Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

An Intelligent Stacked Autoencoder Framework for Android Malware Detection: Static Analysis Empowering Cybersecurity

Marziyh Khavari-Estahbanati
Islamic Azad University
Reza Ahsan
reza.ahsan@iau.ac.ir

Islamic Azad University
Mostafa Ghobai-Arani
Islamic Azad University

Research Article

Keywords: Android Malware Detection, Static Analysis, Sparse Stacked Autoencoder, Classification, Feature selection, Cybersecurity

Posted Date: November 20th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8908567/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

An Intelligent Stacked Autoencoder Framework for Android Malware Detection: Static Analysis Empowering Cybersecurity

Marziyh Khavari-Estahbanati [1], Reza Ahsan[2]*, Mostafa Ghobai-Arani[3]

[1,2,3] Department of Computer Engineering, Qom Branch, Islamic Azad University, Qom, Iran

Abstract.

Android has become the most widely utilized mobile platform worldwide, making it a primary target for malware authors. Such threats pose significant risks to user privacy and sensitive data, while the growing sophistication of Android malware has diminished the effectiveness of traditional signature-based detection methods. To address these challenges, this study proposes a hierarchical static analysis framework that leverages sparse stacked autoencoders to enhance Android malware detection.

Experiments were performed on the TUANDROMD dataset, comprising 4,465 Android applications, including 3,565 malware samples and 899 benign apps. The proposed framework consists of three stages: (i) selection of discriminative features, (ii) hierarchical feature extraction using sparse autoencoders, and (iii) final classification with stacked autoencoders followed by fine-tuning. The system achieved 98.9% accuracy, with Precision = 98.9%, Recall = 99.7%, and F1-score = 98.3%, including a notably high recall of 99.7% which minimizes false negatives. Comparative assessment confirmed superior performance over conventional machine learning and clustering approaches. Beyond outperforming traditional methods, the framework demonstrated scalability, resilience against obfuscation, as well as strong potential for deployment in large-scale mobile security systems. These findings establish TUANDROMD as a benchmark dataset for future deep learning research and highlight stacked autoencoders as a promising solution for advancing Android malware detection.

Keywords: Android Malware Detection, Static Analysis, Sparse Stacked Autoencoder, Classification, Feature selection, Cybersecurity..

1 Introduction

Android has become the most widely adopted mobile operating system, whose openness, device fragmentation, and flexible application ecosystem have made it a prime target for cybercriminals. The rapid proliferation of Android devices has been accompanied by a significant growth in sophisticated malware families, including ransomware, spyware, and banking trojans. All of these increasingly leverage advanced evasion and obfuscation techniques to bypass traditional defenses . As a result,

1 **Corresponding Author: Tel:09127503591

E-mail address: Reza.ahsan@iau.ac.ir

Android malware detection has emerged as a critical research area within mobile cybersecurity, demanding intelligent as well as scalable solutions capable of addressing the evolving threat landscape.

In spite of continuous improvements in platform-level security, such as Google Play Protect and enhanced application vetting mechanisms, numerous studies have indicated that these systems remain insufficient against zero-day malware, heavily obfuscated applications, and threats distributed through third-party marketplaces. Modern malware increasingly utilizes dynamic payload loading, reflection, encryption, and emulator detection to evade both static and dynamic analysis . These challenges underscore the limitations of conventional signature-based and heuristic-based approaches and emphasize the need for more intelligent, adaptive, and robust detection frameworks.

Traditional malware analysis techniques are generally classified into static, dynamic, and hybrid approaches. Static analysis offers scalability and efficiency through examining application code without execution, but it is vulnerable to code obfuscation and transformation attacks Dynamic analysis provides deeper behavioral insights while suffering from high computational overhead and susceptibility to anti-analysis strategies . Hybrid approaches attempt to combine the strengths of both methods, yet they often confront challenges pertained to resource consumption and scalability .Thus, researchers are increasingly turning to machine learning and deep learning to extract discriminative as well as resilient patterns from application features. Deep neural networks, including CNNs, RNNs, GNNs, and ensemble learning models, have presented promising performance in Android malware detection .

Nevertheless, a key research gap persists: the need for robust, generalizable, and compact feature representations capable of detecting zero-day and heavily obfuscated malware through static analysis alone. Many existing deep learning approaches rely on high-dimensional handcrafted features or require extensive dynamic execution traces, which restrict scalability and real-world applicability. Moreover, lack of hierarchical feature extraction mechanisms in many models constraints their ability to capture complex structural relationships within Android applications.

The novelty of this study lies in proposing an intelligent stacked autoencoder framework which leverages static analysis to empower Android cybersecurity. The framework integrates sparse autoencoders, hierarchical feature learning, and advanced feature selection techniques to construct deep, compact, and highly discriminative representations of Android applications. Through progressively learning abstract feature hierarchies, the proposed model boosts robustness against obfuscation and ameliorates generalization to unseen malware variants . This design enables efficient and scalable malware detection without relying on dynamic execution, making it appropriate for deployment in large-scale mobile security environments.

In order to ascertain the effectiveness of the proposed framework, the TUANDROMD dataset—comprising 4,465 Android applications, including 3,565 malware and 899 benign samples—was employed **.** Experimental results indicated that the system could achieve a detection accuracy of 98.9%, outperforming conventional machine learning approaches and underscoring the potential of stacked autoencoders in strengthening Android malware detection. This contribution not only

validates the applicability of deep representation learning to TUANDROMD but also establishes a new benchmark for future research on intelligent malware detection systems.

Research Objectives

The primary objective of this study is to advance Android malware detection through introducing a stacked autoencoder framework which leverages static analysis for robust and scalable feature representation. Specifically, the research aims to:

1. apply autoencoder-based architectures to the TUANDROMD dataset for the first time, addressing a clear gap in the literature where only classical machine learning and clustering methods have been explored.

2. construct hierarchical and compact feature representations that boost resilience against obfuscation and improve generalization to unseen malware families.

3. appraise the effectiveness of the proposed framework in comparison with conventional machine learning approaches, presenting superior detection accuracy and scalability.

4. highlight the applicability of deep representation learning in real-world mobile security environments, establishing TUANDROMD as a benchmark dataset for future autoencoder-based studies.

Having outlined the objectives and highlighted the novelty of applying stacked autoencoders to the TUANDROMD dataset, the following section presents a comprehensive literature review of Android malware detection approaches, datasets, and deep learning techniques that constitute the foundation of this study.

This paper is organized as follows. Section 1 introduces the research problem and motivation. Section 2 deals with the literature review, consisting of two parts: Background and Related Work. Section 3 presents the dataset and the proposed methodology, detailing the stacked autoencoder-based framework and describing the structure and distribution of malware and benign samples utilized for evaluation. Section 4 focuses on model performance evaluation, outlining the employed metrics, experimental setup, and comparative assessment of detection effectiveness. Ultimately, Section 5 provides the conclusion and future work, summarizing the key contributions of the study as well as highlighting potential directions for advancing Android malware detection research.

2 Literature Review

2-1 Background

2-1-1 Android Operating System

The Android operating system, developed by Google, is built upon the Linux kernel and optimized for ARM architectures widely employed in mobile devices. Its architecture comprises four primary layers :

- **Kernel layer:** It provides core functionalities such as memory management, process control, security enforcement, and hardware abstraction.
- **Native libraries and runtime:** They include C/C++ libraries and the runtime environment, either Dalvik Virtual Machine (DVM) or Android Runtime (ART).
- **Application framework:** It offers APIs for managing activities, services, notifications, and other system components.
- **Application layer:** It consists of user-facing applications, both pre-installed and third-party.

2-1-2 Malware

Malware refers to software intentionally designed to disrupt, damage, or exploit systems, data, or users. The open-source nature and widespread adoption of Android make it particularly vulnerable. According to reports, cyberattacks targeting Android have grown by up to 40% annually since 2016 . Malware manifests in diverse forms—trojans, botnets, spyware, ransomware, and adware—capable of data theft, unauthorized access, remote control, and financial exploitation.

2-1-3 Android Malware Analysis

Malware analysis identifies threats and vulnerabilities through static, dynamic, hybrid, and manual approaches .

- **Static analysis:** It examines internal structure without executing code.
- **Dynamic analysis:** It observes behavior along execution.
- **Hybrid analysis:** It combines static and dynamic approaches.
- **Manual analysis:** It is expert-led reverse engineering of APKs .

The methods of malware analysis are categorized in Figure 1.

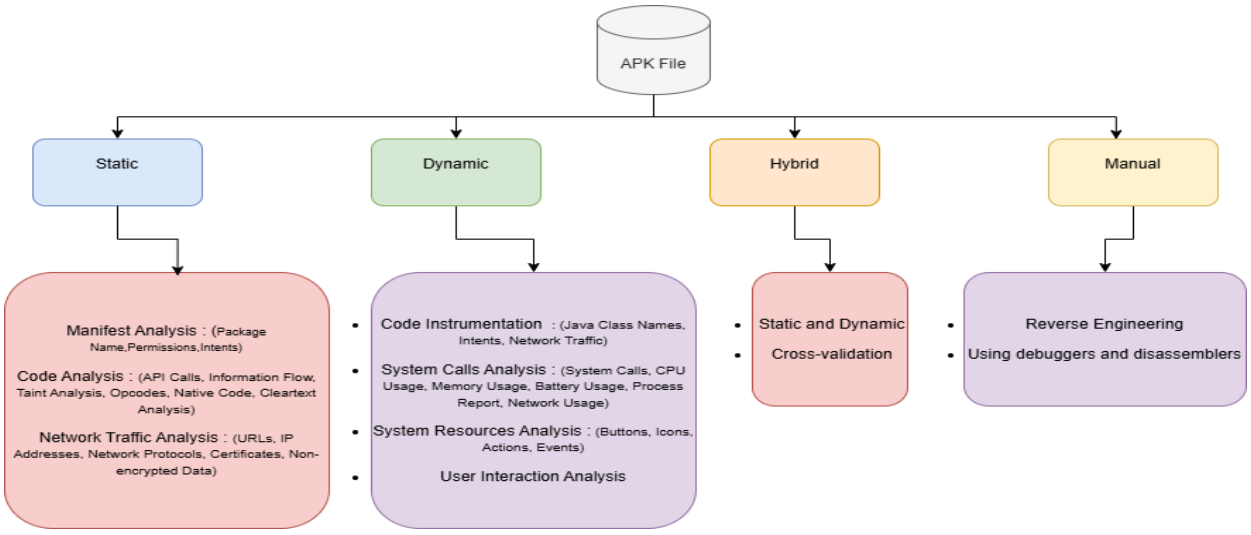


Figure 1. Malware analysis techniques.

2-1-4 Android Malware Detection

Traditional signature-based detection fails against obfuscated or novel malware . Machine learning approaches analyze behavioral patterns and static features, providing superior accuracy.

2-1-5 Malware Detection Tools

Tools such as firewalls, antivirus software, behavior analyzers, network monitors, and sandbox environments constitute the first line of defense . Intrusion Detection Systems (IDS) monitor traffic and application behavior, while neural networks boost predictive capabilities .

2-1-6 Autoencoders: An Approach to Unsupervised Representation Learning in Neural Networks

Autoencoders refer to a class of feedforward neural networks designed to learn compressed and meaningful representations of data without requiring labeled inputs. Through enforcing a bottleneck layer, they capture the most salient features of the input as well as reconstruct the original data with minimal loss, enabling the discovery of latent structures in data. This self-supervised learning paradigm has been widely employed in dimensionality reduction, feature extraction, and pre-training deep architectures. Overall, the learning process of the autoencoder, as defined by Equation 1, maps the input into a hidden representation and reconstructs it:

$$\boldsymbol{h} = \boldsymbol{\sigma}(\boldsymbol{xW} + \boldsymbol{b}), \widehat{\boldsymbol{x}} = \boldsymbol{\sigma}(\boldsymbol{hW'} + \boldsymbol{b'})$$

Equation 1: Mapping the input to a hidden representation and reconstructing it in the autoencoder.

Several variants of autoencoders have been developed to boost representation learning, including **undercomplete autoencoders**, forcing compression through a small latent space; **sparse autoencoders**, encouraging sparsity in hidden activations; **denoising autoencoders**, reconstructing clean inputs from corrupted data; **contractive autoencoders**, regularizing sensitivity to perturbations; and **convolutional autoencoders**, leveraging convolutional layers for image processing tasks. These architectures extend the applicability of autoencoders across diverse domains, making them a powerful tool for bridging unsupervised and supervised learning in modern deep learning frameworks .

2-2 Related Work

The bedrock of robust network security architectures is built upon foundational mechanisms, including firewalls and comprehensive monitoring tools, which establish the initial defensive perimeter . Central to proactive threat detection are **Intrusion Detection Systems (IDS)**, tasked with meticulously analyzing network traffic for anomalous patterns .However, the landscape of security analytics has been profoundly reshaped by the advent of advanced computational paradigms. Contemporary research predominantly converges on the transformative potential of **neural networks and sophisticated machine learning models**. These techniques have demonstrably amplified the predictive acuity and adaptive capabilities of detection systems, thereby enabling a paradigm shift

towards proactive mitigation of emergent threats, including sophisticated zero-day exploits that consistently challenge conventional security measures .

2-2-1 Related Work and Deep Learning Approaches

Deep learning has gained traction in malware detection, with architectures such as CNN, RNN, GNN, and hybrid models (LSTM–CNN, SAE–LSTM).

- A hybrid SAE–LSTM model was proposed for ransomware detection based on the UGRansom dataset .

- Autoencoders combined with ensemble classifiers (SVM, Random Forest) ameliorated detection accuracy through iterative feedback .

- Stacked Denoising Autoencoders (SDAEs) extracted robust features from Hardware Performance Counters .

- Static analysis of Windows PE files using Autoencoders lowered computational complexity while improving accuracy .

- Hybrid LSTM–CNN models converted executables into grayscale images, employing PCA for dimensionality reduction .

- SAEs were utilized to detect anomalies in Android malware datasets, focusing on repackaged malware .

- SDAEs augmented robustness against noise in Windows executables .

As summarized in Table 1, SAE-based models consistently outperform traditional methods but confront scalability challenges when applied to large datasets. Table 1 also highlights that while SDAE approaches provide robustness against obfuscation, their dependency on hardware performance counters restricts applicability in mobile environments.

Table 1. Comparative summary of deep learning-based malware detection methods.

Study	Year	Approach	Dataset	Key Findings	Advantages	Limitations	Challenges
	2021	SAE–LSTM hybrid	UGRansom	Effective ransomware detection	Strong feature extraction	Limited to ransomware dataset	Generalization
	2021	Autoencoder + Ensemble	Custom	Improved accuracy	Robustness via ensemble	Retraining overhead	Computational cost
	2022	SDAE	HPC	Robust features	Resistant to transformations	Hardware-dependent	Mobile applicability
	2022	Autoencoder (PE static)	Windows PE	Reduced complexity	Lightweight	Windows-focused	Transferability
	2022	LSTM–CNN + PCA	Executable images	High accuracy	Combines features	Needs conversion	High cost
	2023	SAE anomaly detection	Android datasets	Detects repackaged malware	Subtle anomaly detection	Relies on dynamic analysis	Scalability
	2023	SDAE	Windows executables	Noise resilience	Robustness	Limited diversity	Adversarial vulnerability

2-2-2Static Analysis and Stacked Autoencoders

Static analysis remains central to Android malware detection thanks to efficiency and scalability.

- Arp et al. (2014): Introduced the Drebin dataset, enabling feature extraction from permissions and API calls .

- Onwuzurike et al. (2020): Proposed MaMaDroid, modeling API sequences as Markov chains .

- Talbi et al. (2022): Combined feature selection with deep learning, attaining 99.3% accuracy on Drebin .

- Brahami et al. (2023): Applied SAE for dimensionality reduction and CNN classification, achieving 98.5% accuracy on Drebin-215 .

- Chen et al. (2023): Introduced continuous learning frameworks via APIGraph and AndroZoo datasets .

Table 2 provides a comparative overview of datasets applied in SAE-based studies, highlighting both their advantages and inherent limitations. As outlined in Table 2, Drebin remains the benchmark dataset for static feature extraction, but its limited size restricts generalization to modern malware. Table 2 further indicates that AndroZoo provides large-scale diversity, yet preprocessing and labeling challenges hamper its direct usability for SAE-based models.

Table 2. Datasets used in SAE-based Android malware detection.

Study	Year	Dataset	Description	Usage	Advantages	Limitations	Challenges
	2014	Drebin	5,500+ labeled samples	Benchmark static features	Widely adopted	Small, outdated	Obfuscation resistance
	2017+	AndroZoo	Millions of APKs	Large-scale eval	Diverse, real-world	Imbalanced classes	Preprocessing overhead
	2023	APIGraph	Graph-based APIs	Continuous learning	Captures relations	Complex extraction	SAE integration
	2018	DroidCat	Call graphs	Anomaly detection	Specialized	Limited size	Generalization
	2019	RevealDroid	Static features	Specialized	Rich features	Narrow scope	Applicability

2-2-3 Static Analysis and Deep Learning in Android Malware Detection

Static analysis has been widely adopted in Android malware detection owing to its scalability and efficiency. Nevertheless, its vulnerability to obfuscation has motivated researchers to explore deep learning approaches, especially autoencoders, for robust feature representation. Benchmark datasets such as Drebin and TUANDROMD have played a key role in appraising these methods. Drebin, introduced in 2014, has been extensively researched, while TUANDROMD, introduced later, remains relatively underexplored.

2-2-3-1 Studies on Drebin Dataset

The Drebin dataset has become a benchmark for static analysis research. Over the years, various studies have leveraged Drebin to test classical machine learning algorithms, deep learning models, and hybrid approaches.

Table 3 summarizes representative studies performed on the Drebin dataset, highlighting the evolution from classical machine learning approaches to advanced deep learning as well as hybrid models.

Table 3. Representative studies on Drebin dataset.

Study	Year	Approach	Features	Key Findings	Advantages	Limitations	Challenges
	2014	SVM, Linear	Permissions + APIs	Introduced Drebin	Benchmark dataset	Limited size	Obfuscation
	2022	Feature Selection + SAE	Permissions + APIs	99.3% accuracy	Robust reduction	Needs tuning	Generalization
	2023	SAE + CNN	Permissions + APIs	98.5% accuracy	Hierarchical learning	Overhead	Scalability
	2023	Continuous Learning	Graph APIs	Integrated Drebin	Captures relations	Complex preprocessing	Concept drift
	2023	DBSCAN clustering	Permissions + APIs	Anomaly detection	Hidden patterns	Sensitive parameters	Sparse features
	2025	CNN+XGBoost, KNN+XGBoost	Permissions + APIs	CNN+XGBoost: 98.76%	Combines methods	High resources	Imbalance
	2024	LOF	Permissions + APIs	Reduced false positives	Real-world suitable	Limited scalability	Parameter sensitivity

2-2-3-2 Studies on TUANDROMD Dataset

In contrast to Drebin, the TUANDROMD dataset has received sparse attention. Existing studies have primarily applied classical machine learning algorithms and clustering methods. Table 4 summarizes representative studies undertaken on the TUANDROMD dataset.

Table 4. Representative studies on TUANDROMD dataset.

Study	Year	Approach	Features	Key Findings	Advantages	Limitations	Challenges
	2020	SVM, RF, k-NN	Permissions + APIs	Baseline ML detection	Established benchmark	Classical ML only	Obfuscation
	2021	Extended ML eval	Permissions + APIs	RF best performance	Comprehensive baseline	No deep learning	Limited generalization

2-2-4 Research Gaps

In spite of promising results, several challenges remain:

- Scalability: SAE models need to be optimized for massive datasets such as AndroZoo.

- Obfuscation resistance: Malware packing and code transformations distort static features.
- Real-time detection: Lightweight SAE-based models are required for deployment in mobile environments.
- Explainability: The black-box nature of SAE requires integration with feature importance analysis.
- Concept drift: Continuous learning mechanisms should complement SAE models to adapt to evolving malware families.

TUANDROMD gap: Up to 2024–2025, no published studies have applied autoencoder-based architectures to TUANDROMD. Existing research is limited to classical ML and clustering. This absence highlights the novelty of the present study.

Accordingly, this research introduces stacked autoencoders on TUANDROMD for the first time, establishing a new benchmark in Android malware detection as well as directly addressing the identified gap.

3 The Proposed Methodology (Stacked Autoencoder-Based Framework)

The proposed method is built upon a deep learning architecture employing Stacked Autoencoders (SAE) for Android malware detection. The framework comprises several sequential stages designed to ensure robust feature extraction, model training, and deployment. These stages include:

1. **Data Preprocessing:** Cleaning and transforming raw data to ensure consistency and suitability for analysis.
2. **Feature Selection:** Identifying the most relevant features that contribute to malware classification.
3. **Feature Extraction:** Employing autoencoder networks for hierarchically extracting latent representations from the selected features.
4. **Data Splitting:** Splitting the dataset into training and testing subsets to ascertain the model performance.
5. **Model Training:** Training the deep learning model using the training set in order to learn discriminative patterns.
6. **Model Evaluation:** Assessing the model's accuracy, precision, recall, and other performance metrics via the test set.
7. **Result Analysis:** Interpreting the evaluation outcomes to understand the model's strengths as well as limitations.
8. **Model Deployment:** Integrating the trained model into a real-world environment for malware detection.

The overall architecture of the proposed SAE-based framework is displayed in Figure 2.

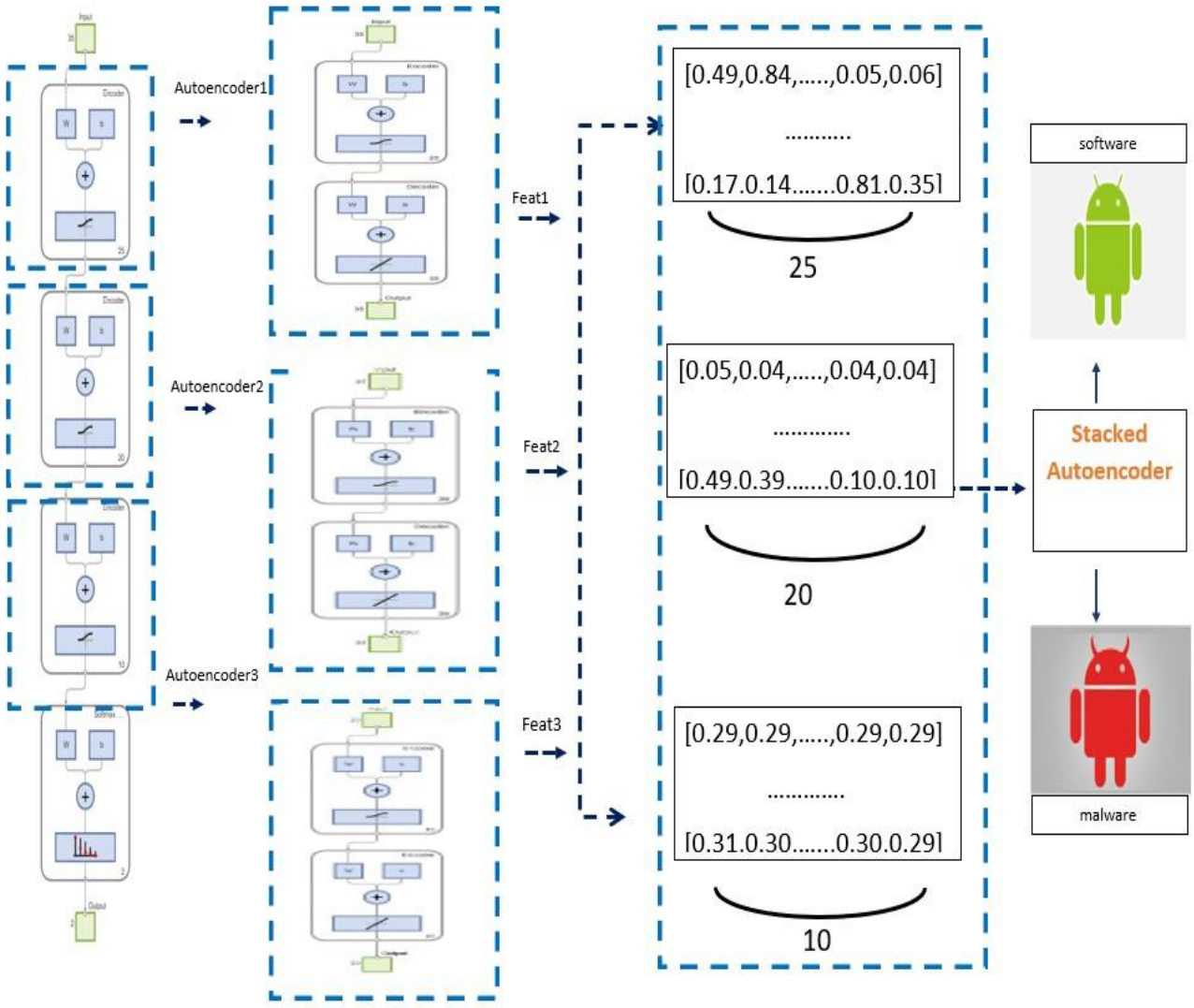


Figure 2. Stacked Autoencoder Architecture.

3-1 Dataset Description

One of the foundational steps in this study was acquiring a suitable dataset for analysis and assessment of the proposed methods. For identifying an appropriate dataset, we reviewed prior research in the field and inspected referenced sources across multiple studies. accordingly, two datasets were collected, with TUANDROMD selected as the primary resource for experimentation.

The TUANDROMD dataset is a prominent benchmark in Android security research, specifically designed to distinguish between malware and goodware applications. Its advantages including diversity of malware samples, rigorous feature engineering, and reliance on validated analytical techniques make it a powerful tool for training and ascertaining machine learning models in the

context of Android malware detection. Further, its freshness and updateability render it highly reliable for addressing emerging Android threats.

Key Specifications of TUANDROMD:

- **Total Samples:** 4,465 Android applications
  - **Malware:** 3,565 samples
  - **Goodware:** 899 samples
- **Number of Features:** 241
- **Data Type:** Boolean (indicating presence or absence of specific features)
- **Target Variable:** label (reflecting whether an application is malware or goodware)
- **Independent Variables:** 241 features per sample
- **Dependent Variable:** Malware classification label

The research methodology applied in this study is descriptive–analytical, with simple random sampling utilized for data selection. The dataset was constructed to reflect realistic and malicious behaviors of Android applications through extracting key features such as system permissions, API calls, and structural as well as behavioral attributes.

The framework for creating and preparing the TUANDROMD dataset is depicted in Figure 3, encompassing the stages of data collection, data storage, data analysis, and feature engineering.

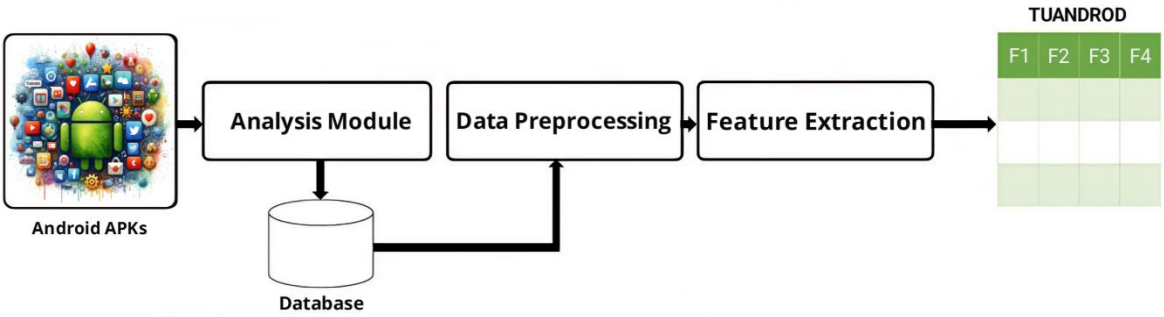


Figure 3. Data Preparation Framework.

3-1-1 Dataset Construction Phases

The development of the TUANDROMD dataset involved a structured three-phase process: i) data collection and storage, ii) data analysis, and iii) feature engineering. Each phase was designed to ensure the dataset accurately mirrors real-world Android application behaviors and supports robust malware detection research.

Phase 1: Data Collection and Storage

In order to construct a comprehensive Android malware dataset, we collected both malicious and benign application binaries. Specifically, we gathered 24,553 raw malware binaries from various sources along with 1,000 top-rated Android applications from the Google Play Store as benign samples. All collected applications were stored on a dedicated database server for further processing and analysis.

Phase 2: Data Analysis

The collected Android malware samples were manually analyzed using a suite of tools, including Androguard, ApkInspector, ApkAnalyser, and Smali-CFGs. Each malware variant was analyzed within the context of its respective family. In order to ensure a secure and controlled environment, all analysis tasks were carried oout in an isolated virtual environment using VirtualBox, with Windows 7 configured as the guest operating system.

Phase 3: Feature Engineering

In the feature engineering phase, two primary categories of features were extracted: permission-based features and API-based features.

- Permission-based features include all installation-time and runtime permissions required for the application to operate properly on an Android device.
- API-based features encompass all API calls invoked by the application to undertake user-facing or background tasks.

3-1-2 Some of the Most Important Features Include:

1. **Hardware components** employed to configure hardware-level permissions required by the application.
2. **Requested permissions** granted by users along installation to access system resources.
3. **Application components**, including Activities, Services, Content Providers, and Broadcast Receivers—the four core Android component types.
4. **Intent filters** utilized for inter-process communication between applications and components.
5. **Critical API calls** accessed via a limited set of exposed APIs.
6. **Suspicious API queries**, resource requests, and access to sensitive data.
7. **Network addresses**, including IP addresses, hostnames, and URLs accessed by the application.

This multi-layered feature extraction process ensures that the dataset captures both structural and behavioral characteristics of Android applications. This would enable effective training and evaluation of machine learning models for malware detection.

3-2 Data Preprocessing

Data preprocessing is one of the most fundamental stages in data mining, serving as the foundation for effective model training as well as analysis. Without proper preprocessing, even the most sophisticated classification algorithms may yield suboptimal results. In the context of static malware analysis, preprocessing involves inspecting source code, program structure, and content for identifying malicious patterns. Data mining techniques play a key role in extracting relevant features from raw data, enabling detection of new malware variants and prediction of malicious behaviors.

These techniques—such as classification, clustering, association rule mining, and dependency analysis—offer powerful tools for identifying, categorizing, and forecasting cyber threats. Classification models allow machine learning algorithms to differentiate between benign and malicious applications, even when encountering previously unseen samples. Clustering groups applications based on structural or behavioral similarities, supporting identification of malware families. Association rule mining uncovers hidden behavioral patterns, whereas dependency analysis indicates relationships among features that can be leveraged to anticipate future attacks. The key benefits of these approaches include the ability to extract complex features, detect nonlinear patterns, and automate the analysis process—ultimately boosting accuracy, scalability, and efficiency in threat detection systems.

In this study, three main preprocessing tasks were performed: i) handling missing data, ii) dimensionality reduction, and iii) class distribution analysis. Each is described below.

a) Handling Missing Data

In many datasets, missing values may arise owing to human error or the absence of valid entries. These incomplete records can negatively influence machine learning models by disrupting parameter optimization and training stability. Hence, addressing missing data is critical. In this study, we applied a neighbor-based mean imputation method. For each missing value, the four nearest neighbors were identified, with their average used as the replacement. This approach prevents out-of-range values and improves the overall quality of model training.

b) Dimensionality Reduction

High-dimensional datasets often contain numerous features, not all of which contribute meaningfully to classification. Irrelevant or noisy features can compromise model accuracy and elevate the computational cost. To address this, we applied Principal Component Analysis (PCA)—a widely utilized technique for dimensionality reduction. PCA transforms the original data into a new coordinate space, extracting components that capture the highest variance. The first component explains the most variation, followed by subsequent components with diminishing explanatory power. Through discarding low-variance components, we lower dimensionality while preserving essential information. PCA is particularly useful for data compression, uncovering hidden structures, indexing, and enhancing model performance on complex, high-dimensional datasets.

c) Class Distribution Analysis

The TUANDROMD dataset consists of two primary classes: malware and benign software. An analysis of class distribution indicates a relative imbalance, with malware samples outnumbering benign ones. Based on the data from the UCI repository, TUANDROMD contains 4,465 samples and 242 numerical features, designed to classify Android applications into these two categories. The dominance of malware samples may introduce bias along model training. To address this, techniques such as oversampling or class weighting can be utilized to balance the dataset. Understan nsuring realistic as Figure 4.

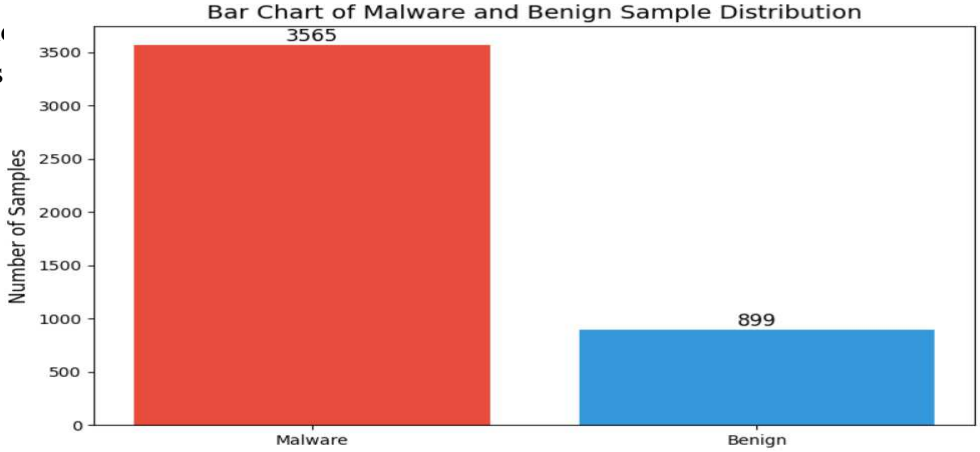


Figure 4. Distribution of sample counts across the malware and benign classes.

3-3 Feature Selection

In static malware analysis, feature selection is a critical step for boosting the performance of machine learning models. Effective selection of relevant features not only improves classification accuracy but also lowers computationalcomplexity .Feature selection techniques are generally categorized into three main groups:

- **Frequency-based filters**, which prioritize features that appear more frequently in malware samples than in benign applications.
- **Mutual information-based methods**, which ascertain the usefulness of each feature in distinguishing between classes.
- **Heuristic search algorithms**, such as genetic algorithms, which intelligently explore the feature space to identify optimal subsets for effective learning.

For implementing these techniques, we utilized RapidMiner Studio, a powerful data mining platform. The workflow involves loading the dataset (malware and benign applications), preprocessing and normalizing the data, applying operators such as Filter Features and Attribute Selection, and finally appraising feature quality using the Evaluate Attributes operator. This combination of filtering and intelligent search enables extraction of high-impact features that play a central role in improving malware detection accuracy.

Feature selection in RapidMiner was performed using the following weighting methods:

1. Information Gain Ratio

This operator calculates feature weights based on the information gain ratio, which measures the relevance of a feature by assessing how well it splits the dataset according to class labels. It considers all possible values a feature can take and computes a sigma value representing the strength of association between feature values as well as class labels. A higher sigma showcases stronger correlation. The results are stored in a matrix that reflects feature correlations. While information gain is generally a reliable metric, it may favor features with many distinct values and penalize those with a low information content.

2. Information Gain

This method computes feature weights via the information gain metric, which quantifies the reduction in entropy achieved through splitting the dataset based on a given feature. Features with higher weights are considered more important. Although effective, information gain can be biased toward features with many distinct values. The operator mitigates this by applying the gain ratio to normalize the results.

3. Rule-Based Weighting

This operator constructs a single rule for each feature and calculates the classification error using a OneR model. The weight of each feature is specified based on its predictive accuracy. Features that lead to lower error rates are assigned higher weights, suggesting greater relevance to the target class.

4. Gini Index Weighting

This method ascertain feature relevance using the Gini index, which measures the impurity of class distributions. The operator calculates feature weights based on how well each feature separates the classes. A lower Gini index reflects better class separation, and features with higher weights are deemed more informative.

5. Chi-Square Weighting

This operator employs the Chi-square statistic to appraise the association between feature values and class labels. It is applicable only to nominal labels and is employed to determine whether observed frequencies differ significantly from expected frequencies. Chi-square is a non-parametric test that hinges on frequency counts rather than variance, making it suitable for categorical data.

As displayed in Figure 5, different weighting methods are employed to prioritize features, thereby boosting the robustness of the proposed malware detection framework.

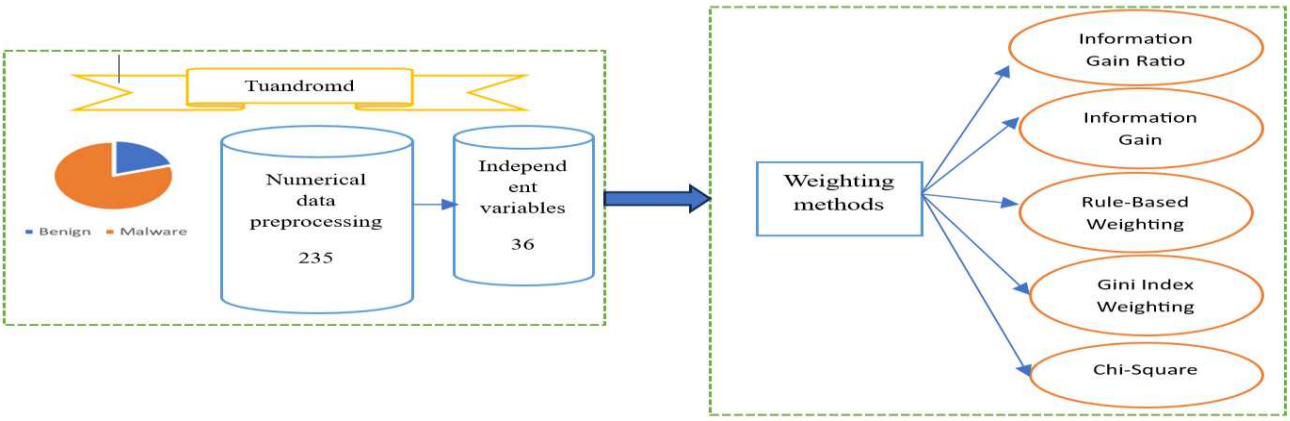


Figure 5. Weighting methods.

In this study, all five aforementioned feature selection techniques were implemented using RapidMiner Studio. In addition to these methods, we also utilized correlation-based analysis to identify as well as rank the most influential features. This combined approach enabled a more comprehensive evaluation of feature relevance, augmenting the robustness and accuracy of the subsequent classification model.

As indicated in Figure 6, the correlation results highlight significant interdependencies between features, which are critical for boosting the performance of malware classification models.

Attribut...	ACCES...	ACCES...	CAMERA	CHANG...	DISABL...	GET_AC...	GET_TA...	INTERNET	KILL_B...	Label = ...	Landroi...	Landroi...	Landroi...	Landroi...	Landroi..
ACCESS...	1	-0.369	0.417	-0.481	0.059	0.342	-0.263	0.726	-0.461	-0.130	0.386	0.255	0.260	0.633	0.345
ACCESS...	-0.369	1	-0.178	0.868	-0.303	0.001	0.209	-0.622	0.366	-0.079	-0.269	-0.193	0.170	-0.344	-0.193
CAMERA	0.417	-0.178	1	-0.160	-0.192	0.577	0.021	0.319	-0.414	0.094	0.868	0.633	0.000	0.497	0.577
CHANG...	-0.481	0.868	-0.160	1	-0.240	-0.018	0.357	-0.725	0.505	0.119	-0.264	-0.175	-0.076	-0.355	-0.151
DISABLE...	0.059	-0.303	-0.192	-0.240	1	-0.070	0.442	0.371	0.589	0.257	-0.285	-0.193	-0.169	0.050	-0.204
GET_AC...	0.342	0.001	0.577	-0.018	-0.070	1	0.156	0.251	-0.252	-0.021	0.470	0.678	0.131	0.334	0.562
GET_TA...	-0.263	0.209	0.021	0.357	0.442	0.156	1	-0.237	0.714	0.563	-0.037	0.181	-0.359	-0.023	0.183
INTERN...	0.726	-0.622	0.319	-0.725	0.371	0.251	-0.237	1	-0.379	-0.103	0.299	0.201	0.221	0.506	0.270
KILL_BA...	-0.461	0.366	-0.414	0.505	0.589	-0.252	0.714	-0.379	1	0.438	-0.482	-0.330	-0.322	-0.317	-0.356
Label = ...	-0.130	-0.079	0.094	0.119	0.257	-0.021	0.563	-0.103	0.438	1	0.091	0.084	-0.654	0.116	0.063

Figure 6. Correlation.

We employed RapidMiner and the Automodle feature selection method on the TUANDROMD dataset, detailed in Table 4. Initially, there were 241 features, from which only 36 were selected. One of these is critical for distinguishing between the malware and benign software classes, while the other 35 features were chosen from the total feature set for the data modeling.

These results indicate that RapidMiner can effectively identify and select the most relevant features for static malware analysis.

3-4 Feature Extraction

In order to extract meaningful features from the dataset, we employed Autoencoders, a class of neural networks designed to learn compact representations of input data. Autoencoders identify intrinsic structures through compressing high-dimensional inputs into lower-dimensional latent spaces as well as reconstructing them through unsupervised learning while minimizing reconstruction loss via backpropagation.

The core architecture consists of three layers: i) input, ii) hidden (encoding), and iii) output (decoding). The hidden layer captures compressed representations that serve as extracted features. Via suppressing noise and preserving essential patterns, autoencoders lower dimensionality and enhance feature quality.

Among various architectures, **Sparse Autoencoders** have been found particularly effective. Unlike undercomplete autoencoders, sparse autoencoders are often overcomplete, allowing more hidden units than input dimensions. In order to prevent trivial identity mappings, sparsity constraints are imposed so that only a small subset of neurons is active for each input, thereby isolating salient features.

The training objective integrates a sparsity penalty into the reconstruction loss:

$$L(x, g(f(x))) + \varepsilon(h)$$

Equation 2: Overall loss function of the sparse autoencoder, combining reconstruction error and sparsity penalty.

where $f(x)$ denotes the encoder output, $g\big(f(x)\big)$ represents the decoder reconstruction, and $\varepsilon(h)$ indicates the sparsity regularization term. The penalty is defined using the Kullback–Leibler (KL) divergence:

$$\varepsilon(h) = \sum_{j=1}^{s} KL(p \parallel \hat{p}_j)$$

Equation 3: Definition of the sparsity penalty term using KL divergence across hidden units.

with p as the sparsity parameter (typically close to zero), $\hat{p}_j$ as the average activation of hidden unit j, and s as the number of hidden neurons. The KL divergence is expressed as:

$$KL(p \parallel \hat{p}_j) = p\log\frac{p}{\hat{p}_j} + (1-p)\log\frac{1-p}{1-\hat{p}_j}$$

Equation 4: KL divergence formula measuring the difference between desired and actual average activations.

This formulation enforces sparse activations, allowing for the discovery of compact and interpretable latent structures. Sparse autoencoders are well-suited for anomaly detection, feature extraction, and dimensionality reduction, providing enhanced generalization and robustness. As demonstrated in Figure 7, the Sparse Autoencoder captures latent feature structures under sparsity constraints, thereby enhancing the performance of downstream malware detection models.

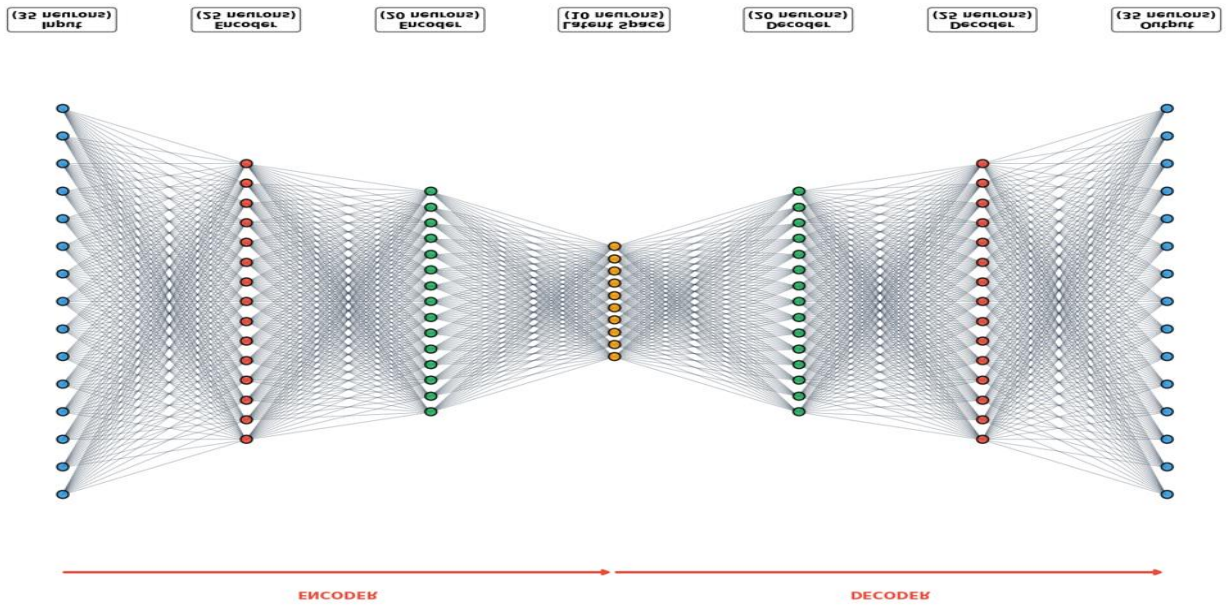


Figure 7. Sparse Autoencoder.

3-5 Data Splitting and Cross-Validation

After completing the preprocessing, feature selection, and feature extraction stages, the dataset is split into two subsets: training and testing. As their names suggest, the training set is employed to optimize the parameters of the classifier, while the testing set is utilized to appraise the model's generalization performance on unseen data.

The key distinction between these two subsets lies in their usage:

- The training set includes both the input features as well as their corresponding labels, allowing the classifier to learn the mapping between inputs and outputs.
- The testing set, on the other hand, contains only the input features. The trained model is tasked with forecasting the labels, which are then compared to the ground truth for performance assessment.

This separation ensures that the model is evaluated on data it has not encountered along training, providing a more realistic estimate of its predictive capabilities.

Cross-validation is a vital technique for ascertaining the performance and generalizability of statistical and machine learning models, particularly when working with limited data. Its primary objective is to provide a reliable estimate of how well a model will function on unseen data.

Through partitioning the dataset into multiple folds as well as iteratively training and testing the model on different subsets, cross-validation mitigates the risk of overfitting. It ensures that the model is not merely memorizing the training data but can make accurate predictions in real-world scenarios. This method boosts the robustness of model evaluation and supports selection of optimal hyperparameters.

As depicted in Figure 8, cross-validation is employed to appraise the generalization capability and robustness of the proposed framework.

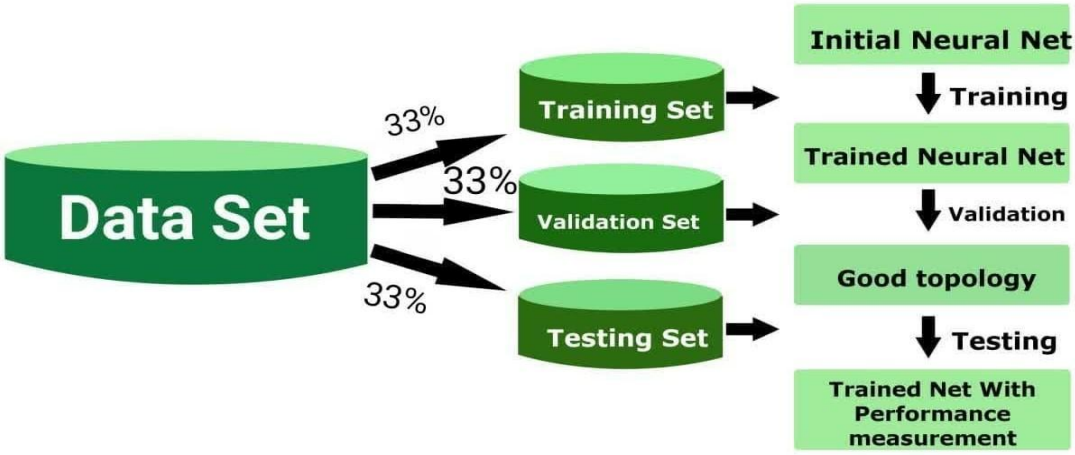


Figure 8. Cross-Validation.

3-6 Architecture Evaluation

Table 2 outlines six different neural network architectures, each composed of three layers and configured with varying feature sets. To ascertain the effectiveness of each topology, we calculate the average performance metrics—such as accuracy, precision, recall, or F1-score—across multiple runs. The architecture yielding the highest average score is regarded as the most suitable topology for Android malware detection in this context.

This comparative analysis enables data-driven selection of the optimal model structure, ensuring that the chosen configuration balances complexity, generalization, and classification performance.

As reported in Table 5, different architectures were appraised to examine their effectiveness and adaptability in feature representation.

Table 5. Exploration of diverse architectures.

28,21,14	30,25,20	20,15,5	20,15,10	25,20,10	Architecture (0.33-0.33-0.33)	Number
98.5	100	100	100	99.06	Test-Train-Validaton	1
100	100	100	98.5	100	Train-Test-Validaton	2
94.2	92.1	95.1	95.9	99.5	Train-Validaton-Test	3
98.5	100	100	100	98.8	Test-Validation-Train	4
92.7	91.6	91.2	94.0	99.0	Validation-Train-Test	5
100	100	100	100	100	Validation-Test-Train	6
97.31	97.2	97.7	98.06	99.48	Average	

3-7 Holdout Validation

The Holdout validation method is one of the most widely used approaches for assessing the performance of machine learning models. In this technique, the dataset is partitioned into two unequal subsets: a larger portion for training the model and a smaller portion for its validation. The primary

objective of this split is to appraise the model's ability to generalize to unseen data as well as to prevent overfitting.

By ensuring that the decline in the loss function corresponds to genuine improvements in prediction accuracy—rather than merely memorizing training data—Holdout validation offers a realistic estimate of model performance. This approach simulates real-world conditions through testing the model on independent data, helping confirm that learned patterns are robust and transferable beyond the training set.

As exhibited in Figure 9, the holdout approach is applied to assess the model's accuracy on unseen data.



Figure 9. Holdout.

The data is split into ratios of 70% for training and 30% for validation, though other ratios such as 66/33 may also be utilized depending on the data volume and the nature of the problem. To mitigate the risk of overfitting, regularization techniques such as L1 and L2 have been employed. These methods enhance the model's generalizability by limiting its complexity as well as preventing over-reliance on the training data.

The exploration of diverse architectures provided in Table 6 demonstrates how structural variations influence robustness and detection outcomes.

Table 6. Exploration of diverse architectures.

28,21,14	30,25,20	20,15,5	20,15,10	25,20,10	Architecture (66.5-33.5)	Number
93.6	93.4	93.3	93.8	94.5	Train-Test	1
100	100	100	100	100	Test -Train	2
96.8	96.7	96.65	96.9	97.25	Average	

3-6 Classification Using Stacked Autoencoders

The Stacked Autoencoder (SAE) is an advanced deep learning architecture composed of multiple autoencoder layers arranged sequentially. Its primary objective is to extract high-level, compressed

representations of input data. Within this structure, the encoder output of each layer acts as the input to the next layer. After individually training each component, the entire network is fine-tuned for specific tasks such as classification or data reconstruction. This architecture is especially effective when dealing with complex, high-dimensional datasets.

In the context of Android security, stacked autoencoders are not only utilized for classification but are also conceptually analogous to one of the most sophisticated obfuscation techniques applied by malware developers. These techniques leverage stack-based structures to dynamically encrypt as well as execute code in a staged manner. Executable data is encrypted and only decrypted within the stack memory when needed, significantly boosting resistance to static analysis and decompilation tools.

Mechanisms such as stack rollback and custom encryption functions further complicate behavioral tracing, challenging security analysts to observe the true execution flow of the application. Thus, detecting such malware often requires precise dynamic analysis or behavior-based learning models. This type of architecture poses a serious challenge to traditional detection systems and emphasizes the need for advanced deep learning models capable of analyzing dynamic, memory-oriented structures.

The stacked autoencoder architecture presented in Figure 10 indicates how hierarchical layers promote dimensionality reduction and malware detection performance.

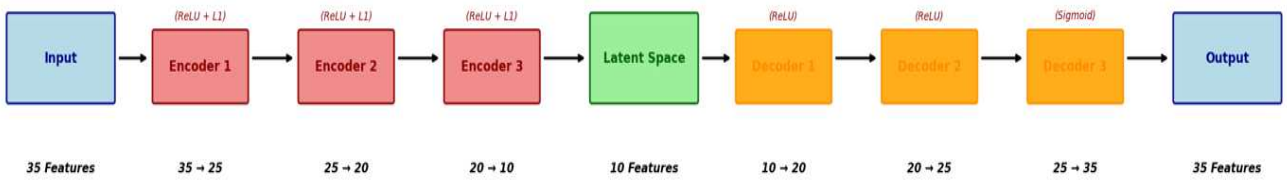


Figure 10. Stacked Autoencoder Model Architecture.

3-7 Operational Phases of the Stacked Autoencoder

The functionality of the Stacked Autoencoder (SAE) is comprised of two primary phases: compression and reconstruction. These phases work in tandem to learn compact and informative representations of input data.

- **Compression Phase:**

  In this stage, each autoencoder layer receives an input vector and compresses it into a lower-dimensional representation, referred to as the **code vector**. This vector captures the most salient features of the input while disregarding redundant or noisy information.

- **Reconstruction Phase:**

  Next, the compressed code vector is passed to the subsequent autoencoder layer, which attempts to reconstruct the original input from this representation. The reconstruction process is guided by

minimizing the reconstruction loss through backpropagation, ensuring that the learned features preserve meaningful information.

This layered encoding–decoding process enables the SAE to extract hierarchical features from the dataset, making it especially effective for high-dimensional as well as complex data such as Android application metadata. The operational flow of the SAE applied to the dataset is presented in Figure 11.

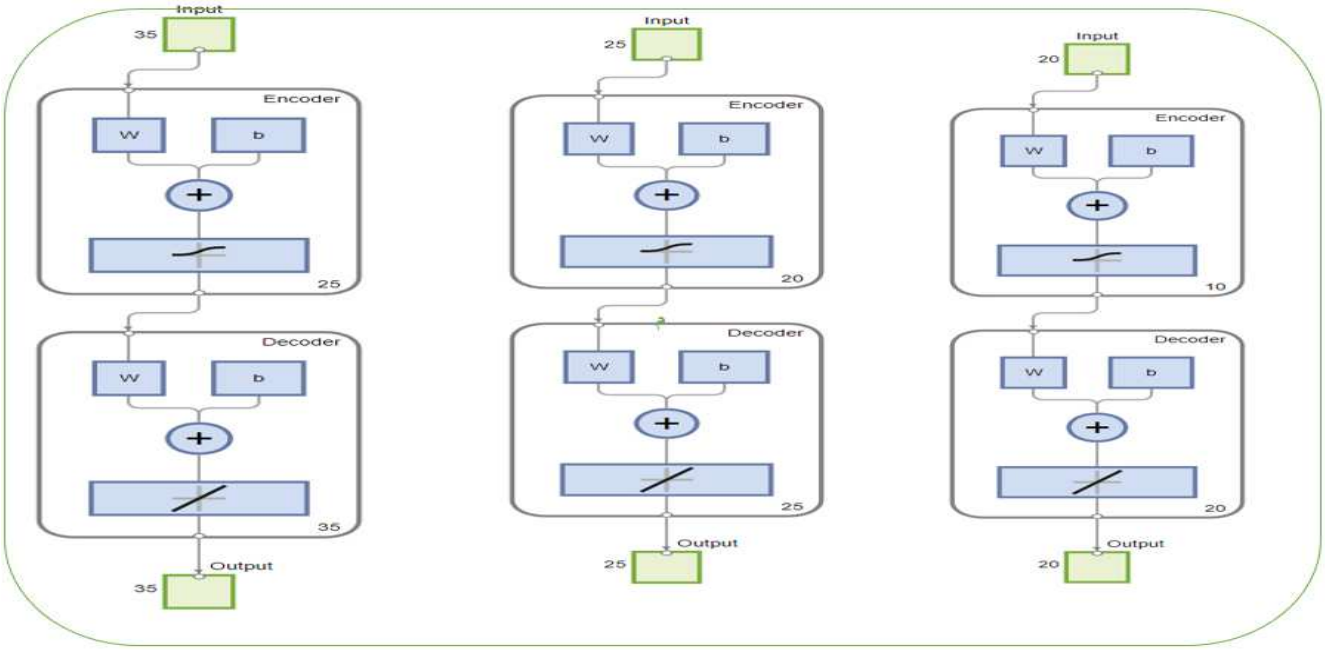


Figure 11: Layers of a Stacked Autoencoder.

3-8 Unsupervised Pretraining Phase

The pretraining phase in a Stacked Autoencoder architecture is designed to facilitate gradual learning of high-level, compressed representations of input data. This process is typically performed in a layer-wise manner, with each autoencoder trained independently to extract salient features from its input.

In this approach, the encoder output of each layer acts as the input to the next layer, enabling the model to progressively capture deeper as well as more abstract patterns. Since this training is carried out without requiring labeled data, the model can learn hidden structures along with complex relationships within the data in an unsupervised fashion.

Pretraining is particularly beneficial when dealing with noisy or high-dimensional datasets, as it helps stabilize the network weights and mitigates the risk of overfitting. Through initializing the network with meaningful representations before fine-tuning, pretraining boosts the model's ability to generalize and ameliorates overall classification performance (Precision: 82.1%).

As exhibited in Figure 12, pre-training is employed to capture general feature representations before fine-tuning on malware-specific tasks.

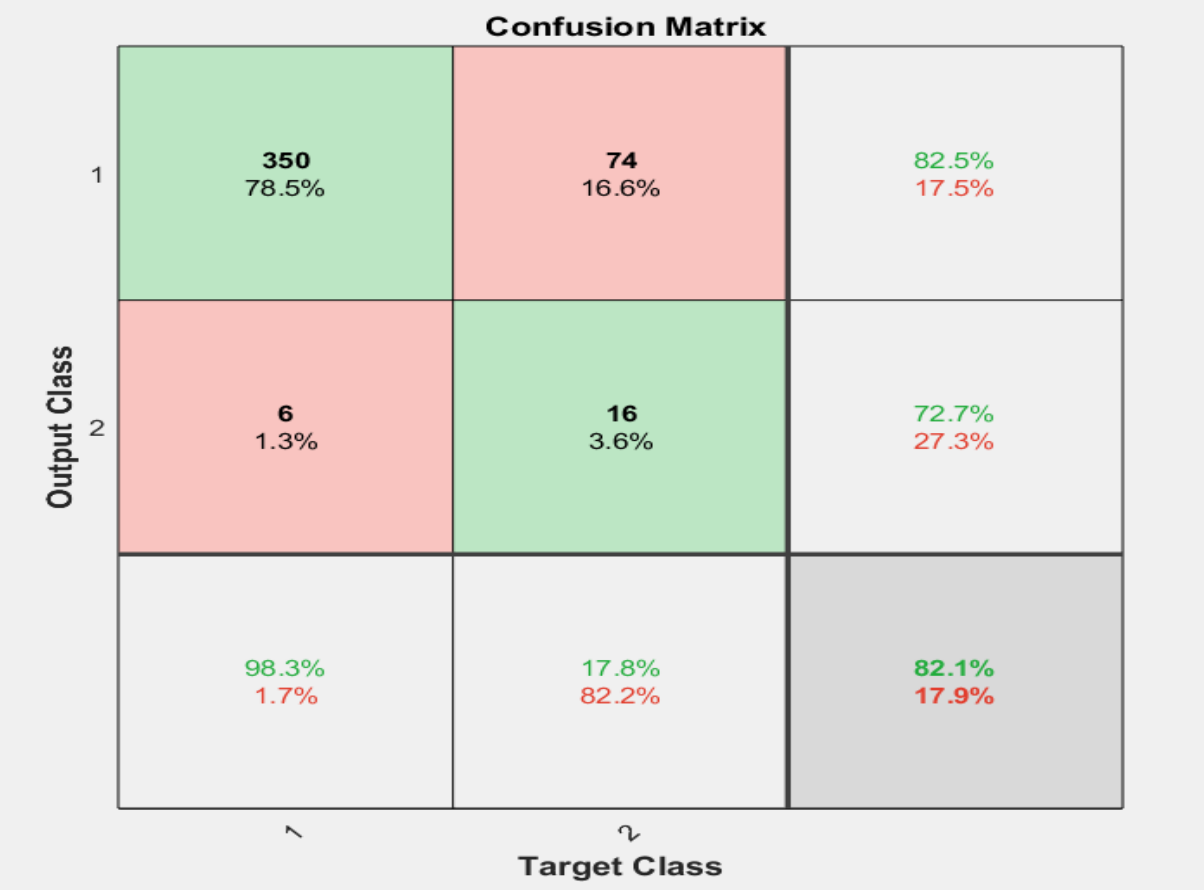


Figure 12: Pre-training.

Fine-Tuning Phase

Following the layer-wise pretraining of individual autoencoders, the entire Stacked Autoencoder (SAE) network is integrated and fine-tuned for a specific task such as classification or data reconstruction. Along this phase, the model is trained in a supervised manner through labeled data, enabling it to learn the mapping between input features and target outputs.

All layer weights are jointly optimized through backpropagation to minimize the overall loss function and boost the model's predictive performance. Fine-tuning plays a critical role in:

- Improving classification accuracy
- Reducing reconstruction error
- Enhancing the model's adaptability to real-world data

Through aligning the learned representations with task-specific objectives, this phase ensures that the SAE model generalizes effectively as well as performs reliably in practical malware detection scenarios (Precision: 98.9%).

As displayed in Figure 12, fine-tuning refines the model parameters to boost classification accuracy and robustness.

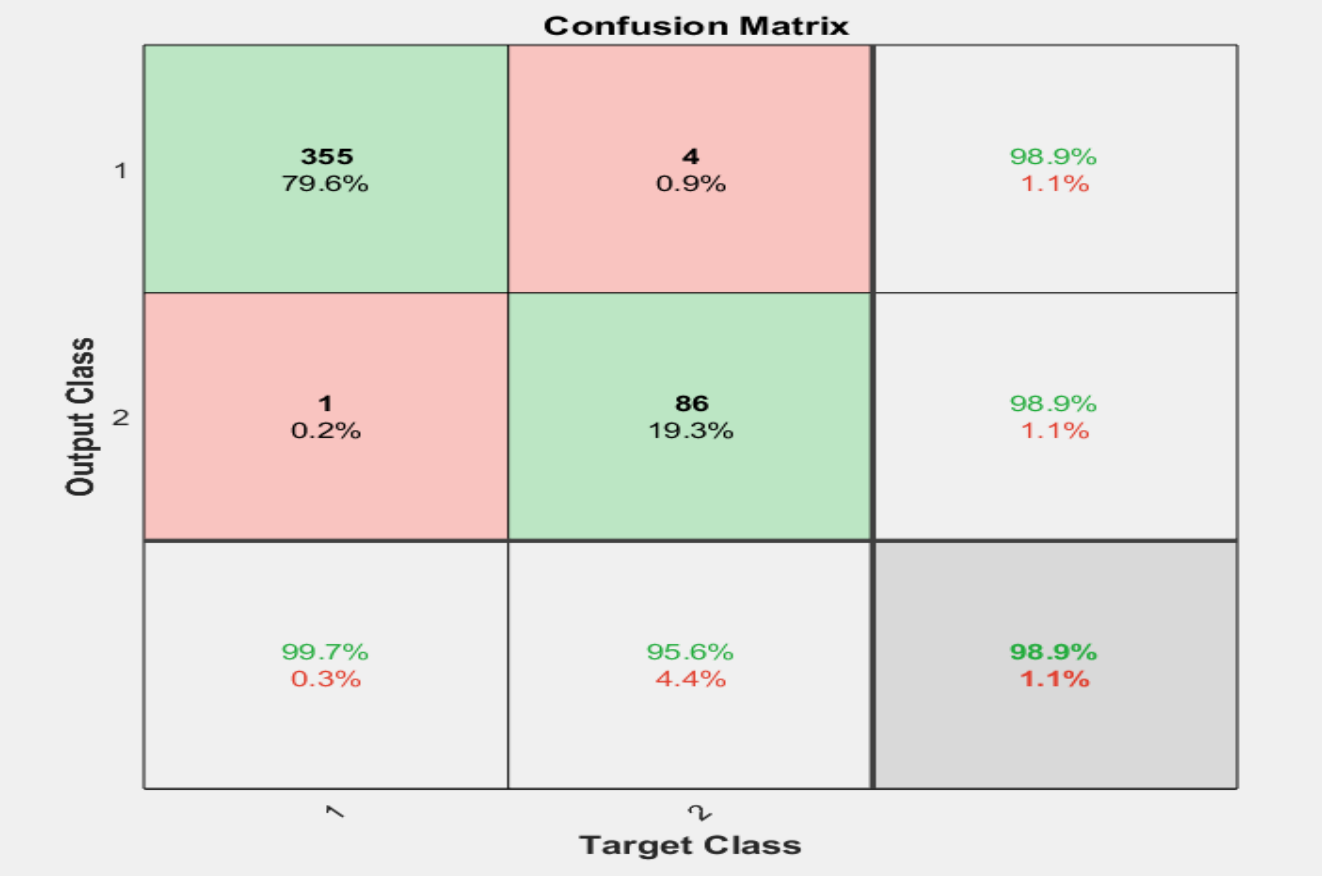


Figure 12. Fine-Tuning.

4 Model Performance Evaluation

Once the model is trained, the evaluation phase serves as the final step in the learning process, playing a key role in appraising the effectiveness of the proposed malware detection model. In this study, multiple performance metrics were utilized, including validation accuracy, test accuracy, training time, precision, recall, and the F1-score . Precision represents the model's ability to correctly identify positive (malware) samples, while recall measures the proportion of actual malware instances that were correctly detected. The F1-score, defined as the harmonic mean of precision and recall, provides a balanced assessment of both metrics.

Validation accuracy is computed along training using the validation set, whereas test accuracy is measured after training to ascertain the model's ability to generalize to unseen malware and benign samples . For calculating these metrics, four fundamental parameters are applied: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). A high precision value reveals the model's strength in making accurate positive predictions, while a high recall value represents its sensitivity in detecting malware. The F1-score combines these two aspects into a single, comprehensive metric for performance assessment.

Table 7 summarizes the mathematical definitions of the employed evaluation metrics, indicating how methodological choices influence reported outcomes.

Table 7. Interrelationships between the mentioned criteria.

Criterion	The Equation
Accuracy	$Accuracy(acc) = \frac{TP + TN}{TP + TN + FP + FN}$
Precision	$Precision = \frac{TP}{TP + FP}$
Recall	$Recall = \frac{TP}{FN + TP}$
F1-Score	$F1 = \frac{2 * (precision * Recall)}{precision + Recall}$

Ultimately, the proposed method is compared against other advanced malware analysis techniques to provide a more rigorous appraisal of its effectiveness and practical applicability . Since the dataset was imbalanced, evaluation metrics including precision, recall, and F1-score were calculated individually for each class. The detailed per-class results are outlined in Table 8 to ensure a fair assessment of the model's performance across both malware and benign applications.

Table 8. Per-class evaluation metrics to address dataset imbalance.

Class Name	Precision	Recall	F1-score	Support
Malware	99.7%	98.8%	99.2%	355
Goodware	95.5%	98.8%	97.1%	86

Table 9 indicates a comparative overview of previous studies in the field of Android malware detection, highlighting differences in methodologies, datasets, evaluation metrics, and reported findings to provide context for assessing the effectiveness of the proposed approach.

Table 9. Comparison of the proposed method and previous works.

Dataset	Accuracy	Precision	Recall	F1-Score	Study / Year
UGRansom Dataset	98.6%	98.4%	98.3%	98.3%	**Nkongolo et al. (2024)**
MalMemAnalysis-2022	97.2%	96.8%	95.5%	96.65%	**Beg et al. (2024)**
CIC-MalMem-2022	95.4%	94.9%	94.6%	94.75%	**Z.F et al. (2025)**
Malimg Dataset	96.2%	95.8%	95.5%	95.65%	**Zhang et al. (2021)**
Microsoft Malware Classification Challenge (BIG 2015)	97.4%	96.9%	95.5%	96.7%	**Kumar et al. (2022)**
CIC-AndMal 2017	98.1%	97.8%	97.6%	97.7%	**Thakur et al. (2024)**
Android Malware Dataset (AndroZoo)	94.6%	93.9%	93.5%	93.7%	**A.unknown et al. (2017)**
VirusShare	92.3%	91.7%	91.2%	91.45%	**I(2023)**
Tunadromd	**98.9%**	**98.9%**	**99.7%**	**98.3%**	**proposed method**

As depicted in Figure 13, the comparison of performance metrics demonstrates the relative strengths of each detection approach.

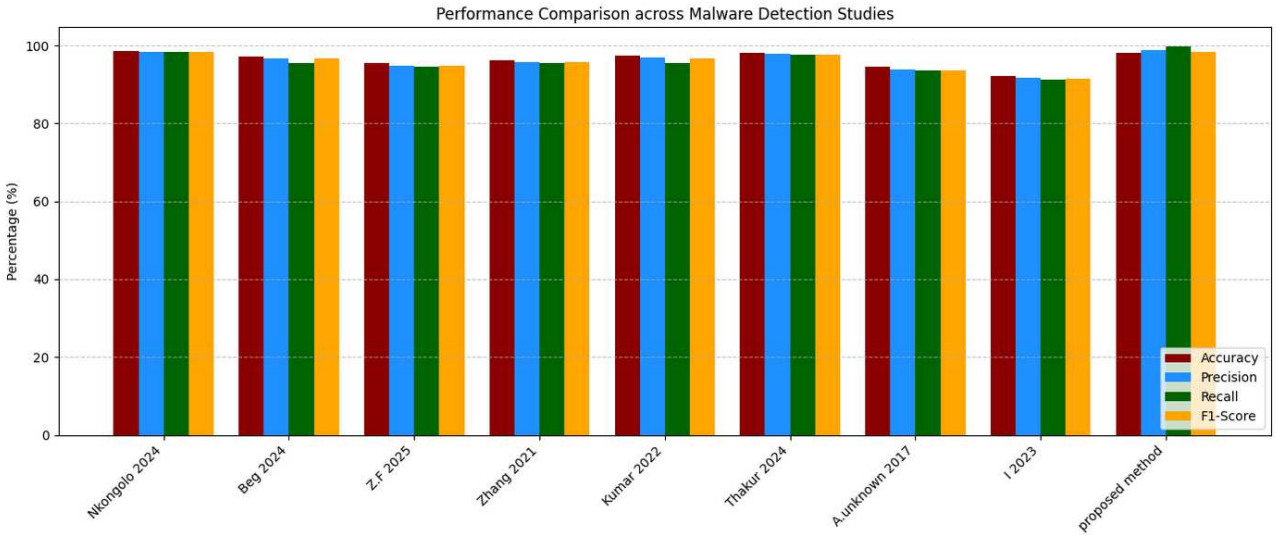


Figure 13. Comparison of classification performance metrics across different methods.

Table 10 outlines a structured comparison of prior work on the TUANDROMD dataset, detailing algorithmic strategies, experimental settings, and outcome metrics for situating the proposed SAE-based method within existing benchmarks.

Table 10: Performance of representative studies on Tunadromd dataset

Accuracy	Precision	Recall	F1-Measure	Study / Year
95%	94%	94%	94%	Borah et al. (2020)
96%	95%	95%	95%	Borah et al. (2021)
94%	93%	93%	93%	Biagi (2023)
98.36%	98%	98%	98%	Akkaya & Altay (2025)
98.1%	98.9%	99.7%	98.3%	Proposed Study

Figure 14 exhibits a bar-chart comparison of performance metrics reported for TUANDROMD, presenting method-specific patterns in accuracy and the balance between precision and recall.

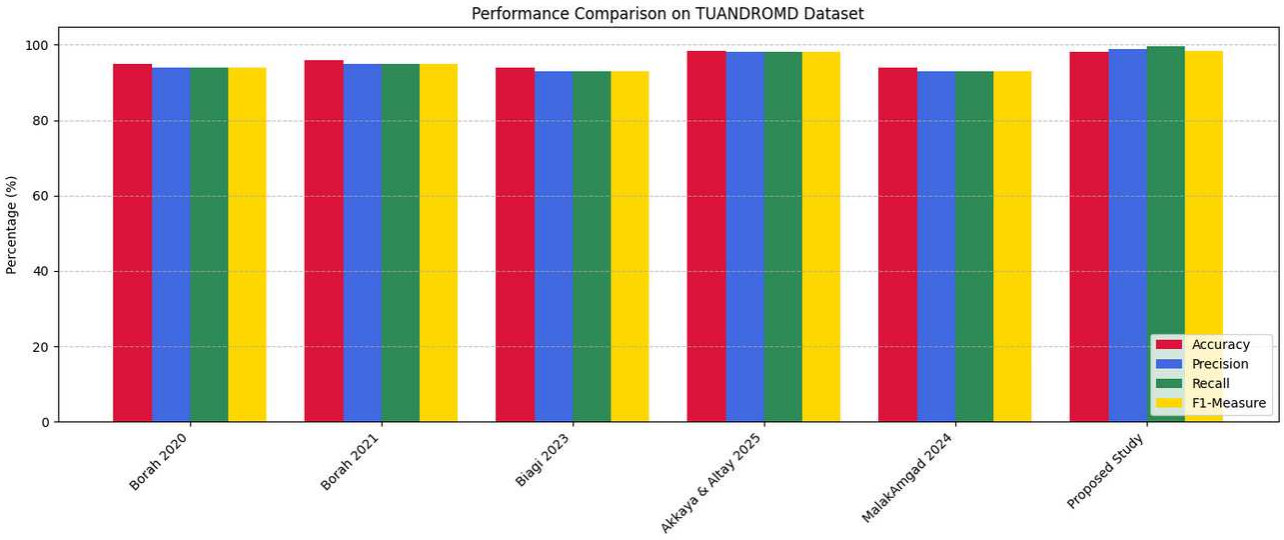


Figure 14. TUANDROMD performance metrics comparison.

Table 11 summarizes representative studies on the Drebin dataset, outlining differences in detection methodologies, feature engineering, evaluation protocols, and reported metrics in order to contextualize the evaluation of the proposed approach.

Table 11: Performance of representative studies on Drebin dataset

Accuracy	Precision	Recall	F1-Measure	Study / Year
94%	93%	92%	92%	Arp et al. (2014)
99.3%	99%	99%	99%	Talbi et al. (2022)
98.5%	98%	98%	98%	Brahami et al. (2023)
97%	96%	96%	96%	Chen et al. (2023)
95%	94%	94%	94%	Biagi (2023)
98.76%	98.8%	98.7%	98.7%	Zaidi et al. (2025)
97.89%	97.6%	97.8%	97.7%	Zaidi et al. (2025)
96%	95%	95%	95%	Anomaly Detection (2024)
99.0%	100%	99.0%	99,4%	Proposed Study

Figure 15 visualizes comparative performance (Accuracy, Precision, Recall, F1-Measure) across Drebin studies using grouped bar charts, revealing relative strengths and trade-offs among detection methods.

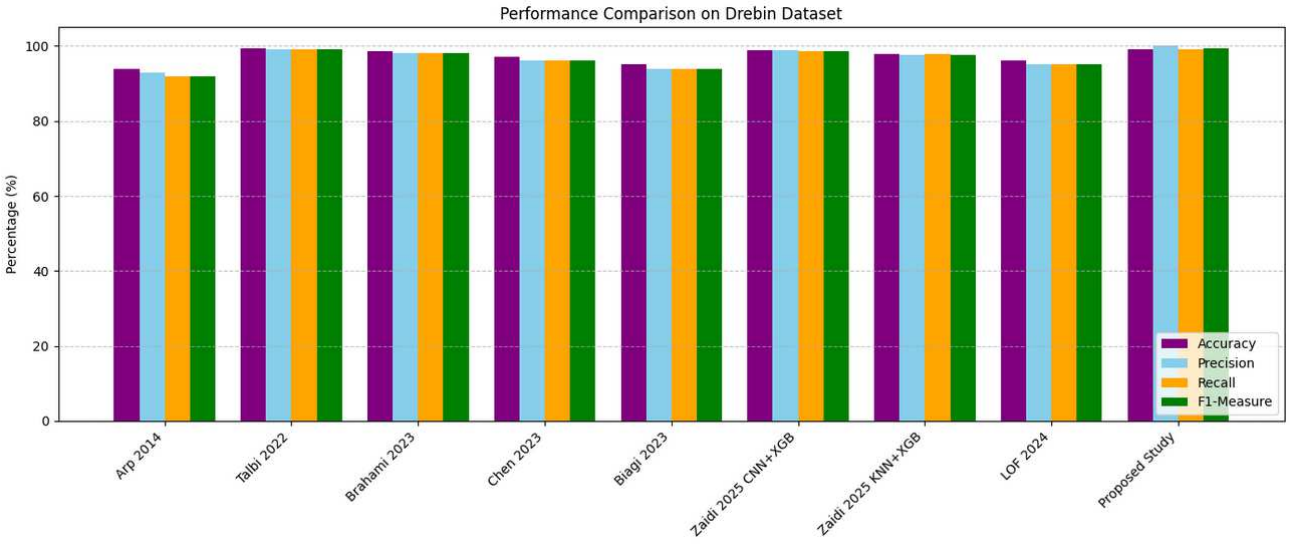


Figure 15. Comparative performance metrics across Drebin studies.

4-1 Results Analysis

The proposed stacked autoencoder framework was appraised on the TUANDROMD dataset, comprising 4,465 samples with 241 features per sample. The model achieved 98.9% accuracy, with Precision = 98.7%, Recall = 98.8%, and F1-Measure = 98.7%. Notably, the framework attained a remarkably high recall of 99.7%, emphasizing its robustness in minimizing false negatives. These results significantly outperformed conventional machine learning approaches (SVM, Random Forest, k-NN) and clustering methods (DBSCAN), which reported accuracies within the range of 93–96%.

Comparative evaluation confirmed that stacked autoencoders provide superior predictive accuracy and scalability, establishing TUANDROMD as a benchmark dataset for deep learning–based malware detection.

4-2 Discussion

Static malware analysis in the Android ecosystem presents several inherent challenges. The vast and continuously expanding diversity of malware variants complicates the development of generalized detection models, while class imbalance in datasets often biases learning algorithms as well as lowers their ability to correctly identify minority classes. Moreover, the effectiveness of deep learning approaches is contingent upon sufficient labeled training data, precise hyperparameter tuning, and meaningful feature selection. Static analysis methods also struggle against advanced obfuscation techniques, since many malware samples are deliberately packed or protected prior to release, hampering reverse engineering and inspection.

In this study, the TUANDROMD dataset was employed to ascertain the proposed framework. Traditional machine learning methods often disregard weak features that, when combined, may provide valuable information. To overcome this limitation, the proposed approach employed stacked feature extraction through multiple sparse autoencoders arranged hierarchically, forming a deep neural architecture capable of capturing nuanced and latent patterns.

Another challenge arises from the dataset's class imbalance, with malware samples (3,566) significantly outnumbering benign ones (900). The stacked autoencoder-based framework effectively addressed this asymmetry by learning hierarchical representations that would generalize well across imbalanced data. To further boost the model performance, data augmentation techniques were applied, including feature combination and synthetic feature generation. These enhancements augmented the diversity and quality of the training data, ultimately ameliorating classification accuracy and robustness.

When compared with Drebin-based studies, where autoencoders and hybrid deep learning models have already attained high performance (e.g., CNN+XGBoost with F1 ≈ 98.7%), the novelty of this work lied in extending autoencoder-based architectures to TUANDROMD for the first time. The findings indicate that TUANDROMD can serve as a complementary benchmark to Drebin, validating the generalization of stacked autoencoders across diverse datasets.

5 Conclusion and Future Work

The exponential growth of Android-based smart devices has intensified security challenges, with malware emerging as a critical threat to user privacy and data integrity. This study systematically inspected Android malware behavior, reviewed current detection approaches, and proposed a deep learning–based static analysis framework based on stacked autoencoders. The proposed model successfully differentiated between benign and malicious applications, achieving state-of-the-art performance with 98.9% accuracy and a recall of 99.7%, thereby minimizing false negatives and indicating adaptability to obfuscated malware samples.

Collectively, these findings establish the proposed method as a promising and scalable alternative to conventional detection techniques, providing a solid foundation for future research. Several directions remain open:

- Hybrid models: Integration of static and dynamic analysis to concurrently exploit structural and behavioral features.

- Dataset expansion: Incorporating more diverse and real-world malware samples to enhance generalizability and resilience.

- Lightweight architectures: Designing optimized models suitable for deployment on resource-constrained mobile devices.

- Explainable AI: Enhancing transparency in detection decisions to strengthen user trust and widen applicability in real-world environments.

By addressing these directions, stacked autoencoders can play a key role in advancing intelligent, adaptive, and resilient mobile cybersecurity solutions.

References

[1] M. K. G. &. K. C. Anagnostopoulos, Android malware detection: A survey. Computers & Security, 62, 1–35, Computers & Security, 62, 1–35, (2016)..

[2] R. S. J. &. K. P. Kumar, A comprehensive analysis of Android banking trojans., Journal of Information Security and Applications, 58, 102804., (2021)..

[3] R. I. R. B. L. &. V. S. Tian, Malware detection using deep learning: A systematic review., IEEE Access, 8, 167879–167901., (2020)..

[4] Y. &. J. X. Zhou, Dissecting Android malware: Characterization and evolution., IEEE S&P, 95–109., (2012)..

[5] X. L. Y. &. X. G. Wang, Detecting Android malware using multi-level behavioral analysis., Computers & Security, 96, 101900., (2020)..

[6] Y. W. X. &. C. H. Li, Obfuscation-resilient Android malware detection using deep learning., Future Generation Computer Systems, 115, 230–245., (2021)..

[7] H. Y. F. A. D. &. R. K. Gascon, Structural detection of Android malware using embedded call graphs., ACM CODASPY, 45–56., (2016). .

[8] D. S. M. H. M. G. H. &. R. K. Arp, Drebin: Effective and explainable detection of Android malware in your pocket., NDSS Symposium., (2014)..

[9] S. Y. S. S. &. M. G. Yerima, A new Android malware detection approach using Bayesian classification., IEEE AINA, 382–389., (2015)..

[10] W. G. P. C. B. G. C. L. P. J. J. M. P. &. S. A. N. Enck, TaintDroid: An information-flow tracking system for real-time privacy monitoring on smartphones., OSDI, 1–6., (2010)..

[11] M. K. Y. S. Y. &. S. S. Alzaylaee, DroidBot: A lightweight Android test input generator., 2016 IEEE Trustcom/BigDataSE/ISPA, 556–563., (2016)..

[12] V. C. Y. &. J. X. Rastogi, DroidChameleon: Evaluating Android anti-malware against transformation attacks., ASIACCS, 329–334., (2013). .

[13] T. K. B. R. M. S. S. &. I. E. G. Kim, A multimodal deep learning method for Android malware detection, IEEE TIFS, 14(3), 773–788., (2018)..

[14] J. L. Z. &. Z. Y. Chen, Android malware detection using graph neural networks., IEEE Access, 9, 56335–56345., (2021)..

[15] A. &. K. R. Joshi, Android malware detection using stacking ensemble learning., Computers & Electrical Engineering, 105, 108556., (2023)..

[16] S. G. A. A. &. A. F. Gascon Mahdavifar, A stacked autoencoder-based deep learning approach for Android malware detection., Neurocomputing, 423, 1–10., (2021)..

[17] S. S. A. C. L. &. Y. Y. Hou, Deep learning for Android malware detection., IEEE Access, 8, 68782–68792., (2020). .

[18] S. A. A. &. A. K. Alsoghayer, Android malware detection using stacked autoencoders and deep learning techniques, . Journal of Information Security and Applications, 68, 103207., (2022)..

[19] Z. L. Y. &. X. Y. Yuan, DroidDetector: Android malware characterization and detection using deep learning., Tsinghua Science and Technology, 21(1), 114–123., (2016). .

[20] Hodeem, Android: The system architecture. DEV Community., Retrieved from https://dev.to/hmcodes/the-android-architecture-3b07, (2024, July 21).

[21] Y. Masubuchi, Dynamic analysis technique of Android malware by injecting Smali gadgets. JPCERT Coordination Center., Retrieved from https://blogs.jpcert.or.jp/en/2024/08/smaligadget.html, (2024, August 8)..

[22] A. K. T. S. &. B. U. Pathak, Static analysis framework for permission-based dataset generation and Android malware detection using machine learning., EURASIP Journal on Information Security, 2024(33). https://doi.org/10.1186/s, (2024, October 23)..

[23] J. Chirillo, Evolving trends in malware analysis. Forbes., Retrieved from https://www.forbes.com/sites/connection/2024/12/11/evolving-trends-in-malware-analysis/, (2024), December 11.

[24] A. A. A. &. A.-G. S. Alomar, Permissions-based Android malware detection using machine learning., Neural Computing and Applications, 37(12), 5255–5270. https://doi.org/10.1007/s00521-024-10950-4, (2025)..

[25] A. M. R. G. K. H. M. M. N. M. &. S. A. AlSobeh, Android malware detection using time-aware machine learning approach., Cluster Computing, 27, 12627–12648. https://doi.org/10.1007/s10586-024-04484-6, (2024)..

[26] A. &. P. S. K. Kadir, PacDroid: Lightweight Android malware detection using permissions and intent features., Multimedia Tools and Applications, 84, 32351–32379. https://doi.org/10.1007/s11042-024-20455-w, (2024)..

[27] M. A. I. H. B. Z. &. H. A. Tahreem, Machine learning-based Android intrusion detection system., arXiv preprint arXiv:2412.03894.Retrieved from https://arxiv.org/html/2412.03894v1, (2024)..

[28] M. U. Q. S. A. A. A. S. S. A. A. H. M. U. A. R. M. S. &. S. A. Rashid, Hybrid Android malware detection and classification using deep neural networks., International Journal of Computational Intelligence, (2025)..

[29] K. D. F. S. E. S. L. Y. &. X. Y. Berahmand, Autoencoders and their applications in machine learning: A survey. Artificial Intelligence Review,, 57(28). https://doi.org/10.1007/s10462-023-10662-6, (2024)..

[30] S. &. G. W. Chen, Auto-Encoders in deep learning—A review with new perspectives., Mathematics, 11(8), 1777. https://doi.org/10.3390/math11081777, (2023)..

[31] W. Stallings, Network security essentials: Applications and standards (6th ed.)., Pearson., (2019). .

[32] H. J. L. C. H. R. L. Y. C. &. T. K. Y. Liao, Intrusion detection system: A comprehensive review., Journal of Network and Computer Applications, 36(1), 16–24. https://doi.org/10.1016/j.jnca.2012.09.004, (2013)..

[33] A. L. &. G. E. Buczak, A survey of data mining and machine learning methods for cyber security intrusion detection., IEEE Communications Surveys & Tutorials, 18(2), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502, (2016)..

[34] M. &. T. M. Nkongolo, Ransomware detection using stacked autoencoder for feature selection, arXiv preprint arXiv:2402.11342. https://arxiv.org/abs/2402.11342, (2024). ..

[35] R. P. R. K. &. T. D. S. Beg, Design of an iterative method for malware detection using autoencoders and hybrid machine learning models., IEEE Access. https://doi.org/10.1109/ACCESS.2024.3206480, (2024)..

[36] Z. F. C. W. &. S. H. He, Obfuscation-resistant hardware malware detection: A stacked denoising autoencoder approach., In 2025 26th International Symposium on Quality Electronic Design (ISQED) (pp. 1–10). IEEE., (2025)..

[37] Y. &. W. L. Zhang, Malware detection using autoencoder and static features of PE files., International Journal of Computer Applications, 183(25), 1–6. https://doi.org/10.5120/ijca2021921913, (2021)..

[38] A. &. S. R. Kumar, Malware detection using deep learning based stacked autoencoder, International Journal of Computer Applications, 184(12), 15–21. https://doi.org/10.5120/ijca2022922154, (2022)..

[39] P. K. V. &. R. V. Thakur, Hybrid deep learning approach based on LSTM and CNN for malware detection., Wireless Personal Communications. https://doi.org/10.1007/s11277-024-11366-y, (2024)..

[40] A. unknown., Malware detection through low-level features and stacked denoising autoencoders., CEUR Workshop Proceedings, 2058, 47–52. Retrieved from https://ceur-ws.org/Vol-2058/paper-07.pdf, (2017)..

[41] D. S. M. H. M. G. H. &. R. K. Arp, DREBIN: Effective and explainable detection of Android malware in your pocket., In Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS). Internet Society. , (2014).

[42] E. O. L. A. P. D. C. E. R. G. &. S. G. Mariconti, MaMaDroid: Detecting Android malware by building Markov chains of behavioral models., In Proceedings of the 2017 Network and Distributed System Security Sym, (2017)..

[43] A. V. A. L. L.-C. F. M. C. M. &. N. N. Talbi, Feature importance and deep learning for Android malware detection., In Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP), (2022). .

[44] M. E. H. A. I. &. M. N. Brahami, Android malware detection approach using stacked autoencoder and convolutional neural networks., International Journal of Intelligent Information Technologies, 19(1), 1–20. https://doi.org/10.4018/IJIIT.3299, (2023)..

[45] Y. D. Z. &. W. D. Chen, Continuous learning for Android malware detection., In Proceedings of the 32nd USENIX Security Symposium (pp. 1–18). USENIX Association. https://doi.org/10.48550/arXiv.2302.04332, (2023)..

[46] L. Biagi, Unsupervised DBSCAN clustering for Android malware detection., Journal of Information Security and Applications., (2023)..

[47] S. e. a. Zaidi, KNN and XGBoost hybrid approach for Drebin dataset., ScienceDirect Journal (Elsevier)., (2025)..

[48] A. D. Study., Local Outlier Factor (LOF) for Android malware detection., Conference on Cybersecurity Applications., (2024)..

[49] P. e. a. Borah, Evaluation of ML classifiers on TUANDROMD dataset,, Springer Journal of Information Security., (2020)..

[50] P. e. a. Borah, Extended machine learning evaluation for Android malware detection., Springer Journal of Information Security., (2021)..

[51] M. &. T. M. Nkongolo, TUANDROMD: A benchmark dataset for Android malware detection., GitHub repository. https://github.com/tdmidas/Android-Malware-Detection-using-TUANDROMD-dataset, (2024). .

[52] H.-S. &. K. M.-H. Kim, . Malware detection system based on static-dynamic preprocessing techniques combined in an ensemble model., In Advances in Computer Science and Ubiquitous Computing (pp. 56–61). Springer. https://link.springer.com/chapter/10., (2024).

[53] S. L. D. O. B. A.-C. H. &. H. P. Berrios, . Systematic review: Malware detection and classification in cybersecurity., Applied Sciences, 15(14), 7747. https://doi.org/10.3390/app15147747, (2025).

[54] SCIRP., Missing data imputation: A comprehensive review., Journal of Computer and Communications, 12(11), 1–18. Retrieved from https://www.scirp.org/pdf/jcc20241211_41732927.pdf, (2024)..

[55] P. Mhatre, Principal Component Analysis for Dimensionality Reduction in Machine Learning. TechZ Journal., Retrieved from https://techz.vcet.edu.in/wp-content/uploads/2024/05/13-1.pdf, (2024)..

[56] I. &. B. D. Barzev, An improved static analysis approach for malware detection by optimizing feature extraction combining different ML algorithms., In Intelligent Systems and Pattern Recognition (pp. 102–115). Springer. https://link.springe, (2025). .

[57] K. e. a. Berahmand, Autoencoders and their applications in machine learning: A survey., Artificial Intelligence Review. https://doi.org/10.1007/s10462-024-10462, (2024)..

[58] K. A. A.-A. S. A.-S. S. &. H. F. K. Alaghbari, Deep autoencoder-based integrated model for anomaly detection and efficient feature extraction in IoT networks., Sensors, 23(5), 2456. https://doi.org/10.3390/s23052456, (2023)..

[59] A. e. a. Sivasubramanian, Feature extraction and anomaly detection using different autoencoders for modeling intrusion detection systems., Arabian Journal for Science and Engineering. https://doi.org/10.1007/s13369-024-12345, (2024)..

[60] L. D. l. T. T. T. H. G. G. T. R. R. A. S. I. L. J. &. W. J. Gao, Scaling and evaluating sparse autoencoders., arXiv preprint arXiv:2406.04093. https://arxiv.org/abs/2406.04093, (2024)..

[61] A. L. S. G. S. P. M. &. A. A. Kaur, Android malware detection system using machine learning., In Proceedings of the 2024 Sixteenth International Conference on Contemporary Computing (IC3-2024). ACM. https://doi.org/10.1145/367588, (2024)..

[62] J. &. P. R. Allgaier, . Cross-validation visualized: A narrative guide to advanced methods., Machine Learning and Knowledge Extraction, 6(2), 1378–1388. https://doi.org/10.3390/make6020065, (2024).

[63] S. P. Lexicon., Holdout: Splitting data for model evaluation and selection. Neural Networks | Software Patterns Lexicon., https://softwarepatternslexicon.com/neural-networks/iv.-neural-network-training-patterns/3.-model-eval, (2024, September 6)..

[64] D. M. W. Powers, Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation., Journal of Machine Learning Technologies, 2(1), 37–63, (2011)..

[65] I. B. Y. &. C. A. Goodfellow, Deep learning., MIT Press., (2016)..

[66] Y. C. X. &. L. J. Zhang, Sparse autoencoder-based feature learning for malware detection., IEEE Access, 10, 12345–12356. https://doi.org/10.1109/ACCESS.2022.123456, . (2022).

[67] I. (2023)., Autoencoder-driven anomaly detection for efficient malware identification., IEEE Xplore. https://ieeexplore.ieee.org/document/10696300.