# LIGHTWEIGHT STATIC FILE ANALYSIS FOR PRE-INSTALLATION ANOMALY SCREENING IN AUTOMOTIVE OTA UPDATES

by

Darwin Liao

Supervisor: Marwa Elsayed

Graduate Program in Computer Science

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

The School of Graduate and Postdoctoral Studies

The University of Western Ontario

London, Ontario, Canada

July 2026

# Abstract

Automotive Over-the-Air (OTA) updates are increasingly important for maintaining modern vehicle software, especially in In-Vehicle Infotainment (IVI) systems that rely on complex Linux-based and Android-based platforms. Existing OTA protections mainly focus on cryptographic trust, integrity verification, authorization, and access control. While these mechanisms are essential, they do not directly determine whether the files inside a valid and authorized update package appear benign for the target vehicle software environment. This leaves a content-level security gap where suspicious or malicious files may still be introduced through compromised build, supplier, repository, or packaging processes before an update is signed or distributed. At the same time, large labelled datasets of malicious automotive OTA packages are limited, making fully supervised detection difficult in this setting. This thesis addresses these challenges by proposing OTA-Sentinel, a lightweight pre-installation anomaly-screening framework that learns normal OTA file structure from benign update data using practical static byte-level representations. Through comparisons of classical, deep, and few-shot anomaly detection methods under data-scarce OTA security conditions, the thesis demonstrates the feasibility and limitations of lightweight static anomaly detection for OTA package inspection and provides an empirical foundation for adding content-level screening to existing automotive OTA security pipelines.

**Keywords** Automotive cybersecurity, Over-the-Air updates, IVI, Android Automotive OS, QNX, Uptane, anomaly detection, unsupervised learning, static malware analysis, autoencoder

# Summary for Lay Audience

Modern vehicles depend on software for navigation, entertainment, connectivity, and other functions. To maintain this software, manufacturers often use Over-the-Air (OTA) updates, which allow vehicles to receive new software remotely instead of requiring a service visit. These updates are useful for fixing bugs and security issues, but they also create a risk: a vehicle could receive an update that appears valid while still containing suspicious or malicious files.

This M.Sc. research studies whether OTA update files for In-Vehicle Infotainment (IVI) systems can be checked before installation using lightweight machine learning methods. The goal is not to replace existing protections such as digital signatures or update verification. Instead, the goal is to add another screening layer that asks whether the files inside an update look normal for the target vehicle software environment.

To study this problem, normal update files were collected from Hyundai-group update material and Android Automotive OS builds. Malicious test files were created using Linux and Android malware samples, as well as modified update files where malicious bytes were inserted into otherwise normal files. Each file was converted into simple measurements such as file size, byte patterns, and entropy, which measures how random or compressed a file appears. Several anomaly-detection models were then tested to see whether they could separate normal update files from suspicious ones.

The results show that lightweight static file analysis can help identify abnormal OTA files before installation, although performance depends on the type of update data and the type of malicious change. Some models performed well when trained on mixed update data, while the Reptile-based approach was more consistent on Android Automotive OS domains. Overall, the findings suggest that pre-installation screening could strengthen automotive OTA security by adding a practical check on update file contents before they are installed on the vehicle.

# Co-Authorship Statement

This thesis includes material drawn from a co-authored publication, while the rest of the work is intended to form the basis of a second publication currently under preparation. The published manuscript is:

> Darwin Liao and Marwa. A. Elsayed, "OTArmor: Securing Automotive Over-the-Air Updates Against Malware Using Generative Modeling," in *2025 9th Cyber Security in Networking Conference (CSNet)*, Abu Dhabi, United Arab Emirates, 2025, pp. 207–214, doi: 10.1109/CSNet67572.2025.11288195.

This publication was authored by the candidate, Darwin Liao, and the candidate's supervisor, Dr. Marwa. A. Elsayed. Additional material from this thesis is expected to be revised and adapted into a future co-authored publication by the same authors.

Darwin Liao was primarily responsible for carrying out the research, including the research design, dataset collection and preparation, feature extraction, model implementation, experimental evaluation, analysis of results, preparation of figures and tables, and drafting of the thesis. Dr. Marwa. A. Elsayed provided close academic supervision throughout, played a central role in shaping the research direction and methodological approach, gave detailed guidance on the experimental design and interpretation of the results, and offered continuous critical insight and editorial feedback throughout the development of the thesis.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Marwa A. Elsayed, for their guidance, support, and mentorship throughout this research, as well as for the countless times they took time out of their weekends and holidays to provide assistance. Their feedback and support were instrumental in shaping this work.

I would also like to thank my family for their constant love, patience, and encouragement throughout my studies. Their support helped me stay grounded during challenging periods and motivated me to keep moving forward. I am especially grateful to my late friend Indy, who generously gave me a place to stay whenever my studies required me to spend the night in London.

# Table of Contents

# List of Tables

# List of Figures

# List of Appendices

# Chapter 1: Introduction

## 1.1 Chapter Overview

This chapter introduces the central problem addressed in the study: pre-installation screening of automotive Over-the-Air (OTA) updates for In-Vehicle Infotainment (IVI) systems. As vehicles become increasingly software-defined, networked, and dependent on remote update mechanisms, OTA infrastructure has become essential for deploying security patches, firmware revisions, and functional improvements. These mechanisms allow manufacturers to maintain vehicle software after deployment, but they also create a point at which new software content enters the vehicle environment.

Existing OTA protections are necessary for verifying delivery trust, including authenticity, integrity, authorization, freshness, and rollback protection. However, those checks do not necessarily assess whether the files inside an accepted update appear benign for the target IVI platform. This chapter motivates the need for an additional pre-installation screening layer, defines the research objective, presents the research questions and contributions, summarizes the methodological approach, and concludes with the structure of the remaining chapters.

## 1.2 Research Motivation

The automotive industry is undergoing a transition toward increasingly connected and software-reliant vehicles, where the maintenance and updating of vehicle software has become progressively more important. Central to this shift is the adoption of OTA updates, which allow Original Equipment Manufacturers (OEMs) to remotely deploy security patches, firmware enhancements, and new features [1, 2]. However, the increasing connectivity of modern vehicles has also expanded their digital attack surface. Although

safety-critical Electronic Control Units (ECUs) are often the primary focus of automotive cybersecurity research, IVI systems have emerged as a significant point of vulnerability. In contrast to specialized real-time systems, modern IVI units frequently operate on Linux-based platforms or Android Automotive OS (AAOS), while supporting complex software stacks, third-party applications, and external connectivity pathways [3–5]. A compromise of the IVI system may therefore provide a point of entry for cross-domain attacks, potentially enabling an adversary to move into the broader vehicle network, including the Controller Area Network (CAN) bus[3, 5].

These risks are especially relevant to OTA updates because accepting an update through a trusted delivery process is not the same as determining that its contents are benign. Secure OTA frameworks such as Uptane are designed to protect update delivery through signed metadata, hash verification, version checks, and authorization controls, which are essential for preventing tampering, rollback, and unauthorized delivery [6, 7]. However, these mechanisms mainly establish whether the vehicle received the update that it was authorized to receive. They do not directly determine whether the files inside that accepted update are normal for the target IVI platform. If malicious content is introduced before the package is signed or distributed, such as through a compromised build, supplier, repository, or packaging process, later verification may still confirm that the delivered package matches the trusted metadata. The remaining problem is therefore content-focused: before installation, the vehicle still needs a way to assess whether the extracted update files appear normal for the intended software environment.

Many traditional malware-detection approaches are difficult to apply directly in this problem setting. Signature-based techniques and many conventional supervised detection methods are limited in their ability to detect zero-day threats, because they rely on patterns derived from previously observed malicious behaviour, known malware families, or labelled attack examples [8]. Supervised approaches also depend on large volumes of la-

belled training data. In the automotive OTA setting, such data are difficult to obtain, since realistic update files often contain safety-critical software and proprietary manufacturer content that restrict access and limit dataset availability, particularly at the file level. In addition, deployment-heavy behavioural or runtime analysis methods are difficult to apply in embedded automotive environments, where computational efficiency and implementation simplicity remain important practical constraints [9]. Taken together, the limited robustness of conventional methods to zero-day threats, their dependence on labelled data, the scarcity of realistic automotive OTA datasets, and the deployment constraints of embedded vehicle environments make existing malware-detection approaches poorly suited to pre-installation OTA file screening.

In response to these limitations, there is a need for a lightweight static screening method that analyzes update-file contents to identify suspicious files within OTA packages before installation. This is especially relevant in IVI update pipelines, where the scale and diversity of software contents increase both the opportunity for misuse and the difficulty of manual inspection. For this reason, the thesis proposes OTA-Sentinal, a benign-data-driven anomaly detection solution. Rather than attempting to learn every possible malicious OTA pattern, the detector learns the normal byte-level structure of benign update files and flags files that deviate from that profile. This formulation better matches the OTA setting, where benign update material is more obtainable than realistic labelled malicious OTA packages.

## 1.3 Research Objective

OTA-Sentinel aims to determine whether lightweight unsupervised anomaly detection can provide a practical pre-installation screening layer for OTA update files in IVI software update pipelines. It examines whether models trained primarily on benign OTA data can learn the normal structure of update files and identify malicious or abnormal deviations,

thereby complementing existing authenticity and integrity protections.

To address this objective, the research pursues the following specific objectives:

- **Problem Formulation:** To define pre-installation OTA screening as an anomaly detection problem, in which files extracted from OTA packages are assessed before installation.
- **Lightweight Static Representation:** To develop and evaluate lightweight static byte-level feature representations for OTA files, including measures such as file size, entropy, byte histograms, and n-gram-based features.
- **Unsupervised Detection Capability:** To determine whether unsupervised learning methods can model the normal characteristics of benign OTA files and detect suspicious deviations without relying on large labelled malware datasets.
- **Model Comparison:** To compare multiple anomaly-detection approaches, including one-class, generative, and meta-learning-based methods, in order to assess their relative suitability for OTA screening.
- **Cross-Domain Evaluation:** To examine model performance across Linux-based and AAOS update domains, and to assess how differences between OTA families affect anomaly-detection performance.
- **Adaptation to OTA Domains:** To investigate whether meta-learning provides measurable benefits for adapting to the normal structure of individual OTA domains when a limited benign support set is available.
- **Computational Considerations:** To evaluate the practical efficiency of the proposed feature representations and anomaly-detection models using measurable factors such as feature-extraction cost, runtime, memory footprint, and inference latency, with respect to their suitability for pre-installation OTA screening.

- **Feature-Use Analysis:** To examine which static feature groups contribute most to the meta-learning-based detector's anomaly scores, and to assess whether the detector relies on byte-level structure or simpler global features such as file size and entropy.

### 1.3.1 Research Questions

Empirically, the study evaluates this problem using two benign OTA sources: Hyundai-group Navigation Updater material as a Linux-based IVI update source, and AAOS-derived update contents generated from AOSP automotive builds.

Toward evaluating OTA-Sentinel, the study is guided by the following research questions:

- **RQ1:** To what extent can lightweight static byte-level features distinguish benign OTA files from malicious or anomalous files in IVI update contexts?
- **RQ2:** Which unsupervised anomaly-detection methods are most effective for pre-installation OTA file screening when trained on pooled benign OTA data and evaluated under varying contamination levels?
- **RQ3:** How well do anomaly-detection models generalize across Linux-based Hyundai-group OTA data and AAOS-derived OTA data?
- **RQ4:** To what extent does meta-learning support adaptation to individual OTA domains when only limited benign support data are available?
- **RQ5:** What are the computational costs of the proposed static feature representations and anomaly-detection models for pre-installation OTA screening?
- **RQ6:** Which static feature groups contribute most to the meta-learning-based detector's anomaly scores, and do the resulting feature-attribution scores indicate reliance on byte-level structure or simpler global features such as file size and entropy?

## 1.4 Main Contributions

The main contributions of OTA-Sentinel are as follows:

- **Problem formulation for pre-installation OTA screening:** The work formulates pre-installation screening of OTA update contents as a file-level anomaly-detection problem for IVI update pipelines. This establishes a security perspective that complements existing OTA protections, which verify authenticity and integrity but do not determine whether update contents are benign.
- **Lightweight static representation for OTA files:** The work develops a lightweight byte-level feature representation for OTA files based on static characteristics such as file size, entropy, byte histograms, and 3-gram-derived information. This representation supports analysis before installation without requiring dynamic execution or heavyweight runtime monitoring.
- **Comparative evaluation of unsupervised anomaly-detection methods:** The work implements and evaluates multiple unsupervised anomaly-detection approaches for OTA file screening, including one-class, generative, and meta-learning-based methods. This provides a comparative assessment of their ability to model normal OTA file structure and detect malicious or abnormal deviations.
- **Cross-domain empirical evaluation across Linux-based and AAOS OTA data:** The work presents an empirical evaluation across Linux-based and AAOS OTA domains, including experiments on limited-support adaptation and varied malware conditions. This shows how anomaly-detection performance changes across OTA families and assesses the potential of meta-learning for domain adaptation in practical OTA settings.
- **Practical and interpretability assessment:** The work evaluates the computational cost of the proposed feature representations and models, including feature-extraction

time, inference latency, and memory use. It also analyzes feature-block contributions for the Reptile-based detector to assess whether the model relies on byte-level structure or simpler global features such as file size and entropy.

## 1.5 Research Methodology

OTA-Sentinel adopts a file-level anomaly-detection approach for securing automotive OTA updates prior to installation. Rather than relying on post-deployment monitoring or signature based scanning, the proposed methodology focuses on screening individual update files before they are installed in the target IVI environment.

Benign OTA files are collected from two automotive-relevant software sources, covering Linux-based infotainment update material and AAOS-based update contents. Malicious samples are then incorporated to approximate realistic threat conditions and to evaluate whether anomalous or malicious update content can be distinguished from benign OTA files.

Each update file is represented using lightweight static features derived from its byte-level contents. These features are designed to capture structural properties across different file types while remaining computationally simple enough for pre-installation analysis. This representation allows the methodology to operate across both system-level binaries and application-level packages without requiring dynamic execution.

Using this representation, this thesis evaluates multiple unsupervised anomaly-detection methods, including generative, boundary-based, and meta-learning-based approaches. These models are trained primarily on benign OTA data in order to learn the characteristics of normal update files and to identify deviations that may indicate malicious content.

Evaluation is conducted across multiple OTA domains and malware conditions in order to assess detection performance and model behaviour under different update environments.

The experimental design includes comparative baseline models, cross-domain analysis between Linux-based and AAOS-based settings, and practical performance considerations related to the efficiency of feature extraction and model use in pre-installation contexts. Particular attention is given to the extent to which different methods can adapt to domain-specific OTA structure. Through this methodology, the study evaluates whether lightweight static analysis combined with unsupervised learning can serve as an effective pre-installation screening layer for automotive OTA security.

## 1.6 Thesis Structure

Chapter 1 introduces the research problem, motivates OTA update security for IVI systems, and presents the research objective, methodological overview, and thesis organization. Chapter 2 reviews automotive cybersecurity, OTA update security, malware detection, anomaly detection, and relevant machine learning approaches. Chapter 3 describes the background, threat model, and problem definition. Chapter 4 presents the proposed OTA-Sentinel framework, analytical methodology, dataset construction, feature representation, task design, and model configurations. Chapter 5 presents the experimental evaluation across OTA domains, malware sources, contamination levels, malicious modification scenarios, Reptile-based adaptation, hash-binned feature variants, interpretability analysis, and computational-efficiency measurements. Finally, Chapter 6 summarizes the findings, discusses limitations, and outlines future work.

# Chapter 2: Literature Review

This chapter reviews the literature most directly related to OTA-Sentinel. It surveys automotive cybersecurity in connected and software-defined vehicles, examines OTA update security and the regulatory frameworks that shape secure update deployment, and reviews the practical constraints of in-vehicle anomaly detection, including real-time operation, resource limits, evaluation realism, and cross-platform generalization. It then considers software analysis approaches used in automotive cybersecurity, with particular attention to why static, pre-installation analysis is a practical choice for screening OTA files. The chapter also reviews meta-learning as an approach for improving adaptation across domains when data distributions vary and labelled malicious samples are limited. Together, these areas of prior work define the research gap that OTA-Sentinel addresses.

## 2.1 Automotive Cybersecurity in Connected Vehicles

CAV cybersecurity has developed into a substantial research area as vehicles have become increasingly connected, software-reliant, and integrated with external networks and services. Recent survey work by Kifor et al. reviews automotive cybersecurity from the perspective of frameworks, standards, testing, and monitoring technologies, showing that the field now extends well beyond isolated attack demonstrations and includes broader questions of validation, governance, and system-level protection [10]. This broader view of CAV security is also connected to how modern vehicles are built and maintained. Moukahal et al. [11] argue that modern CAVs require software engineering methods that account for safety-critical operation, growing system complexity, and continued maintenance after deployment. Since many vehicle functions can now be updated or extended through over-the-air updates, the integrity of the software update pipeline has become an important part of long-term vehicle safety and security.

Within this broader area, much of the existing research has focused on communication security, external attack surfaces, and runtime monitoring of in-vehicle systems. Early experimental work by Koscher et al. showed that a modern automobile contains multiple attackable electronic subsystems, while Checkoway et al. demonstrated the breadth of remotely reachable automotive attack surfaces [12, 13]. Industry reports have further reinforced that these risks are not only theoretical, but also relevant to real connected-vehicle deployments and unauthorized access pathways observed in practice [14].

This attack-surface concern is especially relevant to IVI systems, which are the focus of OTA-Sentinel. Wang et al. provide empirical evidence of this risk through a large-scale study of 649 exploitable Intelligent Connected Vehicle (ICV) vulnerabilities, reporting that vulnerabilities were concentrated primarily in the cloud platform (37.8%) and IVI module (32.0%) [15]. They further report that IVI vulnerabilities accounted for 70.0% of in-vehicle vulnerabilities, which they attribute to IVI exposure to external entry points such as Bluetooth, Wi-Fi, and voice-assistant services, together with open debugging access and legacy software components [15]. These findings support the focus on IVI software as a security-relevant vehicle domain rather than treating the infotainment system as a minor or isolated component.

OTA updates are important in this setting because they provide the mechanism through which IVI and other vehicle software components are maintained after deployment. Vasenev et al. examine OTA security and privacy threats in the context of long-term vehicle support, focusing on update legitimacy, backend communication, and lifecycle risks in automotive software maintenance [16]. Their work is useful in showing that OTA security is already treated as a distinct concern within automotive cybersecurity. However, much of this discussion focuses on threat modelling, risk analysis, and secure update processes rather than on determining whether the update content itself is malicious before installation.

This broader shift in automotive cybersecurity has made OTA updates more than a main-

tenance convenience; they are now part of the vehicle's security-critical software lifecycle. As vehicles increasingly rely on remote software delivery to add features, patch vulnerabilities, and maintain long-term support, securing the update process has become a central concern. This has led to growing attention not only to technical protections for OTA delivery, but also to the standards and regulatory frameworks that govern how automotive software updates are engineered, validated, and managed.

## 2.2 Over-the-Air Update Security and Regulatory Standards

OTA updates have become a central component of modern automotive software maintenance, allowing OEMs to deploy bug fixes, security patches, and feature improvements remotely without requiring repeated service visits. In connected vehicles, this capability is particularly important because software must be updated regularly over a long operational lifetime, both to address newly discovered vulnerabilities and to maintain system functionality. At the same time, the OTA path introduces a highly security-sensitive channel into the vehicle, since it provides a remote mechanism through which software is delivered to in-vehicle systems [17–19].

For this reason, a substantial body of prior work has focused on securing the OTA delivery and installation process itself. Early work such as Chawan et al. examined threats to connected vehicle OTA mechanisms and proposed additional protections for update verification and authorization [17]. More recent work has expanded this focus toward stronger system-level guarantees. Plappert and Fuchs, for example, proposed a secure and lightweight OTA distribution mechanism built around a TPM-based trust anchor, with particular attention to update authenticity, integrity, key protection, and installation authorization under vehicle-state constraints [19, 20]. Seeking to harden the delivery pipeline further, Anwar et al. leveraged LoRaWAN and blockchain technologies to support update

integrity and verification before the software reaches the vehicle [21]. In parallel, practical threat analyses such as that of Vasenev et al. treat OTA security as a broader lifecycle issue involving backend communication, update legitimacy, and long-term support of vehicle software [16]. While these approaches provide important protections for secure delivery, authorization, and update trust, they mainly focus on protecting the OTA process rather than evaluating the contents of the update package itself. They therefore do not directly address whether the files inside an OTA package appear anomalous or potentially malicious before installation, which is the gap addressed by OTA-Sentinel.

Alongside these technical mechanisms, OTA security has increasingly been shaped by formal standards and regulatory requirements. Recent work has highlighted the growing importance of ISO/SAE 21434, ISO 24089, and UNECE regulations R155 and R156 in shaping automotive cybersecurity engineering and software-update governance [22–25]. These frameworks reinforce the view that OTA security is no longer merely a desirable engineering feature, but an essential requirement for modern connected vehicles. These standards and regulations are relevant to this work because they frame software updates as part of a managed cybersecurity and software-update process. However, they do not prescribe a specific file-level method for detecting abnormal contents inside an otherwise accepted OTA package. OTA-Sentinel is therefore positioned as a technical screening layer that could support such update-management processes.

Because part of this work evaluates Android Automotive OS update contents, it is also important to distinguish AAOS from more vendor-specific Linux-based OTA environments. Android Automotive OS (AAOS) introduces a more standardized OTA environment than many vendor-specific Linux-based update paths. AAOS runs directly on in-vehicle hardware as a full Android-based platform, and its update process follows Android's established OTA mechanisms, including A/B and Virtual A/B system updates. In addition, Android supports modular system packaging through APEX, which packages lower-level operat-

ing system components into structured update units. Together, these design choices make AAOS OTA contents more modular and regular at the system level, which is relevant when considering how OTA file contents may differ from those in broader vendor-specific OTA ecosystems [26–30].

Prior work also shows that protecting OTA updates is not limited to cryptographic design alone. Mahmood et al. note that automotive OTA security testing has received less attention than the design of secure update mechanisms themselves, and their systematic evaluation of the Uptane reference implementation shows that even well-established update frameworks may still exhibit weaknesses at the implementation level [18]. This suggests that secure OTA operation depends not only on authenticity and integrity guarantees, but also on careful threat analysis, implementation assessment, and testing under realistic attack conditions. Some prior work has also moved beyond securing the delivery channel to examine the update contents themselves before installation. Qureshi et al. proposed eUF, a framework that augments Uptane with a CNN-based stage for distinguishing benign and malicious executables in OTA updates [31]. This is closer to the problem considered in OTA-Sentinel than runtime in-vehicle intrusion detection, but it remains focused on supervised executable classification rather than static, file-level anomaly detection under limited-label and zero-day conditions. This makes eUF an important point of comparison: it supports the value of pre-installation OTA content screening, while leaving open the benign-only, file-level anomaly-detection setting addressed by OTA-Sentinel.

## 2.3 In-Vehicle Anomaly Detection: Challenges and Constraints

Although OTA-Sentinel does not analyze CAN traffic, in-vehicle anomaly detection is the closest established body of automotive security work using anomaly-based methods. Reviewing this literature helps clarify why the present work focuses on a different point in the

vehicle lifecycle: before update installation rather than during vehicle operation. This field of study is an important research direction in automotive cybersecurity because modern vehicles rely on large numbers of ECUs that exchange messages over internal networks to support both operational and safety-critical functions. Early work on CAN security had already shown that in-vehicle networks expose practical attack opportunities and require dedicated defensive measures [32]. As a result, intrusion detection and anomaly detection methods have been widely studied for identifying abnormal traffic patterns, message injection, replay, spoofing, and other deviations from expected in-vehicle behaviour [33–35]. Existing studies show that these methods can provide an additional defensive layer during vehicle operation, but they also highlight several practical deployment challenges.

One major challenge is the safety-critical and real-time nature of automotive systems. Detection mechanisms must operate with low latency and limited overhead because they share resources with functions that affect vehicle operation [33]. They must also maintain low false-positive rates, since frequent false alarms can trigger unnecessary intervention, reduce trust in the monitoring system, and complicate fail-safe behaviour. Prior work has therefore emphasized that automotive intrusion detection must be designed for both accuracy and fail-operational use under timing and resource constraints [36]. Even lightweight methods based on message timing or frequency can be difficult to deploy robustly, since normal communication patterns may change across driving modes and vehicle states [37].

Prior work on in-vehicle anomaly detection has therefore often emphasized lightweight runtime monitoring of vehicular communication data. For example, Baldini applies entropy-based sliding-window analysis to intrusion detection in automotive in-vehicle networks, reflecting the strong focus that this literature places on efficient CAN-oriented runtime monitoring under resource and timing constraints [38]. More recently, Elsayed and Zincir-Heywood proposed BoostSec, an online attack-detection framework based on incremental ensemble learning for vehicular networks [39]. Although BoostSec addresses important

concerns such as runtime efficiency, adaptation to unseen attacks, adversarial robustness, and interpretability, it remains focused on runtime network data rather than pre-installation analysis of OTA update contents.

Another challenge is that vehicle anomaly detection is inherently cyber-physical. Suspicious behaviour cannot always be identified from message statistics alone, because the meaning of a message often depends on surrounding operational context. This has motivated context-aware approaches that attempt to determine whether observed behaviour is physically plausible rather than merely unusual at the network level [40]. While such methods are promising, they also make the detection problem more difficult. A practical detector must distinguish malicious activity from benign variation caused by driver input, road conditions, mode transitions, software updates, or legitimate faults. Recent surveys similarly note that many approaches remain closely tied to CAN-specific regularities even though contemporary vehicle platforms are increasingly diverse in their protocols, architectures, and software stacks [34, 35].

A further difficulty lies in evaluation. Reported performance in the literature is often shaped by the realism, diversity, and scope of the datasets used for training and testing. Several studies have shown that widely used automotive IDS datasets may not capture the full variability of real vehicle operation, which can make benchmark results appear stronger than expected in practice [41, 42]. Kidmose et al. demonstrate that automotive IDS evaluation is highly sensitive to the properties of the underlying dataset, including how attacks are represented and how well the data capture normal vehicle behaviour [43]. Li et al. further argue that the scarcity of high-quality in-vehicle data continues to limit progress, while the use of synthetic data introduces its own concerns regarding realism and transferability [44]. These issues make it difficult to determine whether a detector trained in one setting will generalize across vehicles, platforms, or attack conditions. This limitation becomes more apparent when moving from runtime in-vehicle IDS research to OTA-focused

analysis. The surveyed anomaly-detection literature is largely built around in-vehicle network traffic, especially CAN-based communication, rather than corpora of real OTA update packages [35]. This matters because public, real-world OTA packages are difficult to obtain, and OTA security studies have often concentrated more on protocol design, framework hardening, and security testing than on large-scale package datasets [18, 45]. As a result, there is still limited public evidence on how well content-based security methods transfer to genuine OTA software distributions.

Robustness under adversarial conditions is another important concern. High nominal accuracy on benchmark datasets does not necessarily imply that a detector will remain reliable when an attacker deliberately attempts to evade it. This is particularly important in automotive settings, where detection failure may have consequences beyond data compromise alone. Longari et al. show that automotive IDS models can be vulnerable to adversarial manipulation, reinforcing the point that strong offline metrics are not sufficient by themselves for safety-critical deployment [46]. Taken together, the literature suggests that in-vehicle anomaly detection remains constrained by strict timing requirements, low-resource execution environments, context dependence, limited dataset realism, uncertain cross-vehicle generalization, and adversarial robustness concerns [33–35, 43, 46].

## 2.4 Software Analysis Approaches in Automotive Cybersecurity

Software security analysis is commonly divided into static, dynamic, and hybrid approaches. Static analysis examines software without executing it, dynamic analysis observes behaviour during execution, and hybrid approaches combine both forms of evidence. This distinction is important for OTA-Sentinel because the proposed screening point occurs before installation or activation, where executing the update contents may be impractical or unsafe. Dynamic and hybrid analysis provide a detailed perspective by observing behaviour

during execution, including network activity, control-flow effects, interface interactions, and other runtime data [15, 47]. In automotive systems, these approaches can reveal behaviours that are difficult to infer from binaries alone, especially when software interacts with hardware-specific drivers, middleware, or communication services [15]. However, they also require controlled execution environments, representative hardware or emulation support, and additional infrastructure for safe testing [18]. These requirements can be difficult to satisfy for OTA workflows, where update files may need to be screened before deployment across platforms [18, 45].

The OTA-specific literature has largely focused on securing the update process itself rather than detecting anomalous content within update packages. Mahmood et al. noted that most prior automotive OTA research emphasized improved mechanisms for securing updates, while systematic security testing of OTA workflows had received less attention. Their evaluation of the Uptane framework showed that, although many experimental attacks were resisted, the reference implementation still exhibited vulnerabilities to denial-of-service and eavesdropping attacks [18]. Related work has also explored the use of OTA to maintain intrusion-detection capabilities rather than to analyze update files directly. For example, Li et al. reported detection accuracies above 99% in an IDS framework based on OTA signature updates, but their evaluation used public CAN-bus attack datasets containing DoS, fuzzy, gear, and RPM attacks rather than real OTA software packages or file-level OTA data [48]. Taken together, these studies suggest that OTA security research has concentrated more on secure delivery, testing, and runtime protection than on lightweight pre-installation analysis of the update contents themselves.

For this reason, static file-level analysis remains attractive for OTA-oriented screening. It can be applied before installation and does not require dynamic execution on the target platform. Prior malware-analysis research has shown that useful discriminative information can be extracted directly from binary content without requiring execution, which supports

the use of compact static representations when computational cost and deployment simplicity are important concerns [49]. For OTA-Sentinel, static byte-level feature extraction is therefore treated not as a replacement for dynamic or runtime defenses, but as a practical complementary layer before deployment across different platforms.

## 2.5 Meta-Learning and Cross-Domain Adaptation

Meta-learning studies how a model can be trained across related tasks so that it can adapt quickly to a new task using only a small amount of additional data. In optimization-based meta-learning, this is commonly expressed as learning model parameters that serve as a strong starting point for rapid task-specific adjustment. Model-Agnostic Meta-Learning (MAML) formalized this idea by learning an initialization that can be fine-tuned in a small number of gradient steps, and Reptile later provided a simpler first-order alternative that keeps the same general goal of fast adaptation while avoiding the cost of second-order optimization [50, 51].

Meta-learning offers one possible way to address cross-domain learning problems because it allows a model to capture structure that is shared across related tasks while still remaining easy to adapt to a specific target task. In particular, first-order methods such as Reptile are relevant to settings where only a modest support set is available, since they are designed to support few-step adaptation without the cost of second-order optimization [51].

Recent work has begun to apply meta-learning directly to anomaly detection under limited target-domain supervision. Kumagai et al. propose a meta-learning method for robust anomaly detection on unseen target tasks that have only unlabelled target data, using an autoencoder-based detector that is adapted to unlabelled target-domain data [52]. Moon et al. similarly combine MAML with a variational autoencoder for anomaly detection, showing that optimization-based meta-learning can be used with reconstruction-based detectors rather than only with supervised classifiers [53]. More recently, Holly et al. formulate

one-class domain adaptation as a meta-learning problem and argue that the goal is rapid adaptation across domains using only normal operational data from the target environment [54]. Taken together, these studies show that optimization-based meta-learning is not limited to few-shot classification, but is also relevant to anomaly detection settings in which labelled attack data are scarce and adaptation must rely mainly on benign or unlabelled target-domain data [52–54].

## 2.6 Summary and Research Gap

The literature reviewed in this chapter shows that modern automotive cybersecurity is shaped by increasing connectivity, software complexity, and expanding OTA use. Prior work has established the importance of securing update delivery through authenticity, integrity, authorization, and regulatory compliance, while anomaly-detection research has focused mainly on runtime monitoring of in-vehicle communication [18–20, 34, 35]. However, the reviewed studies indicate that these approaches do not directly evaluate the abnormality of software content within otherwise legitimate OTA updates for a target platform [18, 19, 45]. In addition, much of the existing evaluation literature relies on network, proxy, or otherwise non-OTA-specific datasets, while realistic file-level OTA datasets remain comparatively limited [34, 35, 41, 43, 44].

Table 2.1: Comparison of OTA security protections and remaining content-level gap

| **Protection** | **Main question answered** | **Remaining gap** |
|---|---|---|
| Encryption/TLS | Was the communication protected? | Does not prove content is benign. |
| Signatures/hashes | Is this the exact authorized package? | If malicious content was signed, it still verifies. |
| Uptane | Is the update authorized, fresh, and correct for the ECU? | Does not inspect file-level benignity. |
| Blockchain | Is update provenance or the audit trail tamper-evident? | Does not prove signed file contents are safe. |
| OTA-Sentinel | Do accepted files look abnormal? | Does not replace cryptographic or provenance protections. |

The resulting gap is therefore not simply the absence of automotive anomaly detection, but the absence of lightweight, file-level OTA content screening under limited labelled malicious OTA data. Existing work provides strong foundations for secure update delivery, runtime intrusion detection, and malware analysis, but it does not fully address whether benign-only static models can flag abnormal files inside OTA update contents before installation. This is the gap addressed by OTA-Sentinel. The literature also suggests that static file-level analysis and meta-learning-based adaptation are promising directions when labelled attack data are limited and software distributions vary across platforms [49, 51, 52]. Building on this gap, the next chapter defines the problem setting, datasets, and experimental framework used by OTA-Sentinel to study anomaly detection for automotive OTA updates.

# Chapter 3: Background and Problem Definition

## 3.1 Automotive OTA Update Overview

Automotive OTA updates provide a mechanism for delivering software, firmware, configuration changes, and security patches to vehicles after production. In a typical update flow, the backend prepares an update package, attaches metadata and cryptographic protections, distributes the package to eligible vehicles, and the vehicle retrieves, verifies, stages, and installs or activates the update. This process creates a practical pre-installation point at which the update has been accepted by the delivery mechanism, but its extracted file contents may still be inspected before they alter the IVI software state.

This section summarizes the OTA update process, the Uptane framework, and the AAOS and QNX update contexts relevant to IVI software updates. The purpose is to establish where OTA-Sentinel is positioned in the update lifecycle and to define the background needed for the threat model that follows.

### 3.1.1 OTA Update Process

In a typical automotive OTA workflow, an update is prepared by the backend provider, associated with metadata such as version information, target identifiers, hashes, and signatures, and then made available to eligible vehicles. The vehicle retrieves the update package through the OTA delivery path, verifies the associated metadata and cryptographic checks, stages the update locally, and then installs or activates it when the required vehicle and system conditions are satisfied [1, 18, 19].

For IVI systems, this process is especially important because the IVI domain contains complex user-facing software, network connectivity, application components, and links to other vehicle services. In modern platforms, an IVI update may include operating-system compo-

nents, application packages, shared libraries, binaries, scripts, configuration files, or other files used by the target software environment. Once installed, these files become part of the vehicle's software state. This makes the pre-installation stage important not only for verifying delivery trust, but also for examining the software contents that are about to be deployed.

### 3.1.2 Uptane Framework

Uptane is a secure software-update framework designed specifically for ground vehicles. It builds on the principles of secure update systems by using signed metadata, repository separation, version checks, and hash verification to reduce the risk of common update attacks such as rollback, freeze, mix-and-match, and unauthorized software delivery [6, 7, 55]. In automotive OTA settings, Uptane is commonly used as a reference framework because it addresses the fact that vehicle updates may involve many ECUs with different computational capabilities, safety roles, and update requirements [6, 7, 55].

A central feature of Uptane is the separation between the Image repository and the Director repository. The Image repository stores software images and metadata describing those images, while the Director repository uses vehicle-specific information to determine which updates should be installed on which ECUs. This separation provides checks and balances: an update selected for a vehicle should match both the metadata describing the available image and the metadata authorizing that image for the target ECU or vehicle configuration [6, 55]. A Primary ECU typically retrieves metadata and update images, verifies the required metadata, and coordinates distribution to Secondary ECUs.

In this structure, Uptane metadata allows the vehicle to check both what software image is available and whether that image is authorized for the intended vehicle or ECU. Hashes bind metadata to the expected software image, version information helps prevent rollback to older vulnerable software, and timestamp or freshness information helps reduce freeze

attacks in which a vehicle is kept on outdated metadata. These checks allow the vehicle to reject update images that do not match the expected metadata or authorization state [6, 7].

Uptane also recognizes that not all ECUs have the same resources. Full-verification ECUs can verify metadata from both the Director and Image repositories, while partial-verification ECUs perform a reduced set of checks, typically relying on Director metadata when resource constraints prevent full verification [6, 7]. These mechanisms strengthen OTA delivery by checking the source, freshness, version, authorization, and integrity of update images. However, they are not designed to determine whether the software content inside an otherwise valid update is benign for the target platform. For this reason, Uptane provides the main reference point for OTA-Sentinel: Uptane protects the update delivery process, while OTA-Sentinel focuses on file-level screening of accepted update contents before installation.

## 3.2 Threat Model and Problem Scope

### 3.2.1 Threat Setting

OTA-Sentinel considers a pre-installation threat setting for OTA updates delivered to the IVI domain. The update is assumed to have passed the normal delivery and platform-level acceptance checks, such as authenticity verification, integrity checking, authorization, version validation, and staging controls [1, 6, 7, 17–19]. Therefore, OTA-Sentinel does not address ordinary network tampering that would already be rejected by cryptographic verification. Instead, it addresses the remaining content-focused question: whether the files inside an accepted update appear normal for the target IVI software environment.

This distinction is important because delivery trust does not always imply content benignity. If malicious or abnormal content is introduced upstream, such as during a compromised

build, supplier, repository, signing, or packaging process, the resulting package may still match the metadata and signatures later checked by the vehicle. In that case, the update may be accepted by the delivery mechanism even though one or more files inside the package are abnormal or malicious for the target platform.

### 3.2.2 Attacker Capability

The attacker is assumed to have the ability to influence the contents of an OTA package before the vehicle installs or activates it. This capability may arise from compromise of an upstream software supply-chain component, update repository, signing process, build environment, supplier-provided component, or packaging stage. Under this assumption, the attacker may insert malicious files, modify existing files, or alter portions of otherwise benign files while preserving enough of the surrounding update structure for the package to remain plausible as an OTA update.

Prior work on secure software update systems has shown that compromised signing keys or repositories can allow attackers to distribute malicious updates that appear authorized to clients [56]. Automotive OTA frameworks such as Uptane are explicitly designed to reduce the impact of partial infrastructure compromise, including repository or signing-key compromise [7, 55]. This assumption is also consistent with OTA threat models that consider compromised repositories, compromised signing keys, malicious software bundles, and arbitrary software installation attacks [6, 55]. More broadly, firmware-update and software supply-chain analyses show that compromise can occur through upstream update infrastructure, supplier code, build processes, or packaging stages before software reaches the target vehicle [57, 58].

### 3.2.3 Protected Asset and Detection Scope

The protected asset is the IVI software state that would result from installing or activating the candidate update. OTA-Sentinel operates on extracted update files before deployment, rather than on runtime execution traces, post-installation behaviour, or in-vehicle network traffic. The goal is therefore not to decide whether the update came from an authorized source, but to determine whether the accepted update contents appear normal for the intended OTA domain before installation is allowed to proceed.

OTA-Sentinel is intended to complement existing OTA trust mechanisms, not replace them. Secure boot, rollback protection, package authentication, update authorization, and metadata validation remain outside the detector's responsibility, as do delivery-focused attacks such as denial of service or freeze attacks that prevent an update from being applied at all [6, 7]. OTA-Sentinel addresses a narrower screening problem: given an update that has reached the pre-installation stage, can file-level static analysis identify contents that deviate from the learned benign profile of the target IVI software environment?

## 3.3 AAOS and QNX Update Context

OTA-Sentinel is intended for IVI update settings in which candidate update contents are available for inspection before installation or activation. AAOS-based and QNX-based IVI platforms differ in their implementation details, but both follow the same broad update logic: update contents are delivered to the vehicle, checked and prepared by the platform's native update mechanism, and then installed or activated only after the relevant update conditions are satisfied. The purpose of this section is not to provide a full platform survey, but to identify the common pre-installation point at which file-level screening can be applied.

For AAOS, this placement aligns naturally with the Android OTA model. AAOS is an IVI

platform built on Android, and its OTA process follows the Android system update path for upgrading the underlying operating system and other read-only system software [26, 27]. Modern Android-based deployments commonly use A/B or Virtual A/B updates, in which the new software is written to an inactive slot or to snapshots while the currently active system continues running, after which the device reboots into the updated state [28, 29]. In the AAOS setting considered here, OTA-Sentinel is placed after the update contents are available for inspection but before the updated system state is accepted for activation.

For QNX-based IVI deployments, the exact update flow is more dependent on the OEM and system integrator because QNX provides platform-specific software update modules rather than a single fixed update stack. Even so, public QNX documentation shows a staged update process consistent with the placement assumed by OTA-Sentinel. In the QNX CAR Platform, the software update daemon "swud" detects an update package, validates its ".manifest" and delta files, notifies the user-facing update component of a pending update, and then, once installation is initiated, coordinates with the "downsize" utility to terminate nonessential processes before applying the update and rebooting into the new software version [59, 60].

In hypervisor-based cockpit systems, the logical placement of OTA-Sentinel does not change: screening still occurs after the platform's native update mechanism has accepted or staged the candidate software, but before the target IVI software state is activated. What may change is the implementation boundary. If the IVI stack is updated within a guest domain, screening can be performed inside that guest's update path. If guest software is deployed by a host or management domain, screening may instead be applied at that higher orchestration layer before the new guest image or software state is committed.

QNX is included here to clarify the broader IVI update context and the placement of pre-installation screening in native-code automotive platforms. The experiments do not claim a complete QNX-specific evaluation because large public QNX OTA datasets and QNX-

targeted malicious OTA packages are not available at useful scale. Instead, the evaluated benign corpora focus on Hyundai-group update material and AAOS-derived update contents, while QNX motivates part of the native-code threat context discussed later in the dataset and malware-source sections.

## 3.4 Machine Learning Background for OTA-Sentinel

OTA-Sentinel uses unsupervised anomaly detection to screen OTA update files when representative benign update data are available but labelled malicious OTA samples are limited. This section introduces the machine-learning concepts needed to understand the model families used later in the experimental design. The detailed model architectures, parameter settings, feature dimensions, and training procedures are provided in the following chapter.

The selected model families cover several common assumptions used in benign-only anomaly detection. PCA provides a simple dimensionality-reduction and reconstruction baseline. One-class SVM models a boundary around normal data. Isolation Forest detects samples that can be separated easily from the rest of the data. Generative and reconstruction-based models, including autoencoders, VAEs, and f-AnoGAN, score files according to how difficult they are to reconstruct or represent. Reptile is included as a meta-learning method because OTA domains may differ from one another, and a detector may need to adapt using only a small benign support set from the target domain.

### 3.4.1 Unsupervised Anomaly Detection

Unsupervised anomaly detection is used when normal data are easier to obtain than labelled examples of all possible attacks. Instead of learning a decision boundary from both benign and malicious samples, the detector learns the structure of benign data and assigns higher anomaly scores to samples that deviate from that structure. This setting is appropriate for

OTA update screening because realistic malicious OTA packages are scarce, while benign update files can be collected from available update sources.

In a static file-screening setting, each candidate file can be represented by features that describe its structure, and an unsupervised detector can assign an anomaly score based on how far the file appears to deviate from benign reference data. A high anomaly score does not prove that a file is malicious, rather, it indicates that the file differs from the benign reference data according to the assumptions of the model. This distinction is important because OTA-Sentinel is intended as a screening layer that flags abnormal content for rejection or further analysis, not as a complete malware attribution system.

Different unsupervised models define abnormality in different ways. Reconstruction-based models treat a file as suspicious when it cannot be reconstructed well from a model trained on benign files. Boundary-based methods treat a file as suspicious when it falls outside the region occupied by normal data. Isolation-based methods treat a file as suspicious when it can be separated from the rest of the data using relatively few partitions. Dimensionality-reduction methods treat a file as suspicious when it is poorly represented by the dominant structure of the benign data.

### 3.4.2 Generative Modeling

Generative and reconstruction-based models are used in anomaly detection to learn the structure of benign data and identify samples that are difficult to represent or reconstruct. This is relevant to OTA-Sentinel because representative benign OTA files are more available than labelled malicious OTA update samples. In this setting, the detector can be trained on benign update files and use reconstruction error or related scores to flag files that deviate from the learned benign profile.

An autoencoder is a neural network composed of an encoder and decoder. The encoder maps an input feature vector to a latent representation, and the decoder reconstructs the

input from that representation. When trained primarily on benign OTA files, high reconstruction error may indicate that a file differs from the normal update-file distribution. A variational autoencoder (VAE), which is one of the models used in OTA-Sentinel, extends this idea by learning a probabilistic latent representation and combining reconstruction accuracy with latent-space regularization [61]. In other words, the VAE learns not only how to reconstruct benign files, but also how to organize their latent representations in a structured and smooth way.

Unlike VAEs, which learn a probabilistic latent representation through reconstruction and regularization, generative adversarial networks (GANs) learn a data distribution through a two-part adversarial training process. A GAN consists of a generator, which attempts to produce samples that resemble the training data, and a discriminator, which attempts to distinguish generated samples from real samples [62]. Through this competition, the generator is encouraged to produce outputs that increasingly match the benign data distribution. f-AnoGAN adapts this idea for anomaly detection by comparing an input with a generated reconstruction or feature-space representation [63]. OTA-Sentinel includes f-AnoGAN as a second generative baseline so that adversarial generative modeling can be compared with VAE-based reconstruction in the static OTA file-screening setting.

### 3.4.3 Base Anomaly-Detection Models

OTA-Sentinel uses several base anomaly-detection models to provide comparison points for the generative and meta-learning methods. These models were selected because they represent different common assumptions about what makes a sample anomalous: poor reconstruction, distance from a learned normal region, or ease of separation from the rest of the data. Including multiple base models allows the evaluation to test whether OTA file anomalies are better detected through reconstruction error, boundary-based scoring, or isolation-based scoring.

Principal component analysis (PCA) is used as a simple reconstruction-based baseline. PCA projects the input features into a lower-dimensional space that captures the dominant directions of variation in the benign training data. Files that are poorly reconstructed from this reduced representation receive higher anomaly scores. In the OTA-Sentinel setting, PCA provides a lightweight baseline for testing whether abnormal OTA files deviate from the main structure of benign file features.

One-class support vector machines (OCSVMs) provide a boundary based approach to anomaly detection [64]. Rather than learning from both benign and malicious samples, an OCSVM estimates a region that contains most of the benign training data. Samples that fall outside this region are treated as anomalous. This makes OCSVMs suitable for benign-only settings, although their performance can be sensitive to feature scaling, kernel choice, and the shape of the benign data distribution.

Isolation Forest provides an isolation-based alternative [65]. The method builds random partitioning trees and assigns higher anomaly scores to samples that can be isolated using fewer splits. The intuition is that unusual samples are often easier to separate from the rest of the data. For OTA-Sentinel, Isolation Forest is included as a non-neural baseline that can operate efficiently on static file-level feature vectors.

Together, PCA, OCSVM, and Isolation Forest provide lightweight baseline detectors against which the VAE, f-AnoGAN, and Reptile-based models can be compared. The exact feature dimensions, training settings, and scoring procedures used for these models are described in the OTA-Sentinel framework and experimental design chapter.

### 3.4.4 Meta-Learning and Reptile

Meta-learning is concerned with learning from a collection of related tasks in a way that supports fast adaptation to a new task with limited data. Instead of training one model only for a single fixed dataset, a meta-learning method learns an initialization, update rule, or

representation that can be adapted using a small support set from a target task. This idea is useful for OTA-Sentinel because OTA update files may differ across vendors, platforms, and software domains, while only a limited number of benign files may be available for a newly observed OTA domain.

Reptile is a meta-learning algorithm that trains a model to start from parameters that can be quickly adjusted to a new task using only a small amount of data [51]. During meta-training, the model is repeatedly copied, adapted for a small number of gradient steps on one task, and then the original initialization is moved toward the adapted parameters. Over many tasks, this process encourages the initialization to become one that can quickly specialize to task-specific data. Reptile is often discussed in few-shot image classification settings, where each task consists of a small classification problem. OTA-Sentinel uses the same adaptation principle in a different setting: each task corresponds to an OTA domain or update family, and the adapted model is an autoencoder used for reconstruction-based anomaly detection rather than a classifier.

For reconstruction-based anomaly detection, the same principle can be applied by meta-learning an initialization for an autoencoder rather than for a classifier. After adaptation to a target domain, the autoencoder can be used to assign reconstruction-based anomaly scores, where larger reconstruction error suggests greater deviation from the adapted benign profile. This use of Reptile is therefore not intended to perform multi-class recognition or image classification. Instead, it tests whether meta-learning can improve domain-specific adaptation for benign-only OTA anomaly detection.

This distinction is important because OTA-Sentinel evaluates Reptile as an adaptation mechanism rather than as a standalone malware classifier. The goal is to determine whether a meta-learned initialization helps the reconstruction model adjust to the normal structure of each OTA domain more effectively than models trained without meta-learning.

# Chapter 4: Proposed Framework and Analytical Methodology

## 4.1 OTA-Sentinel Overview

OTA-Sentinel is a pre-installation file-screening framework for OTA updates delivered to the IVI domain. The screening stage may be deployed centrally at the OEM backend before fleet distribution, at an OTA staging or repository service, or within the vehicle-side staging process after package retrieval and verification. In a centralized OEM deployment, each distinct update package can be screened once before distribution rather than separately on every receiving vehicle, reducing redundant processing across the fleet. Since OTA-Sentinel represents and scores files independently, feature extraction and anomaly scoring can also be parallelized across files, packages, software releases, and regional or vehicle-specific variants. These properties provide a scalable deployment path, although production-scale performance would need to be validated within a complete OTA pipeline. Figure 4.1 shows OTA-Sentinel in a vehicle-side screening configuration, where the package is analyzed after verification and before final installation. The same screening step could also be performed earlier at the OEM backend or at an OTA staging service.

At this point in the pipeline, the update contents are available for inspection as extracted files. OTA-Sentinel represents these files using lightweight static features, scores them with anomaly-detection models trained primarily on benign OTA data, and identifies packages containing files that deviate from the learned benign profile of the target platform. In doing so, OTA-Sentinel adds a targeted pre-installation screening layer for suspicious update contents that may not be captured by authenticity and integrity checks alone. It therefore complements, rather than replaces, established secure OTA frameworks such as Uptane [6, 7]. The following section breaks OTA-Sentinel into its main framework components and

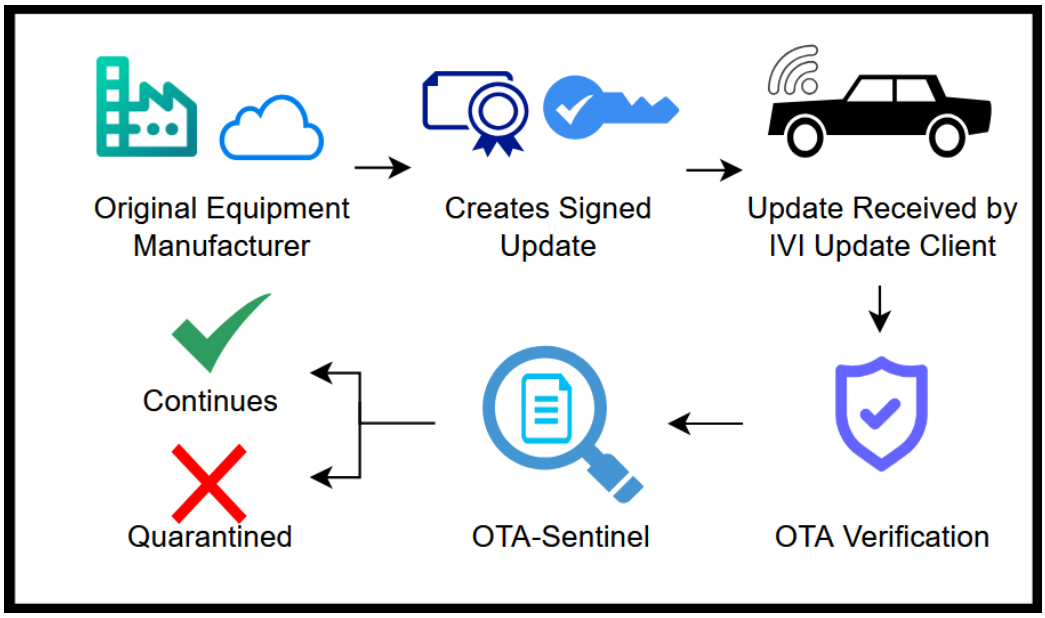



Figure 4.1: High-level overview of OTA-Sentinel in the OTA update pipeline, shown in a vehicle-side screening configuration. After the vehicle receives and verifies a signed OTA package, OTA-Sentinel analyzes the update contents before installation. Packages assessed as benign proceed through the normal update path, while suspicious packages are flagged for blocking, quarantine, or further inspection. The same screening step could also be performed earlier at the OEM backend or at an OTA staging service before distribution.

describes how each component contributes to pre-installation OTA file screening.

The design choices in this chapter are organized around the research questions introduced in Chapter 1. The file-level static representation is used to test whether lightweight byte-level features can separate benign and anomalous OTA contents, addressing RQ1. The baseline model suite is used to determine which unsupervised detection approach is most viable under pooled benign training and varying anomaly conditions, addressing RQ2. The use of Hyundai-group and AAOS update sources supports the cross-domain evaluation in RQ3, while the Reptile and SmallAE adaptation settings address RQ4. The feature-extraction and model-efficiency measurements address RQ5, and the separation of feature blocks supports the later interpretability analysis in RQ6.

## 4.2 OTA-Sentinel Framework

OTA-Sentinel is organized as a file-level screening pipeline that operates on OTA update contents before installation. The framework does not replace cryptographic verification, signature validation, or platform-level update controls. Instead, it adds an additional content-analysis stage after an update has been retrieved and accepted by the normal update mechanism, but before the update is committed to the IVI system. The main components of this pipeline are update extraction, static file representation, benign profile learning, anomaly scoring and calibration, and pre-installation decision logic. Figure 4.2 summarizes these components and shows how the preprocessing stage connects to the training and testing workflow.

### 4.2.1 Update Extraction and File-Level Screening

The first component of OTA-Sentinel is the extraction of OTA package contents into individual files that can be analyzed before installation. This file-level granularity is used because a malicious modification may affect only a small part of an otherwise valid update package. A package-level score alone could hide this type of localized change, especially when most files in the update remain benign. By screening extracted files individually, OTA-Sentinel can identify local deviations inside a larger update before the package is installed or activated.

This component assumes that the update has already passed the normal platform-level acceptance checks and that the file contents are available for inspection during the staging phase. OTA-Sentinel therefore focuses on content screening rather than update authentication. Its role is to examine the files that would be installed and determine whether any of them differ substantially from the expected structure of benign OTA software for the target domain.

### 4.2.2 Static Feature Representation

After extraction, each file is converted into a compact static feature vector derived from its raw byte content. The representation is designed to avoid runtime execution, dynamic tracing, or platform-specific instrumentation. This makes the approach suitable for pre-installation screening, where the goal is to inspect update contents before they are allowed to run on the IVI system.

The feature representation captures several byte-level properties of each file, including file size, Shannon entropy, byte-frequency structure, and short byte-sequence information. File size provides a coarse measure of structural scale, entropy captures patterns associated with compression, packing, or encrypted regions, the byte histogram summarizes overall byte composition, and byte 3-gram information captures local sequence structure. Together, these features provide a lightweight description of file content that can be computed uniformly across both vendor OTA files and AAOS-derived update contents.

### 4.2.3 Benign Profile Learning

OTA-Sentinel learns the expected structure of OTA files primarily from benign update data. This design reflects the practical constraint that representative benign OTA files are more obtainable than labelled malicious automotive OTA files. Instead of training a supervised classifier to recognize known malware families, the framework models the distribution of benign file features for the relevant OTA domain and then treats strong deviations from that distribution as suspicious.

In this setting, benign profile learning can be performed using several anomaly-detection model families. Reconstruction-based models learn to reproduce benign feature vectors and assign higher anomaly scores to files that reconstruct poorly. Boundary-based models learn a region around benign files in feature space and treat samples outside that region as more suspicious. Tree-based and adversarially trained models provide additional ways

to model normality or deviation under the same file-feature representation. The common purpose across these detectors is to learn what normal OTA file structure looks like without requiring a large labelled set of malicious OTA examples.

### 4.2.4 Anomaly Scoring and Calibration

Once a benign profile has been learned, OTA-Sentinel assigns an anomaly score to each extracted file. For reconstruction-based models, this score is based on reconstruction error. For boundary-based models, the score reflects distance from or position relative to the learned benign region. For other detector families, the model-specific score is converted so that larger values indicate stronger deviation from the benign profile.

A calibration step is required to convert continuous anomaly scores into screening decisions. For a benign-only deployment setting, this threshold would need to be estimated from benign calibration files associated with the target platform or update domain. The Reptile-based evaluation follows this structure by using benign calibration data to select the anomaly threshold after target-domain adaptation. The pooled baseline experiments, however, use labelled evaluation data to select operating thresholds for comparative analysis. Those baseline thresholds should therefore be interpreted as retrospective measures of detection capacity rather than as deployable OTA screening thresholds.

### 4.2.5 Pre-Installation Decision Logic

The final component of OTA-Sentinel is the pre-installation decision stage. At this point, the framework has extracted the update contents, represented each file using static features, assigned anomaly scores, and compared those scores against the relevant threshold. Files whose scores exceed the threshold are treated as suspicious because they deviate from the learned benign profile of the target OTA domain.

Although scoring is performed at the file level, the decision can be applied at the update-

package level. If no extracted files are flagged, the update may continue through the normal installation path. If one or more files are flagged, the package can be rejected, quarantined, or sent for further inspection before installation. In this way, OTA-Sentinel provides a targeted screening layer between update verification and final deployment, reducing the chance that abnormal update contents are installed simply because the package passed authenticity and integrity checks.

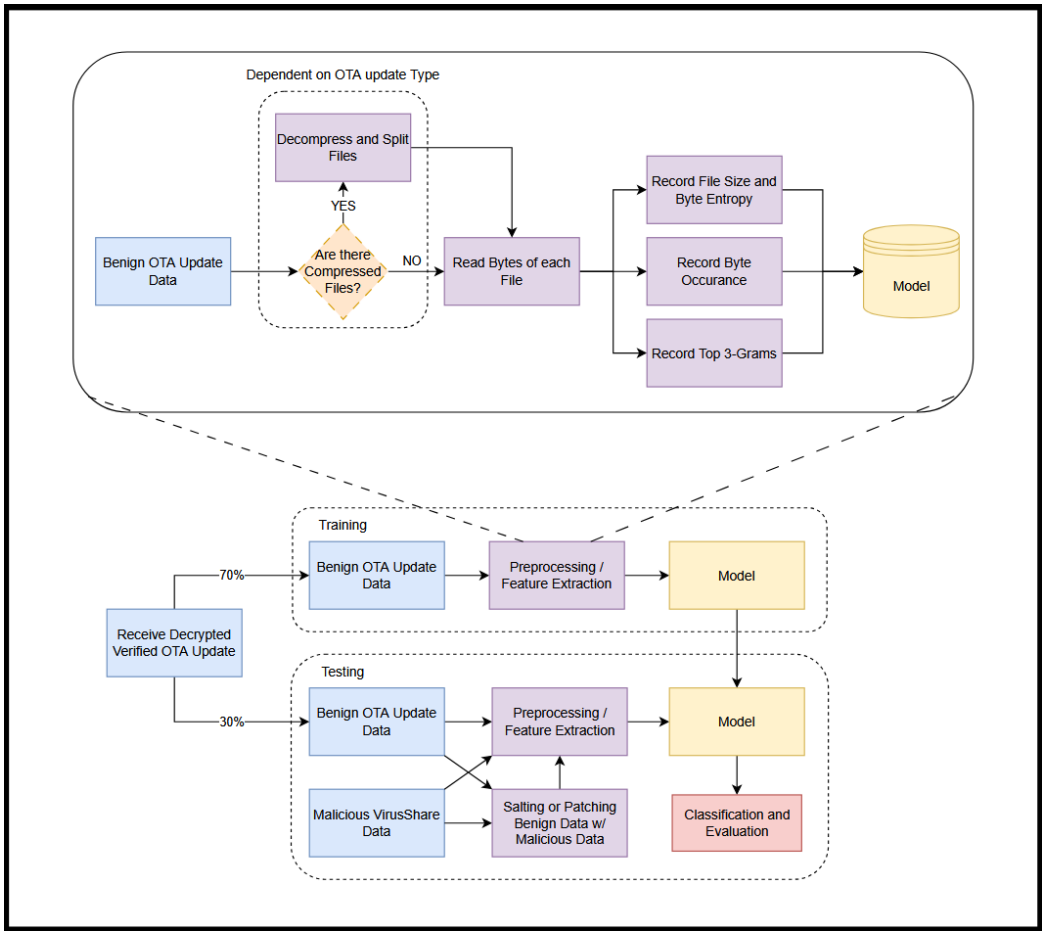



Figure 4.2: The operational overview of the OTA-Sentinel analytics pipeline. Benign OTA update files are extracted or decompressed as needed, read at the byte level, and converted into static features including file size, entropy, byte occurrence, and byte 3-gram information. The lower portion shows how verified OTA data are divided into training and testing partitions: benign files are used for model training, while held-out benign files and VirusShare-derived malicious inputs are used to evaluate detection performance under raw, salted, and patched anomaly conditions.

## 4.3 Analytical Methodology and Evaluation Assumptions

The analytical methodology is organized around the limited-label OTA security setting defined in Chapter 3. Representative benign OTA files are used to learn normal update-file structure, while malicious or modified files are introduced only during evaluation. This design reflects the intended deployment setting: benign update material may be available for a target software domain, but labelled malicious OTA packages for that same domain are unlikely to be available at useful scale.

This methodology is intentionally file-level rather than package-level. OTA updates may contain many files, and a malicious change may affect only a small subset of the package. By scoring files individually, OTA-Sentinel can detect localized deviations that could be diluted or hidden by an aggregate package-level representation. The final package-level decision is then derived from the file-level screening outcome: if one or more files exceed the relevant anomaly threshold, the update can be rejected, quarantined, or forwarded for further analysis before deployment. The experiments therefore evaluate whether models trained primarily on benign OTA file structure can identify abnormal or malicious deviations before installation.

These assumptions follow from the data constraints described in the previous chapter. Production OTA packages are not widely available, file-level malicious OTA examples are difficult to obtain, and manufacturer-specific update contents are often proprietary. As a result, the experimental design treats benign OTA structure as the main source of training information and uses malware-derived or synthetically modified files to test whether the learned benign profile can detect abnormal content. The goal is not to classify known malware families, but to assess whether lightweight file-level screening can flag update contents that differ from expected OTA software distributions before installation.

## 4.4 Dataset Sources and Generation

| Statistic | Hyundai-group | AAOS |
|---|---|---|
| Number of benign files | 18,665 | 64,802 |
| Total benign data (GiB) | 37.52 | 36.55 |
| Mean file size (KiB) | 2104.89 | 591.39 |
| Median file size (KiB) | 424.26 | 16.92 |
| File size IQR (KiB) | 42.63–1384.12 | 3.26–99.34 |
| Mean entropy | 4.556 | 5.040 |
| Median entropy | 4.650 | 5.264 |
| Entropy IQR | 3.721–5.229 | 4.594–6.267 |
| Mean local entropy std. dev. | 0.385 | 0.878 |
| Median local entropy std. dev. | 0.162 | 0.936 |
| Local entropy std. dev. IQR | 0.086–0.696 | 0.000–1.443 |

Table 4.1: Descriptive statistics for the total benign OTA domains used in the evaluation. File-size values are reported in KiB except where noted. IQR denotes the interquartile range, reported as Q1–Q3.

This section describes the datasets used to develop and evaluate the proposed OTA screening method. The dataset design reflects the limited-label setting defined in Chapter 3: benign OTA contents are used to learn normal update structure, while external malware corpora and synthetically modified OTA files are introduced only for evaluation. The use of both Hyundai-group and AAOS-derived benign update contents also supports RQ3 by allowing the experiments to examine whether model behaviour changes across distinct OTA software domains, rather than treating OTA files as a single uniform distribution.

The benign data used in this thesis were drawn from two main sources. The first consists of production-style OTA contents associated with Linux-based automotive infotainment platforms. These files were collected from two Hyundai-group Navigation Updater collection periods [66]. A full list of obtained updates is listed in Appendix A, where list A.1 shows the NaU updates taken after the November 2025 update, and list A.2 shows the NaU updates taken prior to the November 2025 update, where exact software and map versions are no longer available. These files provide a realistic view of the byte-level and structural properties of conventional OTA-delivered IVI software. Before filtering, the archived

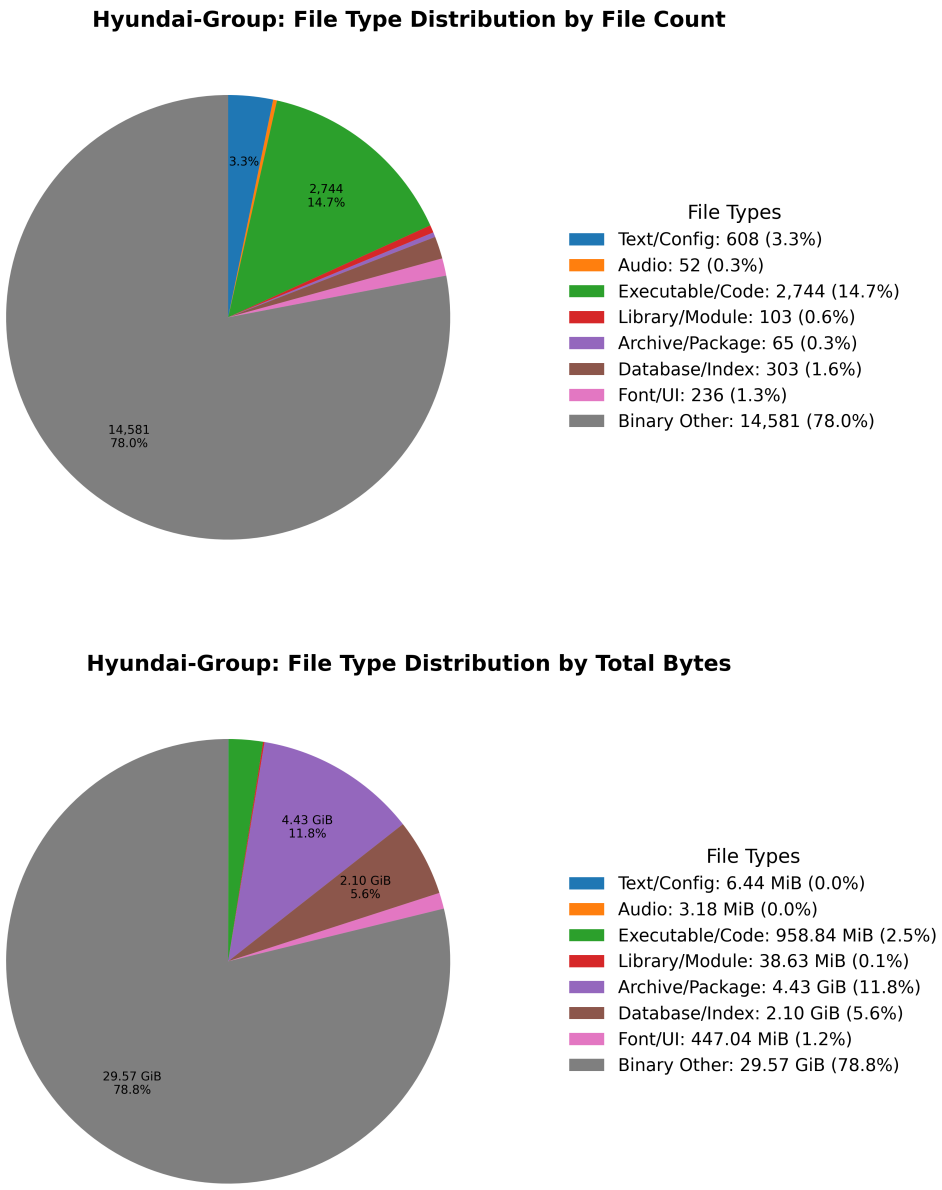



Figure 4.3: Hyundai-group benign OTA file-type distribution by file count (top) and total bytes (bottom).

Hyundai-group update material totalled approximately 162.4 GB. After extraction and removal of encrypted or unusable files, the retained Hyundai-group corpus contained 18,665 benign files and approximately 37.5 GB of processed data. Figure 4.3 summarizes the Hyundai-group benign OTA file-type distribution by file count and total byte volume.

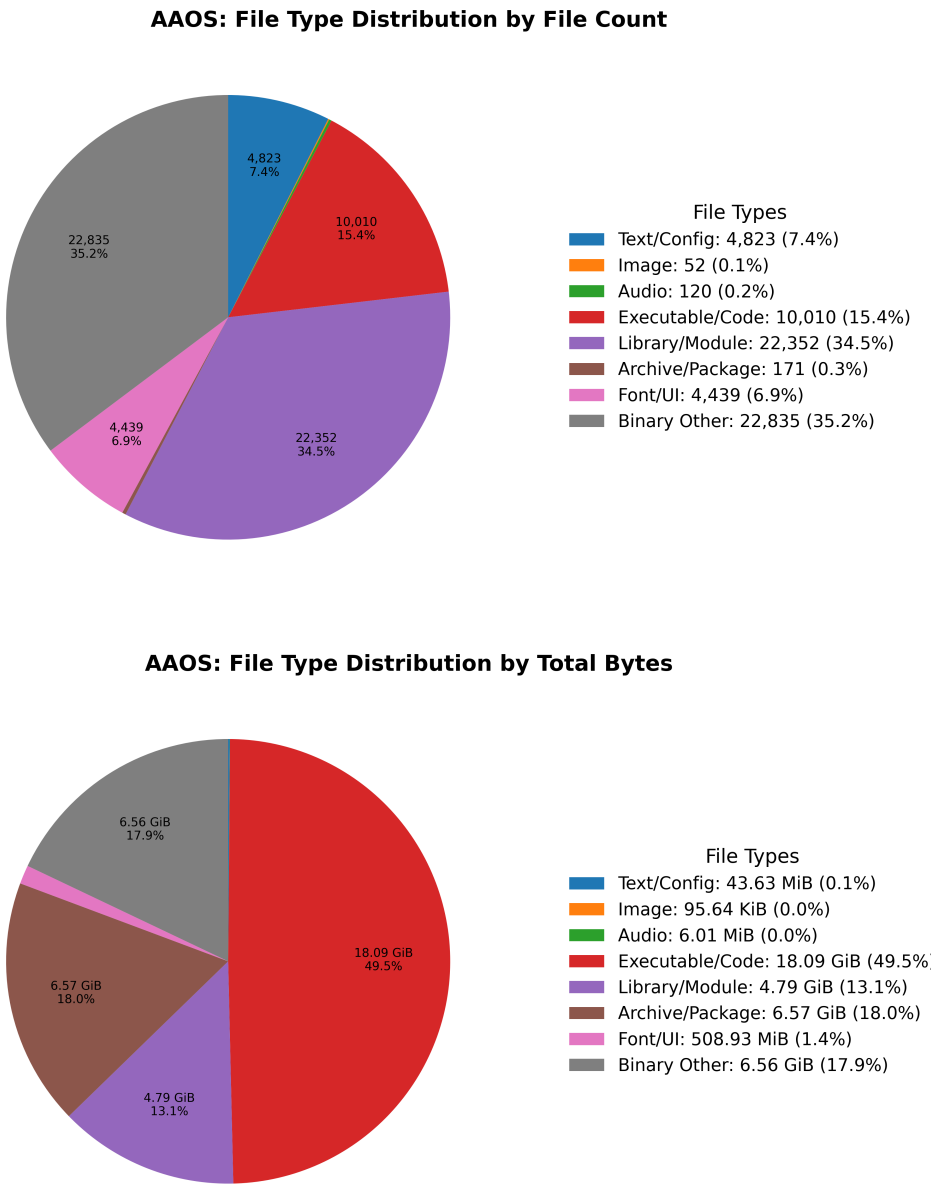



Figure 4.4: AAOS benign OTA file-type distribution by file count (top) and total bytes (bottom).

The second benign source consists of AAOS OTA data derived from Android Automotive OS builds generated from the Android Open Source Project (AOSP). Because AAOS is a full-stack Android-based platform intended to run directly on in-vehicle hardware, it provides a useful representation of modern Android-based IVI software layouts. The dataset was constructed from AAOS build outputs, including system images and OTA-

related contents, obtained through the standard AOSP source retrieval and build workflow. In carrying out this process, official AOSP documentation was used together with a practitioner-oriented AAOS build guide consulted during setup. After extraction, filtering, and normalization, the retained AAOS corpus contained 64,802 files across ten normalized AAOS build-target families, with a total processed size of approximately 36.5 GB. This made the AAOS corpus substantially larger than the Hyundai-group corpus in file count, while remaining comparable in processed data volume. Together, these two benign corpora provide coverage across both traditional Linux-based IVI environments and Android-based infotainment systems [26, 67, 68]. Figure 4.4 summarizes the AAOS benign OTA file-type distribution by both file count and total byte volume. This distribution helps show that the AAOS corpus is not only larger in file count than the Hyundai-group corpus, but also structurally different in the types and sizes of files retained for evaluation. A full list of AAOS builds can be found in Appendix B, and an overview of the retained benign corpora is provided in Table 4.1.

### 4.4.1 Malware Datasets

Two external malware datasets were used in this thesis. The first consists of Linux malware samples obtained from the restricted-access VirusShare repository, specifically the `VirusShare\_Linux\_20160715`, `VirusShare\_ELF\_20190212`, and `VirusShare\_ELF\_20200405` collections. These primarily contain ELF-format files and were used to represent malicious software relevant to Linux-based IVI environments [69]. In this work, the Linux ELF malware dataset is used as a proxy for QNX-specific malware because QNX systems also use ELF-format executables. Although these samples are not QNX-specific, they provide a practical native-code malware source for evaluating file-level anomalies when QNX-targeted malware and malicious QNX OTA datasets are not available. The second consists of Android malware samples obtained from the restricted-access "VirusShare_Android_APK_2022-2" collection. This dataset contains APK files and was

used to represent malicious software relevant to AAOS and other Android-based infotainment platforms [69]. These malware sources were not treated as authentic automotive OTA packages; rather, they were used as external adversarial corpora for evaluating whether benign OTA file profiles could distinguish clearly malicious software from normal update contents. They were also used to construct controlled synthetic contamination scenarios in which malicious content was introduced into otherwise benign OTA files through direct, salted, and patch-based modification conditions.

The choice of malware sources was guided by the platform distribution summarized in Appendix D.1. Publicly available OEM and supplier information shows that modern IVI deployments commonly rely on native software stacks, Android-based infotainment platforms, or QNX-based automotive software platforms. In this work, QNX is not treated as Linux-based. Rather, it is treated as a Unix-like, POSIX-compliant real-time operating system with native executable content that is closer to Linux-like native software than to Android APK application packages. Modern QNX systems also use the ELF executable format, which provides a limited basis for using Linux ELF malware as an approximate native-code adversarial source. However, Linux ELF malware is not treated as QNX malware, nor as a complete substitute for QNX-specific malicious software. It is used only as a practical proxy for native-code byte-level malicious content when real QNX-targeted automotive malware and malicious OTA packages are not available at useful scale.

### 4.4.2 Synthetic Malicious Modification Procedure

Because real malicious automotive OTA packages are not readily available at useful scale, controlled adversarial test cases were constructed by modifying benign OTA files at the file level. The procedure began with files extracted from benign OTA datasets, after which malicious content drawn from the external malware corpora was introduced in ways intended to preserve much of the surrounding benign structure. As shown in Figure 4.5, the resulting test inputs were organized into three modification modes before passing through

the common feature extraction and testing pipeline. This design reflects the pre-installation threat setting considered here: an update may still appear structurally plausible as an OTA package even when some of its internal file contents have been maliciously altered.

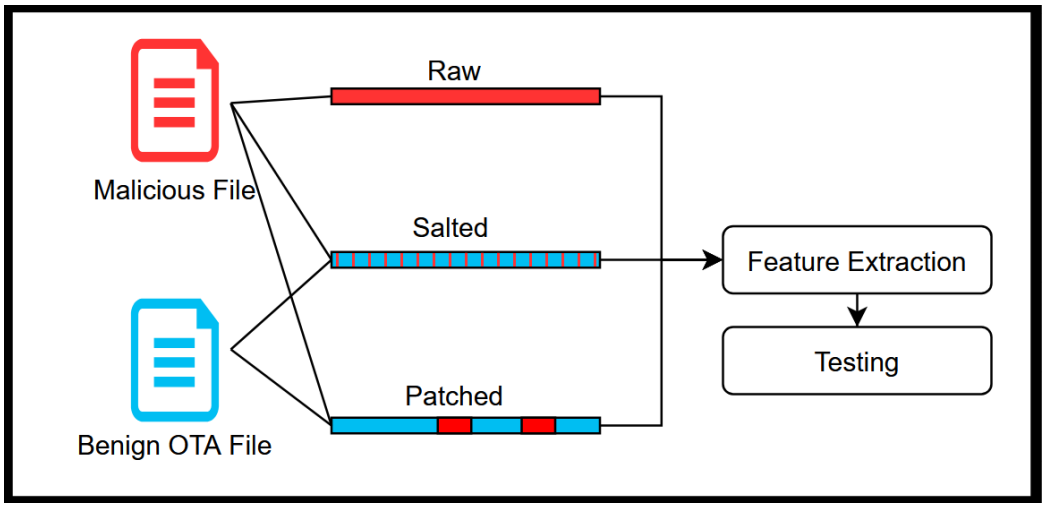



Figure 4.5: Synthetic malicious modification procedures used for evaluation. Adversarial test inputs were generated either as direct malware samples, as salted benign files with dispersed malicious bytes, or as patch-modified benign files with localized malicious regions, after which all files were processed through feature extraction and testing.

Three forms of malicious test input were used. The first used malware files directly as anomalous samples, allowing evaluation against clearly malicious content without mixing it into benign files. The second used byte-level salting, in which a benign file was modified by injecting a controlled fraction of malicious bytes sampled from the malware corpus. Salted variants were generated at 0.1%, 5%, and 10% byte-level contamination. The third used patch-based modification, in which malicious bytes were inserted into a small number of localized regions inside an otherwise benign file. In this setting, the total injected byte budget was kept small and distributed across bounded, non-overlapping patches so that the modification remained spatially concentrated rather than diffused throughout the file. In the experiments reported here, patch-based samples used a 0.1% total byte-injection budget with individual patches capped at 64 KB.

These three constructions were used to test different anomaly conditions under a controlled

pre-installation setting. Direct malware samples measure whether the learned benign profile rejects clearly foreign software. Salting measures sensitivity when malicious content is dispersed sparsely across a largely benign file. Patch-based modification measures sensitivity when only limited localized regions are altered. Together, these cases provide controlled anomaly inputs without requiring a large collection of real malicious automotive OTA packages.

## 4.5 Data Representation and Feature Extraction

The experimental implementation of OTA-Sentinel used two static byte-level feature representations. The primary representation encoded each extracted file as a fixed-length 298-dimensional vector derived directly from its raw byte content. In the baseline representation shown in Figure 4.6, the feature vector combines file size, Shannon entropy, a byte-histogram block, and additional 3-gram-related features that capture selected local byte-sequence structure and associated occurrence information. File size gives a coarse measure of structural scale, entropy captures differences associated with compression, packing, or encrypted regions, the histogram summarizes overall byte composition, and the 3-gram-related features capture short local byte patterns that can recur across related software content.

In later experiments, a binned variant was also used in place of the explicit trigram portion. In that variant, all extracted byte trigrams were converted into a stable string form and mapped into a fixed number of bins using an MD5-based hash. Trigram counts assigned to the same bin were accumulated and then transformed using “log1p”, producing a fixed-length hashed trigram block. The number of bins was configurable during preprocessing, so the binned form preserved the same general byte-level design while replacing explicit trigram coordinates with a compact hashed representation.

These features were chosen because closely related static representations have been ef-

fective in malware detection outside the OTA setting. Kumar and Geetha describe file-independent static features such as byte histograms, whole-file entropy, entropy from different file regions, and n-grams as useful inputs for malware detection, and use them within an explainable ensemble-learning framework [70]. Earlier static malware-detection work also showed that byte-sequence features and statistical summaries of raw file content can be effective for distinguishing malicious and benign executables [71–73]. On that basis, file size, entropy, byte histograms, and byte 3-gram features were selected here as a compact static representation for OTA file screening.

These two representations allow the evaluation to separate two related questions. The primary 298-dimensional representation tests whether a compact static byte-level description is sufficient for OTA anomaly detection. The hash-binned representation tests whether adding broader local byte-sequence information improves detection or instead introduces noise through compression and bin collisions. This comparison supports the feature effectiveness aspect of RQ1 and the model-comparison focus of RQ2, while also providing the feature blocks later used for the interpretability analysis in RQ6.

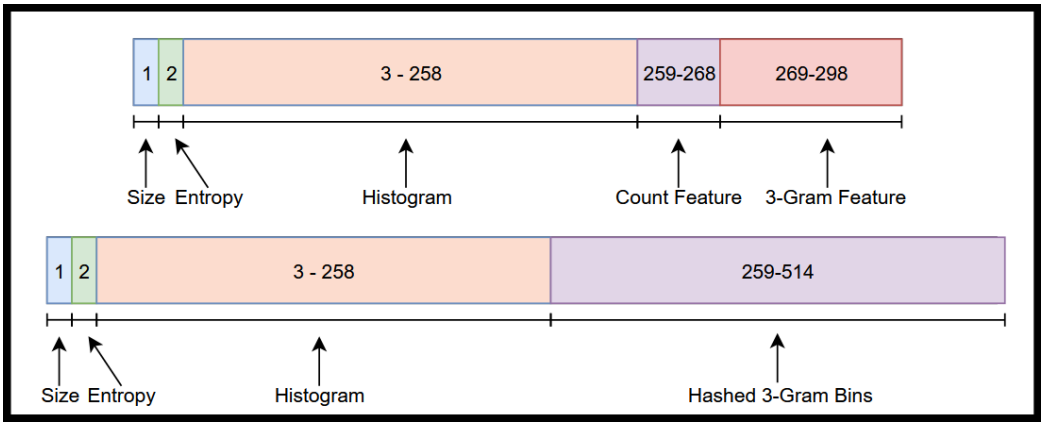



Figure 4.6: File-level feature representations used in the experiments. Both representations encode each extracted file using static byte-level features including file size, entropy, and a byte histogram. The baseline form uses explicit 3-gram-related features, while the binned form replaces that portion with hashed 3-gram bins.

The same feature family was adopted here to test whether byte-level static representations used in broader malware analysis can also support pre-installation OTA screening.

Rather than relying on platform-specific metadata or runtime behaviour, the goal was to use a compact representation that can be computed uniformly for files extracted from both vendor Linux-based OTA packages and AAOS-derived update contents. Prior automotive work has also considered screening software before installation, but typically in supervised executable-classification settings [31]. The present setup instead uses file-level anomaly detection under limited-label and zero-day conditions.

## 4.6 Experimental Tasks, Splits, and Evaluation Setting

The experiments were organized around benign OTA domains rather than supervised malware classes. This reflects the intended screening setting: a manufacturer or evaluator may have clean examples from a target update stream, but may not have labelled malicious OTA packages for that same stream. The pooled baseline setting supports comparison of unsupervised detectors under shared benign training, while the domain-based Reptile setting supports RQ4 by testing whether adaptation from a small benign support set improves detection within individual OTA domains.

For each task, the benign files were partitioned into training and held-out evaluation subsets using a 70/30 split. The training portion was used during model fitting, while the held-out portion was reserved for adaptation, calibration, and final testing. In the Reptile setting, meta-training episodes were formed from benign files within the available source tasks, so that support and query samples came from the same domain during each episode. At evaluation time, the model was adapted using a small benign support set drawn from the target task, calibrated using benign calibration files, and then scored on the remaining held-out benign files together with the malicious or modified test files. This matches the practical assumption that some clean files from the target update stream may be available, but labelled attack examples will not be.

The pooled baseline models used the same train-test separation between benign data and

anomaly exposure, but without task-based adaptation. Benign files from the training partition were combined to fit a single detector, and evaluation was then performed on held-out benign files together with the corresponding anomaly set. Malicious files were excluded from training and were introduced only during evaluation.

## 4.7 Model Components and Training Configuration

The model set was selected to compare several plausible ways of defining abnormality under the same static file representation. Since the training setting is primarily benign-only, each method must be able to learn from benign OTA files and assign anomaly scores to unseen files without requiring labelled malicious OTA examples during training. This comparison provides the basis for answering RQ2, while the Reptile and SmallAE configurations extend the evaluation toward RQ4 by testing whether target-domain benign support data improves adaptation beyond a single pooled benign profile.

### 4.7.1 Baseline Models

**The VAE** baseline used a fully connected encoder–decoder architecture. The encoder mapped the 298-dimensional input through hidden layers of size 512, 256, and 128 before producing latent mean and log-variance vectors, while the decoder mapped from the latent space through layers of size 128, 256, and 512 back to the 298-dimensional input space. ReLU activations were used throughout, layer normalization was applied in most hidden layers, and dropout was set to 0.05 in the encoder and 0.0 in the decoder. Training used Adam with a learning rate of $10^{-4}$, batch size 32, 50 epochs, gradient clipping at 1.0, latent dimension 128, and a KL-divergence weight of $\beta = 0.1$. Reconstruction loss was computed with an equally weighted blockwise mean-squared error over the file-size, entropy, histogram, trigram identity, and trigram count blocks.

**The PCA** baseline modeled normality through linear reconstruction in a lower-dimensional

subspace. It was trained on the same normalized 298-dimensional input vectors used by the other pooled baselines. The implementation used 64 principal components with a fixed random state of 0. At test time, each input vector was projected into the learned subspace and then inverse transformed back to the original feature space, and the anomaly score was defined as the mean squared reconstruction error between the input and its reconstruction.

**The one-class SVM** baseline used an RBF kernel with $\nu = 0.05$ and gamma = scale. The RBF kernel allowed the model to learn a nonlinear boundary around the benign training data rather than being restricted to a linear separation. A StandardScaler was applied within the model pipeline before fitting the one-class classifier so that features with different numeric ranges would not dominate the distance calculations used by the RBF kernel. At test time, anomaly scores were defined as the negative of the model decision function, so that larger values indicated stronger deviation from the learned benign region. This provided a conventional boundary-based baseline for pooled benign training data.

**The Isolation Forest** baseline modeled anomalies through recursive random partitioning of the feature space. The implementation used 50 trees and the default automatic contamination setting. This provided a tree-based unsupervised baseline under the same pooled training setting.

**The f-AnoGAN** baseline used a generator, discriminator, and encoder defined over the same 298-dimensional input size, but with preprocessing adapted for adversarial training. The generator mapped a 100-dimensional latent vector through hidden layers of size 256 and 512 to produce a 298-dimensional output with a tanh final activation. The discriminator mapped the input through layers of size 512 and 256 before a binary classification head, and the encoder mapped the input through two 256-unit hidden layers into the same 100-dimensional latent space. GAN training used binary cross-entropy with logits and Adam optimizers with learning rate $2 \times 10^{-4}$ and betas $(0.5, 0.999)$. The encoder was then trained

separately with Adam at $10^{-4}$ using a weighted sum of input-space L1 reconstruction loss and discriminator feature-space L1 loss, with $\lambda_{\text{img}} = 0.9$. The default configuration used batch size 64, 30 GAN epochs, and 15 encoder epochs.

The baseline models were chosen to represent different anomaly-detection assumptions rather than variations of a single technique. PCA provides a simple linear reconstruction reference point. OCSVM tests a nonlinear boundary around benign files. Isolation Forest provides a non-neural isolation-based detector. VAE tests probabilistic reconstruction, while f-AnoGAN provides an adversarial generative alternative. Keeping all five methods on the same feature representation allows the evaluation to focus on which modelling assumption is more viable for pre-installation OTA file screening.

### 4.7.2 Reptile Model

The Reptile-based detector used a meta-learned autoencoder rather than a supervised classifier. Its base learner was a compact fully connected autoencoder with a 298-dimensional input layer, a 256-unit hidden layer, a 64-dimensional latent layer, and a symmetric decoder of size 64–256–298. Reconstruction loss was computed with the same blockwise mean-squared error used elsewhere in the study, with equal weighting across the file-size, entropy, byte-histogram, trigram-identity, and trigram-count blocks.

Task construction differed from the more common class-labelled use of Reptile. Instead of forming $N$-way, $K$-shot classification episodes, the task builder grouped benign files by OTA family. Only benign samples were used when constructing the domain-level task assignments for meta-training, and each SHA-256 entry was mapped to a resolvable HDF5 group through a hash-to-key lookup built from the deduplicated feature store. Tasks with too few usable benign samples were filtered out before meta-training.

Each OTA domain was first divided internally into separate benign meta-training and evaluation pools; in the experimental setup used here, this corresponded to a 70/30 benign split

within each domain. The 70% split from each domain was used during meta-training, while the remaining 30% from those same domains was reserved for later adaptation, calibration, and benign testing. As a result, the evaluation did not hold out an entire unseen domain. Instead, it measured adaptation and anomaly scoring on unseen benign samples drawn from domains that had already contributed meta-training data.

Meta-training proceeded by sampling one benign domain task at each meta-iteration and drawing a support set from that task. A fast copy of the autoencoder was initialized from the current meta-parameters and then adapted on the sampled support set using Adam. The default meta-training configuration used support size 64, 10 inner-loop steps, inner learning rate $10^{-3}$, meta learning rate 0.25, 2000 meta-iterations, latent size 64, and a minimum task size of 50. Tasks with at least 50 usable benign samples were retained for meta-training; when a retained task contained fewer than the requested support size, support samples were drawn with replacement. After the inner updates, the shared initialization was moved toward the adapted parameters using the standard Reptile update.

Separate training and test HDF5 stores, task mappings, and lookup tables were loaded for the benign train and test splits. For each task in the benign test split, the available pool was shuffled and divided into a support subset for adaptation, a calibration subset for threshold selection, and a held-out benign subset for evaluation. The default evaluation settings used 64 benign support samples, 10 adaptation steps with learning rate $10^{-4}$, 64 benign calibration samples, 256 held-out benign test samples, 512 anomaly samples, and a decision threshold set at the 95th percentile of the calibration reconstruction scores. Anomalies were then introduced only at evaluation time and scored together with the held-out benign files using reconstruction error from the adapted autoencoder. Figure 4.7 provides a high-level overview of the Reptile-based workflow.

This use of Reptile differs from the standard few-shot setting in several important ways. First, tasks were defined by benign OTA domains rather than by supervised class labels.

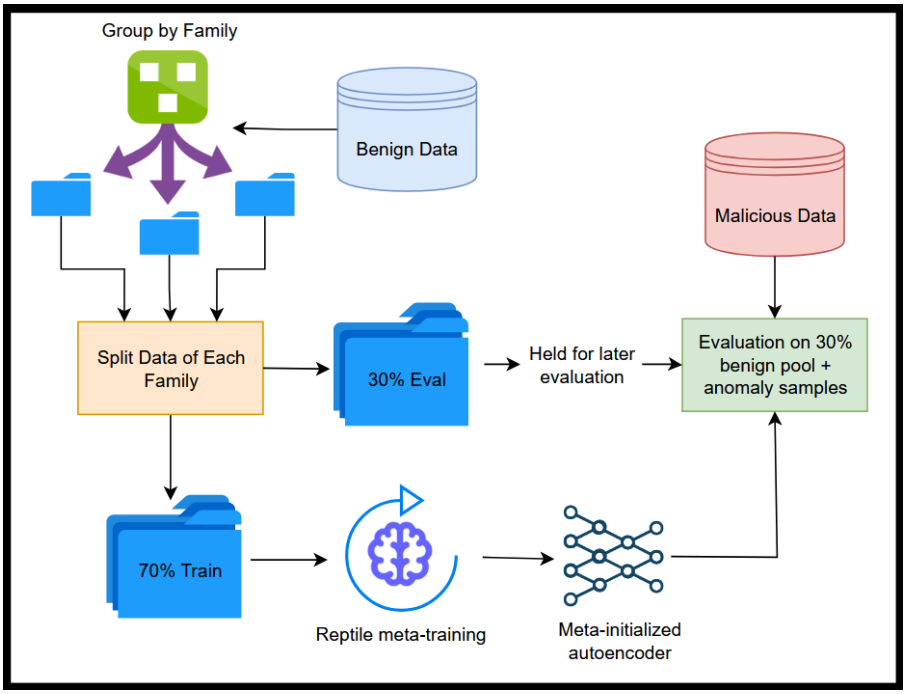



Figure 4.7: Overview of the Reptile-based anomaly-detection workflow. Benign OTA files are grouped by domain and divided into benign meta-training and evaluation pools. Reptile learns a shared autoencoder initialization from the meta-training portions, which is later adapted and evaluated on held-out benign samples from those same domains together with anomaly inputs using reconstruction-error scoring.

Second, performance was measured by benign-only adaptation followed by reconstruction-error scoring, rather than by class prediction on an episodic query set. Third, the evaluation was not designed to test leave-one-task-out transfer to a completely unseen OTA family. Instead, it was designed around the adaptation question posed in RQ4: whether a meta-learned autoencoder initialization can be useful when the detector is adjusted using a small benign support set from the target OTA domain and labelled malicious examples are unavailable. In effect, Reptile was used here as a domain-conditioned meta-initialization method for unsupervised anomaly detection, rather than as a general malware classifier or a few-shot classifier for previously unseen classes. This formulation better matches the structure of OTA screening, where the practical goal is to adapt to the normal file distribution of a target update domain before scoring suspicious deviations.

### 4.7.3 SmallAE Few-Shot Baseline

SmallAE was included as a simple non-meta-learning comparison for the Reptile support-size experiments. It used the same 298-dimensional feature representation, autoencoder structure, and block-weighted reconstruction loss as the Reptile inner learner, but it did not use episodic meta-training. Instead, it was trained as a conventional pooled benign autoencoder before being adapted within each target domain.

During evaluation, SmallAE followed the same support, calibration, and test procedure used for Reptile. For each target domain, the model was adapted using the benign support set, the anomaly threshold was selected from the benign calibration scores, and final performance was measured on held-out benign files together with the corresponding anomalous inputs. This kept the evaluation protocol aligned across the two methods while removing the Reptile meta-initialization stage.

The purpose of this baseline was to separate the effect of target-domain adaptation from the effect of meta-learning itself. If both Reptile and SmallAE perform well after using the same benign support data, then the improvement may come from ordinary adaptation to the target OTA domain rather than from the meta-learned initialization. Conversely, if Reptile outperforms SmallAE under the same support, calibration, and test procedure, then the result provides stronger evidence that meta-learning contributes additional benefit. This comparison is therefore central to interpreting RQ4. Support sizes of 8, 16, 32, 64, and 128 files were evaluated for both methods.

# Chapter 5: Experimental Evaluation

## 5.1 Chapter Overview

This chapter evaluates the proposed OTA-Sentinel framework with respect to the research questions. Before presenting the experimental results, the chapter first defines the evaluation metrics, thresholding procedures, datasets, and anomaly sources used across the study. It then presents the results for mixed-OTA baseline performance, cross-domain transfer, malicious modification detection, Reptile-based domain adaptation, feature and support-size analyses, hash-binned n-gram experiments, interpretability analysis, and computational efficiency.

The experiments in this chapter are structured around the research questions introduced in Section 1.3.1. First, the pooled baseline experiments address RQ1 and RQ2 by examining the extent to which lightweight static byte-level features distinguish benign OTA files from malicious or anomalous files, and by comparing PCA, Isolation Forest, OCSVM, f-AnoGAN, and VAE under mixed-OTA training and varying contamination levels. Next, the cross-domain transfer experiments address RQ3 by assessing how well models trained on one OTA domain generalize to another, including transfer between Hyundai-group and AAOS-derived benign training settings. The malicious modification experiments further inform RQ1 and RQ2 by testing whether the same feature space remains effective when malicious content is embedded within otherwise benign OTA files through raw, salted, and patched anomaly conditions. The Reptile experiments address RQ4 by evaluating the extent to which meta-learning supports adaptation to individual OTA domains using limited benign support data. The support-size, feature-ablation, hash-binned representation, interpretability, and computational-efficiency experiments further examine the practical behaviour of the proposed framework, including the feature-use patterns associated with RQ6

and the computational costs associated with RQ5.

## 5.2 Evaluation Metrics and Thresholding

Performance was evaluated as a binary anomaly-detection problem, where malicious or modified files formed the positive class and benign files formed the negative class. Each method produced an anomaly score $s(x)$, with higher scores indicating greater deviation from the benign reference distribution. Threshold-based predictions were computed by comparing each anomaly score against a decision threshold $\tau$, with a file predicted as anomalous when $s(x) \geq \tau$.

For the pooled baseline, cross-domain transfer, malicious modification, and hash-binned baseline experiments, thresholds were selected retrospectively from the labelled evaluation set for each model and evaluation condition. Specifically, the threshold was chosen from the ROC curve as the point that maximized Youden's $J$ statistic:

$$\tau^* = \arg\max_{\tau} \left(\mathrm{TPR}(\tau) - \mathrm{FPR}(\tau)\right).$$

This procedure was used to compare model capacity under a common retrospective operating point, rather than to represent a deployable OTA threshold-selection method. In a deployed setting, the threshold would need to be selected using benign calibration data, validation updates, or an operational risk policy.

For the Reptile-based experiments, including the support-size, feature-ablation, and hash-binned Reptile evaluations, thresholds were selected without labelled anomaly data. After target-domain adaptation, reconstruction scores were computed on a benign calibration subset, and the anomaly threshold was set to the 95th percentile of those benign calibration scores. The SmallAE support-size baseline used the same benign calibration procedure as Reptile, allowing the two methods to be compared under the same support, calibration, and

test structure.

The main threshold-based metrics were precision, recall, and F1-score:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}, \qquad \text{F1} = \frac{2TP}{2TP + FP + FN}. \tag{5.1}$$

Precision reflects how often files flagged as anomalous were truly anomalous, while recall measures how many anomalous files were successfully detected. The F1-score summarizes the trade-off between these two quantities.

To evaluate score separability independently of a single threshold, OTA-Sentinel also reports ROC-AUC and PR-AUC. The receiver operating characteristic (ROC) curve plots the true positive rate against the false positive rate across thresholds, where

$$\text{TPR} = \frac{TP}{TP + FN}, \qquad \text{FPR} = \frac{FP}{FP + TN}. \tag{5.2}$$

ROC-AUC summarizes how well anomalous files are ranked above benign files across the full threshold range.

The precision-recall curve plots precision against recall across thresholds. Its area, PR-AUC, summarizes the quality of anomaly ranking with respect to the positive class:

$$\text{PR-AUC} = \int_0^1 P(R)\, dR, \tag{5.3}$$

where $P(R)$ denotes precision as a function of recall. PR-AUC is especially useful in the low-contamination settings because the anomalous class is rare, making precision sensitive to false positives.

Together, these measures provide complementary views of detection quality. Precision, recall, and F1-score characterize thresholded detection performance, while ROC-AUC and PR-AUC measure score separability across thresholds. ROC-AUC captures overall ranking

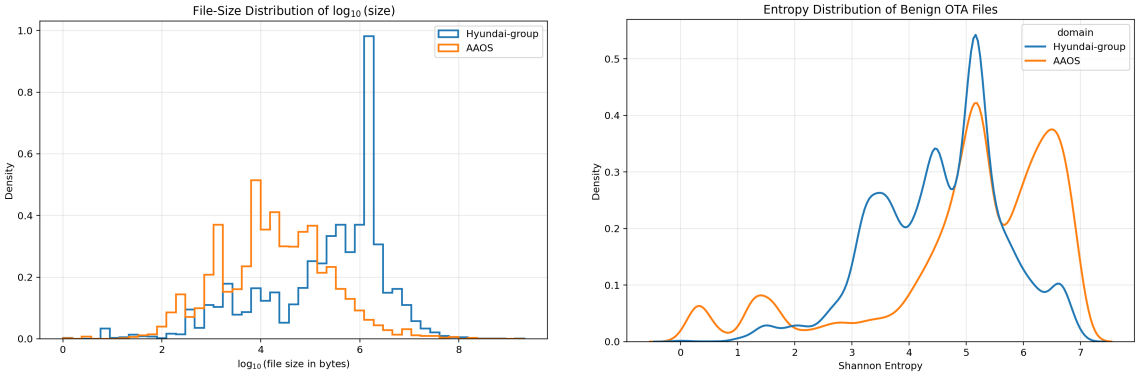
between benign and anomalous files, whereas PR-AUC provides a more positive-class-focused view under class imbalance.

## 5.3 Evaluation Datasets and Anomaly Sources

The dataset construction process is described in Chapter 4. This section summarizes how those datasets are used in the experimental evaluation. The benign evaluation data consist of two OTA domains: Hyundai-group Navigation Updater files and AAOS-derived update files generated from AOSP automotive build targets. These domains are used to test whether OTA-Sentinel can model benign OTA file structure across both vendor IVI update content and Android-based automotive update content.

Anomalous inputs are drawn from three sources. First, raw VirusShare-derived Linux/ELF and Android/APK malware samples are used to evaluate separation between benign OTA files and external malicious files. Second, salted malicious modifications are used to test whether dispersed malware-derived byte content can be detected when inserted into otherwise benign OTA files. Third, patched malicious modifications are used to test whether localized malicious insertions can be detected when most of the original benign file structure remains intact.

The experiments use these datasets in different ways. The pooled baseline experiments train on mixed benign OTA data and evaluate against Hyundai-group, AAOS, and combined benign test settings under varying contamination levels. The cross-domain experiments train on one benign OTA domain and evaluate on another to measure domain transfer. The malicious modification experiments evaluate whether the same feature space remains useful when malicious content is embedded into benign files. The Reptile experiments use domain-specific benign support, calibration, and test splits to evaluate adaptation from limited benign data.




(a) Log-scaled file-size distribution

(b) Comparison of entropy distribution

Figure 5.1: Descriptive comparison of the benign Hyundai-group and AAOS OTA domains using file-size and entropy distributions.

For the pooled baseline contamination experiments, testing was performed under three benign evaluation settings: Hyundai-group OTA files, AAOS OTA files, and a combined set containing both. The benign Hyundai-group test pool contained 5,600 files, while the benign AAOS test pool contained 19,441 files, for a combined benign total of 25,041 files. For each setting, malicious files were added at nominal contamination levels of 1%, 5%, 10%, 20%, 50%, and 100% relative to the benign test pool. This produced 56, 280, 560, 1,120, 2,800, and 5,600 anomalous files in the Hyundai-group setting; 194, 972, 1,944, 3,888, 9,720, and 18,729 anomalous files in the AAOS setting; and 250, 1,252, 2,504, 5,008, 12,520, and 25,041 anomalous files in the combined setting. The slight shortfall at the highest nominal contamination level in the AAOS case reflects the number of available Android malware samples rather than a change in the evaluation procedure.

## 5.4 Baseline Evaluation Under Mixed-OTA Training

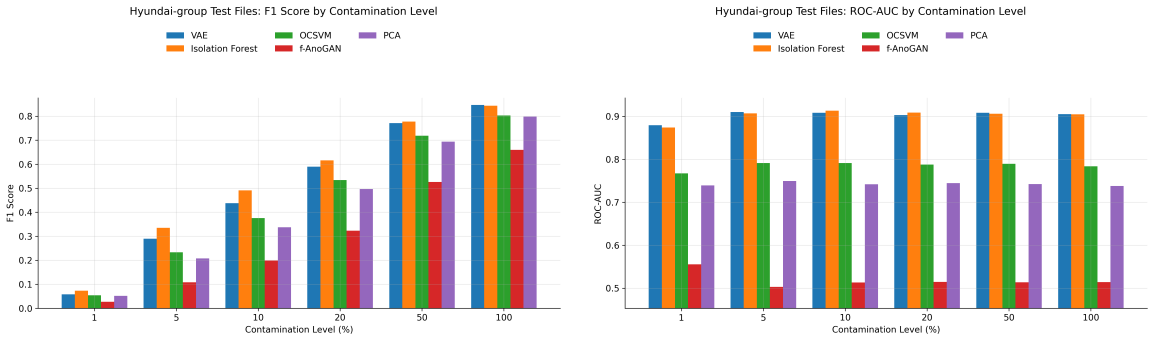
### 5.4.1 Comparison of Benign OTA Domains

Five unsupervised baseline methods were evaluated under a shared mixed-OTA training setting: PCA, Isolation Forest, OCSVM, f-AnoGAN, and VAE. Each model was trained

using benign files drawn from both Hyundai-group and AAOS update sources. This setup exposed the baselines to a single benign reference distribution spanning both Linux-based and AAOS-based OTA content.

Figure 5.1 shows clear differences between the two benign OTA domains. Although the total benign data volume was similar, the AAOS dataset contained substantially more files, indicating that its benign distribution was composed of many smaller files. This is reflected in both the mean and median file sizes, which were markedly lower for AAOS than for the Hyundai-group domain. AAOS files also exhibited higher entropy and greater local entropy variation overall, suggesting a broader range of byte-level structure. These differences indicate that the two domains do not represent interchangeable benign distributions and provide context for the domain-dependent performance differences reported later in the chapter.

### 5.4.2 Results on Hyundai-Group Test Files




(a) Comparison of Baseline F1 Scores for the Hyundai-Group Dataset

(b) Comparison of AUC Scores for the Hyundai-Group Dataset

Figure 5.2: Baseline results on Hyundai-group test files.

Figures 5.2 and 5.3 summarize baseline performance on Hyundai-group test files. The Hyundai-group setting was the most difficult of the three baseline test conditions. Among the five baselines, Isolation Forest and VAE gave the strongest overall results. Their ROC-AUC values remained close to 0.90 across the full contamination range, with Isolation

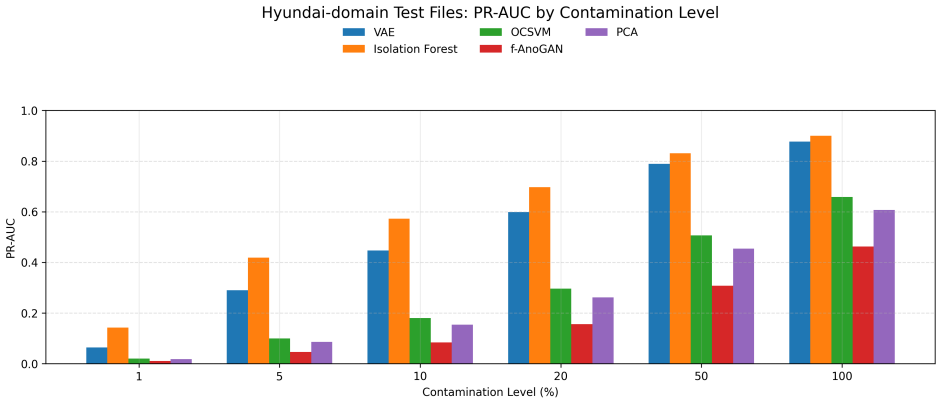
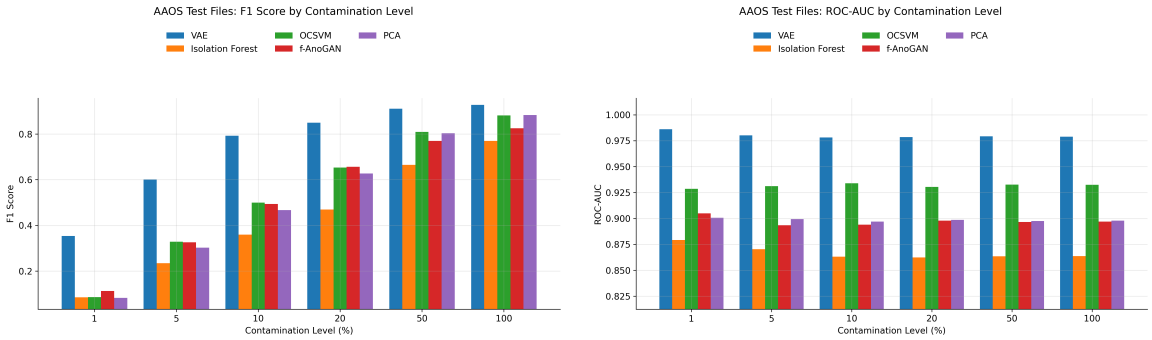



Figure 5.3: PR-AUC results for the 298-D baseline models on Hyundai-group test files across contamination levels.

Forest ranging from 0.874 to 0.913 and VAE ranging from 0.879 to 0.910. The PR-AUC results in Figure 5.3 provide an additional view of this low-contamination difficulty, where class imbalance made precision harder to maintain. OCSVM and PCA formed a lower middle tier, while f-AnoGAN performed substantially worse, with ROC-AUC values near chance level in this setting.

Although Isolation Forest and VAE provided the strongest score separation on Hyundai-group files, thresholded performance at low contamination remained limited. At 1% contamination, all methods produced low F1 scores, with the best result coming from Isolation Forest at 0.073. This indicates that the detectors were not yet reliable as direct binary decision tools when anomalous files were rare, because even a modest false-positive rate translated into a large number of benign files being flagged. The VAE results illustrate this clearly: at 1% contamination, recall reached 0.954, which means that nearly all anomalous files were retrieved, but precision was only 0.030. Given 56 anomalous files and 5,600 benign files in this setting, this corresponds to roughly 53 true positives but more than 1,700 benign files also being flagged. In practical terms, this operating point would be too noisy for automatic rejection, but it could still provide some value as a triage mechanism, since the relatively strong ROC-AUC values indicate that the models were still ranking anomalous files above benign files reasonably well. The main difficulty at 1% contamination was




(a) Comparison of Baseline F1 Scores for the AAOS Dataset

(b) Comparison of AUC Scores for the AAOS Dataset

Figure 5.4: Baseline results on AAOS test files.

therefore not complete failure of score separation, but poor precision under extreme class imbalance.

As the contamination level increased, F1 score improved steadily for all methods. At 100% contamination, VAE achieved the highest F1 score at 0.847, followed closely by Isolation Forest at 0.844. OCSVM and PCA reached 0.803 and 0.798, respectively, while f-AnoGAN remained lower at 0.660. Overall, the Hyundai-group results indicate that mixed-OTA baseline training can provide useful anomaly separation for Linux-based OTA files, but this setting remains comparatively difficult, particularly when anomalous files are rare.

### 5.4.3 Results on AAOS Test Files

Figures 5.4 and 5.5 summarize baseline performance on AAOS test files. The AAOS setting produced substantially stronger baseline performance than the Hyundai-group setting. VAE was the strongest method across all contamination levels. Its ROC-AUC remained between 0.978 and 0.986, and its F1 score increased from 0.354 at 1% contamination to 0.927 at 100%. The PR-AUC results in Figure 5.5 further support this result by showing stronger ranking performance under class imbalance than in the Hyundai-group setting. In contrast to the Hyundai-group case, VAE maintained both strong recall and substantially better precision on AAOS files, leading to the best thresholded performance as well as the best

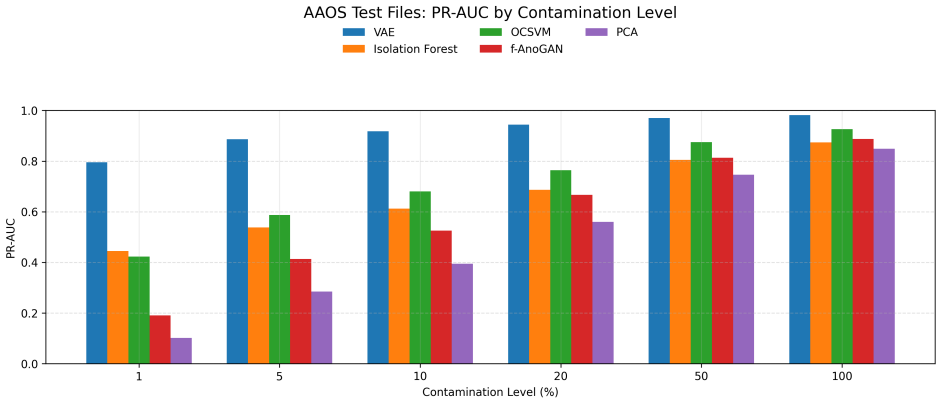



Figure 5.5: PR-AUC results for the 298-D baseline models on AAOS test files across contamination levels.

threshold-free separation.

OCSVM and PCA formed the next strongest group on AAOS. OCSVM achieved ROC-AUC values between 0.929 and 0.934, while PCA remained near 0.90 across all contamination levels. Their F1 scores at 100% contamination were 0.881 and 0.883, respectively. Both methods showed consistently high recall, with PCA in particular exceeding 0.96 recall in every AAOS run. f-AnoGAN was more competitive on AAOS than on Hyundai-group files and reached an F1 score of 0.825 at 100% contamination, but it still remained below VAE. Isolation Forest gave the weakest overall AAOS results, with ROC-AUC values between 0.862 and 0.879 and an F1 score of 0.770 at 100% contamination.

These results indicate that, under the pooled mixed-OTA training setting, baseline performance was consistently stronger on AAOS update content than on Hyundai-group OTA content.

### 5.4.4 Effect of Contamination Level

A consistent trend was observed across all three test settings as the contamination level increased. For each model, ROC-AUC remained relatively stable across the contamination range, whereas F1 score improved substantially. This indicates that the relative ordering of

benign and malicious samples was often reasonably consistent, but the final thresholded results were strongly affected by class imbalance. At low contamination levels, even modest false-positive counts were sufficient to reduce precision sharply, which in turn depressed F1 score.

The same pattern was visible in the combined test setting. VAE again produced the strongest overall performance, with ROC-AUC values between 0.957 and 0.964 and F1 scores increasing from 0.187 at 1% contamination to 0.889 at 100%. OCSVM and PCA followed and ended with F1 scores of 0.857 and 0.856, respectively, at the highest contamination level. Isolation Forest remained competitive but weaker overall, while f-AnoGAN consistently trailed the other baselines in the combined setting.

Taken together, the baseline results show two main points. First, mixed-OTA training provides a workable reference setting for anomaly detection across both Linux-based and AAOS-based OTA files. Second, baseline effectiveness depends strongly on the test domain. AAOS files were detected more reliably by all methods, whereas Hyundai-group OTA files remained the more challenging case, especially under low contamination. This difference is important for interpreting the advantages of the domain-adaptive Reptile setting in the following section.

These results directly address RQ1 and RQ2: they show that the proposed static byte-level representation can provide useful anomaly-separation signal, and they identify VAE, PCA, and OCSVM as the strongest pooled baseline methods under the evaluated settings.

## 5.5 Cross-Domain Transfer Evaluation

### 5.5.1 Evaluation Setting

For the cross-domain transfer experiments, each baseline anomaly-detection model was trained separately on one benign OTA domain and then evaluated on both same-domain and

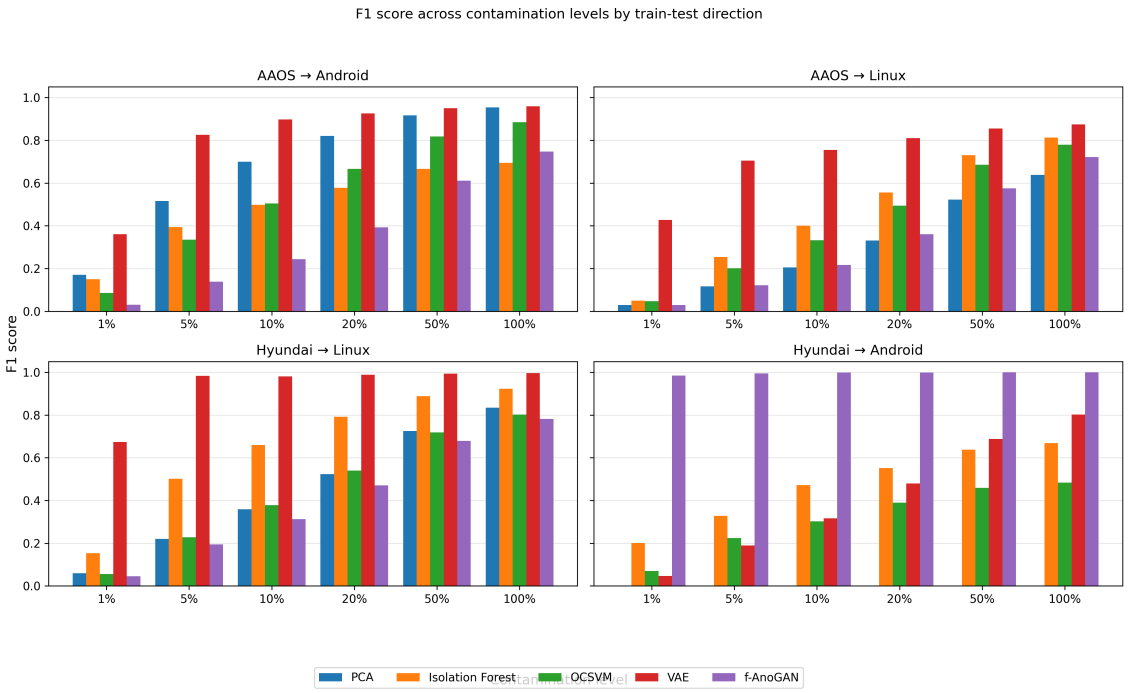



Figure 5.6: F1 score across contamination levels for the cross-domain transfer evaluation.

cross-domain test settings. The evaluated models were PCA, Isolation Forest, OCSVM, f-AnoGAN, and VAE. Each model was trained once using AAOS-derived benign OTA files and once using Hyundai-group benign OTA files, producing separate AAOS-trained and Hyundai-trained versions of each detector.

Same-domain evaluation refers to AAOS-trained models evaluated on Android/AAOS test data and Hyundai-trained models evaluated on Linux/Hyundai-group test data. Cross-domain evaluation refers to AAOS-trained models evaluated on Hyundai-group/Linux test data and Hyundai-trained models evaluated on Android/AAOS test data. This setup tests whether a benign profile learned from one OTA domain transfers to another domain with different file-size, entropy, and byte-distribution characteristics.

### 5.5.2 Cross-Domain Results

Figures 5.6 and 5.7 summarize the direct cross-domain transfer results. Figure 5.6 shows the thresholded F1 performance across contamination levels for each train-test direction,

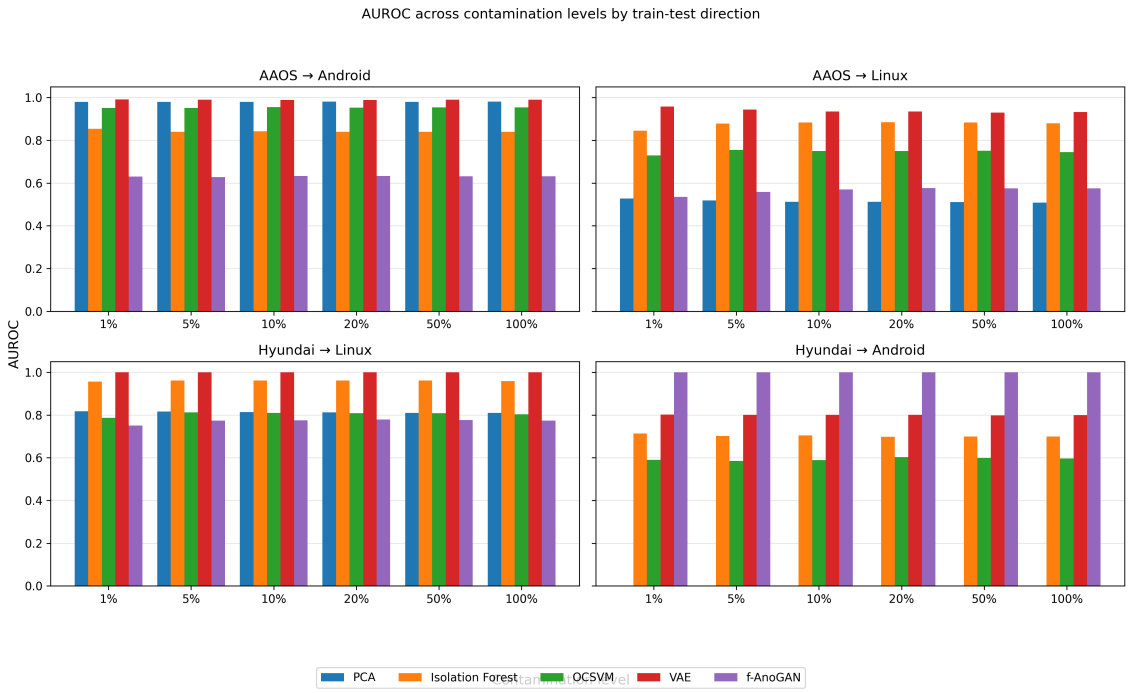



Figure 5.7: AUROC across contamination levels for the cross-domain transfer evaluation.

while Figure 5.7 shows the corresponding score-separation performance using AUROC. Together, the two figures show that same-domain evaluation was generally stronger than cross-domain evaluation. In particular, AAOS-trained models evaluated on Android/AAOS test data and Hyundai-trained models evaluated on Linux/Hyundai-group test data usually outperformed models evaluated across domains.

The F1 results in Figure 5.6 increased with contamination level in most settings, which is consistent with the class-imbalance effects observed in the pooled baseline experiments. At low contamination levels, false positives had a larger effect on precision and therefore reduced F1. The AUROC results in Figure 5.7 were comparatively more stable across contamination levels, indicating that several models still preserved useful ranking separation even when thresholded F1 was lower. Among the evaluated models, VAE gave the most consistent overall transfer behaviour, with strong AUROC in the AAOS-to-Android, AAOS-to-Linux, and Hyundai-to-Linux settings. PCA was especially sensitive to domain shift, with weak transfer in the Hyundai-to-Android direction. Isolation Forest and

OCSVM showed moderate but less consistent transfer. The Hyundai-to-Android f-AnoGAN result was unusually strong, reaching near-perfect AUROC and very high F1 across contamination levels.

### 5.5.3 Interpretation

The gap between F1 and AUROC is important when interpreting these results. AUROC measures score separability across thresholds, while F1 is computed after applying a single decision threshold. At low contamination levels, even a small number of false positives can sharply reduce precision and F1. Therefore, high AUROC with lower F1 does not necessarily indicate a contradiction; it suggests that the model ranked anomalous files above benign files reasonably well, but thresholded detection remained difficult when anomalies were rare.

Overall, the cross-domain results show that generalization across OTA domains is possible but not uniform. Transfer was asymmetric: the AAOS-trained VAE transferred more effectively to Linux test sets than the Hyundai-trained VAE transferred to Android test sets. This suggests that the benign structure learned from AAOS-derived files was more transferable under the evaluated feature representation than the reverse direction. The unusually strong Hyundai-to-Android f-AnoGAN result should be treated cautiously because the same model was much weaker in the other train-test directions. Taken together, these results support the need for domain-specific calibration or adaptation rather than relying only on source-domain training.

These results address RQ3 by showing that cross-domain generalization is possible but asymmetric, with performance depending on the train-test direction and on the OTA domain used for training.

## 5.6 Malicious Modification Experiments

To examine whether the studied methods can detect malicious content hidden inside otherwise benign OTA files, this section evaluates two synthetic modification procedures applied to benign OTA test files: salted injection and patch-based injection. In the salted setting, malware-derived bytes were dispersed across the original benign file at a controlled rate, producing a modified sample with small malicious changes distributed throughout the file. In the patch-based setting, malware-derived bytes were inserted as a small number of localized contiguous regions, producing a more concentrated modification. Full construction details are provided in Chapter 4. In both cases, the starting point was an original benign OTA file, and the resulting modified file retained much of the original structure while containing injected malicious content. These modified files were treated as anomalous during evaluation, while the unmodified OTA files remained in the benign class.

The first procedure, referred to as salted injection, distributed malware-derived bytes throughout the benign file at controlled nominal rates. Three salt levels were evaluated: 0.1%, 5%, and 10%. This produced anomalous samples in which the benign OTA content remained dominant, but the byte-level statistics and local 3-gram patterns were altered by the embedded malicious content. These settings were intended to test whether the anomaly detectors remained sensitive when the modification was spread across the file rather than concentrated in a single location.

The second procedure, referred to as patch-based injection, inserted malware-derived content into localized contiguous regions of the benign file. In contrast to the salted setting, this produced anomalies that were more spatially concentrated within the file. Both patch-based conditions used a 0.1% total byte-injection budget with individual patches capped at 64 KB, while differing in the malware source used to construct the patch bytes. The All Malware patch condition used the combined malware source, whereas the Linux Malware patch con-

| Category | Test set | Benign | Malicious | Malicious:Benign | Malware (%) |
|---|---|---|---|---|---|
| Raw | All Malware | 25,041 | 25,041 | 25,041:25,041 | 50.00 |
| Raw | Android Malware | 25,041 | 18,729 | 18,729:25,041 | 42.79 |
| Raw | Linux Malware | 25,041 | 13,698 | 13,698:25,041 | 35.36 |
| Salted | All Malware 0.1% | 29,223 | 43,414 | 43,414:29,223 | 59.77 |
| Salted | All Malware 5% | 29,223 | 43,414 | 43,414:29,223 | 59.77 |
| Salted | All Malware 10% | 29,223 | 43,414 | 43,414:29,223 | 59.77 |
| Salted | Linux Salt 0.1% | 29,223 | 24,685 | 24,685:29,223 | 45.79 |
| Salted | Linux Salt 5% | 29,223 | 24,685 | 24,685:29,223 | 45.79 |
| Salted | Linux Salt 10% | 29,223 | 24,685 | 24,685:29,223 | 45.79 |
| Patch | All Malware Patch 0.1% | 29,223 | 12,342 | 12,342:29,223 | 29.69 |
| Patch | Linux Malware Patch 0.1% | 29,223 | 6,171 | 6,171:29,223 | 17.44 |

Table 5.1: Benign and malicious sample counts for each test set used in the malicious modification experiments. The raw rows use the corrected 30% combined benign test pool, while the salted and patch-based rows use the separate modified-file evaluation pool which allowed more files to avoid initial removal.

dition used Linux malware only. This distinction is important because a detector that relies mainly on global statistical shifts may respond differently to dispersed byte changes than to a smaller number of concentrated insertions. The ratios in Table 5.1 describe the class composition of each constructed test set, not the amount of malicious content inserted into an individual file. For the salted conditions, the per-file injection levels are the nominal byte-injection rates of 0.1%, 5%, and 10%.

For comparison, this section also reports raw malware conditions in which the anomalous samples are standalone malicious files rather than modified OTA files. These raw settings are not part of the synthetic injection procedure itself. Instead, they serve as a reference point for interpreting the injected cases. In the raw setting, the detector distinguishes benign OTA files from fully malicious inputs, whereas in the salted and patch-based settings it must detect malicious content embedded within an otherwise benign OTA file. The injected conditions are therefore more structurally similar to the benign class and represent a harder test of whether the models remain sensitive once the original OTA file structure is largely preserved.

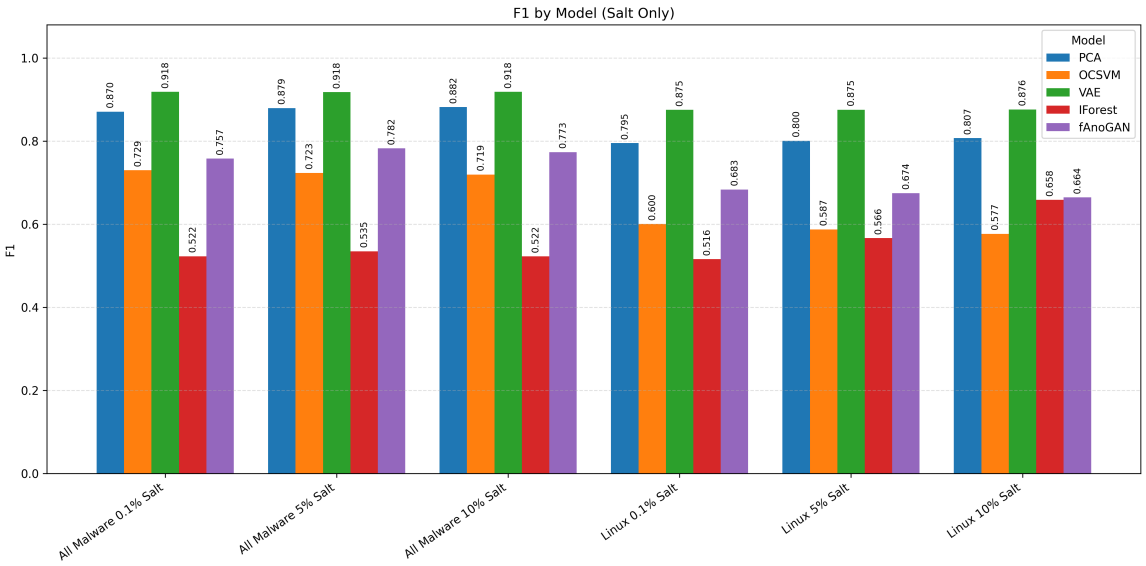



Figure 5.8: Comparison of F1 Scores for Salted Datasets

### 5.6.1 Results on Salted Injection Conditions

Figures 5.8 and 5.9 summarize F1 and ROC-AUC performance under the salted injection conditions. These experiments evaluate whether the baseline methods remain effective when malicious content is dispersed throughout an otherwise benign OTA file. Across these experiments, VAE gave the strongest overall results. On the "All Malware" salted sets, its F1 score remained essentially unchanged across the three salt levels, with values of approximately 0.918, 0.918, and 0.918 at 0.1%, 5%, and 10%, respectively. Its ROC-AUC values were similarly stable, remaining between about 0.896 and 0.898. PCA formed the next strongest group, with F1 scores ranging from about 0.870 to 0.882 and ROC-AUC remaining near 0.863 to 0.864 across the same conditions.

The remaining methods were less consistent. OCSVM produced moderate salted-detection performance on the "All Malware" sets, with F1 scores between about 0.719 and 0.729, while f-AnoGAN remained somewhat higher, between about 0.757 and 0.782. Isolation Forest was the weakest method under these conditions, with F1 scores near 0.52 to 0.53 despite precision above 0.92 in each run. This indicates that its main limitation in the salted

setting was recall rather than false positives: it was conservative, but missed a substantial portion of the modified files.

A similar ranking was observed on the Linux-only salted conditions, although these runs were generally more difficult in thresholded performance than the corresponding "All Malware" salted sets. VAE again gave the strongest results, with F1 values between about 0.875 and 0.876 and ROC-AUC between about 0.902 and 0.906. PCA remained second, with F1 scores between about 0.795 and 0.807. Both methods therefore retained strong detection performance even when the injected malicious source was restricted to Linux malware. OCSVM, f-AnoGAN, and Isolation Forest were weaker in this setting. OCSVM dropped to F1 values between about 0.577 and 0.600, while f-AnoGAN remained in the mid-0.66 to low-0.68 range. Isolation Forest was the most variable of the five methods, ranging from about 0.516 at 0.1% salt to about 0.658 at 10% salt.

One notable pattern is that the nominal salt ratio had only a limited effect on the stronger methods. For both "All Malware" and Linux-only salted conditions, VAE changed very little as the injection rate increased from 0.1% to 10%. PCA also remained highly stable, with only small improvements at the higher salt levels. This suggests that, once malicious content was distributed throughout the file, even the lowest tested salt ratio was often sufficient to shift the file representation away from the benign reference distribution in a way that these reconstruction-based methods could detect reliably. At the same time, malware source still mattered. For most models, the Linux-only salted conditions gave lower F1 scores than the corresponding "All Malware" salted conditions. This can be seen in PCA, where F1 decreased from roughly 0.87–0.88 on the "All Malware" salted sets to about 0.80 on the Linux-only salted sets, and in OCSVM, which dropped from roughly 0.72–0.73 to about 0.58–0.60. Even VAE, although still the strongest method overall, showed lower thresholded performance on Linux-only salt because its precision decreased relative to the "All Malware" salted runs.

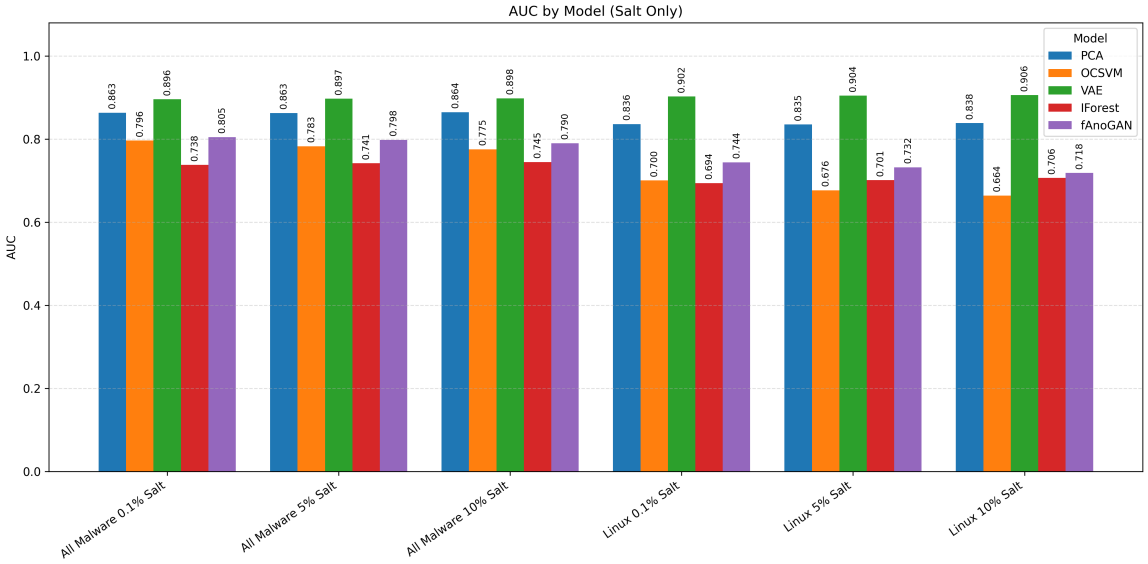



Figure 5.9: Comparison of AUC Scores for Salted Datasets

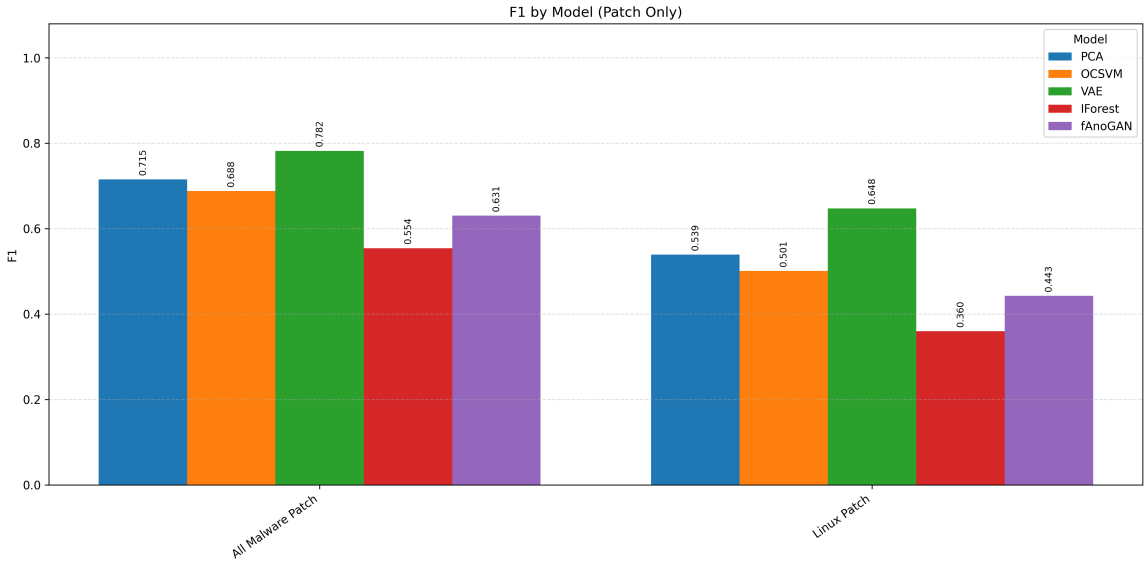



Figure 5.10: Comparison of F1 Scores for Patched Datasets

Overall, the salted injection results show that dispersed malicious content remained detectable for the stronger methods in this setting, especially VAE and PCA, with malware source type playing a larger role than the nominal salt percentage.

### 5.6.2 Results on Patch-Based Injection Conditions

Figures 5.10 and 5.11 summarize F1 and ROC-AUC performance under the patch-based injection conditions. The patch-based injection conditions were more difficult than the corresponding salted settings for all five baseline methods. In this setting, the malicious content was inserted into localized contiguous regions rather than being distributed across the full file, which reduced the global statistical shift seen by the detector. As a result, both threshold-free separation and thresholded detection performance declined relative to the salted conditions, particularly on the Linux-only patch set.

VAE gave the strongest overall performance under patch-based injection. On the "All Malware" patch condition, it achieved an F1 score of approximately 0.782 with ROC-AUC near 0.895, clearly exceeding the other baselines. PCA formed the next strongest group, reaching about 0.715 F1 with ROC-AUC near 0.867. OCSVM followed at about 0.688 F1, while f-AnoGAN and Isolation Forest were lower at roughly 0.631 and 0.554, respectively. These results show that the reconstruction-based methods remained the most effective even when the malicious content was concentrated into smaller regions of the file.

The Linux-only patch condition was substantially harder. VAE again produced the strongest result, with F1 near 0.648, and PCA remained second at about 0.539. OCSVM dropped to roughly 0.501, f-AnoGAN to about 0.443, and Isolation Forest to about 0.360. Although VAE still maintained the highest F1 score, this condition produced a clear reduction in thresholded performance across all methods when compared with the "All Malware" patch set. The same ordering was also reflected broadly in ROC-AUC, where VAE and PCA remained strongest, while OCSVM, f-AnoGAN, and Isolation Forest showed weaker separation.

A clear pattern in the patch results is the gap between recall and precision. VAE and PCA retained very high recall in both patch conditions, exceeding roughly 0.89 recall in all four

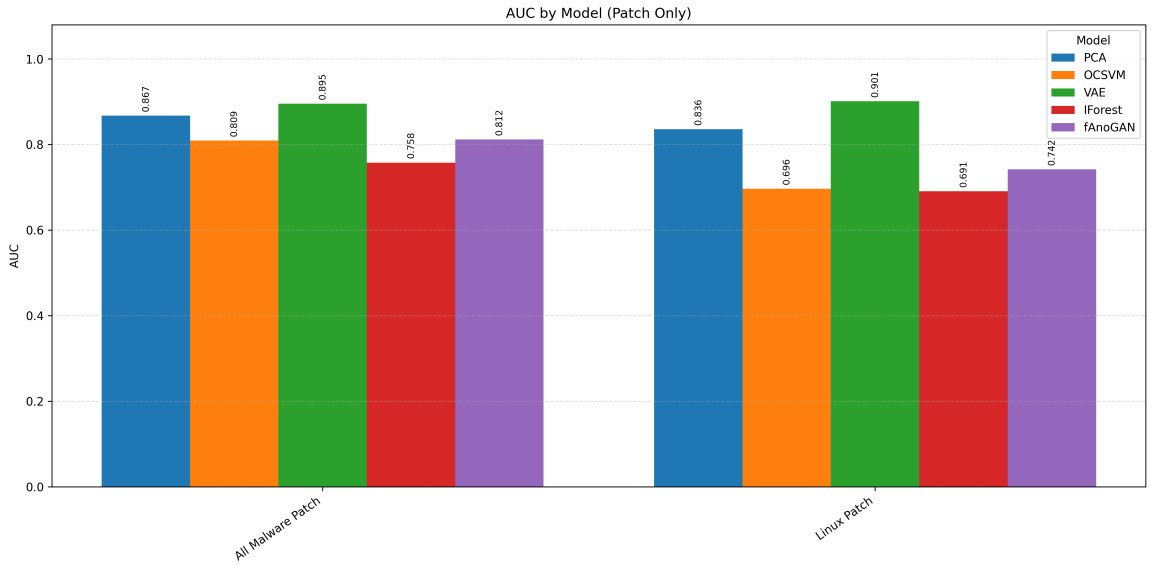



Figure 5.11: Comparison of AUC Scores for Patched Datasets

runs and reaching about 0.98 recall for VAE. However, their precision remained more limited, especially on the Linux-only patch condition. This indicates that the stronger methods were still able to rank most patched files above benign files, but the final thresholded decisions were affected by the increased overlap between benign and anomalous samples. OCSVM showed the opposite trade-off on the "All Malware" patch set, where precision remained relatively high at about 0.755 but recall fell to about 0.632, indicating a more conservative detector that missed a larger portion of the patched files.

Overall, the patch-based results show that localized malicious insertion is harder to detect than dispersed salted injection. Even so, the stronger baselines, particularly VAE and PCA, still retained useful detection capability. The drop from the "All Malware" patch condition to the Linux-only patch condition also suggests that malware source remained important in the patch setting, just as it did under salted injection. In this section, the effect of injection method appears more pronounced than the exact nominal salt ratio considered earlier, since concentrating the malicious content into localized regions reduced detector performance much more clearly than changing the salt percentage from 0.1% to 10%.

### 5.6.3 Effect of Malware Source and Injection Method

The modified-condition results show that detection performance depended more strongly on how malicious content was inserted than on the nominal salt percentage. Across the evaluated models, salted injection was generally easier to detect than patch-based injection. When malicious bytes were distributed throughout the file, the resulting feature shifts were more pronounced. When the same type of content was inserted into localized regions, the modified files retained more of the structure of the original OTA samples and were therefore harder to separate from the benign set.

Malware source also affected performance. For most models, the "All Malware" conditions were easier than the corresponding Linux-only conditions under both salted and patch-based evaluation. This suggests that some injected sources produced feature patterns that were closer to those of benign OTA files under the static representation used in this work.

By contrast, the exact salt percentage had a smaller effect. For the stronger methods, especially VAE and PCA, performance changed only slightly between the 0.1%, 5%, and 10% salted conditions. Overall, these results show that the studied methods remain effective under several forms of malicious modification, but that detection becomes more difficult as the altered file more closely preserves the characteristics of benign OTA content.

These experiments further support RQ1 and RQ2 by showing that malicious content can remain detectable even when embedded into benign OTA files, while also showing that detection difficulty depends on the injection method and malware source.

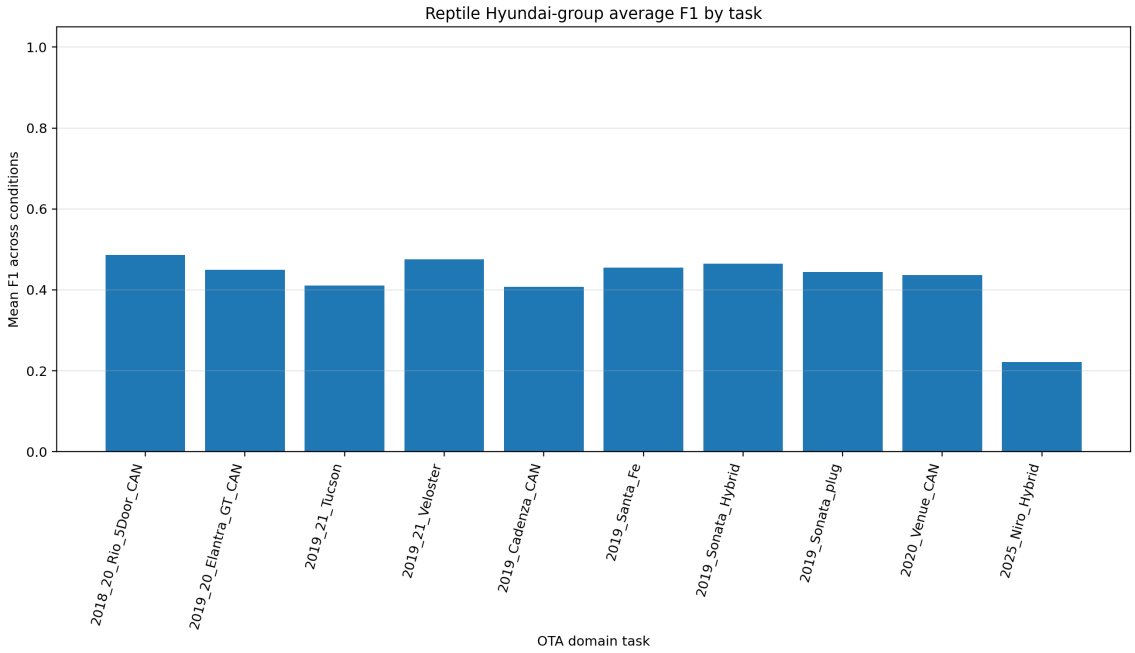



Figure 5.12: Comparison of Reptile F1 Scores for the Hyundai-Group Domains

## 5.7 Meta-Learning-Based Domain Evaluation Using Reptile

This section evaluates the meta-learning component of OTA-Sentinel with respect to RQ4, which asks to what extent meta-learning supports adaptation to individual OTA domains when only limited benign support data are available. Reptile is adopted as the meta-learning technique for this component. In contrast to the pooled baseline models, which learn a single benign profile across mixed OTA data, the Reptile-based approach learns an initialization that can be adapted to a specific OTA family using a small benign support set. The following experiments evaluate whether this adaptation strategy improves file-level anomaly detection across Hyundai-group and AAOS OTA domains.

### 5.7.1 Results on Hyundai-Group OTA Domains

The Hyundai-group experiment addresses RQ4 by testing whether the Reptile meta-learning component can adapt to individual vendor OTA domains using limited benign support data. Each Hyundai-group OTA family was treated as a separate evaluation task, with benign support files used for adaptation, benign calibration files used for threshold selection, and held-out benign files plus anomalous inputs used for testing.

Figure 5.12 summarizes Reptile F1 performance across the Hyundai-group OTA domains. On these domains, Reptile produced moderate and task-dependent detection performance across the evaluated anomaly conditions. Averaged over all conditions, the strongest Hyundai-group tasks were 2018_20_Rio_5Door_CAN ($F1 \approx 0.487$), 2019_21_Veloster ($F1 \approx 0.476$), and 2019_Sonata_Hybrid ($F1 \approx 0.465$). Several other tasks remained somewhat lower, including 2019_21_Tucson ($F1 \approx 0.411$) and 2019_Cadenza_CAN ($F1 \approx 0.408$). The most difficult Hyundai-group task was 2024_Niro_Hybrid CAN, whose average performance across conditions fell to about $F1 \approx 0.221$. These results show that Reptile can adapt to Hyundai-group benign support data, but the quality of that adaptation is not uniform across OTA domains.

A clear pattern in the Hyundai-group results is the gap between precision and recall. For most tasks, precision remained high, often in the range of roughly 0.87 to 0.94, while recall was much lower, typically near 0.26 to 0.34. This indicates that when the adapted model labelled a file as anomalous, it was usually correct, but a substantial portion of anomalous files still went undetected. In practical terms, the Hyundai-group domains were affected more by missed detections than by excessive false positives. Average AUROC values followed the same general pattern, remaining moderate for most tasks, usually around 0.56 to 0.62, with 2025_Niro_Hybrid again showing the weakest separation at about 0.469.

The anomaly source also influenced performance. The raw Linux malware condition was

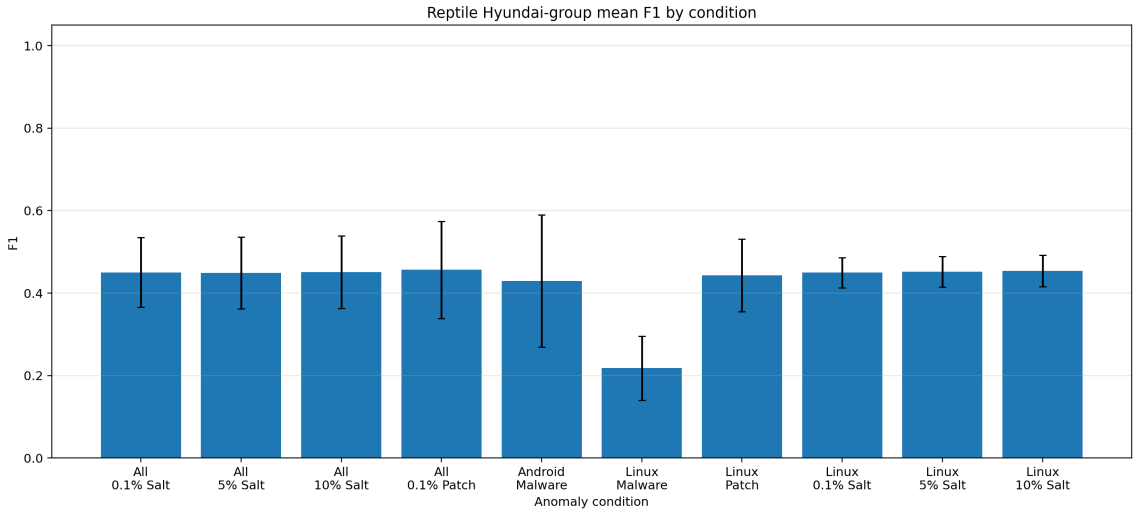



Figure 5.13: Average F1 Scores Across Different Perturbation Levels For Hyundai-Group Domains

the hardest setting for the Hyundai-group domains, yielding the lowest mean domain-level performance overall ($F1 \approx 0.218$ across Hyundai-group tasks). The raw Android malware condition was stronger, but still below most of the mixed or injected settings ($F1 \approx 0.429$ across Hyundai-group tasks). By comparison, the mixed Linux-and-Android malware conditions and the injected conditions produced Hyundai-group means near $F1 \approx 0.44$ to 0.45. This indicates that Hyundai-group tasks were better matched to the mixed and injected anomaly settings than to the raw malware conditions, although the improvement was modest rather than large.

Another notable observation is that the salt ratio had little effect on aggregate Hyundai-group performance as shown in Figure 5.13. The 0.1%, 5%, and 10% salt conditions remained very close to one another, both for the mixed Linux-and-Android malware settings and for the injected settings. This suggests that, within the Hyundai-group domains, performance was shaped more by the general relationship between the benign support set and the anomaly source than by the exact contamination level. The patched conditions followed the same overall trend. They were generally more favorable than the raw malware settings, but they did not produce a major jump in detection quality.

At the task level, the Hyundai-group domains were not uniformly difficult. Tasks such as 2018_20_Rio_5Door_CAN, 2019_21_Veloster, and 2019_Sonata_Hybrid remained relatively stable across conditions and consistently achieved the strongest results within this group. In contrast, 2025_Niro_Hybrid behaved as a clear failure case under raw malware evaluation, dropping to nearly zero F1 under both raw Android and raw Linux malware before recovering only partially in the injected settings. Overall, these results show that Reptile can adapt effectively to some Hyundai-group OTA domains, but that success depends strongly on the structure of the target domain and on the anomaly condition being evaluated.

With respect to RQ4, the Hyundai-group results provide mixed support for the benefit of meta-learning-based adaptation. Reptile was able to adapt to several Hyundai-group OTA domains from limited benign support data and achieved moderate detection performance on the stronger tasks, including 2018_20_Rio_5Door_CAN, 2019_21_Veloster, and 2019_Sonata_Hybrid. However, performance remained task-dependent, and weaker domains such as 2019_21_Tucson, 2019_Cadenza_CAN, and 2025_Niro_Hybrid indicate that meta-learning did not eliminate domain sensitivity. Therefore, the Hyundai-group evaluation suggests that Reptile can support domain-specific adaptation, but the benefit is not uniform across vendor OTA families.

### 5.7.2 Results on AAOS OTA Domains

The AAOS experiment also addresses RQ4 by testing whether the Reptile meta-learning component can adapt to Android-based automotive OTA domains using limited benign support data. As in the Hyundai-group setting, each AAOS build-target family was treated as a separate evaluation task. Benign support files were used for adaptation, benign calibration files were used for threshold selection, and held-out benign files plus anomalous inputs were used for testing. This experiment evaluates whether the meta-learned initialization remains useful when the target domains are drawn from AAOS-derived update contents

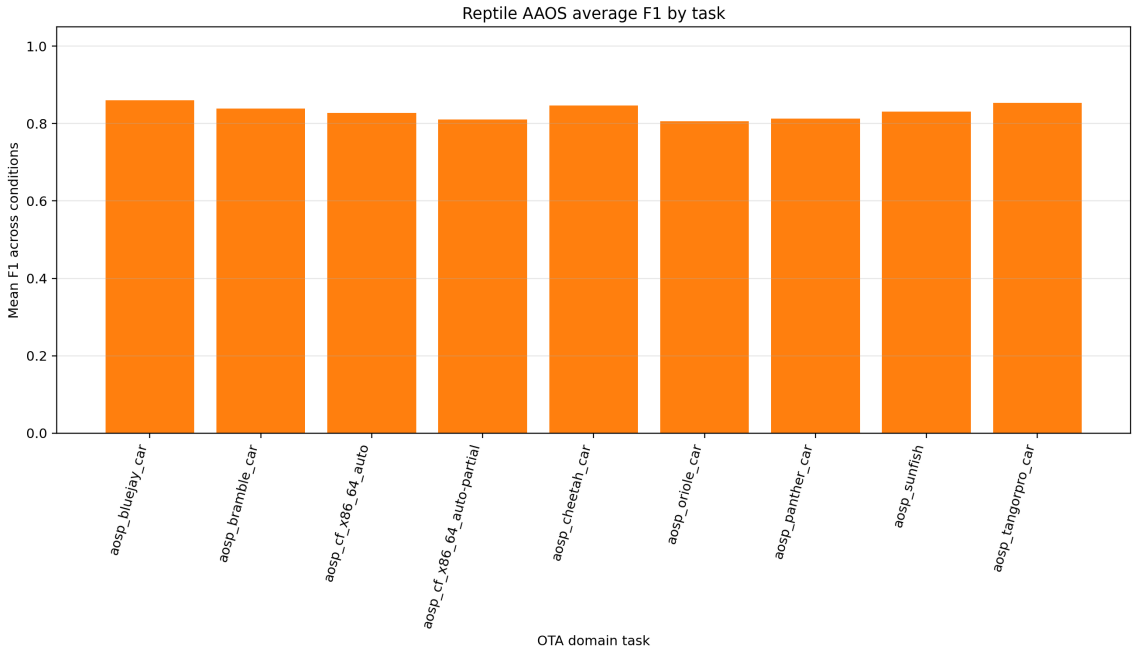



Figure 5.14: Comparison of Reptile F1 Scores for the AAOS Domains

rather than vendor Navigation Updater packages.

Figure 5.14 shows that, on AAOS OTA domains, Reptile produced consistently stronger detection performance than on the Hyundai-group domains. Across the AAOS tasks, mean F1 scores were generally high and remained more stable across anomaly conditions. Averaged over all evaluated settings, the strongest AAOS tasks included aosp_bluejay_car, aosp_tangorpro_car, aosp_ cheetah _car, and aosp_bramble_car, each of which remained in the low-to-mid 0.8 range on average across conditions. This indicates that, for many AAOS domains, the benign support data provided a sufficiently coherent domain structure for Reptile to adapt effectively.

A central pattern in the AAOS results is that performance remained relatively strong even when the anomaly source changed. The raw Android malware condition was the weakest AAOS setting, but still produced a moderate mean domain-level result of about $F1 \approx 0.673$. In contrast, the raw Linux malware condition was much stronger, with a mean AAOS-domain F1 of about 0.880. The mixed Linux-and-Android malware settings and

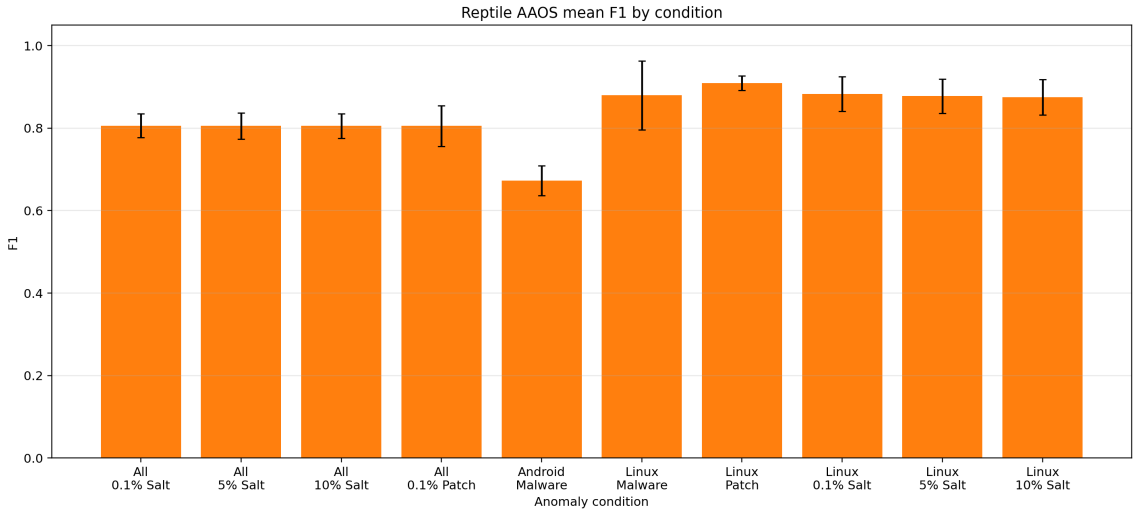



Figure 5.15: Average F1 Scores Across Different Perturbation Levels For AAOS-Group Domains

the injected settings were similarly strong, generally remaining between about $F1 \approx 0.81$ and 0.91 across the AAOS domains. These results show that the AAOS tasks were not only easier overall, but also less sensitive to the anomaly condition than the Hyundai-group tasks.

The strong AAOS performance under raw Linux malware is particularly notable. Unlike the Hyundai-group domains, which degraded sharply under raw Linux malware, the AAOS domains continued to separate benign and anomalous samples effectively. This suggests that the benign AAOS support sets were structurally distinct enough from the raw Linux malware files that the adapted model could detect them with relatively little ambiguity. The raw Android malware setting was more difficult, which is consistent with the closer relationship between Android malware and the AAOS software environment. Even so, AAOS performance under raw Android malware remained well above the corresponding Hyundai-group levels.

As with the Hyundai-group results, the contamination ratio itself had only a limited effect. The 0.1%, 5%, and 10% salt settings remained close to one another at the aggregate level, and the patched conditions followed the same general pattern as shown in Figure 5.15.

Performance was slightly higher in some combined or injected settings than in the raw settings, but these differences were modest compared with the larger difference between AAOS and Hyundai-group domain behaviour. This indicates that, within the AAOS domains, the dominant factor was again the relationship between the benign support set and the anomaly source, rather than the exact level of contamination.

At the task level, the AAOS domains were also more consistent than the Hyundai-group domains. While performance still varied across tasks, the variation was narrower and there was no AAOS task that collapsed in the same way as 2025_Niro_Hybrid in the Hyundai-group results. Instead, the stronger AAOS domains remained strong across most anomaly conditions, and even the weaker AAOS tasks generally preserved usable detection performance. This consistency suggests that the AAOS OTA domains form a more favorable setting for Reptile-based adaptation, at least under the feature representation and adaptation procedure used here.

With respect to RQ4, the AAOS results provide stronger support for meta-learning-based adaptation than the Hyundai-group results. Reptile achieved consistently higher F1 scores across several AAOS domains, including `aosp\_bluejay\_car`, `aosp\_tangorpro\_car`, and `aosp\_cheetah\_car`. These results suggest that the meta-learned initialization adapted more effectively to the AAOS-derived OTA families, likely because the AAOS data were more structurally consistent across build targets. However, performance still varied by anomaly source, with Android malware generally producing weaker results than Linux malware in some AAOS settings. Therefore, the AAOS evaluation suggests that Reptile can support limited-support adaptation more effectively in structured AAOS domains, but the benefit remains dependent on the target domain and anomaly condition.

### 5.7.3 AAOS OTA Structure and Expected File-Level Regularity

The stronger AAOS domain results are consistent with the AAOS update context and dataset construction described earlier in Sections 3.3 and 4.4. Because AAOS updates are produced within a more standardized Android-based OTA framework, they may expose more regular file-level patterns within each domain than the broader vendor-specific Linux-based OTA collections used in this thesis. In the present dataset, this interpretation is also supported by the composition of the benign AAOS tasks, which contained more numerous and generally smaller files. Taken together, these characteristics likely gave Reptile a denser and more internally consistent support set for adaptation, which helps explain why the AAOS domains were generally stronger and more stable than the Hyundai-group domains.

At the same time, the raw-malware results show that anomaly difficulty still depended on the relationship between the anomaly source and the target domain. Raw Android malware was the weaker condition for AAOS, whereas raw Linux malware was the weaker condition for the Hyundai-group tasks. This suggests that the advantage of the AAOS domains did not come from anomaly detection being uniformly easier, but rather from the benign AAOS domains providing a clearer and more consistent notion of normal behaviour at the file level. In this sense, the AAOS results are consistent with the platform structure discussed in the related-work chapter while still reflecting the domain-specific interaction between benign support data and anomaly type.

### 5.7.4 Support-Size Analysis and Comparison with a Simple Few-Shot Baseline

To examine how strongly Reptile depended on the amount of benign support data available at adaptation time, additional experiments were conducted using support sizes of 8, 16, 32, 64, and 128 samples. Reptile was then compared against a simpler few-shot baseline,

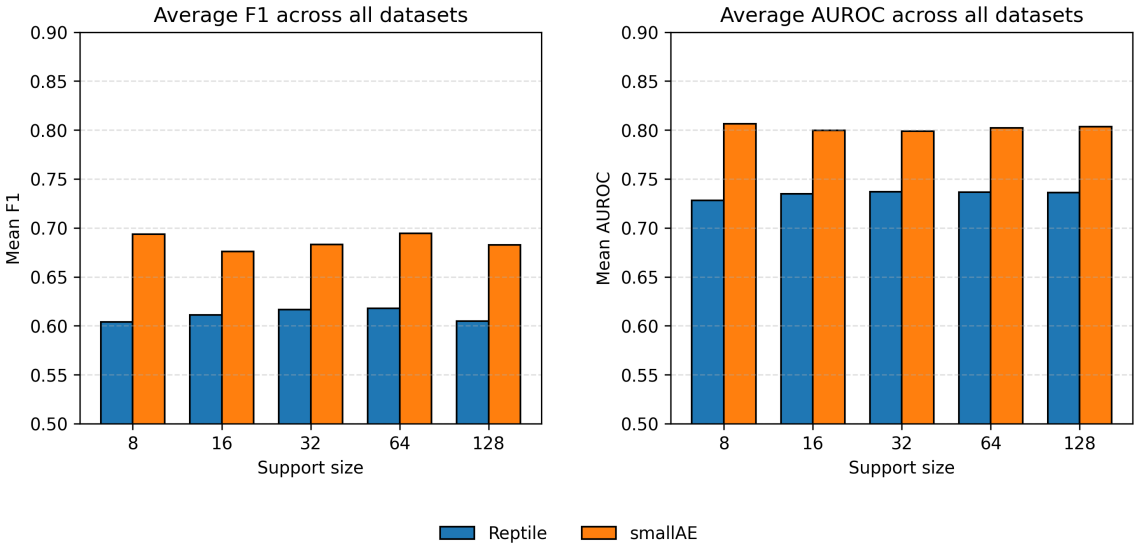



Figure 5.16: Average F1 scores across different support sizes compared to a simple few-shot learner

denoted here as smallAE, under the same support settings. Figure 5.16 summarizes the average F1 score across these shared conditions as the support size increases.

The results show that increasing the support size produced only modest changes in Reptile performance. Mean F1 rose from 0.6039 at support size 8 to a local maximum of 0.6179 at support size 64, before decreasing slightly to 0.6048 at support size 128. This suggests that the benefit of additional benign support data was limited, with most of the gain already realized by the mid-range support settings. In contrast, the simple few-shot baseline achieved higher average F1 scores at every tested support size, with values of 0.6937, 0.6760, 0.6829, 0.6946, and 0.6827 for support sizes 8, 16, 32, 64, and 128, respectively. The same overall pattern was also observed in AUROC, where the SmallAE baseline remained ahead across the shared support settings.

One possible explanation is that the task structure in the OTA corpus did not strongly favor a meta-learned initialization. Although multiple OTA families were included, many files were drawn from relatively closely related software ecosystems, particularly within the Hyundai-group updates and within the AAOS builds derived from the AOSP toolchain.

Under these conditions, a standard autoencoder trained on pooled benign data may already learn a strong prior over normal OTA content, leaving limited room for Reptile to improve adaptation. At the same time, the broader difference between vendor Linux-based updates and AAOS may have made it difficult for a single initialization to transfer equally well across all domains. This interpretation also suggests that the composition of the OTA corpus may have limited the extent to which the benefits of meta-learning could be observed. A wider collection of OTA updates from more vendors and software stacks may result in a more suitable setting for evaluating whether meta-learning can offer a clearer advantage.

These results address RQ4 by showing that limited benign support data is useful for target-domain adaptation, but that the Reptile configuration used here did not outperform the simpler few-shot autoencoder baseline.

### 5.7.5 Ablation Study on File Size and Entropy

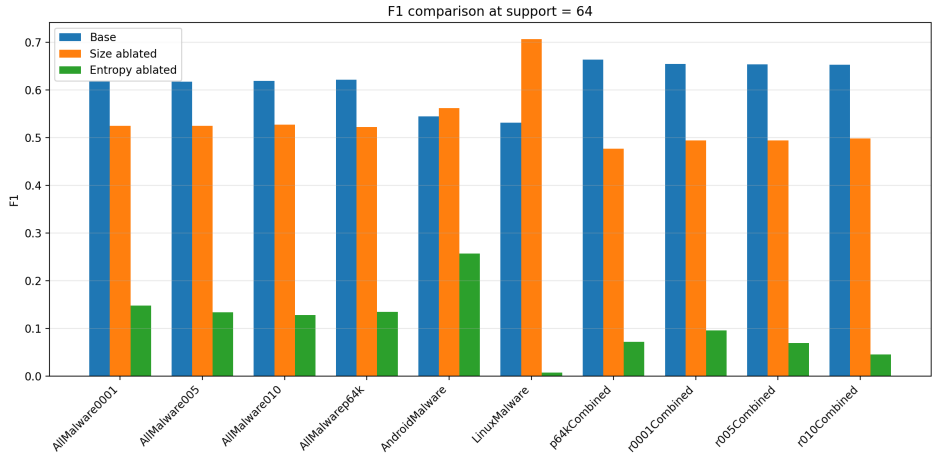



Figure 5.17: F1 comparison at support size 64 for the baseline 298-dimensional Reptile model and the two ablated variants. Removing size reduced performance in several settings but did not eliminate detection capability, while removing entropy caused a substantially larger decline across most conditions.

To examine whether the Reptile detector depended too heavily on the scalar size and entropy features, separate ablation experiments were conducted in which each feature was

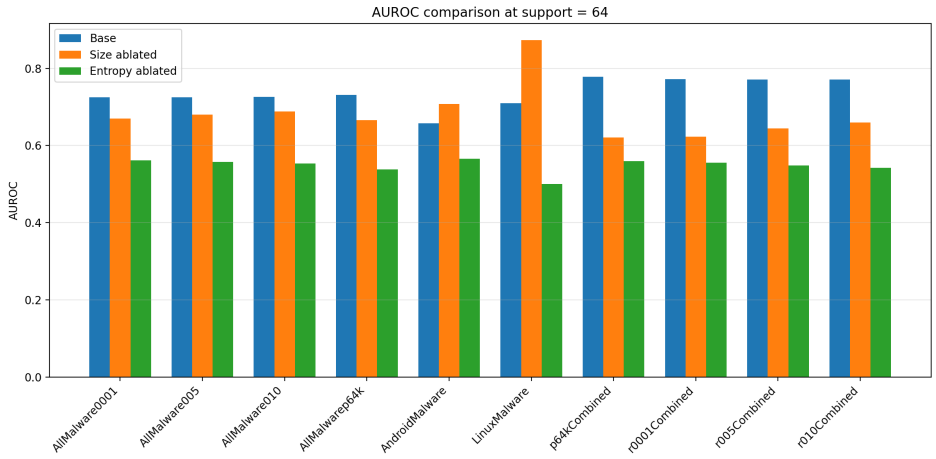



Figure 5.18: AUROC comparison at support size 64 for the baseline 298-dimensional Reptile model and the two ablated variants. The baseline model remained strongest overall, the size-ablated model showed mixed degradation with some stronger raw-malware results, and the entropy-ablated model declined more substantially across nearly all conditions.

removed individually while the remaining 298-dimensional representation was left unchanged. Figures 5.17 and 5.18 compare the resulting F1 and AUROC values at support size 64. The baseline model remained strongest overall across most settings, indicating that both scalar features contributed useful discriminative information in the original representation. However, the two ablations had different effects. Removing size reduced performance in many of the mixed and injected settings, but the detector remained viable and even improved on some raw-malware cases, most notably Linux malware. This suggests that file size was helpful, but was not solely responsible for the model's behaviour.

Removing entropy caused a much larger decline across nearly all settings. The drop was especially pronounced for the mixed and injected conditions, where both F1 and AUROC fell substantially relative to the baseline and to the size-ablated model. This indicates that entropy carried more central information for discrimination than file size alone. Taken together, the ablation results do not support the view that the detector was driven only by a simple size-based shortcut. Instead, they show that both scalar features contributed meaningful signal, with entropy appearing to play the stronger role in the full 298-dimensional model.

## 5.8 Effect of Hash-Binned 3-Gram Representations

### 5.8.1 Evaluation Setting

This experiment addresses RQ1 and RQ2 by testing whether a richer static byte-sequence representation improves OTA anomaly detection across the evaluated unsupervised methods. The earlier experiments used the primary 298-dimensional feature representation, which included selected 3-gram-related features. In contrast, the hash-binned representation maps byte 3-gram information into fixed hash bins, allowing a broader set of local byte-sequence patterns to be represented while keeping the feature vector size bounded. This experiment therefore evaluates whether expanding the byte-level representation improves detection performance, and whether the effect is consistent across model families.

For the Reptile-based results in this section, the experiment also informs RQ4 by testing whether the meta-learning-based detector benefits from the same hash-binned representation during limited-support domain adaptation.

The hash-binned 3-gram representation was considered as a more scalable alternative to explicit 3-gram features for file-level OTA analysis. Its main purpose is to retain a broader summary of local byte-pattern behaviour while keeping the feature space fixed and computationally manageable. This is particularly useful for OTA data, where update contents vary across vendors, operating systems, and file types, but where the anomaly detector must still operate on a common input representation. By compressing many possible 3-gram patterns into a fixed number of bins, the representation reduces storage and preprocessing cost and supports consistent evaluation across models. The main limitation is that bin collisions may obscure some exact pattern identity, so improved scalability may come at the cost of reduced fine-grained discrimination.

The hash-binned 3-gram representation was constructed by mapping extracted byte tri-

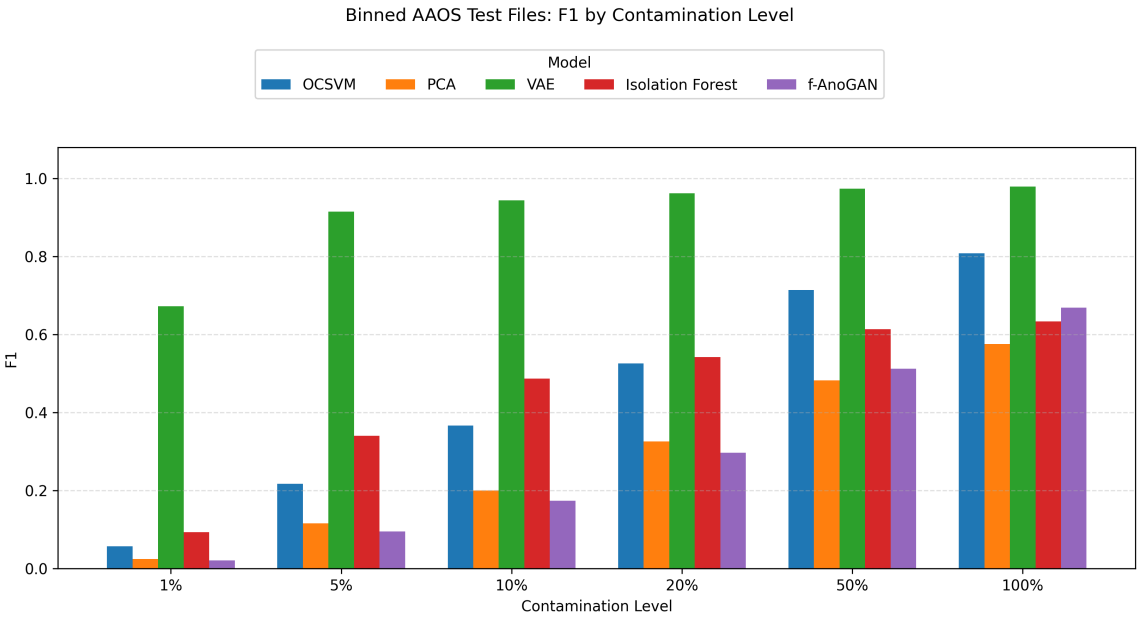



Figure 5.19: Binned Android F1 Results

grams into a fixed number of bins using a stable MD5-based hash function. For each file, each extracted trigram was converted into a consistent string form and hashed to a bucket index by taking the hash value modulo the number of bins. The corresponding trigram counts were accumulated within their assigned bins, so that multiple distinct trigrams could contribute to the same bucket. After accumulation, a $\log(1 + x)$ transform was applied to the resulting bin vector. This produced a fixed-length representation that retained approximate local byte-pattern frequency information while avoiding explicit storage of all trigram identities.

### 5.8.2 Results on Baseline Models

Compared with the original explicit 3-gram baseline representation, the hash-binned representation did not produce a uniform improvement across the five baseline models. Its effect depended strongly on the test setting and on the model itself, as shown by the F1 results in Figures 5.19 and 5.20 and the PR-AUC results in Figures 5.21 and 5.22.

In the AAOS setting, the hash-binned representation weakened several models relative to

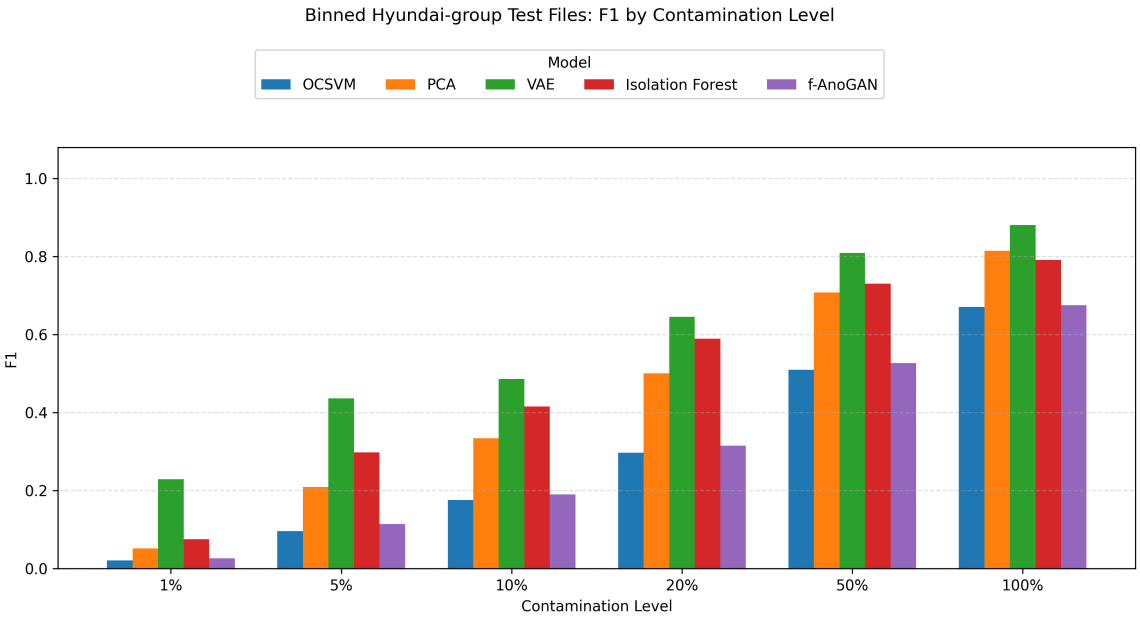



Figure 5.20: Binned Linux F1 Results

the original 298-dimensional representation. At 100% contamination, OCSVM produced the strongest F1 score among the binned baselines at 0.808, followed by f-AnoGAN at 0.669, VAE at 0.658, Isolation Forest at 0.634, and PCA at 0.576. The threshold-free metrics show the weakness of the binned VAE more clearly. Its AUROC remained near 0.156 across the AAOS contamination range, and its PR-AUC reached only 0.327 at 100% contamination. By contrast, OCSVM retained much stronger threshold-free separation on AAOS, with AUROC near 0.88 and PR-AUC increasing to 0.877 at 100% contamination. These results indicate that the hash-binned representation did not improve VAE on AAOS files in the current run.

The combined mixed-OTA setting was also weaker under the hash-binned representation than under the original baseline representation. In the original combined baseline results, VAE, OCSVM, and PCA reached F1 scores of 0.889, 0.857, and 0.856, respectively, at 100% contamination. Under the hash-binned representation, the corresponding binned results were lower, with OCSVM at 0.777, PCA at 0.653, and VAE at 0.678. Isolation Forest reached 0.730 and f-AnoGAN reached 0.675 in the same setting. At 1% contamination, the

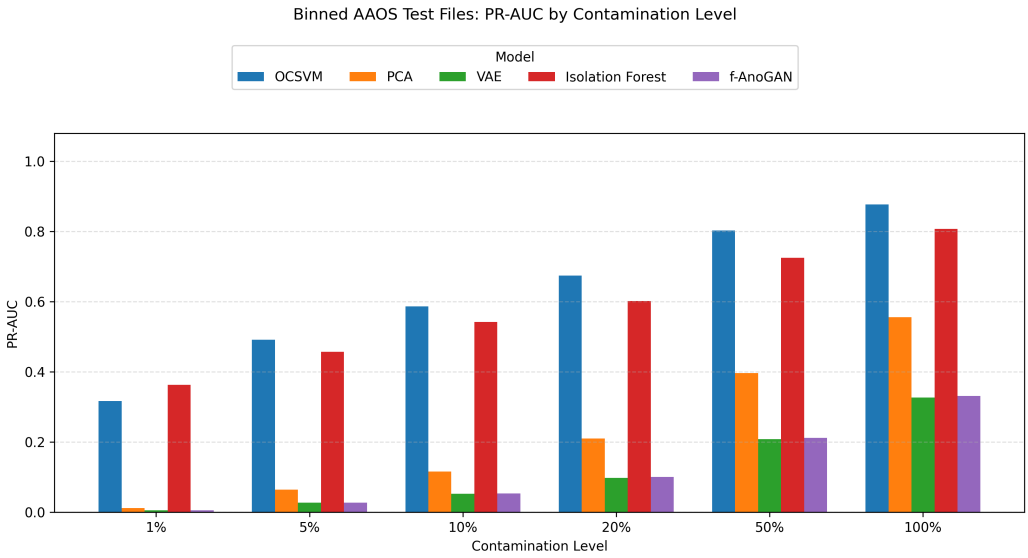



Figure 5.21: PR-AUC results for the hash-binned baseline models on AAOS test files across contamination levels.

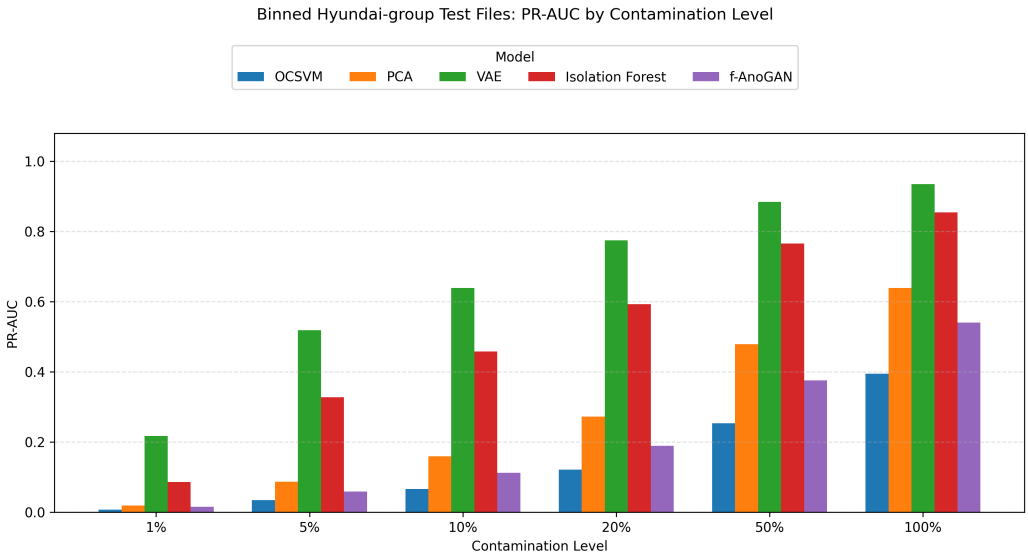



Figure 5.22: PR-AUC results for the hash-binned baseline models on Hyundai-group test files across contamination levels.

best binned combined result was 0.088 under Isolation Forest, indicating that thresholded detection remained weak when anomalies were rare.

The Linux-based Hyundai-group setting showed the strongest binned VAE behaviour. In the original Hyundai-group evaluation, VAE and Isolation Forest were the strongest methods, reaching F1 scores of 0.847 and 0.844 at 100% contamination. Under the hash-binned

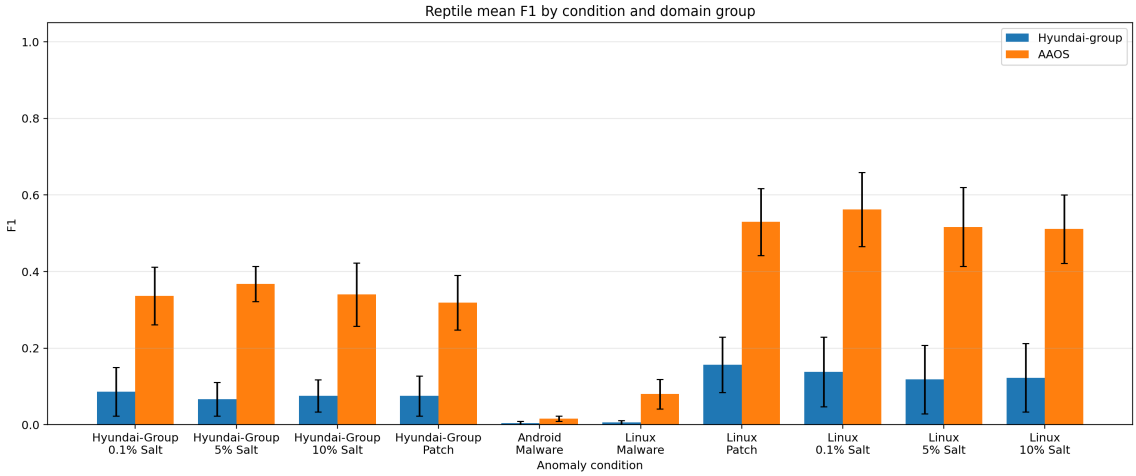



Figure 5.23: Binned Reptile F1 Results

representation, VAE remained strong and reached an F1 score of 0.881 at 100% contamination, with AUROC remaining near 0.95 across the contamination range. PCA and Isolation Forest were also competitive at 100% contamination, reaching F1 scores of 0.815 and 0.791, respectively, while f-AnoGAN reached 0.675 and OCSVM reached 0.671. The PR-AUC results show the same pattern: VAE reached 0.935 at 100% contamination, followed by Isolation Forest at 0.854 and PCA at 0.639. Therefore, the hash-binned representation did not uniformly degrade VAE performance. Instead, its effect was domain-dependent: it weakened VAE substantially on AAOS test files but preserved strong performance on Hyundai-group test files.

### 5.8.3 Results on Reptile

For Reptile, the hash-binned representation produced a clear degradation across all evaluated conditions. This contrasts with the baseline-model comparison, where the effect of binning was mixed and occasionally beneficial. Under Reptile, the 298-D representation consistently outperformed the binned version in both thresholded and threshold-free evaluation, indicating that the meta-learned detector relied more strongly on the original explicit feature structure.

Figure 5.23 shows that the largest drop appeared in the raw-malware conditions. Under the 298-D representation, mean F1 reached 0.545 on Android Malware and 0.531 on Linux Malware. Under the binned representation, these fell sharply to 0.009 and 0.037, respectively. AUROC followed the same pattern, decreasing from 0.658 to 0.141 on Android Malware and from 0.710 to 0.444 on "Linux Malware". This indicates that the decline was not only a thresholding issue: score separation itself was substantially weaker once the trigram information was compressed into hash bins.

The same behaviour was observed under the injected conditions. For the "All Malware" salted sets, mean F1 was about 0.618–0.619 under the 298-D representation, but only about 0.189–0.196 under the binned version. On the corresponding Linux-only salted sets, F1 fell from about 0.653–0.655 to about 0.289–0.320. Patch-based results also declined, with "All Malware Patch" decreasing from 0.622 to 0.180 and "Linux Malware Patch" decreasing from 0.664 to 0.316. Across all of these settings, precision remained moderate in the binned runs, but recall dropped sharply, showing that the detector became much less sensitive to anomalous files after the representation change.

### 5.8.4 Summary of Hashed 3-Gram Results

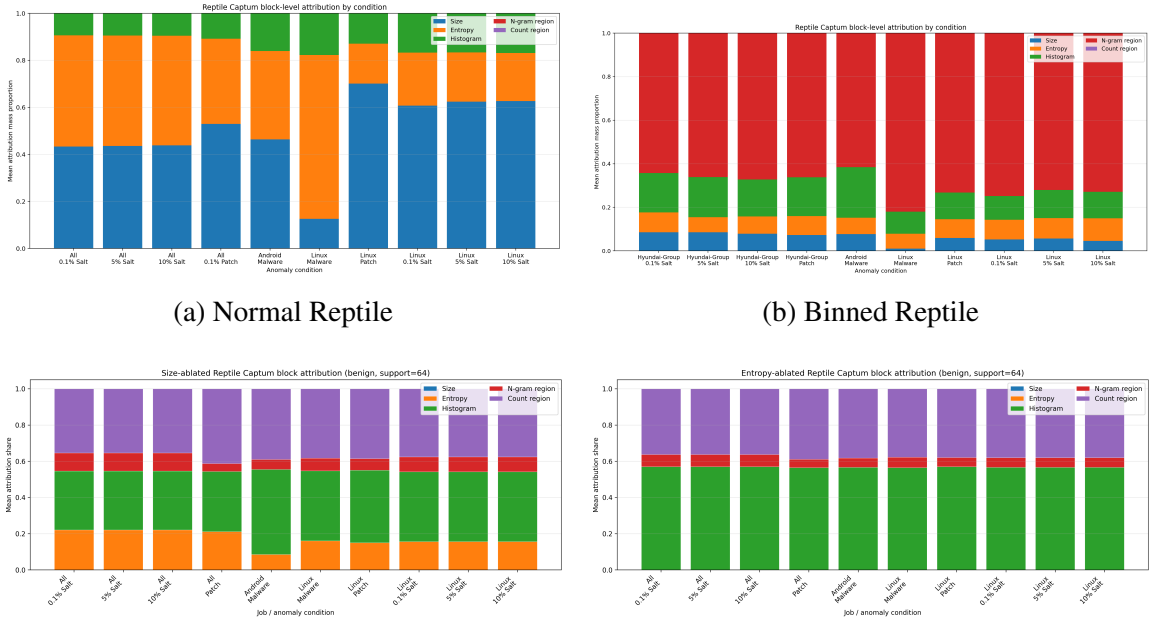
With respect to RQ1 and RQ2, the hash-binned representation did not provide a consistent improvement over the primary 298-dimensional feature representation. Although the expanded representation captured additional byte-sequence information, the resulting performance remained model-dependent and, in several cases, weaker than the corresponding 298-dimensional results. For RQ4, the Reptile hash-binned results suggest that increasing the n-gram representation alone did not improve limited-support meta-learning performance. These findings indicate that richer static features are not automatically beneficial unless the model can use the added byte-sequence information reliably.

### 5.8.5 Interpretability Analysis of Meta-Learning

The preceding Reptile experiments evaluate detection performance, but they do not show which parts of the static feature vector contributed most to the model's anomaly scores. This section addresses RQ6 by examining the feature groups used by the meta-learning-based detector after adaptation. The goal is to determine whether Reptile relied on meaningful byte-level structure, such as byte histograms and 3-gram features, or whether its decisions were driven mainly by simpler shortcut features such as file size and entropy.

Captum is the open-source PyTorch interpretability library used for this analysis [74]. Integrated Gradients was applied to the Reptile autoencoder to estimate the contribution of each input feature to the reconstruction-based anomaly score [75]. The resulting attributions were aggregated into feature blocks corresponding to size, entropy, byte-histogram features, and 3-gram-related features. This block-level view provides a compact way to compare how the normal, hash-binned, size-ablated, and entropy-ablated Reptile variants used the available static features.

Figure 5.24 compares the Captum block-level attribution summaries for the normal, binned, size-ablated, and entropy-ablated Reptile models. The Captum results changed substantially between the two Reptile representations, which suggests that the performance difference was accompanied by a real shift in what the model treated as informative. In the 298-D setting, attribution was dominated by the global file-level features. Across the saved explanations, approximately 53% of the attribution mass was assigned to entropy and about 41% to file size, while the histogram block accounted for only about 6%. By contrast, in the hash-binned setting most of the attribution moved to the hashed trigram-bin block, which absorbed roughly 72% of the total attribution mass. The remaining importance was divided mainly across the histogram, entropy, and file-size features. This indicates that the binned Reptile model relied far less on the simpler global cues that were prominent in the 298-D setting and instead based its decisions primarily on the compressed local-pattern




(a) Normal Reptile

(b) Binned Reptile

(c) Size-ablated Reptile

(d) Entropy-ablated Reptile

Figure 5.24: Captum block-level attribution summaries for the Reptile models and ablation variants.

representation.

This shift is consistent with both the construction of the binned features and the observed drop in detection performance. Hash binning maps many distinct trigrams into a fixed set of buckets, which improves scalability but also introduces collisions and removes exact 3-gram identity. As a result, the model sees a denser but less explicit local-feature block, and the attribution maps suggest that Reptile over-relied on that block after binning. Since the binned runs also showed markedly worse AUROC and F1 across the raw, salted, and patch-based conditions, the altered Captum pattern is better interpreted as a change in what the model relied on rather than as an improvement in what it learned. In the 298-D setting, Reptile appears to have depended largely on coarse global cues such as file size and entropy, which may have acted as an effective shortcut for separating the evaluated benign and anomalous files. After hash binning, attribution moved toward the compressed trigram-bin block. Because hash binning preserves local 3-gram information only approximately, this shift exposed the model to a broader but noisier representation and was accompanied by

weaker detection performance.

The ablation results provide a more direct test of whether these attributed features were actually important for detection performance. Compared with the baseline Reptile model, removing file size produced a moderate average decline, with mean F1 decreasing from 0.618 to 0.533 and mean AUROC decreasing from 0.737 to 0.683. The AUPRC drop was smaller, from 0.861 to 0.840, indicating that the size-ablated model still retained much of its ranking ability. The full per-condition results are reported in Appendix E.1. This suggests that file size contributed to the baseline model, but was not the only source of useful signal.

By contrast, entropy ablation caused a much larger performance collapse. The mean F1 decreased from 0.618 to 0.109, while mean AUROC dropped from 0.737 to 0.548. This places the entropy-ablated model close to random ranking in several conditions, most clearly for LinuxMalware, where AUROC was 0.501 and F1 was only 0.006. These results are consistent with the Captum attribution patterns, where entropy received the largest share of attribution in the 298-D Reptile model. More importantly, they show that entropy was not only highly attributed, but also necessary for much of the model's detection performance. File size appears to be useful but partly replaceable, whereas entropy appears to provide a stronger and more stable signal for separating benign OTA files from the evaluated anomaly conditions.

With respect to RQ6, the interpretability analysis shows that Reptile's decisions were strongly shaped by the feature representation. In the primary 298-dimensional setting, the model placed substantial importance on compact global features such as size and entropy, while the hash-binned setting shifted the attribution pattern but did not produce stronger detection results. The ablation results further show that removing size or entropy changed both the attribution profile and the resulting performance. These findings suggest that interpretability analysis is useful for diagnosing whether the meta-learning detector is using

robust byte-level structure or relying on simpler shortcut features. In this evaluation, the Captum results indicate that Reptile's behaviour was partly explainable, but also sensitive to feature design.

## 5.9 Computational Efficiency and Practical Considerations

### 5.9.1 Feature Extraction Cost

| Rep. | Proc. | Wall (min) | Mean ms/file | Median ms/file | MB/s | Peak RSS |
|---|---|---|---|---|---|---|
| Size/Entropy | 18,665 | 25.52 | 26.82 | 6.44 | 52.51 | 152.35 |
| Histogram | 18,665 | 3.88 | 12.30 | 2.87 | 168.61 | 113.51 |
| 3-grams | 18,665 | 63.51 | 2353.27 | 490.68 | 0.72 | 48.30 |

Table 5.2: Preprocessing efficiency for the three feature representations on all Hyundai OTA updates. Wall time reports elapsed end-to-end runtime for the full preprocessing job. Mean and median ms/file report measured per-file processing times and are not computed from wall time divided by the number of processed files; for multiprocessing runs, per-file worker times may exceed elapsed wall-clock averages. Peak Resident Set Size (RSS) is reported in MB as a process-level memory measure.

Table 5.2 summarizes the computational cost of constructing the three feature representations from the Hyundai OTA files. The byte-histogram representation was the most efficient overall, requiring the shortest total runtime, the lowest mean and median processing time per file among the two higher-throughput representations, and the highest average throughput at 168.61 MB/s. The chunked size/entropy representation was slower and used more memory, but still remained within a practical range, with a mean processing time of 26.82 ms per file and a mean peak RSS of 152.35 MB. These results indicate that both representations can be extracted at relatively low cost from large OTA file collections.

In contrast, byte 3-gram extraction was substantially more expensive in runtime. Although its recorded mean peak RSS was lower than that of the other two representations, its total

| | Android mixed | | Linux mixed | |
|---|---|---|---|---|
| **Model** | **ms/sample** | **Peak RSS (MB)** | **ms/sample** | **Peak RSS (MB)** |
| PCA | 0.0024 | 435.96 | 0.0016 | 356.93 |
| Isolation Forest | 0.0040 | 379.65 | 0.0031 | 339.36 |
| OCSVM | 0.3775 | 378.47 | 0.3460 | 339.66 |
| VAE | 0.0121 | 696.44 | 0.0158 | 679.29 |
| f-AnoGAN | 0.0247 | 363.11 | 0.0260 | 329.37 |
| Reptile[†] | 0.0064 | 465.73 | 0.0060 | 434.25 |

Table 5.3: Inference-efficiency results for the evaluated models on the Android and Linux mixed test sets. Peak RSS is reported in MB as a process-level inference memory measure. For Reptile, inference values are averaged across tasks within each malware condition.

[†] Reptile values are reported as the mean across evaluated OTA-domain tasks. Mean adaptation time for Reptile was approximately 0.063 s per task for the Android-malware condition and 0.062 s per task for the Linux-malware condition.

runtime was much longer, with a mean processing time of 2353.27 ms per file and throughput of only 0.72 MB/s. This suggests that the main cost of the 3-gram representation was computational rather than memory-related. Taken together, the preprocessing results show a clear trade-off between representational richness and extraction cost: histogram- and size/entropy-based features were comparatively inexpensive to compute, whereas 3-gram extraction imposed a much heavier runtime burden.

### 5.9.2 Inference Cost

Table 5.3 compares the inference cost of the evaluated models on the Android and Linux mixed test sets. The results show that the classical baselines, particularly PCA and Isolation Forest, had the lowest per-sample scoring cost, with inference times remaining below 0.005 ms per sample in both settings. Reptile also remained efficient at scoring time after adaptation, with mean inference costs of 0.0064 ms per sample on the Android mixed set and 0.0060 ms per sample on the Linux mixed set. VAE and f-AnoGAN were slower than these methods, but still operated at a low per-sample cost overall. In contrast, OCSVM was substantially slower than the other models, at approximately 0.35–0.38 ms per sample, making it the least favorable of the evaluated methods from an inference-time perspective.

The memory results show a similar trade-off. VAE produced the highest process-level peak RSS in both test settings, followed by Reptile and PCA, whereas Isolation Forest, OCSVM, and f-AnoGAN remained somewhat lower. Even so, the recorded inference memory usage remained within a few hundred megabytes for all evaluated methods, which supports the practicality of static file-level screening in a pre-installation setting. For Reptile, it is also important to distinguish between adaptation and scoring cost. Although the scoring stage itself was efficient, the method incurred an additional mean adaptation time of approximately 0.063 s per task for Android malware and 0.062 s per task for Linux malware. Taken together, these results suggest that inference-time cost was modest for most models, although the strength of the lightweight claim is more convincing for PCA, Isolation Forest, and Reptile scoring than for OCSVM, and less so for VAE in terms of memory usage alone.

These measurements address RQ5 by showing that static pre-installation screening can be computationally practical, especially for lower-cost feature blocks and efficient scoring models, while 3-gram extraction, VAE memory use, and OCSVM inference cost remain important deployment considerations.

## 5.10 Discussion of Findings

This section summarizes the principal findings drawn from the empirical results and relates them to the research questions introduced in Section 1.3.1. Across the experiments, OTA-Sentinel showed that lightweight static byte-level features can provide useful anomaly-detection signals for pre-installation OTA file screening, but the strength of those signals depends on the target domain, anomaly type, and model family. The strongest results generally appeared in the AAOS setting and in raw-malware or salted-modification conditions, while Hyundai-group domains and patch-based modifications produced more difficult detection settings.

The results also show that no single model dominated across all experiments. VAE was generally the strongest pooled baseline, while PCA and OCSVM remained competitive in several settings. Reptile showed that domain-specific benign support data can be useful, especially for AAOS domains, but the support-size comparison indicated that the Reptile configuration used here did not clearly outperform the simpler SmallAE adaptation baseline. The hash-binned and interpretability experiments further showed that adding richer byte-sequence information did not automatically improve performance and that Reptile's behaviour was sensitive to feature design.

Finally, the computational-efficiency results indicate that the proposed static screening approach is practical in principle, but deployment choices should account for feature-extraction cost, memory usage, and inference latency. Byte-histogram, size, and entropy features were comparatively efficient, while byte 3-gram extraction was more costly. These findings support the feasibility of OTA-Sentinel as a pre-installation screening approach, while also identifying the main limitations that affect its robustness across domains and anomaly conditions.

**RQ1: To what extent can lightweight static byte-level features distinguish benign OTA files from malicious or anomalous files in IVI update contexts?**

The results indicate that static byte-level features can support meaningful separation between benign OTA files and anomalous or malicious files, although the strength of that separation depended on the evaluation setting, the target domain, and the detection method. Under the pooled baseline setting, several methods achieved strong ROC-AUC values, particularly on AAOS test files, while the malicious modification experiments showed that the same feature space could still retain useful anomaly information when malicious content was embedded inside otherwise benign OTA-like files. The Reptile results further showed that these features remained informative in a domain-adaptive setting, especially for several AAOS OTA domains. At the same time, the weaker results under some Hyundai-group and

patch-based conditions indicate that the feature representation does not make all anomaly settings equally easy. Overall, the findings support the view that lightweight static byte-level features can distinguish benign and anomalous OTA files to a meaningful extent, but that their effectiveness remains dependent on domain structure and anomaly type.

**RQ2: Which unsupervised anomaly-detection methods are most effective for pre-installation OTA file screening under mixed-OTA training and varying contamination levels?**

No single method dominated in every setting, but clear patterns emerged across the experiments. In the pooled mixed-OTA baseline evaluation, VAE was generally the strongest overall method, particularly on AAOS data and under the modified-file conditions, while PCA and OCSVM also remained competitive in several settings. Isolation Forest performed reasonably in some Linux-based conditions but was less consistent overall, and f-AnoGAN generally trailed the stronger baselines. These results suggest that reconstruction-based methods were the most consistently effective overall in the mixed-OTA setting, while the relative ranking of models still depended on the target domain, contamination level, and anomaly source.

**RQ3: How well do anomaly-detection models generalize across Linux-based Hyundai-group OTA data and AAOS-derived OTA data?**

The results show that generalization across these domains is possible, but uneven. In the pooled baseline evaluation, AAOS files were generally handled more effectively than Hyundai-group files, especially by stronger methods such as VAE, PCA, and OCSVM. The direct cross-domain transfer experiments in Section 5.5 further showed that train-test direction mattered: same-domain evaluation was generally stronger than cross-domain evaluation, and transfer was asymmetric across the AAOS-to-Linux and Hyundai-to-Android directions. In particular, the AAOS-trained VAE transferred more effectively to the Hyundai-group/Linux test setting than the Hyundai-trained VAE transferred to the Android/AAOS

test setting. Under the Reptile evaluation, the same broader pattern remained, with AAOS domains generally producing stronger and more stable results, while Hyundai-group domains were more variable and more sensitive to anomaly source. Overall, the findings indicate that model behaviour does not transfer uniformly across OTA domains, and that domain-specific calibration or adaptation is important when applying anomaly detection across different IVI software environments.

**RQ4: To what extent does meta-learning support adaptation to individual OTA domains when only limited benign support data are available?**

The Reptile experiments show that limited benign support data can support target-domain adaptation, but they do not show that Reptile was consistently superior to simpler few-shot adaptation. On several AAOS tasks, Reptile achieved strong and relatively stable results across anomaly conditions, suggesting that the meta-learned initialization was able to make use of regularity within those target domains. On the Hyundai-group tasks, performance was more uneven, with some domains responding well and others remaining difficult. However, the support-size analysis showed that the simple few-shot autoencoder baseline achieved higher average F1 and AUROC than Reptile across the tested support sizes. Therefore, the results support the value of target-domain benign support data and show that meta-learning can support adaptation in some OTA domains, but they provide only limited evidence that the Reptile configuration used here offers a clear advantage over simpler adaptation methods.

**RQ5: What are the computational costs of the proposed static feature representations and anomaly-detection models for pre-installation OTA screening?**

The efficiency results show that computational cost depends on both the feature representation and the model. For feature extraction, byte histograms were the most efficient representation, while file-size and entropy extraction also remained practical for large OTA collections. In contrast, byte 3-gram extraction imposed a much higher runtime cost, in-

dicating that richer local byte-sequence features should be used carefully in deployment settings where screening time is limited. At inference time, PCA and Isolation Forest were the fastest baseline methods, both remaining below 0.005 ms per sample in the Android and Linux mixed test sets. Reptile also had low scoring cost after adaptation, at approximately 0.006 ms per sample, although it required a small additional adaptation step for each target task. VAE and f-AnoGAN were slower than PCA, Isolation Forest, and Reptile scoring, but still remained within a low per-sample inference range. OCSVM was the least favourable model from an inference-time perspective, requiring roughly 0.35–0.38 ms per sample. Memory usage also varied across methods, with VAE producing the highest peak RSS, followed by Reptile and PCA. Overall, the results indicate that static pre-installation screening can be computationally practical, especially when using lower-cost feature blocks and efficient scoring models such as PCA, Isolation Forest, or Reptile after adaptation. However, the cost of 3-gram extraction, the higher memory usage of VAE, and the slower inference time of OCSVM should be considered when selecting a deployment configuration.

**RQ6: Which static feature groups contribute most to the meta-learning-based detector's anomaly scores, and do these attributions indicate reliance on byte-level structure or simpler global features such as file size and entropy?**

The Captum-based interpretability analysis showed that Reptile relied heavily on compact global features such as size and entropy in the primary representation, while the hash-binned and ablated variants produced different attribution patterns. In the 298-dimensional setting, entropy and file size accounted for most of the attribution mass, while the hash-binned setting shifted attribution toward the compressed 3-gram bin block without improving detection performance. The ablation results further showed that removing entropy caused a much larger decline than removing file size, indicating that entropy provided a central detection signal for the evaluated Reptile model. These results show that feature-block attribution can help explain how the meta-learning-based detector uses static OTA file

features, while also revealing sensitivity to feature design and possible reliance on simpler global indicators rather than only richer byte-sequence structure.

# Chapter 6: Conclusion

This chapter concludes the research by summarizing the main findings, identifying the main limitations, and outlining future work. The central contribution is to show that lightweight, static models trained primarily on benign OTA files can be used as a practical pre-installation inspection step for IVI OTA update pipelines. The proposed screening layer is positioned after update retrieval and verification, but before installation or activation, where it complements authenticity and integrity checks by asking whether the extracted files in an accepted update appear consistent with the expected benign structure of the target software environment. The results show that this approach can provide useful file-content anomaly signals before installation, while also identifying the conditions that limit its performance, including model choice, feature representation, domain transfer, anomaly type, thresholding, adaptation strategy, and deployment setting.

## 6.1 Summary of Findings

The results show that lightweight static models trained primarily on benign OTA files can support practical pre-installation inspection of IVI update contents. By treating extracted OTA files as individual samples, the proposed approach adds an additional layer of file-content protection after update retrieval and verification, but before installation or activation. This screening step is useful in realistic OTA security settings where labelled malicious OTA samples are scarce, because it can identify files that deviate from the expected structure of the target OTA environment without requiring malicious OTA examples during training. At the same time, the results identify the limits of this approach, showing that performance depends on feature representation, software domain, anomaly type, adaptation strategy, thresholding, and deployment conditions.

The baseline experiments showed that static byte-level features provide effective anomaly-

detection signal. File size, Shannon entropy, byte histograms, and byte 3-gram information allowed several models to separate benign OTA files from malicious or modified samples. The 298-dimensional representation was especially effective in the Reptile experiments, while the hash-binned representation often weakened score separation. These results show that compact static features can support OTA file screening, but that feature compression must be evaluated carefully.

The mixed-OTA baseline experiments showed that reconstruction-based methods were the most reliable overall. The VAE provided the most consistent performance across several settings, especially on AAOS files and in the malicious modification experiments. PCA also remained competitive in several cases, showing that a simpler linear reconstruction method can still capture useful structure in the feature space. OCSVM was useful in some settings but had higher computational cost, while Isolation Forest and f-AnoGAN were less stable across conditions. The metric analysis also showed that ROC-AUC, PR-AUC, precision, recall, and F1 must be interpreted together, since strong ranking performance did not always translate into strong precision under low-contamination conditions.

The malicious modification experiments showed that anomaly type strongly affects detection difficulty. Raw malware was generally easier to separate from benign OTA files, while salted and patch-based modifications provided stricter tests because malicious content was inserted into otherwise benign OTA-derived files. Patch-based insertion was especially difficult because it preserved more of the original benign file structure. These results show that evaluation on standalone malware alone is not sufficient for assessing OTA file-screening methods.

The domain-based experiments showed that OTA anomaly detection depends strongly on the software environment. Within the evaluated datasets, AAOS files were generally handled more effectively than Hyundai-group files in the pooled baseline setting. The Reptile experiments showed a similar pattern: the evaluated AAOS domains produced stronger

and more stable results, while Hyundai-group domains were more variable. These findings show that OTA screening methods require domain-aware evaluation rather than assuming clean transfer across software environments.

The adaptation experiments showed that target-domain benign support data is valuable, but that the tested Reptile configuration did not consistently outperform simpler few-shot adaptation. Reptile used target-domain structure effectively in some settings, but the support-size comparison showed that smallAE remained a strong baseline. This result narrows the interpretation of the meta-learning experiments: the evidence supports the value of target-domain adaptation, but not a clear advantage for the tested Reptile initialization over simpler autoencoder adaptation.

Viewed against the research questions, the results show where lightweight static OTA file screening is effective and where it remains limited. For RQ1, the baseline experiments showed that static byte-level features can separate benign OTA files from multiple malicious and modified conditions. For RQ2, the contamination and metric analysis showed that imbalanced evaluation requires PR-AUC, precision, recall, and F1 in addition to ROC-AUC. For RQ3, the domain-based results showed clear differences between Hyundai-group and AAOS software environments. For RQ4, the Reptile and smallAE comparison showed that benign support data improves adaptation, while the tested Reptile configuration did not consistently outperform simpler few-shot adaptation. For RQ5, the computational-efficiency results showed that the proposed static screening approach is suitable for lightweight file-level analysis before installation, while also identifying 3-gram extraction, VAE memory use, and OCSVM inference cost as important practical considerations. For RQ6, the Captum attribution and ablation results showed that the Reptile-based detector relied strongly on compact global features, especially entropy and file size, and that feature representation strongly affected the model's attribution pattern and detection behaviour.

## 6.2 Limitations

The main limitation of this work is dataset coverage. The benign OTA data covers useful Hyundai-group and AAOS-based software settings, but it does not represent the full range of production automotive OTA systems. The findings therefore support the feasibility of file-level OTA anomaly screening, but they should not be read as evidence of consistent performance across all OEMs, update formats, IVI platforms, or package structures.

The malicious evaluation conditions are also approximations of real OTA compromise scenarios. The raw malicious files came from external Linux ELF and Android APK malware corpora, while the OTA-like modified files were created through salted and patch-based injection procedures. These conditions provided controlled tests of anomaly-detection behaviour, but they do not fully capture realistic OTA-specific attacks or malicious content embedded within production-like update packages.

The analysis is limited to file-level static screening. This scope is appropriate for pre-installation analysis because it does not require executing update contents. However, it does not model package-level dependencies, installation order, update metadata, permissions, version relationships, script behaviour, or runtime behaviour after activation. The detector should therefore be viewed as a screening layer rather than a complete update validation mechanism.

The feature representation also limits the conclusions that can be drawn. File size, entropy, byte histograms, and byte 3-gram features are lightweight and practical, but the feature-importance and ablation results showed that some models may rely heavily on coarse properties such as size and entropy. This creates a possible shortcut if malicious changes preserve those properties.

The adaptation results limit the conclusions about meta-learning for OTA domains. Target-domain benign support data was useful, but the tested Reptile configuration did not con-

sistently outperform the simpler smallAE baseline. The findings therefore support target-domain adaptation, but not the superiority of the evaluated meta-learning approach.

A further limitation is threshold selection. The pooled baseline experiments used ROC-derived thresholds selected from labelled evaluation data, which was useful for retrospective comparison but not deployment-ready. The Reptile experiments used benign calibration data, which is closer to the intended setting, but practical thresholding would still depend on the target OTA domain and acceptable false-positive rate.

Deployment also requires an operational response policy. In a real OTA pipeline, an anomaly score alone is not enough. The system must decide whether to block the update, flag the package for review, run deeper analysis, or allow installation with additional monitoring. This work evaluates useful pre-installation screening signals, but not the full decision policy for production use.

Finally, the computational evaluation was conducted in the experimental environment rather than on IVI or embedded automotive hardware. The results support the feasibility of lightweight file-level screening, but they do not capture IVI hardware constraints such as CPU limits, memory pressure, storage speed, thermal behaviour, or resource contention. The efficiency results should therefore be read as evidence of file-level practicality, not as a complete deployment benchmark.

## 6.3 Future Work

A major direction for future work is a broader and more production-like dataset evaluation. A stronger evaluation would include OTA packages from more OEMs and suppliers, multiple IVI operating systems, additional update formats, and a wider range of software versions and regional variants. It should also include packages that more closely match production update structures, including realistic metadata, staging layouts, package orga-

nization, and file dependencies. This would make it possible to test not only whether the screening approach works on additional benign software distributions, but also how well thresholds, feature representations, and model choices transfer across realistic OTA environments.

Although OTA-Sentinel supports scalability through various features such as centralized OEM-side screening, independent file-level processing, and parallelizable feature extraction and scoring, future work should integrate and evaluate it within a complete OTA update pipeline. A comparative analysis could evaluate the same pipeline with and without the OTA-Sentinel screening stage to quantify its added detection capability, computational overhead, and effect on update processing time. This comparison would demonstrate the practical value of content-level screening by testing whether OTA-Sentinel can identify anomalous or malicious files within packages that would otherwise pass existing authenticity, integrity, and signature-verification checks.

The feature representation could also be expanded in future evaluations while preserving the lightweight design goal of the screening stage. The current representation based on file size, entropy, byte histograms, and byte 3-gram features shows that simple static signals can support pre-installation inspection, but these features capture only a limited view of file structure and software context. A stronger version of the approach could add file-type-specific metadata, ELF and APK structural properties, permission and manifest information, strings, section-level properties, import and export information, and package-level relationships between files. Comparing these features against the current representation would show whether additional static context improves robustness without making the detector too costly for IVI deployment.

A further direction is to evaluate the screening approach against more realistic malicious update scenarios. The salted and patch-based experiments provide controlled OTA-like anomaly conditions, but future experiments should include attacks that better reflect how

update contents could be manipulated in practice. Examples include dependency-aware tampering, update-script modification, configuration changes, downgrade attempts, repackaged applications, and malicious changes designed to preserve coarse statistical properties such as size and entropy. A stronger evaluation would also require the creation or collection of a real OTA malware dataset, where malicious examples are embedded in or packaged as OTA update contents rather than represented only by standalone malware files or synthetic byte-level modifications. Such a dataset would make it possible to test whether the detector remains effective when abnormal content is intentionally shaped to resemble normal OTA files and realistic update package structures.

Another important direction is to develop and test deployment-oriented thresholding and response policies. Rather than relying only on fixed percentile thresholds or retrospective operating points, future work could compare domain-specific calibration strategies across OTA families, software versions, and update types. This evaluation should measure how different thresholds affect false positives, missed detections, and update continuity under realistic deployment constraints. It should also define and test response actions for different risk levels, such as allowing installation, sending files for deeper analysis, delaying activation, triggering manual review, or blocking the update when the risk is high. This would turn anomaly scores into operational decisions that can be evaluated in the context of an OTA pipeline.

A final direction is to further investigate adaptation methods for OTA domains. The results suggest that target-domain benign support data can be useful, but the tested Reptile configuration did not consistently outperform the simpler smallAE baseline. Future work should therefore separate the value of adaptation itself from the value of a specific meta-learning method. This could involve comparing alternative meta-learning approaches, stronger non-meta-learning autoencoder baselines, different support-set sizes, support-set selection strategies, and domain-specific calibration methods. Such experiments would

help identify when adaptation provides practical benefit for OTA screening and when simpler target-domain training is sufficient.

## 6.4 Closing Remarks

Overall, the findings show that lightweight, static, benign-trained models can add a practical file-content screening layer to IVI OTA update pipelines. The value of the approach is not in replacing existing OTA protections, but in adding a pre-installation content-screening layer that can flag anomalous files even when an update package has passed authenticity and integrity verification. At the same time, the results show that this screening layer is affected by model choice, feature design, anomaly type, software domain, thresholding, and adaptation strategy. The contribution of this work is therefore twofold: it demonstrates that benign-trained static models can support practical pre-installation OTA inspection, and it identifies the technical and operational conditions that must be addressed before such screening can be deployed more broadly.

# References

[1] S. Halder, A. Ghosal, and M. Conti, "Secure over-the-air software updates in connected vehicles: A survey," *Computer Networks*, vol. 178, p. 107343, 2020.

[2] A. Giannaros, A. Karras, L. Theodorakopoulos, C. Karras, P. Kranias, N. Schizas, G. Kalogeratos, and D. Tsolis, "Autonomous vehicles: Sophisticated attacks, safety issues, challenges, open topics, Blockchain, and future directions," *Journal of Cybersecurity and Privacy*, vol. 3, no. 3, pp. 493–543, 2023.

[3] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," IOActive, White Paper, 2015, accessed: Jan. 15, 2025. [Online]. Available: https://www.ioactive.com/remote-exploitation-of-an-unaltered-passenger-vehicle/

[4] Google for Developers, "Android for cars," 2025, accessed: Dec. 1, 2025. [Online]. Available: https://developers.google.com/cars

[5] M. D. Pesé, K. G. Shin, J. Bruner, and A. Chu, "Security analysis of Android automotive," SAE International, SAE Technical Paper 2020-01-1295, 2020, accessed: Dec. 1, 2025. [Online]. Available: https://www.macheforum.com/site/attachments/2020-01-1295_official-pdf.14616/

[6] Uptane Project, "Uptane standard for design and implementation 2.1.0," 2025, accessed: Sep. 15, 2025. [Online]. Available: https://uptane.org/docs/latest/standard/uptane-standard

[7] ——, "Securing delivery of software updates for ground vehicles," 2021, uptane white paper. Accessed: Sep. 15, 2025. [Online]. Available: https://uptane.org/papers/Uptane_first_whitepaper.pdf

[8] Y. Guo, "A review of machine learning-based zero-day attack detection: Challenges and future directions," *Computer Communications*, vol. 198, pp. 175–185, 2023.

[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366422004248

[9] J. Reeves, A. Ramaswamy, M. Locasto, S. Bratus, and S. Smith, "Lightweight intrusion detection for resource-constrained embedded control systems," in *Critical Infrastructure Protection V*, 2011, pp. 31–46, accessed: Jan. 15, 2025. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-24864-1_3

[10] C. V. Kifor and A. Popescu, "Automotive cybersecurity: A survey on frameworks, standards, and testing and monitoring technologies," *Sensors*, vol. 24, no. 18, p. 6139, 2024.

[11] L. J. Moukahal, M. A. Elsayed, and M. Zulkernine, "Vehicle software engineering (vse): Research and practice," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 137–10 149, 2020.

[12] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[13] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Security Symposium*, 2011, accessed: Jan. 15, 2025. [Online]. Available: https://www.usenix.org/conference/usenix-security-11/comprehensive-experimental-analyses-automotive-attack-surfaces

[14] D. Keuper and T. Alkemade, "The connected car: Ways to get unauthorized access and potential implications," Computest, Zoetermeer, The Netherlands, Research Report, 2018, accessed: Jan. 15, 2025. [Online]. Available: https://www.computest.nl/wp-content/uploads/2018/04/connected-car-rapport.pdf

[15] Y. Wang, Y. Ning, Y. Sun, X. Xie, Z. Xie, Y. Chen, Z. Guo, S. Xue, J. Wang, and S. Chen, "Towards understanding and characterizing vulnerabilities in intelligent connected vehicles through real-world exploits," 2026, accessed: Apr. 15, 2026. [Online]. Available: https://arxiv.org/abs/2601.00627

[16] A. Vasenev, F. Stahl, H. Hamazaryan, Z. Ma, L. Shan, J. Kemmerich, and C. Loiseaux, "Practical security and privacy threat analysis in the automotive domain: Long term support scenario for over-the-air updates," in *Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2019)*. SCITEPRESS, 2019, pp. 550–555.

[17] A. Chawan, W. Sun, A. Javaid, and U. Gurav, "Security enhancement of over-the-air update for connected vehicles," in *Wireless Algorithms, Systems, and Applications*, ser. Lecture Notes in Computer Science, vol. 10874. Springer, 2018, pp. 853–864.

[18] S. Mahmood, H. N. Nguyen, and S. A. Shaikh, "Systematic threat assessment and security testing of automotive over-the-air (OTA) updates," *Vehicular Communications*, vol. 35, p. 100468, 2022.

[19] C. Plappert and A. Fuchs, "Secure and lightweight over-the-air software update distribution for connected vehicles," in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC '23. ACM, 2023, pp. 268–282.

[20] A. C. P. Henriques, "Secure over-the-air vehicle updates using trusted execution environments (TEE)," Master's thesis, Universidade do Porto, 2022, accessed: Jan. 15, 2025. [Online]. Available: https://repositorio-aberto.up.pt/bitstream/10216/147646/2/605924.pdf

[21] A. Anwar, N. Chakraborty, and M. Zulkernine, "Secure OTA software updates for connected vehicles using LoRaWAN and Blockchain," in *2024 IEEE 24th Interna-*

*tional Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE, jul 2024, pp. 1067–1076.

[22] International Organization for Standardization and SAE International, "ISO/SAE 21434:2021 Road vehicles — Cybersecurity engineering," 2021, international standard. Accessed: Apr. 15, 2026. [Online]. Available: https://www.iso.org/standard/70918.html

[23] International Organization for Standardization, "ISO 24089:2023 Road vehicles — Software update engineering," 2023, international standard. Accessed: Apr. 15, 2026. [Online]. Available: https://www.iso.org/standard/77796.html

[24] United Nations Economic Commission for Europe, "UN Regulation No. 155: Cyber security and cyber security management system," 2021, uNECE vehicle regulation. Accessed: Apr. 15, 2026. [Online]. Available: https://unece.org/sites/default/files/2023-02/R155e%20%282%29.pdf

[25] ——, "UN Regulation No. 156: Software update and software update management system," 2021, uNECE vehicle regulation. Accessed: Apr. 15, 2026. [Online]. Available: https://unece.org/sites/default/files/2024-03/R156e%20%282%29.pdf

[26] Android Open Source Project, "What is Android automotive?" 2025, accessed: Dec. 1, 2025. [Online]. Available: https://source.android.com/docs/automotive/start/what_automotive

[27] ——, "OTA updates," 2025, accessed: Dec. 1, 2025. [Online]. Available: https://source.android.com/docs/core/ota

[28] ——, "A/B (seamless) system updates," 2025, accessed: Dec. 1, 2025. [Online]. Available: https://source.android.com/docs/core/ota/ab

[29] ——, "Virtual A/B overview," 2025, accessed: Dec. 1, 2025. [Online]. Available: https://source.android.com/docs/core/ota/virtual_ab

[30] ——, "Apex file format," 2025, accessed: Dec. 1, 2025. [Online]. Available: https://source.android.com/docs/core/ota/apex

[31] A. Qureshi, M. Marvi, J. A. Shamsi, and A. Aijaz, "euf: A framework for detecting over-the-air malicious updates in autonomous vehicles," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 8, Part A, pp. 5456–5467, 2022.

[32] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks—practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, 2011.

[33] W. Wu, R. Li, G. Xie, J. yao An, Y. Bai, J. Zhou, and K. Li, "A survey of intrusion detection for in-vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2020.

[34] Ö. Özdemir, M. T. İşyapar, P. Karagöz, K. W. Schmidt, D. Demir, and N. A. Karagöz, "A survey of anomaly detection in in-vehicle networks," 2024, accessed: Apr. 15, 2026. [Online]. Available: https://arxiv.org/abs/2409.07505

[35] M. Althunayyan, A. Javed, and O. Rana, "A survey of learning-based intrusion detection systems for in-vehicle networks," *Computer Networks*, vol. 277, p. 112031, 2026.

[36] C. Young, J. Zambreno, and G. Bloom, "Towards a fail-operational intrusion detection system for in-vehicle networks," in *Proceedings of the Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS)*, nov 2016, accessed: Apr. 15, 2026. [Online]. Available: https://www.ece.iastate.edu/~zambreno/assets/pdf/YouZam16A.pdf

[37] C. Young, H. Olufowobi, G. Bloom, and J. Zambreno, "Automotive intrusion detection based on constant CAN message frequencies across vehicle driving modes," in *Proceedings of the ACM Workshop on Automotive Cybersecurity (AutoSec)*, 2019, pp. 9–14.

[38] G. Baldini, "On the application of entropy measures with sliding window for intrusion detection in automotive in-vehicle networks," *Entropy*, vol. 22, no. 9, p. 1044, 2020.

[39] M. A. Elsayed and N. Zincir-Heywood, "Boostsec: Adaptive attack detection for vehicular networks," *Journal of Network and Systems Management*, vol. 32, no. 1, p. 6, 2024.

[40] A. R. Wasicek, M. D. Pesé, A. Weimerskirch, Y. Burakova, and K. Singh, "Context-aware intrusion detection in automotive control systems," in *Proceedings of the 5th ESCAR USA Conference*, Detroit, MI, USA, jun 2017, accessed: Apr. 15, 2026. [Online]. Available: https://mpese.com/publication/wasicek-2017-context/Escar17_CAID_Paper.pdf

[41] S. Lee, W. Choi, I. Kim, G. Lee, and D. H. Lee, "A comprehensive analysis of datasets for automotive intrusion detection systems," *Computers, Materials & Continua*, vol. 76, no. 3, pp. 3413–3442, 2023.

[42] D. H. Blevins, P. Moriano, R. A. Bridges, M. E. Verma, M. D. Iannacone, and S. C. Hollifield, "Time-based CAN intrusion detection benchmark," in *Workshop on Automotive and Autonomous Vehicle Security (AutoSec)*, 2021.

[43] B. Kidmose, A. Kidmose, and W. Meng, "can-sleuth: Sleuthing out the capabilities, limitations, and performance impacts of automotive intrusion detection datasets," *International Journal of Information Security*, vol. 24, no. 5, p. 193, 2025.

[44] J. Li, N. Ersotelos, M.-A. Tsompanas, and G. Epiphaniou, "Challenges and opportu-

nities of synthetic data generation for machine learning-based intrusion detection systems in in-vehicle networks," *Vehicular Communications*, vol. 58, p. 100998, 2026.

[45] B. Li, W. Hu, L. Da, Y. Wu, X. Wang, Y. Li, and C. Yuan, "Over-the-air upgrading for enhancing security of intelligent connected vehicles: a survey," *Artificial Intelligence Review*, vol. 57, p. 314, 2024.

[46] S. Longari, P. Cerracchio, M. Carminati, and S. Zanero, "Assessing the resilience of automotive intrusion detection systems to adversarial manipulation," *ACM Transactions on Cyber-Physical Systems*, vol. 9, no. 3, pp. 31:1–31:27, 2025.

[47] M. Dibaei, X. Zheng, K. Jiang, R. Abbas, S. Liu, Y. Xiang, Y. Zhang, and S. Yu, "Attacks and defences on intelligent connected vehicles: a survey," *Digital Communications and Networks*, vol. 6, no. 4, pp. 399–421, 2020.

[48] B. Li, W. Hu, X. Qu, and Y. Li, "A novel multi-attack IDS framework for intelligent connected terminals based on over-the-air signature updates," *Electronics*, vol. 12, no. 10, p. 2267, 2023.

[49] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, 2015, pp. 11–20.

[50] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1126–1135, accessed: Jan. 15, 2026. [Online]. Available: https://proceedings.mlr.press/v70/finn17a.html

[51] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, accessed: Jan. 15, 2026. [Online]. Available: https://arxiv.org/abs/1803.02999

[52] A. Kumagai, T. Iwata, H. Takahashi, and Y. Fujiwara, "Meta-learning for robust anomaly detection," in *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, F. Ruiz, J. Dy, and J.-W. van de Meent, Eds., vol. 206. PMLR, 2023, pp. 675–691, accessed: Jan. 15, 2026. [Online]. Available: https://proceedings.mlr.press/v206/kumagai23a.html

[53] J. Moon, Y. Noh, S. Jung, J. Lee, and E. Hwang, "Anomaly detection using a model-agnostic meta-learning-based variational auto-encoder for facility management," *Journal of Building Engineering*, vol. 68, p. 106099, 2023.

[54] S. Holly, T. Bierweiler, S. von Dosky, A. Frikha, C. Heitzinger, and J. Eder, "One-class domain adaptation via meta-learning," 2025, accessed: Jan. 15, 2026. [Online]. Available: https://arxiv.org/abs/2501.13052

[55] T. K. Kuppusamy, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch, and J. Cappos, "Uptane: Securing software updates for automobiles," in *Proceedings of the 14th Embedded Security in Cars Conference (escar Europe)*, 2016, accessed: Sep. 15, 2025. [Online]. Available: https://ssl.engineering.nyu.edu/papers/kuppusamy_escar_16.pdf

[56] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. Association for Computing Machinery, 2010, pp. 61–72.

[57] R. Bielawski, R. Gaynier, D. Ma, S. Lauzon, and A. Weimerskirch, "Cybersecurity of firmware updates," National Highway Traffic Safety Administration, Tech. Rep. DOT HS 812 807, oct 2020, accessed: Jan. 15, 2026. [Online]. Available:

https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/cybersecurity_of_firmware_updates_oct2020.pdf

[58] European Union Agency for Cybersecurity, "ENISA Threat Landscape for Supply Chain Attacks," European Union Agency for Cybersecurity, Tech. Rep., jul 2021, accessed: Jan. 15, 2026. [Online]. Available: https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Threat%20Landscape%20for%20Supply%20Chain%20Attacks.pdf

[59] QNX Software Systems Limited, *QNX CAR Architecture Guide*, QNX Software Systems Limited, sep 2014, accessed: Jan. 15, 2026. [Online]. Available: https://www.qnx.com/download/download/26204/QNX_CAR_Architecture_Guide.pdf

[60] ——, *QNX CAR Platform for Infotainment System Services Reference*, QNX Software Systems Limited, aug 2014, accessed: Jan. 15, 2026. [Online]. Available: https://www.qnx.com/download/download/26213/System_Services_Reference.pdf

[61] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the International Conference on Learning Representations*, 2014, arXiv:1312.6114. Accessed: Sep. 15, 2025. [Online]. Available: https://arxiv.org/abs/1312.6114

[62] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, 2014, pp. 2672–2680, accessed: Sep. 15, 2025. [Online]. Available: https://papers.neurips.cc/paper/5423-generative-adversarial-nets

[63] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks," *Medical Image Analysis*, vol. 54, pp. 30–44, 2019.

[64] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[65] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proceedings of the 2008 IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[66] Hyundai Motor Company, "Official Hyundai Motors Navigation Update Website," 2025, accessed: Nov. 20, 2025. [Online]. Available: https://update.hyundai.com/CA/EN/navigationUpdate

[67] Android Open Source Project, "Build Android," 2025, accessed: Dec. 1, 2025. [Online]. Available: https://source.android.com/docs/setup/build/building

[68] Harsh, "Building Android automotive os: A beginner-friendly guide," jun 2024, dEV Community, posted June 9, 2024; edited June 11, 2024. Accessed: Dec. 1, 2025. [Online]. Available: https://dev.to/hpnightowl/building-android-automotive-os-a-beginner-friendly-guide-4f67

[69] VirusShare, "VirusShare.com," 2025, restricted-access malware repository for research use. Accessed: Sep. 15, 2025. [Online]. Available: https://virusshare.com/

[70] R. Kumar and S. Geetha, "Explainable machine learning for malware detection using ensemble bagging algorithms," in *Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing (IC3-2022)*, 2022, pp. 453–460.

[71] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001, pp. 38–49.

[72] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild,"

in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 470–478.

[73] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics (CSI-KDD '09)*, 2009, pp. 23–31.

[74] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan, and O. Reblitz-Richardson, "Captum: A unified and generic model interpretability library for PyTorch," 2020, accessed: Feb. 1, 2026. [Online]. Available: https://arxiv.org/abs/2009.07896

[75] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 3319–3328, accessed: Feb. 1, 2026. [Online]. Available: https://proceedings.mlr.press/v70/sundararajan17a.html

[76] Tesla, "Linux Graphics Engineer, Infotainment Platforms, Vehicle Software," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.tesla.com/careers/search/job/linux-graphics-engineer-infotainment-platforms-vehicle-software-253096

[77] ——, "Additional Resources – Open Source," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.tesla.com/en_ca/legal/additional-resources

[78] Waymo, "Android OS Engineer, Waymo Infotainment," 2023, accessed: Apr. 19, 2026. [Online]. Available: https://jobs.spacetalent.org/companies/waymo/jobs/28249372-android-os-engineer-in-car-experience

[79] General Motors, "The software behind the screen: engineer Stanley Fok on scaling GM infotainment," 2026, accessed: Apr. 19, 2026. [Online]. Available:

https://news.gm.com/home.detail.html/Pages/topic/us/en/2026/feb/0217-engineer-Stanley-Fok-scaling-GM-infotainment.html

[80] BlackBerry QNX, "QNX Automotive Solution Guide," 2025, accessed: Jan. 15, 2026. [Online]. Available: https://qnx.software/content/dam/qnx-xwalk/pdf/solution-guides/qnx-automotive-solution-guide.pdf

[81] Automotive Grade Linux, "Volkswagen Joins Automotive Grade Linux and the Linux Foundation," 2019, accessed: Apr. 19, 2026. [Online]. Available: https://www.automotivelinux.org/announcements/volkswagen-joins-agl/

[82] Honda, "Cars & SUVs with Google Built-In Technology," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://automobiles.honda.com/google-built-in

[83] Acura, "Acura SUV Models — Compare Benefits," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.acura.com/suvs

[84] ——, "2026 MDX Premium Performance SUV," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.acura.com/suvs/mdx

[85] Ford Motor Company, "Ford and Google to Accelerate Auto Innovation, Reinvent Connected Vehicle Experience," 2021, accessed: Apr. 19, 2026. [Online]. Available: https://media.ford.com/content/dam/fordmedia/fmea/South-Africa/2021/01%20FEB%2021%20FMCSA%20release%20-%20Ford%20and%20Google%20to%20Accelerate%20Auto%20Innovation.pdf

[86] Lincoln, "Lincoln Owner Support," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.lincoln.com/support/

[87] ——, "Lincoln Digital Experience Frequently Asked Questions," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.lincoln.com/support/how-tos/lincoln

-technology/lincoln-digital-experience/lincoln-digital-experience-frequently-asked-questions/

[88] Volvo Cars, "Volvo Cars and Google expand partnership with Gemini and accelerate innovation in cars," 2025, accessed: Apr. 19, 2026. [Online]. Available: https://www.volvocars.com/intl/media/press-releases/6B411DC79F4F8965/

[89] Polestar, "Polestar 2," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://media.polestar.com/models/polestar-2?lang=en_CA

[90] Renault Group, "Renault Group and Google Accelerate Partnership to Develop the Vehicle of Tomorrow and Strengthen Renault Group's Digital Transformation," 2022, accessed: Apr. 19, 2026. [Online]. Available: https://media.renaultgroup.com/renault-group-and-google-accelerate-partnership-to-develop-the-vehicle-of-tomorrow-and-strengthen-renault-groups-digital-transformation/

[91] Renault, "openR Link Multimedia System - Renault Connect," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.renault.co.uk/renault-connect/openr-link.html

[92] Nissan Motor Co., Ltd., "Overview of Connected Cars and Services," 2024, accessed: Apr. 19, 2026. [Online]. Available: https://www.nissan-global.com/EN/TECHNICALREVIEW/PDF/TOPIC/NISSAN_TECHINICAL_REVIEW_90_En_SP3.pdf

[93] Nissan Canada, "Google Maps, Assistant, & More Built-In," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.nissan.ca/services/apps/google-built-in.html

[94] INFINITI, "Google Built-In: Assistant, Maps, & Play — INFINITI InTouch," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.infinitiusa.com/intouch/features-apps/google-built-in.html

[95] ——, "2026 INFINITI QX80 Connected Services & Technologies," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.infinitiusa.com/vehicles/suvs/qx80/features/connected-services.html

[96] Mitsubishi Motors Europe, "Connectivity," 2024, accessed: Apr. 19, 2026. [Online]. Available: https://www.mitsubishi-motors-europe.com/connectivity_old/

[97] ——, "New Mitsubishi Grandis," 2025, accessed: Apr. 19, 2026. [Online]. Available: https://www.mitsubishi-motors-europe.com/grandis/

[98] Mazda USA, "2026 Mazda CX-5 Crossover SUV," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.mazdausa.com/vehicles/cx-5

[99] Chevrolet, "Chevrolet with the New Google Built-In Feature," 2026, accessed: Apr. 19, 2026. [Online]. Available: https://www.chevrolet.com/technology/google-built-in

# Appendix A: Archived OEM Update Listings

This appendix summarizes the archived Hyundai and Kia update material used during dataset collection. The first table lists archived updates that were matched to visible public OEM update entries. The second table summarizes the retained OEM update sets that are no longer publicly available as of December 2025.

## A.1 Archived Updates with Corresponding Public Entries

Table A.1: Archived Hyundai-group OTA updates with corresponding public entries

| **Model** | **Navigation System** | **SW Version** | **MAP Version** | **Model Name** |
|---|---|---|---|---|
| 2019 Cadenza | Standard Gen5 Navigation | YG19.CAN.SOP.V140.250506.STD_H | NAM.18.42.61.013.301.0 | YGAS.S5AMC.CA |
| 2022–23 Forte 4Door | Standard Gen5W Navigation | BDPE.CAN.S5W_M.V014.010.250818 | NAM.18.47.55.021.561.0 | BDAS.S5BMC.D4CA |
| 2025 Niro Hybrid | Standard Gen5W Navigation | SG225HEV.CAN.S5W_M.V014.010.250818 | NAM.18.47.55.021.561.0 | SG2BH.S5BMC.CA |
| 2025 EV6 | ccNC Navigation | CV1A.CAN.ccNC.001.002.250930 | NAM.18.47.55.021.961.0 | CV1AE.SCBMC.CA |
| 2025 Carnival | ccNC Navigation | KA4.CAN.ccNC.001.002.250930 | NAM.18.47.55.021.961.0 | KA4AS.SCBMC.CA |
| 2018–20 Rio 5Door | Standard Gen5 Navigation | SC.CAN.SOP.V140.250506.STD_H | NAM.18.42.61.013.301.0 | SCAS.S5AMC.CA |

| Model | Navigation System | SW Version | MAP Version | Model Name |
|---|---|---|---|---|
| 2023 Soul | Standard Gen5W Navigation | SK3_PE.CAN.S5W_M.V01 4.010.250818 | NAM.18.47.55.021 .561.0 | SK3DS.S5BMC. CA |

## A.2 Additional Archived Folders

- 2019_20_Elantra_GT_CAN
- 2016_Genesis_9_2_in_touch_CAN
- 2019_Sonata_plug_in_hybrid_CAN
- 2019_21_Tucson_CAN
- 2019_21_Veloster_CAN
- 2020_Venue_CAN
- 2019_Santa_Fe_7seat_CAN
- 2017_18_Ioniq_Electric_CAN
- 2019_Sonata_Hybrid_CAN

## A.3 Hyundai Group Raw OTA update Sizes

Table A.2: Hyundai Group Raw OTA Update Sizes

| Update Set | Share | Size | Items | Files |
|---|---|---|---|---|
| 2022_Forte_4Door_CAN | 18.3% | 29.9 GB | 12 | 12 |
| 2023_Soul_CAN | 9.2% | 14.9 GB | 1,104 | 860 |
| 2016_Genesis_9_2_in_touch_CAN | 8.1% | 13.2 GB | 697 | 663 |
| 2025_Niro_Hybrid_CAN | 6.1% | 9.9 GB | 1,574 | 1,524 |
| 2017_18_Ioniq_Electric_CAN | 6.1% | 9.9 GB | 685 | 655 |
| 2018_20_Rio_5Door_CAN | 5.8% | 9.5 GB | 2,449 | 2,378 |
| 2019_Cadenza_CAN | 5.8% | 9.5 GB | 2,440 | 2,370 |
| 2020_Venue_CAN | 5.8% | 9.5 GB | 2,386 | 2,316 |
| 2019_21_Tucson_CAN | 5.8% | 9.5 GB | 2,386 | 2,316 |
| 2019_Santa_Fe_7seat_CAN | 5.8% | 9.4 GB | 2,381 | 2,311 |
| 2019_21_Veloster_CAN | 5.8% | 9.4 GB | 2,381 | 2,311 |
| 2019_Sonata_plug_in_hybrid_CAN | 5.8% | 9.4 GB | 2,381 | 2,311 |
| 2019_Sonata_Hybrid_CAN | 5.8% | 9.4 GB | 2,381 | 2,311 |
| 2019_20_Elantra_GT_CAN | 5.8% | 9.4 GB | 2,381 | 2,311 |
| **Total** | **100.0%** | **162.4 GB** | **25,638** | **24,649** |

## A.4 Hyundai Group Processed OTA Update Sizes

Table A.3: Processed Hyundai and Kia update sets after extraction and removal of encrypted files.

| **Update Set** | **Share** | **Size** | **Files** |
|---|---|---|---|
| 2025_Niro_Hybrid_CAN | 19.5% | 7.3 GB | 1,511 |
| 2016_Genesis_9_2_in_touch_CAN | 13.6% | 5.1 GB | 469 |
| 2017_18_Ioniq_Electric_CAN | 13.4% | 5.0 GB | 444 |
| 2018_20_Rio_5Door_CAN | 5.7% | 2.1 GB | 1,790 |
| 2019_Cadenza_CAN | 5.7% | 2.1 GB | 1,783 |
| 2019_21_Tucson_CAN | 5.7% | 2.1 GB | 1,726 |
| 2020_Venue_CAN | 5.7% | 2.1 GB | 1,726 |
| 2019_Sonata_plug_in_hybrid_CAN | 5.6% | 2.1 GB | 1,723 |
| 2019_Santa_Fe_7seat_CAN | 5.6% | 2.1 GB | 1,723 |
| 2019_Sonata_Hybrid_CAN | 5.6% | 2.1 GB | 1,723 |
| 2019_20_Elantra_GT_CAN | 5.6% | 2.1 GB | 1,723 |
| 2019_21_Veloster_CAN | 5.6% | 2.1 GB | 1,723 |
| 2023_Soul_CAN | 1.6% | 0.6 GB | 598 |
| 2022_Forte_4Door_CAN | 1.2% | 0.4 GB | 3 |
| **Total** | **100.0%** | **37.5 GB** | **18,665** |

# Appendix B: AAOS Build Targets Used in Dataset Construction

This appendix summarizes the AAOS build targets retained in the benign AAOS corpus. For each target, the table reports the relative share shown in the archive view, the approximate on-disk size, and the extracted file count observed during dataset preparation.

## B.1 AAOS Build Targets Included in the Dataset

Table B.1: AAOS build targets retained in the benign dataset.

| Build Target | Share | Size | Items | Files |
|---|---|---|---|---|
| aosp_cf_x86_64_auto | 20.4% | 23.6 GB | 12,959 | 11,532 |
| aosp_cheetah_car | 13.0% | 15.0 GB | 11,373 | 10,314 |
| aosp_panther_car | 12.9% | 15.0 GB | 11,376 | 10,317 |
| aosp_oriole_car | 12.7% | 14.7 GB | 11,340 | 10,296 |
| aosp_sunfish | 10.1% | 11.6 GB | 9,660 | 8,733 |
| aosp_tangorpro_car | 9.4% | 10.9 GB | 7,999 | 7,201 |
| aosp_bluejay_car | 8.5% | 9.8 GB | 7,556 | 6,862 |
| aosp_cf_x86_64_auto-partial | 5.6% | 6.4 GB | 5,717 | 5,083 |
| aosp_bramble_car | 4.9% | 5.7 GB | 3,418 | 3,082 |
| sdk_car_x86_64 | 2.5% | 2.9 GB | 42 | 28 |
| **Total** | **100.0%** | **115.8 GB** | **81,454** | **73,448** |

## B.2 AAOS Processed OTA Update Sizes by Normalized Family

Table B.2: Processed AAOS OTA update sizes and file counts by normalized family.

| **Update Set** | **Share** | **Size** | **Count** |
|---|---|---|---|
| `aosp_cf_x86_64_auto` | 15.2% | 5.5 GB | 10,210 |
| `aosp_panther_car` | 13.7% | 5.0 GB | 9,165 |
| `aosp_cheetah_car` | 13.7% | 5.0 GB | 9,165 |
| `aosp_oriole_car` | 13.1% | 4.8 GB | 9,144 |
| `aosp_sunfish` | 10.7% | 3.9 GB | 7,647 |
| `aosp_tangorpro_car` | 9.9% | 3.6 GB | 6,215 |
| `aosp_bluejay_car` | 8.7% | 3.2 GB | 6,092 |
| `aosp_cf_x86_64_auto-partial` | 6.3% | 2.3 GB | 4,438 |
| `sdk_car_x86_64` | 4.3% | 1.6 GB | 16 |
| `aosp_bramble_car` | 4.3% | 1.6 GB | 2,710 |
| **Total** | **100.0%** | **36.5 GB** | **64,802** |

# Appendix C: Baseline Statistics

## C.1 Baseline-model results for modified experiments

Table C.1: Complete baseline-model results for modified experiments.

| Category | Condition | Model | AUROC | Precision | Recall | F1 |
|---|---|---|---|---|---|---|
| Raw | All Malware | OCSVM | 0.911 | 0.838 | 0.923 | 0.878 |
| Raw | All Malware | PCA | 0.883 | 0.838 | 0.868 | 0.853 |
| Raw | All Malware | VAE | 0.837 | 0.788 | 0.919 | 0.849 |
| Raw | All Malware | f-AnoGAN | 0.863 | 0.796 | 0.799 | 0.798 |
| Raw | All Malware | Isolation Forest | 0.830 | 0.802 | 0.685 | 0.739 |
| Raw | Android Malware | OCSVM | 0.923 | 0.753 | 0.949 | 0.840 |
| Raw | Android Malware | PCA | 0.899 | 0.761 | 0.922 | 0.834 |
| Raw | Android Malware | VAE | 0.887 | 0.733 | 0.969 | 0.835 |
| Raw | Android Malware | f-AnoGAN | 0.885 | 0.743 | 0.785 | 0.763 |
| Raw | Android Malware | Isolation Forest | 0.795 | 0.877 | 0.522 | 0.654 |
| Raw | Linux Malware | OCSVM | 0.895 | 0.813 | 0.801 | 0.807 |
| Raw | Linux Malware | PCA | 0.860 | 0.678 | 0.786 | 0.728 |
| Raw | Linux Malware | VAE | 0.768 | 0.562 | 0.932 | 0.701 |
| Raw | Linux Malware | f-AnoGAN | 0.832 | 0.567 | 0.858 | 0.683 |
| Raw | Linux Malware | Isolation Forest | 0.877 | 0.631 | 0.873 | 0.733 |
| Salted | All Malware 0.1% Salt | OCSVM | 0.796 | 0.912 | 0.608 | 0.729 |
| Salted | All Malware 0.1% Salt | PCA | 0.863 | 0.832 | 0.912 | 0.870 |
| Salted | All Malware 0.1% Salt | VAE | 0.896 | 0.867 | 0.976 | 0.918 |
| Salted | All Malware 0.1% Salt | f-AnoGAN | 0.805 | 0.818 | 0.705 | 0.757 |
| Salted | All Malware 0.1% Salt | Isolation Forest | 0.738 | 0.945 | 0.361 | 0.522 |
| Salted | All Malware 5% Salt | OCSVM | 0.783 | 0.911 | 0.599 | 0.723 |
| Salted | All Malware 5% Salt | PCA | 0.863 | 0.824 | 0.942 | 0.879 |
| Salted | All Malware 5% Salt | VAE | 0.897 | 0.866 | 0.975 | 0.918 |
| Salted | All Malware 5% Salt | f-AnoGAN | 0.798 | 0.784 | 0.781 | 0.782 |

| Category | Condition | Model | AUROC | Precision | Recall | F1 |
|---|---|---|---|---|---|---|
| Salted | All Malware 5% Salt | Isolation Forest | 0.741 | 0.922 | 0.376 | 0.535 |
| Salted | All Malware 10% Salt | OCSVM | 0.775 | 0.909 | 0.595 | 0.719 |
| Salted | All Malware 10% Salt | PCA | 0.864 | 0.825 | 0.946 | 0.882 |
| Salted | All Malware 10% Salt | VAE | 0.898 | 0.866 | 0.977 | 0.918 |
| Salted | All Malware 10% Salt | f-AnoGAN | 0.790 | 0.780 | 0.766 | 0.773 |
| Salted | All Malware 10% Salt | Isolation Forest | 0.745 | 0.945 | 0.361 | 0.522 |
| Salted | Linux Salt 0.1% | OCSVM | 0.700 | 0.822 | 0.473 | 0.600 |
| Salted | Linux Salt 0.1% | PCA | 0.836 | 0.716 | 0.894 | 0.795 |
| Salted | Linux Salt 0.1% | VAE | 0.902 | 0.788 | 0.983 | 0.875 |
| Salted | Linux Salt 0.1% | f-AnoGAN | 0.744 | 0.658 | 0.710 | 0.683 |
| Salted | Linux Salt 0.1% | Isolation Forest | 0.694 | 0.654 | 0.425 | 0.516 |
| Salted | Linux Salt 5% | OCSVM | 0.676 | 0.818 | 0.458 | 0.587 |
| Salted | Linux Salt 5% | PCA | 0.835 | 0.719 | 0.903 | 0.800 |
| Salted | Linux Salt 5% | VAE | 0.904 | 0.790 | 0.980 | 0.875 |
| Salted | Linux Salt 5% | f-AnoGAN | 0.732 | 0.649 | 0.702 | 0.674 |
| Salted | Linux Salt 5% | Isolation Forest | 0.701 | 0.612 | 0.527 | 0.566 |
| Salted | Linux Salt 10% | OCSVM | 0.664 | 0.814 | 0.446 | 0.577 |
| Salted | Linux Salt 10% | PCA | 0.838 | 0.716 | 0.925 | 0.807 |
| Salted | Linux Salt 10% | VAE | 0.906 | 0.791 | 0.981 | 0.876 |
| Salted | Linux Salt 10% | f-AnoGAN | 0.718 | 0.634 | 0.698 | 0.664 |
| Salted | Linux Salt 10% | Isolation Forest | 0.706 | 0.551 | 0.817 | 0.658 |
| Patch | All Malware Patch | OCSVM | 0.809 | 0.755 | 0.632 | 0.688 |
| Patch | All Malware Patch | PCA | 0.867 | 0.587 | 0.916 | 0.715 |
| Patch | All Malware Patch | VAE | 0.895 | 0.650 | 0.982 | 0.782 |
| Patch | All Malware Patch | f-AnoGAN | 0.812 | 0.520 | 0.800 | 0.631 |
| Patch | All Malware Patch | Isolation Forest | 0.758 | 0.793 | 0.426 | 0.554 |
| Patch | Linux Malware Patch | OCSVM | 0.696 | 0.536 | 0.470 | 0.501 |
| Patch | Linux Malware Patch | PCA | 0.836 | 0.387 | 0.890 | 0.539 |
| Patch | Linux Malware Patch | VAE | 0.901 | 0.484 | 0.978 | 0.648 |
| Patch | Linux Malware Patch | f-AnoGAN | 0.742 | 0.323 | 0.704 | 0.443 |
| Patch | Linux Malware Patch | Isolation Forest | 0.691 | 0.275 | 0.521 | 0.360 |

# Appendix D: Public Examples of Automotive IVI Software Platforms

## D.1 Public Examples of Automotive IVI Software Platforms

Table D.1: Public examples of OEMs and brands associated with Android-based or QNX-based automotive software platforms.

| Company / Brand | Operating System / Platform |
|---|---|
| Tesla | Custom Linux-based infotainment / software platform [76, 77] |
| Waymo (Google) | Android-related in-car software work; evidence is primarily from public job postings rather than a formal platform announcement [78] |
| General Motors | Android-based infotainment; QNX-related automotive deployments are also documented by BlackBerry QNX [79, 80] |
| Volkswagen Group | QNX-based automotive deployments are documented by BlackBerry QNX; Volkswagen also joined Automotive Grade Linux [80, 81] |
| Toyota | QNX-based automotive deployments documented by BlackBerry QNX [80] |
| Honda | Google built-in / Android-based infotainment; QNX-related automotive deployments are also documented by BlackBerry QNX [80, 82] |

| **Company / Brand** | **Operating System / Platform** |
|---|---|
| Acura | Google built-in / Android-based infotainment [83, 84] |
| BMW | QNX-based automotive deployments documented by BlackBerry QNX [80] |
| Bosch | BlackBerry QNX automotive Tier-1 supplier [80] |
| Continental | BlackBerry QNX automotive Tier-1 supplier [80] |
| Dongfeng Motor | QNX-based automotive deployments documented by BlackBerry QNX [80] |
| Ford | Google built-in / Android-powered vehicle experience; QNX-related automotive deployments are also documented by BlackBerry QNX [80, 85] |
| Lincoln | Google built-in / Android-based Lincoln Digital Experience [86, 87] |
| Mercedes-Benz | QNX-based automotive deployments documented by BlackBerry QNX [80] |
| Hyundai | QNX-based automotive deployments documented by BlackBerry QNX [80] |
| Jaguar Land Rover | QNX-based automotive deployments documented by BlackBerry QNX [80] |
| Fiat Chrysler | QNX-based automotive deployments documented by BlackBerry QNX [80] |
| Volvo Cars | Android Automotive / Google built-in [88] |
| Polestar | Android Automotive / Google built-in [89] |
| Renault Group | Android Automotive / Google built-in / Android-based car OS [90, 91] |
| Nissan | Google built-in / Android-based infotainment [92, 93] |
| INFINITI | Google built-in / Android-based infotainment [94, 95] |

| **Company / Brand** | **Operating System / Platform** |
|---|---|
| Mitsubishi Motors | Google built-in / Android-based infotainment [96, 97] |
| Mazda | Google built-in / Android-based infotainment [98] |
| Chevrolet | Google built-in / Android-based infotainment [99] |

# Appendix E: Ablation Results

## E.1 Reptile Size and Entropy Ablation Results

Table E.1: Comparison of the baseline 298-dimensional Reptile model with the size-ablated and entropy-ablated variants at support size 64. Reported values are mean scores across the evaluated malware and injected-anomaly settings.

| | Baseline | | | Size Ablated | | | Entropy Ablated | | |
|---|---|---|---|---|---|---|---|---|---|
| Condition | F1 | AUROC | AUPRC | F1 | AUROC | AUPRC | F1 | AUROC | AUPRC |
| AllMalware0001 | 0.619 | 0.725 | 0.861 | 0.525 | 0.670 | 0.836 | 0.148 | 0.561 | 0.720 |
| AllMalware005 | 0.618 | 0.725 | 0.861 | 0.522 | 0.666 | 0.833 | 0.139 | 0.557 | 0.714 |
| AllMalware010 | 0.619 | 0.726 | 0.861 | 0.527 | 0.689 | 0.848 | 0.137 | 0.553 | 0.709 |
| AllMalwarep64k | 0.622 | 0.731 | 0.864 | 0.528 | 0.679 | 0.844 | 0.126 | 0.538 | 0.697 |
| AndroidMalware | 0.545 | 0.658 | 0.826 | 0.562 | 0.708 | 0.842 | 0.088 | 0.566 | 0.739 |
| LinuxMalware | 0.531 | 0.710 | 0.861 | 0.707 | 0.873 | 0.948 | 0.006 | 0.501 | 0.632 |
| p64kCombined | 0.664 | 0.778 | 0.878 | 0.477 | 0.621 | 0.791 | 0.103 | 0.560 | 0.696 |
| r0001Combined | 0.655 | 0.774 | 0.873 | 0.494 | 0.623 | 0.807 | 0.119 | 0.555 | 0.692 |
| r005Combined | 0.654 | 0.771 | 0.873 | 0.494 | 0.645 | 0.816 | 0.118 | 0.548 | 0.681 |
| r010Combined | 0.653 | 0.771 | 0.872 | 0.498 | 0.660 | 0.818 | 0.108 | 0.542 | 0.672 |
| Mean | 0.618 | 0.737 | 0.861 | 0.533 | 0.683 | 0.840 | 0.109 | 0.548 | 0.695 |

# Curriculum Vitae

| | |
|---|---|
| **Name:** | Darwin Liao |
| **Post-Secondary Education and Degrees:** | University of Western Ontario<br>London, ON, CA<br>2020 - 2024 B.Sc |
| **Related Work Experience:** | Teaching Assistant<br>The University of Western Ontario<br>2024 - 2026 |

**Publications:**

1. D. Liao and M. A. Elsayed, ”OTArmor: Securing Automotive Over-the-Air Updates Against Malware Using Generative Modeling,” 2025 9th Cyber Security in Networking Conference (CSNet), Abu Dhabi, United Arab Emirates, 2025, pp. 207-214, doi: 10.1109/CSNet67572.2025.11288195.