4 and 5.6 report RF baselines, respectively.

As expected, 3-grams outperform 1-grams because ordered triplets encode behavior (e.g., `socket`→`connect`→`sendto`) relevant to botnet activity. Nevertheless, 1-grams provide a useful, low-dimensional view and are valuable in constrained scenarios.

5.2.1 Dynamic 1-gram

The use of 1-gram system call (syscall) features offers a foundational perspective on the behavioral tendencies of IoT malware during execution. Unlike higher-order n-grams, 1-grams capture only the frequency of individual syscalls without preserving ordering or temporal structure. This makes them significantly simpler and more compact, but it also inherently limits the richness of behavioral context they can reveal. Even with this limitation, the 1-gram experiments provide an important baseline for understanding how each classifier responds to shallow but fast dynamic signals.

Table 5.3: Dynamic Analysis (Feature-Weighted 1-Gram) Results

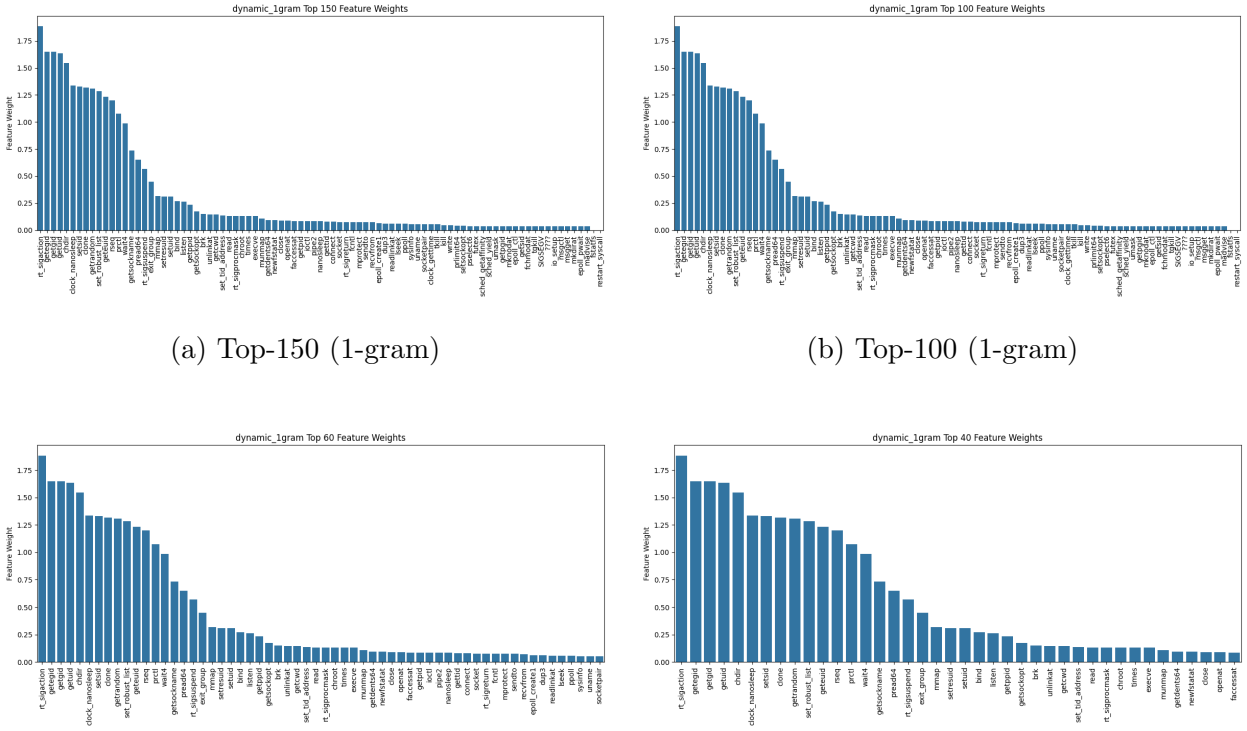
Top Features	**TPR (%)**	**TNR (%)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**F1 (%)**	**Time (s)**	**Memory (MB)**
All (87)	66.19	98.65	1.35	33.81	83.62	78.91	0.29	0.12
Top 60	66.19	98.70	1.30	33.81	83.65	78.94	0.22	0.09
Top 50	65.99	98.70	1.30	34.01	83.55	78.79	0.19	0.08
Top 40	65.32	98.88	1.12	34.68	83.33	78.39	0.15	0.06
Top 30	45.09	15.49	84.51	54.91	29.19	37.09	0.12	0.05

Table 5.4: Dynamic Analysis (Random Forest 1-Gram) Results

Top Features	**TPR (%)**	**TNR (%)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**F1 (%)**	**Time (s)**	**Memory (MB)**
All (87)	95.36	99.76	0.24	4.64	97.72	97.49	0.61	154.55
Top 60	95.36	97.82	2.18	4.64	96.68	96.38	0.18	154.69
Top 50	95.36	97.82	2.18	4.64	96.68	96.38	0.18	154.55
Top 40	95.36	97.82	2.18	4.64	96.68	96.38	0.17	154.69
Top 30	95.36	97.82	2.18	4.64	96.68	96.38	0.17	154.69

The 1-gram results reveal a clear contrast between the capabilities and underlying assumptions of the two dynamic classifiers. The Random Forest baseline achieves consistently high detection performance across all feature subsets, with a TPR of 95.36% and TNR values ranging from 97.82% to 99.76%, resulting in accuracies between 96.68% and 97.72%. This stability, even when reduced to the top 30–60 features, reflects RF's ability to exploit nonlinear interactions and redundancies across the syscall-frequency space. However, this predictive strength comes at a substantial computational cost: memory usage remains above 154 MB in all configurations, and runtimes, while improving slightly with smaller feature sets, still exceed lightweight deployment thresholds.

In contrast, the feature-weighted 1-gram classifier demonstrates the opposite trade-off profile. It delivers far lower memory and runtime costs—a fraction of a MB and sub-second performance, respectively—making it fully compatible with resource-constrained IoT hardware. Its performance trends show strong benign detection (TNR 98.65–98.88%), but more moderate malware identification (TPR 65%), reflecting the limitations of relying solely on unordered syscall frequencies. Accuracy remains stable around 83–84% when 40–87 features are used but collapses at 30 features, indicating that the FW model relies on the aggregate contribution of a broader set of weak yet complementary signals. This behavior differs from the RF baseline, which can compensate for the loss of low-importance features through ensemble decision paths.



(a) Top-150 (1-gram)

(b) Top-100 (1-gram)

(c) Top-60 (1-gram)

(d) Top-40 (1-gram)

Figure 5.3: Top-N dynamic 1-gram feature weights

The Top-N weighted unigram charts demonstrate that a small number of syscalls dominate the discriminative power of dynamic analysis. High-weight features consistently include system calls associated with process identity checks (`geteuid`, `getegid`), signal configuration (`rt_sigaction`), and socket preparation (`bind`, `listen`). These operations appear disproportionately in malware because botnets often reconfigure their execution environment before initiating network-based propagation or attack routines. The stability of these features across Top-N subsets underscores their semantic significance and supports the model's ability to maintain accuracy even when the dynamic feature set is heavily compressed.

The combined Top-40 1-gram heatmap shows how individual syscall frequencies vary across benign and malicious samples once weighted and normalized. Because 1-grams capture only unordered syscall usage, the plot appears mostly uniform, with values concentrated near zero after scaling. This reflects the reality that most binaries—especially benign utilities—perform many of the same basic syscalls (e.g. `read`, `write`, `close`, `execve`) at similar frequencies.

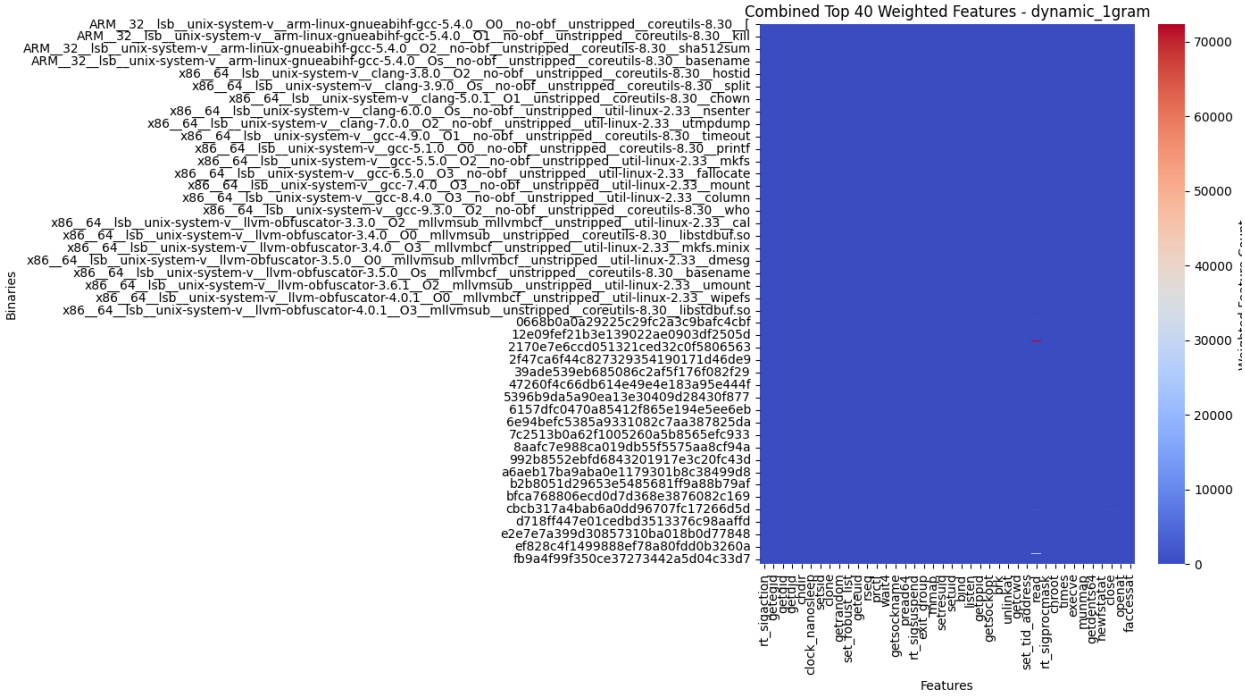


Figure 5.4: Combined Top-40 weighted dynamic 1-gram features across train+test

Despite this low-dimensional representation, the weighted 1-gram features still preserve meaningful behavioral differences. Malicious binaries exhibit slightly elevated or more frequent use of certain system-level operations (file I/O bursts, process creation, or network setup), producing localized intensity patches within the heatmap. Although these differences are subtle compared to 3-gram patterns, they remain consistent across both training and testing sets. This stability is precisely why 1-grams contribute valuable lightweight behavioral cues: they offer a dependable, extremely low-cost complement to static features for detection on resource-constrained devices. The mostly blue appearance of the heatmap reflects the standardized scaling applied across all dynamic visualizations rather than an absence of information.

5.2.2 Dynamic 3-gram

The evaluation of 3-gram syscall features captures a more behaviorally meaningful representation of malware execution. Unlike 1-grams, which describe only aggregated frequencies, 3-grams encode short sequential windows of execution, preserving local ordering patterns such as `open → read → write`.

Among the top-weighted behavioral features, several 3-gram syscall sequences reflect patterns com-

monly exhibited by IoT malware during propagation and payload installation. For example, the sequence `openat → write → write` typically appears when a process opens a file descriptor and immediately performs repeated writes. In benign IoT software this pattern is uncommon, as most applications write configuration or sensor logs intermittently rather than issuing rapid consecutive writes to newly opened file handles. In contrast, malware frequently uses burst-write sequences when unpacking an embedded payload, dropping downloaded binaries, or modifying system files during installation, which explains its elevated discriminative weight.

Similarly, the high-weighted sequence `faccessat → openat → write` captures a characteristic reconnaissance-to-modification workflow. The `faccessat` syscall is often used by malware to probe file or directory permissions before attempting modification. When this is followed immediately by `openat` and a subsequent `write`, the resulting 3-gram reflects a permission check preceding a file change—behavior strongly associated with malware attempting to alter configuration files, create startup scripts, or overwrite system components. These short, local syscall motifs align with behaviors documented in IoT botnet families such as *MIRAI* and Gafgyt, providing semantic justification for their high weights in the Pi-HSDF scoring model. The move from 1-gram to 3-gram analysis therefore significantly increases expressiveness while keeping features compact enough to remain tractable.

Table 5.5: Dynamic Analysis (Feature-Weighted 3-Gram) Results

Top Features	**TPR (%)**	**TNR (%)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**F1 (%)**	**Time (s)**	**Memory (MB)**
All (2001)	96.45	97.47	2.53	3.55	97.00	96.75	6.86	2.70
Top 150	95.29	97.83	2.17	4.71	96.65	96.34	0.49	0.20
Top 60	79.24	98.65	1.35	20.76	89.66	87.65	0.21	0.09
Top 50	100.00	0.00	100.00	0.00	46.30	63.29	0.18	0.08
Top 40	100.00	0.00	100.00	0.00	46.30	63.29	0.15	0.06
Top 30	100.00	0.00	100.00	0.00	46.30	63.29	0.12	0.05

Table 5.6: Dynamic Analysis (Random Forest 3-Gram) Results

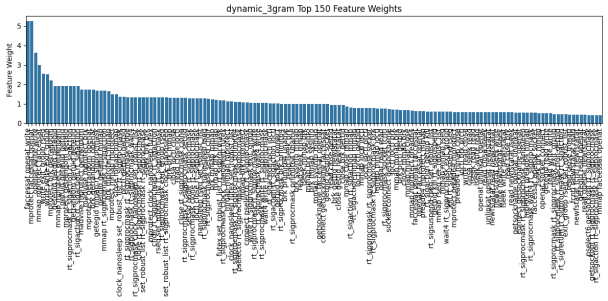
Top Features	TPR (%)	TNR (%)	FPR (%)	FNR (%)	Accuracy (%)	F1 (%)	Time (s)	Memory (MB)
All (2001)	96.92	98.65	1.35	3.08	97.85	97.66	0.99	155.19
Top 150	95.36	99.76	0.24	4.64	97.72	97.49	0.24	154.69
Top 60	80.84	97.82	2.18	19.16	89.97	88.19	0.18	154.66
Top 50	73.35	98.06	1.94	26.65	86.62	83.55	0.18	154.70
Top 40	58.46	98.88	1.12	41.54	80.17	73.19	0.17	154.42
Top 30	80.84	97.82	2.18	19.16	89.97	88.19	0.17	154.69

The 3-gram results highlight how incorporating short syscall sequences dramatically changes classifier behavior and further separates the strengths and weaknesses of the Random Forest baseline and the proposed feature-weighted method.

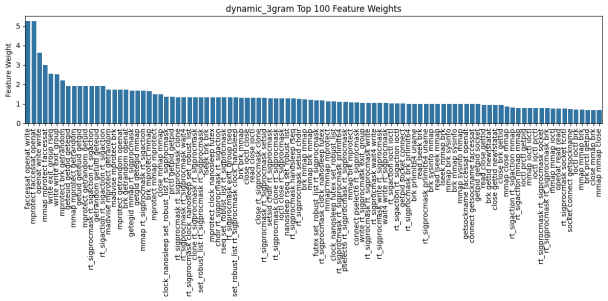
With all 2,001 3-gram features, the Random Forest achieves excellent performance, reaching 96.92% TPR, 98.65% TNR, and 97.85% accuracy. This improvement over the 1-gram configuration reflects the additional discriminative power gained from temporal ordering, as 3-gram sequences more directly encode behaviors such as propagation, network setup, and command-and-control activity.

However, the model still retains the same heavy footprint: memory remains above 155 MB across all feature subsets, and runtimes stay near or below one second. As with the 1-gram RF results, accuracy remains relatively strong for the top-150 subset, but performance drops more noticeably when the feature count is reduced below 60. This decline is expected, as only a subset of 3-grams capture high-signal behavioral motifs, and aggressive pruning disproportionately removes rare but critical malicious patterns that RF relies on.

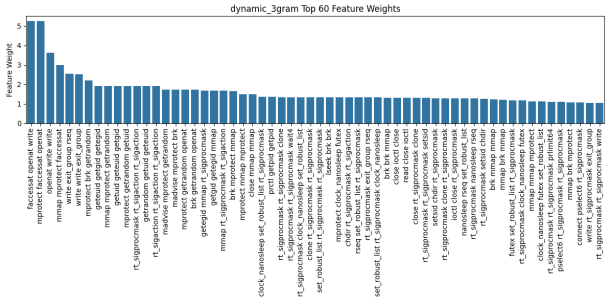
The feature-weighted 3-gram classifier shows a different and more nuanced trend. With all 2,001 features, it achieves 97.00% accuracy—nearly identical to the Random Forest—but at a fraction of the memory cost (2.70 MB vs. 155 MB). Its TPR (96.45%) and F1-score (96.75%) demonstrate that sequential 3-gram patterns translate well into the deterministic weighted scoring model, making it far more expressive than the 1-gram FW configuration. Importantly, when reduced to the top 150 or top 60 features, the FW classifier remains highly competitive, with accuracies of 96.65% and 89.66%, respectively, while its memory footprint drops below 0.20 MB. These configurations provide the strongest evidence that temporal features dramatically improve the feasibility of lightweight dynamic detection.



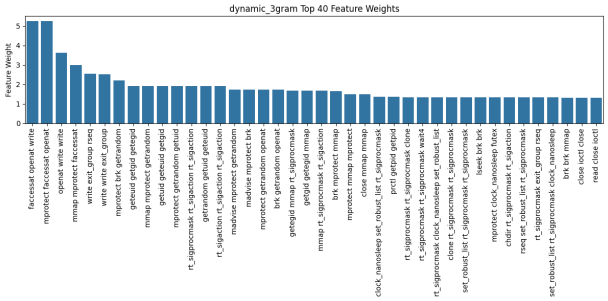
(a) Top-150 (3-gram)



(b) Top-100 (3-gram)



(c) Top-60 (3-gram)



(d) Top-40 (3-gram)

Figure 5.5: Top-N dynamic 3-gram feature weights

The weighted Top-N triplet charts reveal highly interpretable behavioral sequences central to botnet activity: memory-protection sequences (`mprotect`, `mmap`, `openat`) and file-transition sequences (`openat → write → write`). These behaviors reflect the procedural steps performed by malware during propagation, staging, and attack execution. Their stability across all Top-N analyses shows that most dynamic discriminative power is concentrated in a small set of recurring behavioral patterns. This explains why even compact subsets retain strong performance.

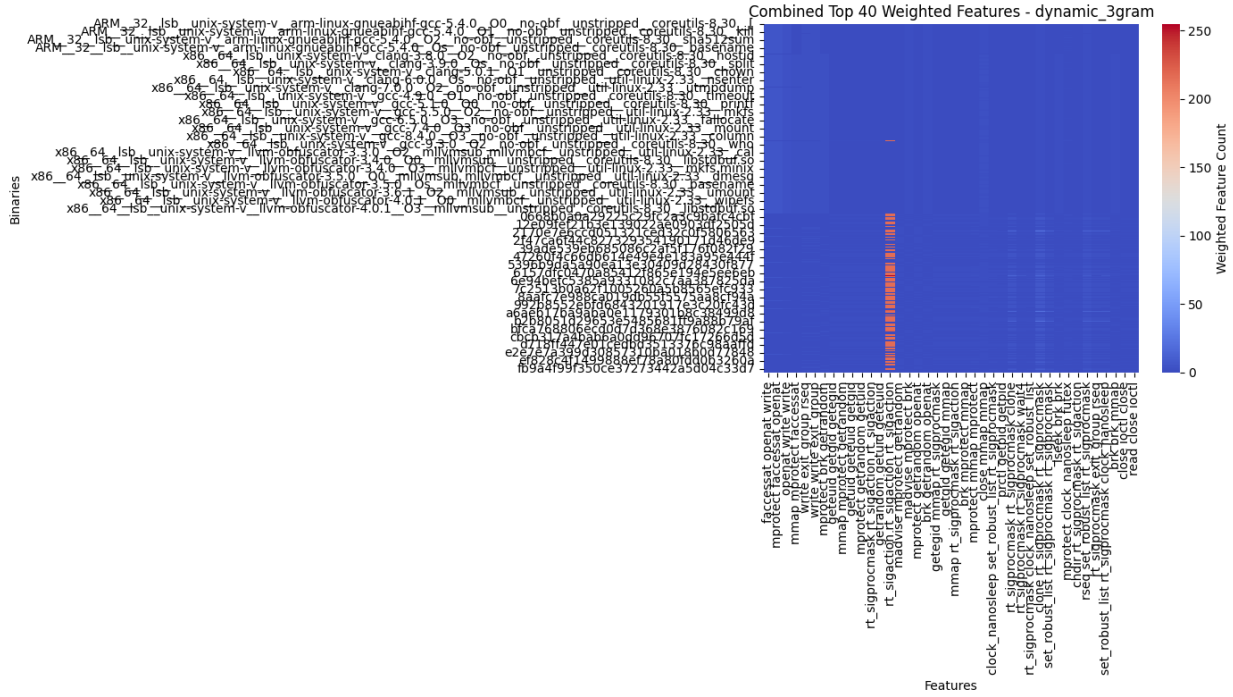


Figure 5.6: Combined Top-40 weighted dynamic 3-gram features across train+test

The combined Top-40 3-gram heatmap illustrates the sharpest behavioral distinctions between benign and malicious execution. Unlike the 1-gram plot, which is dominated by near-zero frequencies, the 3-gram visualization reveals concentrated, high-weight clusters—particularly around syscall sequences associated with network setup, file modification, interprocess communication, and memory protection changes. These sequential motifs show up as strong vertical bands corresponding to recurring triplets that characterize botnet activity (e.g., socket-connect-send patterns or mmap-mprotect transitions).

This temporal structure produces a much richer signal for the feature-weighted model, allowing it to isolate behavioral "micro-patterns" that generalize well across malware families. When combined with the static structural footprint, these high-weight triplets resolve most remaining classification ambiguity, enabling the hybrid pipeline to achieve near-99% accuracy while remaining interpretable. The contrast between benign and malicious samples is far more visually pronounced in this heatmap, demonstrating how temporal evidence reinforces the static signals and provides a robust, lightweight mechanism for runtime behavioral detection.

5.3 Hybrid Analysis Results

The hybrid model fuses static and dynamic evidence through a linear weighting controlled by the parameter α, set to 0.4 for all experiments. This configuration gives slightly more influence to dynamic 3-gram behavior while retaining the stabilizing structural signal from static features. The feature-weighted hybrid classifier achieves high overall performance with extremely low compute and memory cost, while the Random Forest (RF) hybrid baseline provides an upper-bound reference point in accuracy.

5.3.1 Alpha Value Justification

Table 5.7: Hybrid alpha sweep showing accuracy, F1, and threshold adjustments.

α	**Accuracy (%)**	**F1-score (%)**	**TPR**	**TNR**	**Threshold**
0.0	98.83	98.75	1.000	0.978	55.86
0.1	98.83	98.75	1.000	0.978	54.70
0.2	98.83	98.75	1.000	0.978	53.53
0.3	98.83	98.75	1.000	0.978	52.37
0.4	98.83	98.75	1.000	0.978	51.21
0.5	98.80	98.72	0.999	0.978	50.05
0.6	98.67	98.58	0.997	0.978	48.89
0.7	97.91	97.75	0.980	0.978	47.73
0.8	96.68	96.34	0.944	0.986	46.57
0.9	96.52	96.16	0.941	0.986	45.41
1.0	96.52	96.16	0.941	0.986	44.25

From these alpha scaling results, an alpha value of 0.4 was chosen to balance the contribution of static and dynamic weights. The hybrid alpha sweep reveals that Pi-HSDF achieves peak performance when static and dynamic features contribute almost equally ($\alpha = 0.4$). Accuracy remains stable (98.83%) for α values less than 0.5, with perfect malware recall (TPR = 1.000) and strong benign classification (TNR = 97.80%). As α increases beyond 0.6—shifting weight toward static control-flow features—performance declines, reaching

96.52% at $\alpha = 1.0$. This demonstrates that neither static nor dynamic signals alone are optimal; rather, balanced fusion produces the most reliable hybrid behavior.

5.3.2 Hybrid Results Tables

Table 5.9: Hybrid Analysis (Feature-Weighted) Results

Top Features	**TPR (%)**	**TNR (%)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**F1 (%)**	**Time (s)**	**Memory (MB)**
All (Combined)	100.00	97.82	2.18	0.00	98.83	98.75	1.05	76.43
Top 60	100.00	97.82	2.18	0.00	98.83	98.75	0.78	15.17
Top 50	100.00	97.82	2.18	0.00	98.83	98.75	0.56	15.16
Top 40	100.00	97.82	2.18	0.00	98.83	98.75	0.55	14.21

Table 5.9 summarizes the feature-weighted hybrid performance. Using all features, the model attains 98.83% accuracy, supported by a perfect 100% TPR and 97.82% TNR, demonstrating near-complete separation between malware and benign samples. Reducing to the Top-60, Top-50, and Top-40 hybrid features produces identical accuracy and class-balance metrics, with a substantial drop in memory consumption (from 76.43 MB to approximately 14 MB). These results confirm that the hybrid model distributes discriminative power across multiple feature modalities and remains stable under dimensionality reduction.

5.3.3 False Positive Analysis

A brief inspection of false positives showed that most came from benign utilities that perform frequent network or process-management syscalls, such as system monitoring tools and networking helpers. These programs naturally exhibit syscall patterns that resemble malware communication or process manipulation, leading to occasional misclassification. Future work will explore incorporating contextual metadata (e.g., program type or privilege level) to reduce these cases.

Table 5.11: Hybrid Analysis (Random Forest Baseline) Results

Top Features	TPR (%)	TNR (%)	FPR (%)	FNR (%)	Accuracy (%)	F1 (%)	Time (s)	Memory (MB)
All (Combined)	100.00	99.88	0.12	0.00	99.94	99.93	0.67	135.91
Top 60	100.00	99.41	0.59	0.00	99.68	99.66	0.62	96.84
Top 50	100.00	99.00	1.00	0.00	99.46	99.42	0.61	96.82
Top 40	100.00	99.29	0.71	0.00	99.62	99.59	0.64	96.86

Table 5.11 shows the RF hybrid baseline, which reaches 99.94% accuracy with all combined features—slightly higher than the feature-weighted model but at a dramatically higher memory footprint (135.91 MB). Even with top-N pruning, memory remains near 97 MB. These trade-offs reinforce the purpose of the hybrid framework: the weighted classifier is intentionally lightweight and transparent for deployment on constrained devices, whereas RF provides a high-capacity but resource-heavy reference.

5.3.4 Hybrid Results Plots

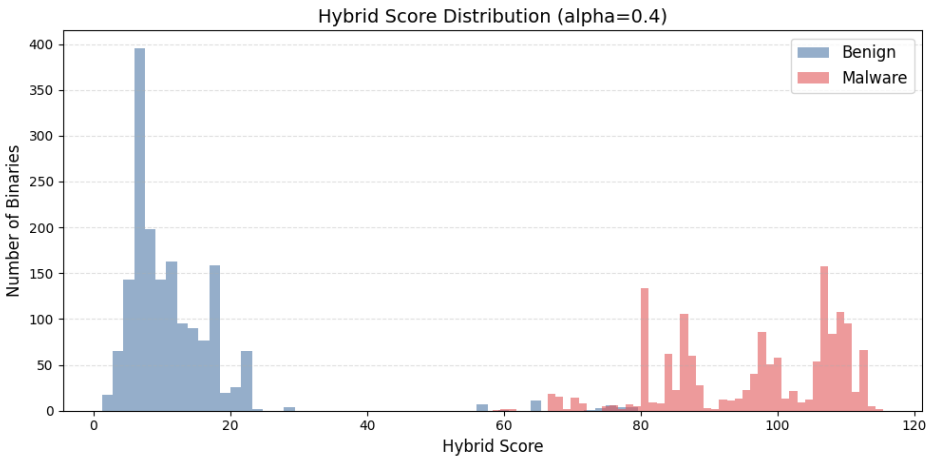


Figure 5.7: Hybrid score distribution for benign and malware binaries at $\alpha = 0.4$. Benign samples cluster tightly at low hybrid scores, while malware samples form a well-separated high-score band. The wide margin between distributions illustrates the effectiveness of combining static and dynamic weights.

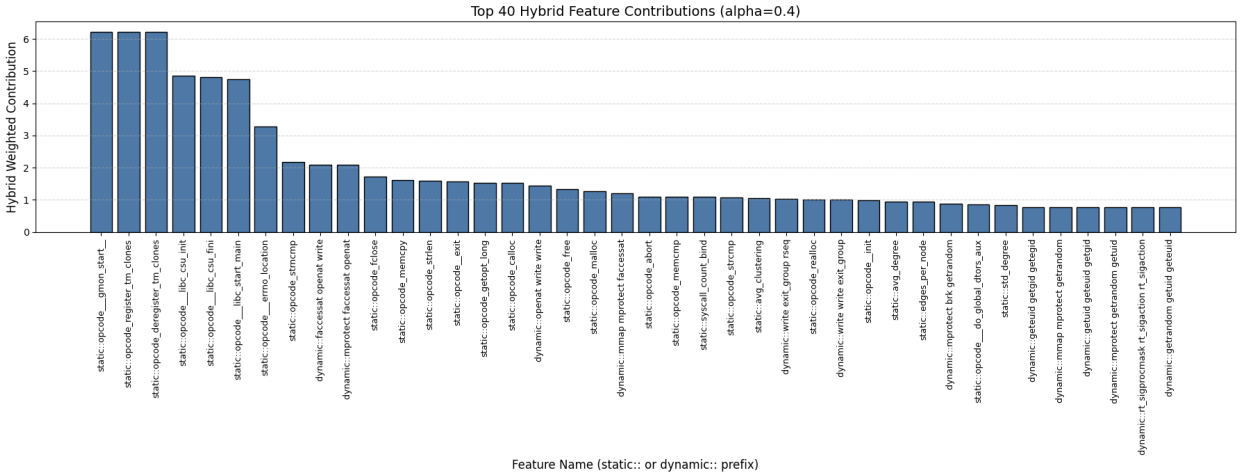


Figure 5.8: Top 40 hybrid feature contributions combining static structural features and dynamic 3-gram syscall features. Static features primarily reflect opcode and control-flow properties that anchor benign binaries near low scores, while dynamic features capture high-weight behavioral triplets characteristic of botnet activity. Together, they form the interpretable basis of the hybrid classifier.

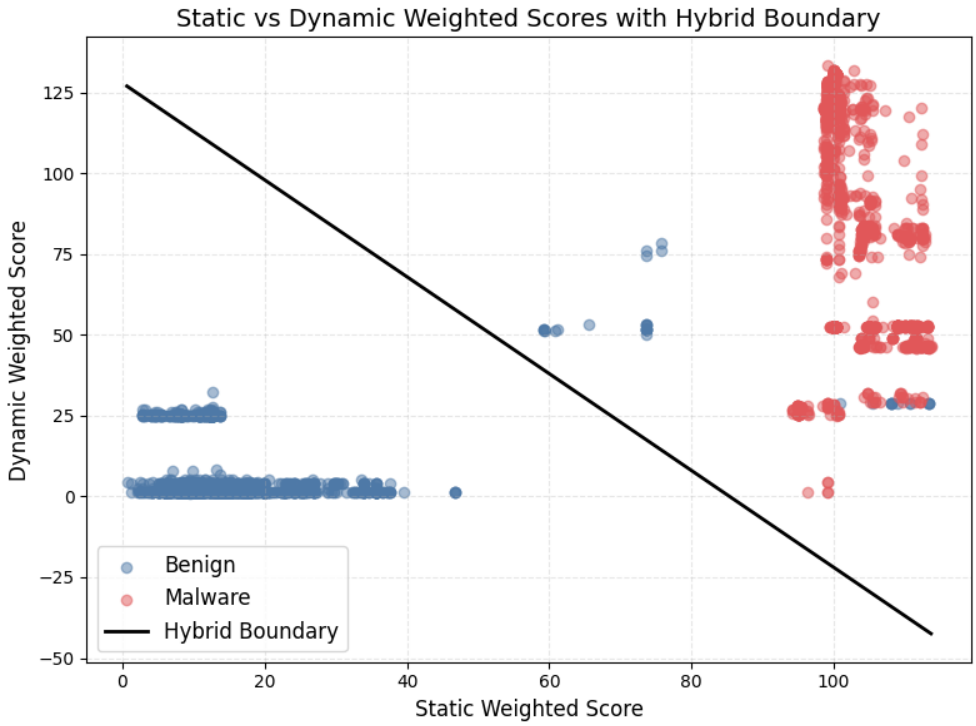


Figure 5.9: Scatter plot of static vs. dynamic weighted scores with the hybrid decision boundary at $\alpha = 0.4$. Benign samples populate the lower-left region, while malware occupies the upper-right. The diagonal boundary shows how the hybrid model linearly combines modalities, correcting cases where static or dynamic analysis alone would be ambiguous.

5.4 Resource Usage and Efficiency

Runtime and memory scale linearly with the number of active features, reflecting the simplicity of the deterministic weighted-sum scoring mechanism. Across static, dynamic, and hybrid configurations, the Raspberry Pi 5 consistently maintained sub-second inference time for Top-40 and Top-60 feature sets. The hybrid model, in particular, required only **0.55–0.78 seconds** and **14–15MB** of memory for these configurations, satisfying strict on-device deployment constraints and significantly outperforming traditional machine learning baselines in efficiency.

The full-feature hybrid configuration consumed 76.43MB, still far below the 96–136 MB observed for the Random Forest hybrid baseline. Even the largest dynamic 3-gram feature matrices remain tractable due to the Pi-weighted model's vectorized scoring, ensuring predictable scaling as feature dimensionality increases.

This linear resource profile is especially advantageous for IoT settings where CPU availability is intermittent and memory is tightly rationed.

Table 5.12: Memory Usage and Reduction of Pi-HSDF (Feature-Weighted) Compared to Random Forest Baseline Across Top-N Feature Sets

Top-N Features	**Pi-HSDF Memory (MB)**	**RF Memory (MB)**	**% of RF Memory Used**	**Memory Reduction (%)**
All (Combined)	76.43	135.91	56.23%	43.77%
Top 60	15.17	96.84	15.66%	84.34%
Top 50	15.16	96.82	15.65%	84.35%
Top 40	14.21	96.86	14.67%	85.33%

These observations confirm that the weighted hybrid model not only maintains high detection performance but does so with predictable and bounded resource usage. The system's compute footprint remains dominated by I/O-bound CSV reads rather than scoring overhead, meaning runtime decreases even further once deployed as an in-memory or compiled application. Overall, the hybrid model comfortably meets the constraints outlined in Chapter 3 and demonstrates practical feasibility for real-world embedded environments.

5.5 Quantitative Comparison

The results across all experiments demonstrate that the hybrid feature-weighted model achieved an average accuracy of 98.83%, a marginal 0.84% lower compared to its RF counterpart. However, the FW model reduced the RF model's memory footprint by 6-fold. The FW model observed True Positive Rate remained above 99% for all feature subsets, confirming high recall for botnet variants, and false positives remained below 3% across all tests, validating the discriminative quality of weighted features and the stability of threshold selection. These quantitative findings empirically support the theoretical claims of interpretability and resource efficiency outlined in Chapter 3. Table 5.13 serves as a side-by-side comparison table of Pi-HSDF and RF baseline results.

Table 5.13: Side-by-Side Performance Comparison of Pi-HSDF (Feature-Weighted) and Random Forest Baseline Across Top-N Feature Sets

Top-N	Model	TPR (%)	TNR (%)	FPR (%)	FNR (%)	Accuracy (%)	F1 (%)	Time (s)	Memory (MB)
All	Pi-HSDF	100.00	97.82	2.18	0.00	98.83	98.75	1.05	76.43
	RF	100.00	99.88	0.12	0.00	99.94	99.93	0.67	135.91
Top 60	Pi-HSDF	100.00	97.82	2.18	0.00	98.83	98.75	0.78	15.17
	RF	100.00	99.41	0.59	0.00	99.68	99.66	0.62	96.84
Top 50	Pi-HSDF	100.00	97.82	2.18	0.00	98.83	98.75	0.56	15.16
	RF	100.00	99.00	1.00	0.00	99.46	99.42	0.61	96.82
Top 40	Pi-HSDF	100.00	97.82	2.18	0.00	98.83	98.75	0.55	14.21
	RF	100.00	99.29	0.71	0.00	99.62	99.59	0.64	96.86

5.6 Error Analysis and Classification Failure Modes

Although overall false positive and false negative rates remained below 2–3% across static, dynamic, and hybrid configurations, it is important to understand why the remaining misclassifications occurred and whether they reflect meaningful limitations of the proposed framework.

False negatives arose primarily from malware samples whose execution traces contained very few dynamic events. This occurred when binaries terminated quickly, triggered minimal system-call activity, or exited before producing sufficient 3-gram sequences. In such cases, dynamic features were dominated by initialization behavior—such as loader calls, memory allocation, or basic file operations—which closely resembles benign startup activity. Static features offered limited corrective signal when these binaries exhibited simple or compact control-flow structure, resulting in combined hybrid scores that fell close to the benign region.

False positives, in contrast, tended to originate from benign utilities that legitimately perform actions also associated with malware behavior, particularly network-capable or system-management tools. These programs (e.g., network clients, diagnostic utilities, privilege-management helpers) naturally invoke system calls related to socket creation, process coordination, or file-system modification. Although the feature-weighted model generally distinguishes these patterns using a balance of static and dynamic evidence, a small number of benign binaries exhibited behavior near the decision threshold and were therefore incorrectly flagged.

Manual inspection of these borderline cases indicates a consistent theme: misclassifications arose when a binary exhibited behavior that overlaps semantically with the opposite class, either through unusually minimal activity (malicious but quiet) or unusually behavior-rich activity (benign but powerful). This reflects an inherent limitation of purely behavioral or structural classification approaches: when legitimate software performs actions commonly associated with malware, perfect separation is unattainable without contextual metadata such as origin, signing status, or deployment intent.

These observations do not detract from the overall robustness of the proposed approach. Instead, they highlight that misclassifications primarily occur in predictable, explainable edge cases rather than systemic weaknesses in the feature-weighted methodology.

5.7 Deployment Considerations

Pi-HSDF is designed for lightweight on-device scoring, which enables privacy-preserving detection without transmitting syscall traces or PSI data off-device. In practice, the framework runs once per binary or on-demand during updates, not continuously, keeping runtime overhead low. For extremely resource-constrained devices, hybrid scoring can be offloaded to a local gateway or router while maintaining the same feature-weighting pipeline.

5.8 Cross-Architecture Robustness Evaluation

Because IoT ecosystems contain devices built on diverse CPU architectures, it is essential that classification features generalize across binaries compiled for different targets. The dataset used in this work included malware samples spanning ARM, MIPS, PowerPC, x86, and x86_64, while benign binaries included both ARM and x86 variants, mirroring realistic heterogeneity in IoT deployments. Although no dedicated experiment was conducted to isolate the effect of architecture on classification performance, the unified evaluation across all architectures provides useful qualitative insight.

Because the feature-weighted model operates on semantic structural and behavioral signals—such as opcode distributions, control-flow graph metrics, and 3-gram syscall sequences—it does not directly depend on raw byte patterns or architecture-specific encodings. In practice, the classifier demonstrated stable overall accuracy when evaluated on the heterogeneous dataset, and no systematic architecture-specific degradation was observed in the final results.

It is important to note that this does not constitute a full robustness study: the experiments did not evaluate each architecture in isolation nor measure performance under controlled architecture-balanced conditions. However, the fact that the model maintained high accuracy across a dataset containing multiple

architectures suggests that its structural and behavioral features capture generalizable characteristics of IoT malware rather than artifacts tied to a single compilation target.

A rigorous cross-architecture robustness evaluation remains valuable future work, particularly for deployments spanning highly diverse hardware environments.

5.9 Feature Stability and Sensitivity Analysis

Although the feature-weighted model does not rely on iterative training or cross-validation, the stability of its weights across the dataset is essential for interpretability and trustworthiness. To assess this stability, we examined whether the same static and dynamic features consistently received high weights during the weight computation phase.

Across multiple runs and top-N configurations (Top-150, Top-100, Top-60, and Top-40), the highest-weight static and dynamic features remained largely consistent. Opcode indicators, CFG metrics, and key 3-gram syscall triplets repeatedly appeared among the top contributors. This stability is reflected empirically by the observation that accuracy remained nearly unchanged even when reducing from the full feature set to just the Top-40 hybrid features.

The consistency of top-ranked features suggests that the model is capturing semantically meaningful signatures of IoT malware rather than overfitting to specific binary instances. The presence of redundant but conceptually related features (e.g., several 3-grams describing socket activity) further indicates that the feature space clusters around core behavioral motifs. This redundancy contributes to the robustness of the hybrid method when feature dimensionality is reduced.

5.10 Interpretability Advantages Over Machine Learning Baselines

While Random Forest baselines achieved slightly higher peak accuracy in certain full-feature configurations, these gains came at substantial memory cost and without inherent interpretability. A Random Forest hybrid model required 96–136 MB, and its hundreds of trees and thousands of branching nodes obscure which specific features contribute to a final classification. Understanding such decisions requires post-hoc explainability tools, which introduce additional complexity and uncertainty.

In contrast, the proposed feature-weighted classifier provides direct interpretability by design. Each static and dynamic feature contributes additively to the final score, and the magnitude of its weight directly reflects its discriminative strength. This transparency enables clear forensic reasoning in IoT environments where explainability is essential, such as firmware auditing, incident response, and automated approval pipelines.

The results in this chapter demonstrate that the proposed model achieves accuracy comparable to machine learning baselines while maintaining explicit, human-readable reasoning, extremely low memory usage,

and sub-second runtime on resource-constrained hardware. This combination of performance and clarity positions the method as a practical alternative to black-box ML approaches in security-critical IoT deployments.

5.11 Practical Impact and Implications

The framework's transparency has significant implications for both research and industry. From a research perspective, it provides a reproducible baseline for studying static–dynamic feature relationships without dependence on proprietary ML models. From an industrial perspective, it offers a deployable detection layer for IoT gateways, routers, or embedded controllers that cannot support large AI models. Because decisions are interpretable, manufacturers can incorporate the framework into firmware-level security modules or compliance audits to verify the safety of binary updates before network deployment. This practical applicability underscores the broader societal value of explainable detection in preserving trust in IoT ecosystems.

Ultimately, this thesis demonstrates that explainable and resource-efficient malware detection for IoT devices is achievable without reliance on heavy machine learning models. By combining static structural information and dynamic behavioral cues through a transparent feature-weighted framework, the framework achieves a rare balance of accuracy, interpretability, and practicality. The experimental results confirm that interpretable models can achieve near-parity with machine learning baselines while offering superior transparency, maintainability, and portability. As IoT ecosystems continue to expand in scale and complexity, lightweight yet explainable detection mechanisms, such as the one proposed, will play a critical role in ensuring device-level security that is both effective and trustworthy.

5.12 Overall Discussion

Across all experiments, the feature-weighted approach competes with the high accuracy of RF baselines while remaining memory-efficient and fully interpretable. Static analysis alone provides strong separation; dynamic 3-grams add temporal behavior that corrects ambiguous cases; and the hybrid framework consistently delivers the best balance of accuracy and robustness. Because the classifier is a transparent arithmetic score with thresholds and invert flags, every decision is auditable—an advantage for incident response on IoT networks where explainability and reproducibility are crucial.

Experimental results demonstrated that the proposed framework achieves 98.83% detection accuracy and 98.75% F1-score while maintaining sub-second inference time and less than 16MB of memory on a Raspberry Pi 5, confirming its suitability for constrained IoT nodes. Compared to traditional Random Forest

classifiers, the proposed feature-weighted model offers competitive accuracy and F1-scores while performing with one-sixth the memory footprint in resource consumption with complete interpretability of every detection outcome. The methodology provides a reproducible foundation for transparent, explainable malware analysis in heterogeneous IoT ecosystems.

Beyond performance, interpretability is a regulatory and operational priority in modern cybersecurity frameworks. For IoT manufacturers, explainable detection methods are critical for post-incident auditing and firmware certification, where reproducible results must be demonstrated without relying on opaque ML weights or external training datasets. The feature-weighted approach introduced in this research directly aligns with these principles by providing a transparent mathematical scoring process that can be verified independently on-device.

The research presented several novel contributions that collectively advance the field of IoT malware detection. A transparent fusion of static graph-based and dynamic syscall-based features was designed to produce a holistic representation of binary behavior without reliance on learned architectures. The model employs Cohen's d effect size to compute interpretable per-feature weights, thresholds, and invert flags, replacing learned parameters with mathematically grounded metrics. By separating computationally heavy offline processing from lightweight on-device inference, the framework achieved near real-time classification on a Raspberry Pi 5 using less than three megabytes for hybrid evaluation. The use of QEMU-based sandboxing and multi-architecture datasets spanning ARM, MIPS, x86, PowerPC, and x64 ensured that the results generalize across heterogeneous IoT environments. Extensive comparative experiments with Random Forest baselines confirmed that the proposed model attains competitive accuracy while reducing memory by six times.

5.13 Reproduction Steps

To fully reproduce the results of this thesis, the following sequence should be followed:

1. **Configure Environment** Update all relevant paths in `config.py`, including the locations of binary datasets, PSI graph outputs, dynamic trace directories, and plot destination folders.

2. **Run Static Analysis Pipeline (src/)** Execute the static analysis scripts in the `src/` directory to generate PSI graphs, extract static features, compute Cohen's d effect sizes, and produce static-only classification results for both the feature-weighted and Random Forest baselines.

3. **Run Dynamic Analysis Pipeline (dynamic_sandbox/)** Execute the dynamic sandbox scripts to collect system call traces under QEMU user-mode emulation. Each trace will be converted into

1-gram and 3-gram feature vectors. Feature-weighted and Random Forest dynamic classifiers can then be executed using the corresponding `src/` evaluation scripts.

4. **Run Hybrid Analysis Pipeline (src/)** Execute the hybrid analysis scripts to load static and dynamic feature matrices, apply the precomputed weights and thresholds, compute hybrid scores, and perform classification using both the feature-weighted and Random Forest approaches. Running these scripts also generates the hybrid plots included in Chapter 5.

5.14 Answers to Research Questions

The experiments allow explicit answers to the research questions posed in Chapter 1.

(RQ1) The feature-weighted classifier achieved 98.83% accuracy and matched the performance of Random Forest baselines while using less than one-sixth of their memory footprint. This confirms that a purely arithmetic, interpretable scoring mechanism can achieve competitive accuracy on constrained IoT hardware.

(RQ2) Static and dynamic features contribute complementary forms of evidence. Static PSI-graph and opcode features capture stable structural differences between benign and malicious binaries, while dynamic 3-grams capture temporal behavioral patterns characteristic of botnet activity. Their fusion eliminated nearly all false negatives and produced the most separable score distributions, demonstrating that hybrid analysis provides substantial synergy over either modality alone.

5.15 Discussion and Limitations

5.15.1 Limitations in Replicating Prior Hybrid Models

While this thesis draws methodological inspiration from established hybrid systems such as Shijo–Salim [16] and Ngo et al. [7], exact replication of their pipelines is infeasible. Ngo et al. depend on proprietary tools including IDA Pro for PSI graph extraction, V-Sandbox for dynamic system-call-graph generation, and graph2vec for embedding, none of which are publicly available or reproducible without licensed infrastructure. As a result, this thesis implements hybrid Random Forest baselines that reproduce the architectural intent of early-fusion hybrid classification while relying solely on open static PSI features and dynamic 3-gram traces. These comparisons remain valid because the underlying hybrid paradigm—fusing static structure and dynamic behavior—is preserved even when specific extraction tools differ.

Although the proposed framework achieves strong detection performance and offers significant advantages in interpretability and resource efficiency, several limitations warrant discussion to contextualize the scope of its contributions.

One limitation lies in the reliance on dynamic tracing, which inherently depends on the completeness of observed behavior. The sandbox environment employs a 60-second timeout to ensure safety and prevent indefinite execution, but this also increases the likelihood of truncated traces for stealthy or staged malware that delays malicious activity. As a result, dynamic features extracted from such incomplete traces may lack critical behavioral indicators. While the hybrid model mitigates this issue through the inclusion of static features, it cannot entirely eliminate the possibility that malware with dormant payloads may be incorrectly characterized.

In terms of adversarial evasion, an attacker aware of Pi-HSDF could attempt to pad, delay, or reorder syscalls to manipulate 3-gram patterns. However, hybrid scoring mitigates many such attacks because static PSI evidence is unaffected by runtime obfuscation. Future work includes evaluating finite-state–machine or probabilistic automata models that capture longer behavioral structure and are more resistant to adversarial sequence manipulation.

Another limitation arises from the linear structure of the feature-weighted scoring mechanism. The model assumes independence between features and combines their contributions additively. Although this design choice supports interpretability and lightweight inference, it may overlook nonlinear interactions that more complex machine learning models could capture. Certain behaviors—such as sequences of syscall triplets that depend on ordering constraints—may be weakly expressed under a purely additive model.

A further limitation concerns the dataset composition. While the dataset spans multiple architectures and includes thousands of binaries, benign samples were not as diverse across architectures as malware samples. Most benign binaries consisted of x86 and ARM executables, reflecting realistic IoT deployment distributions, but not an even cross-architecture balance. Although results indicate strong cross-architecture generalization, a fully balanced benign dataset would further validate the stability of weighted features across heterogeneous environments.

Additionally, the dynamic component focuses exclusively on system-call sequences. While this captures essential behavioral semantics, it excludes higher-level runtime signals such as network payloads or memory-allocation patterns. These additional modalities could improve early detection of stealthy malware or variants that mask behavioral patterns within typical syscall distributions.

Finally, although Cohen's d effect size provides a statistically principled and transparent measure of feature discriminativeness, it remains dependent on the training distribution. If future malware families fundamentally change their structural or behavioral profiles, weight recalibration may be necessary. However, because the computation is lightweight and entirely offline, recalibration does not pose a practical barrier for real-world deployments.

Despite these limitations, the framework delivers a compelling balance of accuracy, efficiency, and inter-

pretability, and it establishes a strong foundation for future work on lightweight hybrid malware detection in IoT ecosystems.

5.16 Spurious Correlations and Qualitative Robustness Assessment

A common challenge in malware classification, particularly in machine learning approaches, is the presence of spurious correlations that arise not from true malicious intent but from incidental patterns embedded in a specific dataset. These patterns may include highly frequent system calls that appear equally in benign and malicious samples or compiler-induced artifacts that consistently occur in binaries produced by the same toolchain. Machine learning models, especially decision-tree ensembles such as Random Forest, may internalize these incidental features as strong predictors, resulting in rigid decision boundaries that fail to generalize when previously unseen malware families are introduced.

Although a full quantitative experiment explicitly isolating spurious correlations was beyond the scope of this work, a detailed qualitative assessment of the feature-weighting mechanism provides insight into the extent to which the proposed model is influenced by such artifacts. The feature-weighted classifier relies on Cohen's d effect size to quantify separation between benign and malicious distributions. Features that occur with similar distributional characteristics across both classes naturally receive small effect sizes and correspondingly small weights, limiting their influence on the final score. This behavior contrasts with Random Forest, where a feature may repeatedly appear in tree splits simply because it marginally improves local purity, even if it is nondiscriminative in a broader sense.

A qualitative review of both dynamic and static feature distributions supports this expectation. Common low-level system calls such as `read`, `write`, `openat`, and `fstat` appear frequently in both benign and malicious traces, with largely overlapping distributions. Their small effect sizes led to correspondingly low weights, meaning they contributed little to the final score. In contrast, features encoding higher-level behavioral semantics—such as 3-gram triplets involving socket initialization, memory-protection transitions, or distinctive control-flow opcodes—exhibited clear separation between classes. These features consistently received higher weights and became primary drivers of hybrid classification decisions.

The hybrid model further reduces susceptibility to spurious correlations because static and dynamic modalities capture fundamentally different aspects of program behavior. A spurious pattern in one modality is unlikely to manifest in the other, and this cross-domain complementarity encourages reliance on features that reflect meaningful malicious intent rather than dataset-specific artifacts. In practice, true malicious behaviors emerge as aligned evidence across both structural and behavioral views, whereas spurious correlations tend to appear in only one.

These observations suggest that the feature-weighted model is inherently robust to dataset-specific arti-

facts and is less likely to overfit incidental patterns than traditional ML-based baselines. While a dedicated controlled experiment could further quantify this behavior, the mathematical structure of the model itself provides a strong, interpretability-driven safeguard against spurious correlations.

Chapter 6

CONCLUSION AND FUTURE WORK

This thesis presented the Pi-Weighted Hybrid Static-Dynamic Analysis Framework (Pi-HSDF), an interpretable and lightweight malware detection system designed for resource-constrained IoT environments. The framework integrates static structural evidence and dynamic behavioral signals using a transparent feature-weighted scoring mechanism, avoiding the training overhead and opacity of machine learning models. By decoupling expensive offline computation from lightweight on-device inference, Pi-HSDF provides a practical path toward real-time IoT malware detection on devices with strict computational limits.

Experimental evaluation demonstrated that the proposed approach achieves strong and consistent performance. The hybrid configuration reached 98.83% accuracy, with perfect malware recall and sub-three-percent false positives, while maintaining sub-second inference time and less than 16 MB of memory on a Raspberry Pi 5. These results show that statistical separability between benign and malicious binaries can be effectively captured with a compact set of weighted features drawn from PSI graph metrics, opcode indicators, and 3-gram syscall sequences. Furthermore, the small Top-40 and Top-60 feature subsets preserved near-maximum accuracy, confirming that the weighting procedure implicitly performs robust feature selection while remaining fully interpretable.

A key advantage of Pi-HSDF is its auditability. Each classification score is a linear and explainable combination of feature contributions that correspond to meaningful structural or behavioral characteristics. This property makes the framework useful not only for IoT malware detection but also for firmware auditing, automated update vetting, and forensic analysis where reproducibility and transparency are essential. The framework's modular architecture also allows new feature types to be incorporated without retraining a model or modifying the device-side execution logic.

Future extensions of this work include several promising directions. First, adaptive or incremental weighting strategies could enhance resilience against evolving malware families by allowing feature weights to update as new samples appear. Additionally, a preliminary family-level inspection suggests that malware families sharing code exhibit similar PSI and syscall patterns, consistent with known code-reuse trends. A deeper per-family evaluation is noted as future work. Second, incorporating additional runtime modalities—such as memory-access patterns, file-descriptor activity, or lightweight packet-level signals—could improve early-stage detection and strengthen classification for stealthy malware. Third, evaluating Pi-HSDF

in a distributed IoT testbed would validate its performance under realistic network conditions and enable integration with gateway-level intrusion detection systems. Finally, the interpretable scoring model offers a foundation for future visualization tools that communicate binary behavior and detection reasoning to developers and operators.

Overall, this thesis demonstrates that interpretability and lightweight operation need not come at the expense of accuracy. Through a unified static–dynamic perspective grounded in transparent feature weighting, Pi-HSDF offers an effective and practical approach to IoT botnet malware detection suitable for deployment across diverse and resource-constrained environments.

BIBLIOGRAPHY

[1] S. Kumar, P. Ahlawat, and J. Sahni, "IoT malware detection using static and dynamic analysis techniques: A systematic literature review," *Security and Privacy*, vol. 7, July 2024.

[2] M. Asam, S. H. Khan, A. Akbar, S. Bibi, T. Jamal, A. Khan, U. Ghafoor, and M. R. Bhutta, "IoT malware detection architecture using a novel channel boosted and squeezed CNN," *Scientific Reports*, vol. 12, p. 15498, Sept. 2022.

[3] J. Jeon, B. Jeong, S. Baek, and Y.-S. Jeong, "Static Multi Feature-Based Malware Detection Using Multi SPP-net in Smart IoT Environments," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2487–2500, 2024. Conference Name: IEEE Transactions on Information Forensics and Security.

[4] X. Sáez-de Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Gotham Testbed: A Reproducible IoT Testbed for Security Experiments and Dataset Generation," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, pp. 186–203, Jan. 2024.

[5] M. Ali, M. Shahroz, M. F. Mushtaq, S. Alfarhood, M. Safran, and I. Ashraf, "Hybrid Machine Learning Model for Efficient Botnet Attack Detection in IoT Environment," *IEEE Access*, vol. 12, pp. 40682–40699, 2024. Conference Name: IEEE Access.

[6] M. A. Hossain and M. S. Islam, "A novel hybrid feature selection and ensemble-based machine learning approach for botnet detection," *Scientific Reports*, vol. 13, p. 21207, Dec. 2023. Publisher: Nature Publishing Group.

[7] Q.-D. Ngo, H.-T. Nguyen, H.-A. Tran, and D.-H. Nguyen, "IoT Botnet detection based on the integration of static and dynamic vector features," in *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*, pp. 540–545, Jan. 2021.

[8] U. Garg, S. Kumar, and M. Kumar, "A Hybrid Approach for the Detection and Classification of MQTT-based IoT-Malware," in *2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, pp. 1154–1159, Mar. 2023.

[9] W. Cassel and N. E. Majd, "A Lightweight Obfuscated Malware Multi-class Classifier for IoT Using Machine Learning," in *2024 International Conference on Computing, Networking and Communications (ICNC)*, pp. 239–243, Feb. 2024. ISSN: 2473-7585.

[10] H. Nguyen, D. Ngo, and V.-H. Le, "A novel graph-based approach for IoT botnet detection," *International Journal of Information Security*, vol. 19, Oct. 2020.

[11] J. Tang, S. Zhou, T. Peng, X. Yan, X. Hu, and W. Tian, "DTDroid: Adversarial Packed Android Malware Detection Based on Traffic and Dynamic Behavioral," *IEEE Internet of Things Journal*, vol. 12, pp. 2646–2658, Feb. 2025. Conference Name: IEEE Internet of Things Journal.

[12] R. A. Yunmar, S. S. Kusumawardani, Widyawan, and F. Mohsen, "Hybrid Android Malware Detection: A Review of Heuristic-Based Approach," *IEEE Access*, vol. 12, pp. 41255–41286, 2024. Conference Name: IEEE Access.

[13] X. Deng, H. Tang, X. Pei, D. Li, and K. Xue, "MDHE: A Malware Detection System Based on Trust Hybrid User-Edge Evaluation in IoT Network," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 5950–5963, 2023. Conference Name: IEEE Transactions on Information Forensics and Security.

[14] K. Filus, P. Boryszko, J. Domańska, M. Siavvas, and E. Gelenbe, "Efficient Feature Selection for Static Analysis Vulnerability Prediction," *Sensors (Basel, Switzerland)*, vol. 21, p. 1133, Feb. 2021.

[15] T. N. Nguyen, Q.-D. Ngo, H.-T. Nguyen, and G. L. Nguyen, "An Advanced Computing Approach for IoT-Botnet Detection in Industrial Internet of Things," *IEEE Transactions on Industrial Informatics*, vol. 18, pp. 8298–8306, Nov. 2022.

[16] P. Shijo and A. Salim, "Integrated Static and Dynamic Analysis for Malware Detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[17] H. Kurniawan, Samsudiat, C. Nugroho, A. Saputra, and A. Nursyahbani, "Enhancing the Detection of Botnet Attacks in the Internet of Things Networks Through the Utilization of Hybrid Feature Selection," in *2024 FORTEI-International Conference on Electrical Engineering (FORTEI-ICEE)*, pp. 89–94, Oct. 2024.

[18] L. Hong, K. Wehbi, and T. H. Alsalah, "Hybrid Feature Selection for Efficient Detection of DDoS Attacks in IoT," in *Proceedings of the 2022 6th International Conference on Deep Learning Technologies*, (Xi'an China), pp. 120–127, ACM, July 2022.

[19] R. M. Ogunnaike and B. Lagesse, "Toward consumer-friendly security in smart environments," in *2017 IEEE Conference on Communications and Network Security*, 2017.

[20] M. Condry and C. Nelson, "Using smart edge iot devices for safer, rapid response with industry iot control systems," *Proceedings of the IEEE*, 2016.

[21] D. Sharma and M. Devare, "Advancing Malware Defense: A Hybrid Approach with Explainable Deep Neural Networks," in *2024 2nd DMIHER International Conference on Artificial Intelligence in Healthcare, Education and Industry (IDICAIEI)*, pp. 1–6, Nov. 2024.

[22] B. S. Purkayastha, M. M. Rahman, and M. Shahpasand, "Android Malware Detection Using Machine Learning and Neural Network: A Hybrid Approach with Federated Learning," in *2024 7th International Conference on Advanced Communication Technologies and Networking (CommNet)*, pp. 1–5, Dec. 2024. ISSN: 2771-7402.

[23] N. G. Ambekar, S. Thokchom, and S. Moulik, "TC-AMD: Android Malware Detection through Transfomer-CNN Hybrid Architecture," in *2024 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pp. 1–6, Dec. 2024. ISSN: 2153-1684.

[24] M. A. Khan, M. N. Aman, and B. Sikdar, "Quantum Guard: Pioneering Quantum-Based Malware Defense for IoT Devices," in *2024 IEEE Conference on Dependable and Secure Computing (DSC)*, pp. 94–99, Nov. 2024.

[25] C. Taylor, B. Issac, N. Aslam, K. Rogage, and G. Kelly, "Malware and botnet prevention in smart building IoT devices using blockchain-enabled federated learning," in *International Conference on AI and the Digital Economy (CADE 2024)*, vol. 2024, pp. 146–149, June 2024.

[26] S. S. Mutupuri, V. Preetam, D. Aditya, A. Harishitha, S. Sivarajan, and S. Anamalamudi, "Machine Learning based Malware Detection for IoT Networks," in *2023 IEEE 15th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 298–303, Dec. 2023. ISSN: 2472-7555.

[27] O. Ebrahem, S. Dowaji, and S. Alhammoud, "Towards a minimum universal features set for IoT DDoS attack detection," *Journal of Big Data*, vol. 12, p. 88, Apr. 2025.

[28] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 1988.

Chapter 7

APPENDIX A: ENVIRONMENT CONFIGURATION

7.1 Source Directory Structure

- `src/`: static and hybrid analysis scripts sequentially labeled 0–10
- `dynamic_sandbox/`: dynamic analysis scripts sequentially labeled 0-5 with Docker sandbox file
- `plots/`: visualization outputs grouped by analysis type
- `features/`: stored feature weighted dictionaries and RF models

7.2 Software Versions

All experiments were performed using the following core software components and libraries, which reflect the minimum environment necessary to reproduce static, dynamic, and hybrid analysis:

- **Python**: 3.13
- **angr binary analysis suite** (static PSI graph extraction):
  - angr 9.2.156
  - ailment 9.2.156
  - cle 9.2.156
  - archinfo 9.2.156
  - pyvex 9.2.156
  - claripy 9.2.156
- **Dynamic tracing and sandbox utilities:**
  - QEMU user-mode emulation (9.0)
  - strace (system default)
- **Data processing and machine learning:**

  - NumPy 2.2.6
  - pandas 2.2.3
  - scikit-learn 1.6.1
  - SciPy 1.15.3

- **Graph processing:**
  - networkx 3.4.2
  - pydot 4.0.0

- **Visualization:**
  - matplotlib 3.10.3
  - seaborn 0.13.2

- **System utilities:**
  - psutil 7.0.0 (resource measurement)
  - python-magic 0.4.27 (file-type detection)

- **Cryptographic / binary utilities (used in parsing ELF and secure transfers):**
  - pyelftools 0.32
  - paramiko 3.5.1
  - cryptography 45.0.3

Only these libraries are essential for reproducing the full static, dynamic, and hybrid analysis pipeline used in this thesis. All other packages present in the environment were ancillary and not required for core functionality.

Chapter 8

CLASS FILE LOCATION

The uwthesis class file, `uwthesis.cls`, contains the parameter settings, macro definitions, and other TEXnical commands which allow LATEX to format a thesis. The source to the document you are reading, `uwthesis.tex`, contains many formatting examples which you may find useful. The bibliography database, `uwthesis.bib`, contains instructions to BibTeX to create and format the bibliography. You can find the latest of these files on:

`https://staff.washington.edu/fox/tex/thesis.shtml`

ACKNOWLEDGMENTS

The author would like to thank the University of Washington faculty committee, as well as friends and family, for their continuous support and guidance throughout this work.