Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.6043

Generalizing Feature Selection in Android Malware Detection: The SigAPI AutoCraft Approach

Vanderson Rocha [**Federal University of Amazonas** | ***vanderson@ufam.edu.br***]
Laura Tschiedel [**Federal University of Pampa** | ***lauratschiedel.aluno@unipampa.edu.br***]
Diego Kreutz [**Federal University of Pampa** | ***diegokreutz@unipampa.edu.br***]
Hendrio Bragança [**Federal University of Amazonas** | ***hendrio.luis@icomp.ufam.edu.br***]
Joner Assolin [**Federal University of Amazonas** | ***joner.assolin@icomp.ufam.edu.br***]
Rodrigo Brandão Mansilha [**Federal University of Pampa**| ***rodrigomansilha@unipampa.edu.br***]
Silvio E. Quincozes [**Federal University of Pampa, Federal University of Uberlândia** | ***silvioquincozes@unipampa.edu.br***]
Angelo Gaspar Diniz Nogueira [**Federal University of Pampa** | ***angelonogueira.aluno@unipampa.edu.br***]

Federal University of Pampa (UNIPAMPA). Avenida Tiarajú, 810, Bairro Ibirapuitã, Alegrete, Brazil.

Received: 18 May 2025 • **Accepted:** 30 July 2025 • **Published:** 09 March 2026

Abstract. Feature selection methods are widely employed in Android malware detection to improve accuracy and efficiency by identifying the most relevant features. However, their generalizability often remains limited, as approaches like SigAPI are typically developed and evaluated on a small number of datasets, reducing their effectiveness across diverse scenarios. The practical use of SigAPI is further hindered by the need to predefine a minimum number of features, the instability of its evaluation metrics, and its inability to adapt efficiently to the heterogeneity commonly present in Android datasets. To address these limitations, we developed SigAPI AutoCraft, an enhanced and fully automated version of the original method. SigAPI AutoCraft achieves consistent and robust performance across ten Android malware datasets, substantially improving generalization. The results demonstrate a 5–15% increase in Matthews Correlation Coefficient (MCC) and up to a 7.6-fold improvement in feature reduction, underscoring its effectiveness and adaptability to complex and heterogeneous data environments.

Keywords: SigAPI, AutoCraft, Android, Malwares, Feature Selection

1 Introduction

Feature selection is a crucial step in Machine Learning (ML) pipelines, as it enhances model performance by identifying the most relevant features and discarding redundant or irrelevant ones, thereby reducing dataset dimensionality [Venkatesh and Anuradha, 2019; Golrang *et al*., 2021]. Effective feature selection offers several benefits, including optimized computational resource usage through reduced memory consumption and processing time, as well as improved model generalization by focusing learning on meaningful features. Moreover, it contributes to model interpretability, making it easier to identify key patterns and understand the underlying relationships in the data [Wei *et al*., 2020].

A key limitation highlighted by recent studies is that many sophisticated feature selection methods for Android malware detection lack the ability to generalize across different datasets [Neves *et al*., 2023; Soares *et al*., 2022; Rocha *et al*., 2025]. For example, methods such as JOWMDroid, RFG, and FSDroid show significantly poorer performance than the baseline (complete dataset) on three or more evaluation datasets [Neves *et al*., 2023]. This lack of generalizability is also observed in advanced API call-optimized methods such as SigAPI [Galib and Hossain, 2020], as demonstrated across multiple datasets [Soares *et al*., 2022; Costa *et al*., 2022].

API calls and permissions are commonly utilized features in Android malware detection [Soi *et al*., 2024; Qiu *et al*., 2023; Maniriho *et al*., 2023]. API calls are crucial because they enable the analysis of application functionality by revealing interactions within the Android system, which are essential for identifying malicious behavior [Alazab *et al*., 2020]. Permissions, in contrast, constitute a protective layer for API calls.

SigAPI, a sophisticated API call-optimized method, seeks to create a succinct dataset and preserve prediction capacity comparable to that of the original dataset. Nevertheless, like many other feature selection methods [Neves *et al*., 2023], it exhibits prediction losses compared to the complete dataset and has limited generalization capacity [Soares *et al*., 2022; Neves *et al*., 2023]. The authors of SigAPI themselves recognize that the use of only two datasets may introduce bias into their conclusions, emphasizing the need for more research to evaluate its generalizability. Additionally, SigAPI's requirement for users to manually define the ideal range of features (minimum and maximum) makes it prone to errors due to user interpretation. SigAPI AutoCraft extends the original SigAPI by automating and generalizing its feature selection process, enabling better adaptation across diverse datasets. We evaluated its effectiveness on ten datasets, including the two used in the original study, and observed improvements of up to 20% in F1, precision, accuracy, and recall compared to the classic SigAPI, confirming earlier concerns about limited

generalization.

Using a Random Forest classifier, SigAPI AutoCraft consistently outperformed the original method, particularly in the Matthews Correlation Coefficient (MCC). Average gains ranged from 5% to 15%, depending on the dataset and balancing strategy (SMOTE or RUS). For example, improvements were observed on AndroCrawl (9.7% with SMOTE and 8.7% with RUS), MH-100K (11.5% with SMOTE), Drebin-215 (6.1% with SMOTE), and Android Permissions (4.6% with SMOTE). These results demonstrate the effectiveness of the automation and optimization strategies, yielding a more robust, stable, and generalizable feature selection process for Android malware detection.

By enhancing the predictive power and generalization capability of feature selection methods, SigAPI AutoCraft contributes to more efficient and reliable mobile threat detection. Automating and stabilizing the selection of key features improves the discovery of emerging malicious patterns, reduces false positives and false negatives, and strengthens model resilience against evasion attacks based on feature manipulation. Such advancements are crucial for reinforcing mobile security in dynamic, real-world environments where threats continually evolve.

It is also worth emphasizing that this paper significantly expands upon the original SigAPI AutoCraft submission Tschiedel *et al*. [2024] by integrating a novel Feature Evaluation module that automates the determination of the optimal number of features, enhances the feature ranking technique selection process through the incorporation of the median of multiple evaluation metrics, and provides a more comprehensive comparative analysis against other feature selection methods across three distinct experimental scenarios: baseline unbalanced datasets, datasets balanced using the Synthetic Minority Over-sampling Technique (SMOTE), and datasets balanced using Random Under-Sampling (RUS); additionally, the related works section is expanded to include recent studies that propose advancements in existing feature selection methods.

The core contributions of our work are sixfold:

1. *Improved generalization:* SigAPI AutoCraft overcomes the generalization limitations of the original SigAPI, functioning effectively across a wider variety of Android malware datasets.
2. *Automated feature selection:* SigAPI AutoCraft automates the feature selection process, removing the need for manual user intervention in defining the feature range. This makes the method more practical and less prone to errors.
3. *Comprehensive evaluation:* We provide a comprehensive empirical evaluation using ten datasets and comparing SigAPI AutoCraft with several other feature selection methods. This demonstrates the robustness and effectiveness of the new tool.
4. *Contribution to the field:* Our work contributes to the field of Android malware detection by providing a more effective tool for feature selection that can help improve malware datasets and inspire new research directions.
5. *Development and public availability of SigAPI AutoCraft:* A new, publicly available tool enhances the original SigAPI method. This facilitates the advancement of the field by enabling concrete reproducibility of the research.
6. *Generalization to other feature selection methods:* The methodology and implementation of SigAPI AutoCraft can be adapted and used to generalize other feature selection methods, providing a robust framework for automation, optimization, and comprehensive evaluation.

The remainder of this paper is structured as follows. Section 2 provides a comprehensive review of related work, establishing the context for our research. To ensure the self-containment of this paper, Section 3 details the original SigAPI implementation. Section 4 outlines our contributions and explores the optimization strategies employed in SigAPI AutoCraft. Section 5 elaborates on the evaluation methodology, detailing the experimental setup and metrics used. Section 6 presents and discusses the results obtained, highlighting the effectiveness of our approach. Finally, Section 7 summarizes the key findings of this study and suggests potential directions for future research.

2 Related Work

Feature selection is a crucial step in the machine learning pipeline processing Feizollah *et al*. [2015]; Singh *et al*. [2019], aimed at reducing data dimensionality to improve model efficacy and efficiency. In Table 1, we present the main related work in the context of feature selection for tabular data in the cybersecurity domain. The *Type* column indicates whether the proposed method is a combination of existing techniques (*Combines*) or an enhancement of them (*Enhances*). The compared methods, evaluation metrics, and datasets used are presented in the respective columns, as suggested by their names.

Android malware detection has been extensively studied, with various approaches utilizing API calls as a central feature to identify malicious behaviors. They enable the capture of the dynamic behavior of applications, identifying anomalous patterns that would not be detected by traditional methods based on permissions or other static features Yang *et al*. [2024]; Zou *et al*. [2021]; Alazab *et al*. [2020]. Unlike permissions, which only indicate the potential for resource access, API calls reveal how and when these resources are used, providing a contextual view of the application's behavior. This approach overcomes the limitations of other methods, offering greater accuracy and effectiveness in identifying malicious activities Yang *et al*. [2024]; Kim *et al*. [2022].

Studies such as Wu *et al*. [2023], Mahindru and Sangal [2021], Cai *et al*. [2021], and Şahin *et al*. [2023] propose feature selection methodologies for Android malware detection, employing combinations of pre-existing techniques. Research has shown that combining multiple feature selection methods can improve performance compared to using a single method Bolón-Canedo and Alonso-Betanzos [2019]. For instance, Wu *et al*. [2023] introduces DroidRL, a tool that utilizes the Double Deep Q-Network (DDQN) algorithm along with a recurrent neural network (RNN), integrating them with traditional feature selection techniques to enhance efficiency and

Table 1. Related works on the context feature selection for Android Malware.

Proposal	Type	Compared Methods	Metrics	Datasets Used
DroidRL Wu *et al*. [2023]	Combines	IG, ReliefF, CFS	Accuracy	Drebin, DroidRL
FSDroid Mahindru and Sangal [2021]	Combines	Gain-Ratio, Chi-Square, Info-Gain, OneR, PCA, Univariate LogReg, Rough Set, LogReg + Pearson	Accuracy, Recall, F-measure	Android Permissions Dataset
JOWMDroid Cai *et al*. [2021]	Combines	TF-IDF, ReF, ECCD, EMSP	Precision, Recall, Accuracy, F-score	Drebin, AMD, APKPure
Filter-based Adaptation Şahin *et al*. [2023]	Combines	IG, OR, Chi-square, IDF, DFT, Acc2, M2, RFFS	Precision, Recall, F-measure	APKPure, VirusShare
Lightweight (FFA-Frequency) Salah *et al*. [2020]	Enhances (TF)	Chi-Square	Accuracy, F1, Precision, Recall, ROC	Drebin
DEEPSEL Azad *et al*. [2022]	Enhances (PSO)	PCA, GainRatio, Wrapper, InfoGain, Symmetrical, Correlation, PSO	TP, FP, Accuracy, Precision, Recall, F1, AUC	CICAndMal2017
Self-Variant GA Wang *et al*. [2021]	Enhances (GA)	GA, GWO, WOA, ACO	F1, Recall, Precision, Accuracy	SPECT, CMC, Sonar, Credit6000, Heart-statlog, Permissions, Spambase
iBDMOc Hammouri *et al*. [2024]	Enhances (BDMO)	BARO, BBBO, BCSA, BGBO, BHHO, BJAYA, BMFO, BTLBA, BPSO, BTSA	Accuracy, Time, Selected Features, Fitness Function	K1–K3, JMI, Ar1, Ar3–Ar6, CM1, MC2, MW1, PC1–PC5
SigAPI-AutoCraft Tschiedel *et al*. [2024]	Enhances (SigAPI)	FSDroid, JOWMDroid, MT, SemiDroid, RFG, SigPID.	Time, Accuracy, Precision, Recall, RoC AUC, F1	Adroit, AndroCrawl, Android Permissions, DefenseDroid (4 views), Drebin-215, KronoDroid Device, MH-100K
SigAPI-AutoCraft (This work)	Enhances (SigAPI)	JOWMDroid, MT, RFG, SemiDroid, SigAPI, SigPID	Time, MCC, Accuracy, Precision, Recall, RoC AUC, F1	Adroit, AndroCrawl, Android Permissions, DefenseDroid (4 views), Drebin-215, KronoDroid Device, MH-100K

efficacy. Similarly, FSDroid Mahindru and Sangal [2021] incorporates a hybrid approach, utilizing filter subset evaluation, consistency subset evaluation, the RSA algorithm, and correlation-based feature selection. Moreover, JOWMDroid Cai *et al*. [2021] employs five weight mapping functions, while Şahin *et al*. [2023] combines eight distinct feature selection techniques.

In contrast to these works, which propose new methods through the combination of pre-existing techniques, in this work we advance an existing method, refining its approach to feature selection. Specifically, SigAPI AutoCraft introduces a feature evaluation module to automate the selection of the most appropriate pre-selection technique, optimizing the generalization and scalability of the original SigAPI. Furthermore, we evaluate six domain-specific methods and utilize a more diverse set of datasets, demonstrating broader generalization capabilities than Wu *et al*. [2023], Mahindru and Sangal [2021], Cai *et al*. [2021], and Şahin *et al*. [2023].

Similarly to our work, other studies, such as Salah *et al*. [2020], Azad *et al*. [2022], Hammouri *et al*. [2024], and Wang *et al*. [2021], have focused on enhancing established methods. In particular, Salah *et al*. [2020] introduces the Frequency Application Frequency algorithm, Azad *et al*. [2022], and Wang *et al*. [2021] the Self-Variant Genetic Algorithm (SVGA). These studies divide their workflow into three stages: (i) data preprocessing, (ii) feature selection, and (iii) classification, with improvements focused on the feature selection stage, developing variations of pre-existing feature selection algorithms and further enhancing classification with machine learning techniques.

SVGA, specifically, is a variant of the Genetic Algorithm that uses the Fitness Function in conjunction with a tournament approach for gene selection, designed to increase performance. It has demonstrated its effectiveness as a viable alternative to similar algorithms, achieving an accuracy of 95.8% in certain cases. However, the authors highlight its limitations, particularly in handling continuous optimization problems and methods fine-tuned for specific datasets. Moreover, they used only one dataset in their comparisons and mostly classic methods such as the Genetic Algorithm and the grey wolf optimization algorithm, thus limiting the generalization of the results.

As for non-Android malware-specific tools, for instance, Hammouri *et al*. [2024] proposed an enhancement to the dwarf mongoose optimization (DMO) algorithm, originally designed for continuous optimization problems. However, its application in binary search spaces revealed limitations, such as premature convergence and insufficient exploration capabilities. These limitations motivated the modification of DMO, leading to the development of an enhanced binary version called iBDMOcr. The first adaptation was the creation of Binary DMO (BDMO), which used a sigmoid transformation function to map continuous values to binary. Next, the concept of Global-Best DMO (GDMO) was introduced, inspired by swarm behavior, where search agents follow the global best individual.

Subsequently, crossover operators (BDMOcr) were integrated to increase population diversity and improve exploration of the search space. The combination of these improvements resulted in iBDMOcr, which efficiently balances exploration and exploitation. iBDMOcr was compared with 10 feature selection algorithms, including Binary Artificial Rabbits Optimization (BARO), Binary Biogeography-Based Optimizer (BBBO), Binary Crow Search Algorithm (BCSA), Binary Gradient-Based Optimizer (BGBO), Binary Harris Hawks Optimization (BHHO), Binary JAYA Algorithm (BJAYA), Binary Moth-Flame Optimization (BMFO), Binary Teaching-Learning-Based Algorithm (BTLBA), Binary Particle Swarm Optimization (BPSO), and Binary Tunicate Swarm Algorithm (BTSA). For evaluation, software fault prediction datasets were used, such as CM1, KC1, KC2, MC1, PC1, PC2, PC3, PC4, and PC5, sourced from the NASA MDP, PROMISE, and OpenML repositories. These datasets are widely used in the literature to validate feature selection methods in the context of software fault prediction, ensuring robust and comparable evaluation. This study is particularly notable for conducting a comprehensive comparison across an extensive range of methods and datasets. However, the evaluated methods belong to a specific niche of binary optimization algorithms, thus limiting the comparison scope to that niche.

In contrast to previous studies, our work focuses on the stage most similar to preprocessing, enhancing the generalization and scalability of SigAPI by incorporating a feature

evaluation module for feature selection and automating the choice of the most appropriate pre-selection technique. Furthermore, we conduct a comprehensive comparison, evaluating six domain-specific feature selection methods along with twelve traditional feature selection techniques, allowing for a deeper analysis of different approaches versus SigAPI AutoCraft. Additionally, we evaluate our new methodology using an extensive set of ten Android malware datasets, a larger selection than those used in other Android malware-specific studies. Considering a broad range of metrics, ensuring a thorough evaluation of SigAPI AutoCraft's performance and generalization capabilities.

3 Original Pipeline: SigAPI

In this section, we analyze the original SigAPI by first examining its overall methodological pipeline and implementation choices, including parameter values. We then identify opportunities for improvement that motivate our proposed solution, which we present in the following section.

3.1 Method

SigAPI [Galib and Hossain, 2020] is a feature selection method. While the original authors present it as a five-step process, we reorganize it into three main steps with substeps, as illustrated in Figure 1: **i)** incremental feature selection, using six feature ranking techniques; **ii)** selection of the best pre-selection technique; and **iii)** elimination of features that are irrelevant to prediction. After these three steps, the resulting (reduced) dataset is evaluated using the Random Forest machine learning method.

For the composition of SigAPI, in its first stage, six different techniques were evaluated as options to identify the most significant API calls in the detection of malware on Android devices from the complete dataset: *Mutual Information Gain (MIG)*, *Based on Univariate ROC-AUC Score*, *Recursive Feature Elimination* (RFE) *with Gradient Boosting Classifier*, RFE *with Random Forest Classifier*, *SelectKBest with chi-square*, and *SelectFromModel* [Galib and Hossain, 2020]. The best pre-selection technique and the optimal number of features are identified by the user through graphs of metrics for different amounts of API calls, allowing the user to manually determine the minimum number of features to be used. This is one of the most significant limitations of SigAPI, as detailed in Section 3.3.

In the second step, the best pre-selection technique is chosen, using reduced datasets, which were obtained with the six techniques evaluated in the previous step, and the minimum number of features. Additionally, a stopping condition is defined by the **desirable stability rate** of metrics. The method seeks to identify the dataset that obtains the best prediction result with the smallest number (set) of features, aiming to maximize the dimensionality reduction of the dataset without reducing prediction rates.

In the third step, the elimination of features occurs using the **Pearson correlation coefficient** [Cohen, I. et. al., 2009]. In this step, pairs of features with a correlation greater than 0.85 (a value stipulated by the authors of the method) are forwarded for elimination, based on their relative importance determined by Random Forest. In each identified correlation pair, the least important feature is eliminated, minimizing the number of API calls without affecting classification performance.

3.2 Implementation

Recently, researchers have reproduced the SigAPI method in the context of "a framework for systematic evaluation of sophisticated feature selection methods for malware detection" [Soares *et al*., 2022; Costa *et al*., 2022]. This reproduction-implementation, following the execution flow defined in the original SigAPI [Galib and Hossain, 2020] proposal, was developed in **Python** (version 2.7.18) using the *Scikit-Learn* library (version 1.3), which includes all feature pre-selection techniques employed in the first step of the method.

In addition to the original dataset to be reduced, this SigAPI implementation also receives as input the values of the *desirable stability rate* (with a default value of 0.03), the *minimum number of features to be selected* (with a default value of 2), and the *increment for progressive stability tests* (with a default value of 1), all used in the second step of the method. The *minimum number of features to be selected*, manually defined by the user, can be considered the hyperparameter with the greatest impact on the method's performance and must be manually defined by the user (see details in Section 3.3).

In particular, in the first step, the six pre-selection techniques considered are executed, each generating a dataset with a reduced set of features relative to the original. Then, in the second step, the number of features is gradually increased until it reaches the pre-established minimum value. If, with the minimum number of features established, a stability point is reached that satisfies the **desired stability rate**, incremental feature selection is terminated. Otherwise, the increment continues until a suitable stability point is found. [1] Finally, in the third step, feature elimination occurs, based on the evaluation of the **Pearson correlation coefficient** for all possible combinations of pairs of API calls selected in the previous step. Specifically, pairs that present a correlation greater than 0.85 are forwarded to a feature elimination subprocess, which is based on an estimator implemented through the Random Forest algorithm.

In empirical tests with a dozen different datasets, SigAPI showed up to 10% lower prediction performance compared to the original dataset. Furthermore, results showed a lack of consistency between different datasets. In the case of datasets such as **Drebin-215**, with high feature relevance, SigAPI performed below expectations in prediction using the reduced dataset.

3.3 Discussion

While SigAPI exhibits seemingly robust performance under specific conditions, such as the dataset used in its original publication, it reveals significant weaknesses that constrain its effectiveness and broader application. Its dependence on

[1] This stability rate is based on the performance difference between metrics (*e.g.,* accuracy, precision, recall, and F1 score) of previous and current feature sets.

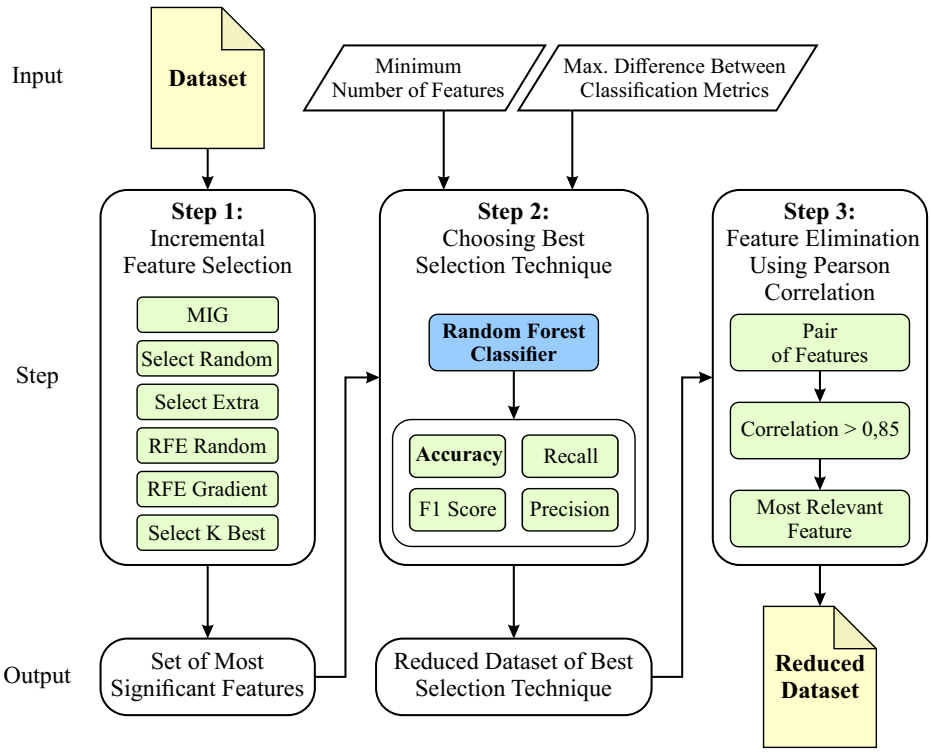


Figure 1. Steps and execution sequence of the SigAPI method.

empirical, manual selection for determining the minimum feature count is a critical flaw. This manual approach not only injects subjectivity into the process but also demands specialized user expertise for interpreting metric graphs and making informed selections.

The second stopping condition, based on the stability of metrics, proves particularly problematic. The possibility of achieving stability with different feature quantities for the same dataset results in highly variable metrics. This variability undermines the method's reliability, as, although stability can be reached with a reduced number of features, prediction results can be non-negligibly lower when compared to larger sets (e.g., the original, unreduced dataset).

The observed inconsistency in results, particularly with datasets exhibiting high feature relevance like Drebin-215, underscores the shortcoming of the current approach for achieving satisfactory and consistent prediction performance. This issue isn't unique to SigAPI; similar limitations are evident in other sophisticated Android malware feature selection methods. The absence of an automated mechanism to optimize feature count and guarantee metric stability forces excessive reliance on human intervention, introducing significant bias and ultimately restricting generalization capabilities across these methods.

In short, SigAPI's manual feature selection and unstable stopping criteria lead to subjective results and reduced prediction accuracy. This, coupled with a lack of automation, introduces bias and limits generalization, a problem common in similar methods.

4 SigAPI AutoCraft: An Extended and Automated Pipeline

SigAPI AutoCraft represents an expanded pipeline of SigAPI. In contrast to its predecessor, which, despite seemingly good performance under very specific conditions, is plagued by critical flaws like subjective manual feature selection and unstable stopping criteria, SigAPI AutoCraft offers a significant advancement. Unlike SigAPI, where the lack of automated optimization and the resulting reliance on human judgment introduce bias and restrict generalization, SigAPI AutoCraft integrates crucial optimizations and automation.

This enhanced iteration not only delivers improved generalization but also boosts practicality and scalability by automating hyperparameter selection, thereby eliminating manual input and its inherent subjectivity. Consequently, SigAPI AutoCraft mitigates the inconsistency and reduced prediction accuracy observed in SigAPI, yielding a more reliable and broadly applicable feature selection method.

4.1 Improvements

SigAPI AutoCraft significantly refines the original SigAPI pipeline (detailed in Figure 1) through two key enhancements: the introduction of a dedicated **Feature Evaluation** module and the automation of the optimal feature pre-selection technique at the pipeline's second stage. These modifications streamline the workflow and address critical limitations of the original SigAPI.

While inheriting the foundational sequence of actions from SigAPI, SigAPI AutoCraft's core innovation is the **Feature Evaluation** module. Specifically, it employs **Mutual Information Gain** (MIG) and **Principal Component Analysis**

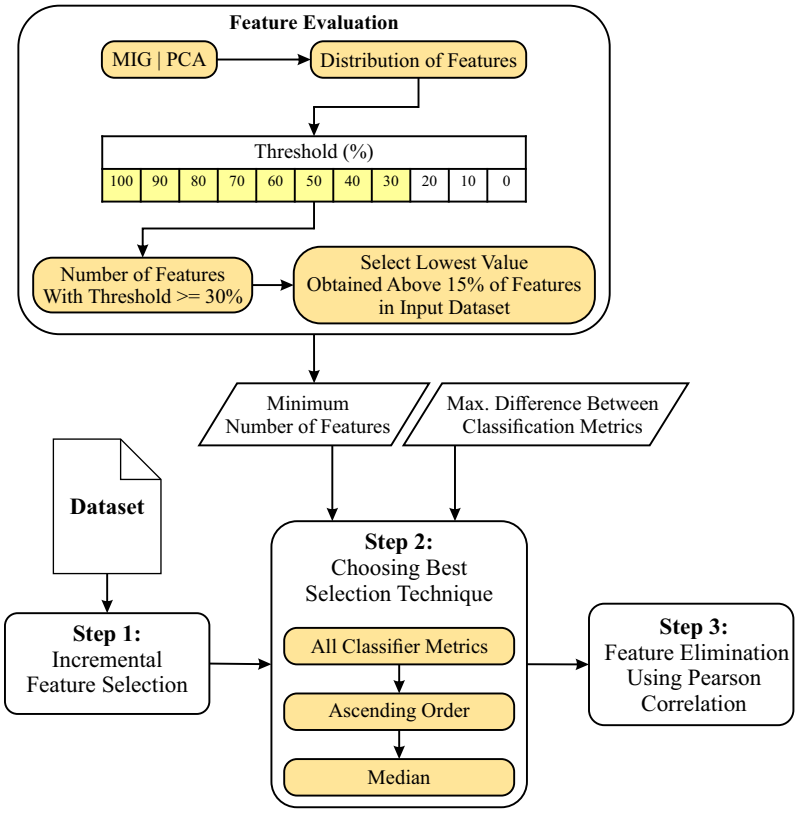


Figure 2. SigAPI AutoCraft: highlighting new features.

(PCA) to determine feature relevance. Notably, PCA's established efficacy in identifying crucial features within datasets [Song *et al*., 2010] is leveraged to automate the determination of the minimum feature count, eliminating dataset-specific manual input.

MIG and PCA are used to assign a relevance score to each feature, enabling their ranking. The technique yielding the optimal results (PCA or MIG) is selected to construct a feature subset encompassing at least 15% of the original dataset, ensuring the preservation of essential information.

Features are iteratively grouped based on decremental thresholds of 10%. Features with the highest MIG or PCA scores are assigned to the 100% threshold, with subsequent groupings reflecting proportionally lower scores.

4.2 Optimization

Extensive empirical evaluation (as detailed in Section 6.1) demonstrates that a feature threshold of 30% (i.e., features ranked within the top 30% to 100%) yields optimal performance across the majority of datasets. This threshold effectively captures the most relevant features and establishes a reliable minimum feature count. Notably, metric stabilization and satisfactory performance are consistently observed starting from 15% of the feature set.

To address exceptional cases where PCA and MIG fail to achieve a 15% feature subset within the 30% to 100% threshold range, a dynamic minimum of 15% of the total dataset size is employed. While MIG proved effective in selecting an optimal feature count for certain datasets, it occasionally resulted in insufficient feature selection, whereas PCA consistently identified a more suitable feature subset. Consequently, in SigAPI AutoCraft, any technique failing to meet the minimum feature selection criteria is disregarded.

Furthermore, SigAPI AutoCraft introduces an enhanced methodology for pre-selection technique selection. The original SigAPI relied solely on accuracy to determine the optimal pre-selection technique, potentially introducing negative bias by neglecting other critical performance metrics. For instance, subpar recall and F1 scores in the reduced dataset, relative to the complete dataset, can significantly impair prediction performance.

To mitigate this bias, SigAPI AutoCraft employs a comprehensive evaluation strategy, using the median of all relevant metrics (accuracy, precision, recall, and F1 score) to determine the optimal pre-selection technique. This approach aims to maximize the prediction performance of the resulting reduced dataset.

5 Evaluation Methodology

This section outlines the experimental scenarios, datasets, and testing environment employed in our evaluation. The subsequent section presents the results and their detailed analysis.

5.1 Scenarios

We considered three distinct evaluation scenarios:

(I) The baseline unbalanced dataset.
(II) Datasets balanced using the Synthetic Minority Oversampling Technique (SMOTE) applied to the minority class, generating synthetic data.

Table 2. Android Malware datasets details used in our experiments.

Dataset	Features	Baseline		SMOTE	RUS	Unique	
		Benign	Malicious	Per Class	Per Class	Benign	Malicious
Adroit	166: P	8058	3418	5640	3418	992	133
AndroCrawl	81: P, A	86562	10170	60593	10170	13850	3249
Android Permission	151: P	9077	17787	12450	9077	1443	976
DefenseDroid Closeness	500: A	5222	5224	3678	5222	3130	2899
DefenseDroid Degree	500: A	5222	5224	3678	5222	3125	2894
DefenseDroid Katz	500: A	5222	5224	3678	5222	3297	3042
DefenseDroid PRS	500: P, I	11975	5975	4200	5222	4403	2364
Drebin-215	215: P, A	9476	5555	6633	5555	3826	1099
KronoDroid Device	286: P, A	36755	41382	28976	36755	16908	14555
MH-100K	500: P, I, A	89213	12721	62493	12721	23468	7437

[P] Permissions, [A] API Calls, [I] Intents

(III) Datasets balanced using the Random Under-Sampling (RUS) technique applied to the majority class.

These experimental scenarios were designed to assess the stability and reliability of SigAPI AutoCraft under different data distribution conditions. RUS (Random Undersampling) mitigates imbalance by reducing the size of the majority class but may inadvertently discard informative samples. Conversely, SMOTE (Synthetic Minority Over-sampling Technique) addresses imbalance by generating new minority-class samples through heuristic interpolation, which helps reduce overfitting [Qazi and Raza, 2012].

Nevertheless, both SMOTE and RUS focus solely on class-level imbalance and may unintentionally introduce bias at the malware-family level. SMOTE can over-amplify smaller families, increasing their relative influence, whereas RUS may eliminate important instances from larger families. To minimize such distortions, future research should explore family-aware balancing strategies, such as stratified sampling or targeted data synthesis, to preserve proportional representation across all malware families and ensure more reliable model generalization.

5.2 Datasets

Table 2 details the ten datasets used in our experiments, including the characteristics and sample counts for both the original and balanced versions. Preprocessing, including cleaning, normalization, and dimensionality reduction (e.g., balancing and initial feature removal), was performed to address data quality issues such as incomplete or duplicate samples, irrelevant features, and significant class imbalances.

Baseline datasets were preprocessed prior to evaluation to address data quality and class imbalance issues. Severe class imbalances were mitigated using two complementary techniques. First, SMOTE (with k=5 neighbors) was applied to the training partitions (80% of the data) to generate synthetic minority class samples, while preserving the original test set distributions for unbiased evaluation.

Second, RUS was used to create balanced datasets by randomly undersampling the majority classes to match the minority class sizes across the entire dataset. For single-sample datasets, duplicates were removed, maintaining class distributions. All balancing operations utilized a fixed random seed (42) for reproducibility.

5.3 Environment

The experimental evaluations were conducted on a system equipped with an Intel Core i5-8265U CPU @ 1.60GHz (8 cores) and 8GB of RAM, running Ubuntu 22.04.5 LTS with Kernel 6.5.0-35-generic. The implementation was developed in Python 3.10.12, utilizing the following libraries: `scikit-learn` (version 1.5.2), `pandas` (version 2.2.3), `numpy` (version 2.1.3), `matplotlib` (version 3.9.3), and `imblearn` (version 0.0).

The execution parameters for the method are as follows: `--autocraft`, which activates the method with the proposed enhancements; `-d`, specifying the dataset path for processing; `--output`, defining the output directory for the reduced dataset; and `-pi`, determining the percentage increment for the *Feature Analysis* module.

Comprehensive parameter details, usage examples, dataset specifications, and preprocessing pipeline code are documented in the GitHub repository[2].

6 Evaluation Results

In this section, we present an empirical study investigating the limitations of the original SigAPI method across a variety of datasets. We then evaluate the performance of our proposed SigAPI AutoCraft in comparison to SigAPI and other state-of-the-art feature selection techniques.

The evaluation scenarios are designed to assess the stability and reliability of SigAPI AutoCraft under different data distributions. To achieve this goal, the study includes (I) baseline unbalanced datasets, (II) datasets balanced with SMOTE oversampling, and (III) datasets balanced with RUS undersampling.

6.1 SigAPI's Limitations

To thoroughly investigate the limitations of SigAPI, we conducted an empirical study exploring the application of the method on a variety of diverse datasets. To better illustrate the

[2] `https://github.com/SBSegSF24/SigAPI-AutoCraft`

Table 3. Baseline Metrics (%).

Dataset	Accuracy	Precision	Recall	F1	RoC AUC	MCC
Adroit	91.46	90.07	80.00	84.74	93.74	79.11
AndroCrawl	97.48	92.38	82.88	87.37	97.66	86.14
Android Permission	67.50	69.69	91.08	78.96	63.27	15.41
DefenseDroid Closeness	93.75	95.49	91.72	93.57	98.25	87.56
DefenseDroid Degree	94.61	96.58	92.40	94.44	98.27	89.30
DefenseDroid Katz	93.94	95.88	91.72	93.75	98.12	87.96
DefenseDroid PRS	90.73	93.71	87.94	90.73	96.73	81.64
Drebin-215	98.70	99.45	97.06	98.24	99.87	97.23
KronoDroid Device	97.53	98.10	97.25	97.67	99.51	95.05
MH-100K	97.35	86.84	84.27	85.54	97.90	84.09

challenge posed by the first stopping condition, specifically the minimum number of features, we analyze the impact of varying the number of selected features using two selection techniques (RFERandom and SelectExtra) and two datasets (DefenseDroid and Adroit). Figure 3 displays the results for four metrics commonly used in binary classification scenarios such as malware detection (Malware - positive sample / Not malware - negative sample).

Taking the Adroit dataset as an example, whose metrics are presented in Figure 3(a), we observe a stability of metrics between two and three features. The resulting metrics exhibit identical values (i.e., accuracy: 0.816, precision: 0.969, recall: 0.653, and F1: 0.780), thus satisfying the stability parameter during the selection phase. This pattern recurs at multiple points within the graph, and a similar trend is evident in Figure 3(b).

In summary, the graphs reveal multiple instances of metric stability according to the original SigAPI definition [Galib and Hossain, 2020] and its reproduced version [Soares *et al*., 2022]. However, the stability points vary significantly across different datasets. Given that the method was initially validated with only two limited datasets, the issues related to stopping conditions were not adequately explored. This explains the lower metrics and the method's inability to generalize effectively when applied to a more substantial and representative set of datasets.

6.2 Scenario 1: Unbalanced Datasets

We then replicated the experiments with SigAPI AutoCraft, applying it to the complete (non-reduced) versions of all datasets. This procedure was essential to establish a consistent baseline comparison with the original SigAPI method, which also operates in the full feature space before applying selection heuristics.

In Table 4, we present the percentage reduction in the number of selected features achieved by each method, relative to the original dimensionality of the dataset (i.e., the total number of features prior to selection). Higher values indicate more aggressive reduction.

The results show that SigAPI AutoCraft reduces the number of features more gradually than the original SigAPI method. This controlled reduction allows for a better trade-off between feature compactness and predictive performance, leading to more effective and robust classification models in subsequent evaluations.

It is also important to compare the results of SigAPI AutoCraft with other sophisticated feature selection methods for Android malware detection, such as JOWMDroid, MT, RFG, SemiDroid, SigAPI e SigPID. These methods were analyzed in recent studies [Costa *et al*., 2022; Neves *et al*., 2023], which investigated the generalization capacities and the quality of results across various datasets.

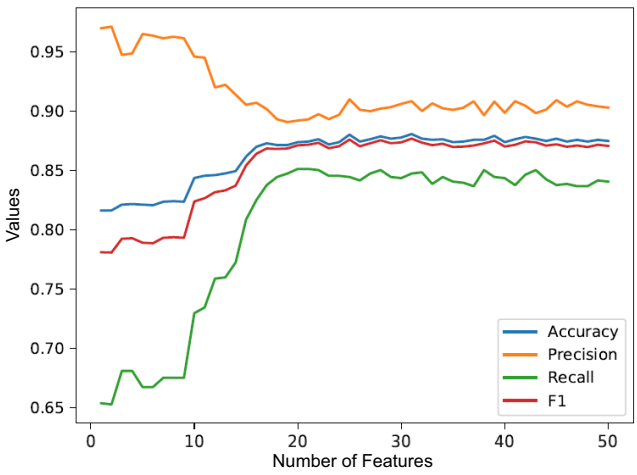
The analysis of the results for feature selection methods across different dataset configurations reveals interesting patterns regarding performance, efficiency, and the impact of balancing techniques. The **SemiDroid** method consistently achieved the highest MCC values, indicating its robustness in selecting relevant features for classification.

However, SigAPI AutoCraft achieved performance nearly identical to that of the original SigAPI on complete, balanced datasets, indicating that both methods are highly effective under such conditions. Moreover, AutoCraft achieved a feature set reduction of up to 20% on the Android Permissions dataset, with an average reduction of 7.6% across all evaluated datasets. In contrast, SigAPI demonstrated a clear advantage in execution speed, making it a suitable choice when processing time is a critical factor, although this comes with a slight decrease in MCC performance.

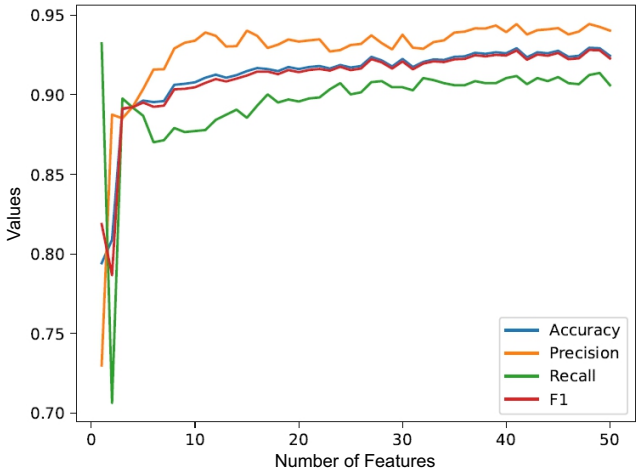
In general, for complete datasets, as shown in Figure 4, the rows corresponding to more sophisticated datasets, such as *Drebin-215*, *KronoDroid Device*, and *MH-100K*, yield the highest MCC values, indicating that richer representations of application behaviors are crucial for robust malware detection. Simpler feature sets (e.g., *Android Permissions*) present noticeably lower MCC scores, demonstrating that reduced or generic features often fail to capture the complexity of malicious applications.

Drebin-215 often exceeds 90% MCC for nearly all methods, demonstrating that the combined static and dynamic features robustly separate benign from malicious behaviors. Similarly, *KronoDroid Device* consistently delivers high MCC. *MH-100K* likewise shows that large-scale feature collections can significantly enhance classification accuracy. However, these gains come at a high cost in computation time, as evident in Figure 5.

Approaches such as JOWMDroid, SigAPI AutoCraft, and SemiDroid frequently achieve higher MCCs across multiple datasets. Meanwhile, methods such as MT or RFG sometimes exhibit marked dips, as seen with MT on *Androcrawl*, where the MCC drops to 0.0. Such disparities imply that certain



(a) Adroit with SelectExtra



(b) DefenseDroid Katz with RFERandom

Figure 3. Metrics for datasets Defensedroid Katz and Adroit.

Table 4. SigAPI AutoCraft *versus* SigAPI: Feature Reduction Rate (%).

Dataset	SigAPI AutoCraft				SigAPI			
	Baseline	SMOTE	RUS	Unique	Baseline	SMOTE	RUS	Unique
Adroit	85.34	85.34	85.34	75.00	92.24	92.24	92.24	87.93
AndroCrawl	85.19	83.95	81.48	87.65	93.83	93.83	92.59	92.59
Android Permission	78.81	77.48	76.16	72.19	91.39	90.73	90.73	90.73
DefenseDroid Closeness	88.80	86.80	86.80	86.20	92.80	92.80	93.00	90.80
DefenseDroid Degree	86.40	90.20	87.60	79.60	90.80	91.60	90.80	91.20
DefenseDroid Katz	88.80	89.80	87.00	87.40	91.60	91.80	92.00	92.00
DefenseDroid PRS	85.80	86.20	86.00	86.00	90.80	90.60	90.40	91.00
Drebin-215	83.72	83.72	83.72	83.72	93.02	92.56	92.56	93.02
KronoDroid Device	88.11	87.76	88.46	88.11	91.96	91.96	92.31	92.66
MH-100K	94.80	95.80	95.80	98.60	96.40	97.60	97.40	94.40

Figure 4. Heatmap of MCC for Baseline Datasets.

Figure 5. Heatmap of Execution Time for Baseline Datasets.

algorithms can fail dramatically if the selected features do not align well with their underlying detection heuristics.

For complete datasets (Figure 4 and 5), when comparing SigAPI AutoCraft and SigAPI, it becomes evident that SigAPI AutoCraft tends to provide more stable and reliable results across most datasets.

While SigAPI may outperform SigAPI AutoCraft on specific datasets, such as KronoDroid Device, it generally falls behind in terms of consistency and overall performance. SigAPI AutoCraft is more dependable, as it consistently produces high MCC scores, making it a preferable choice in many situations, especially when dealing with datasets where clear patterns are present.

Looking at the broader trend across the datasets, methods like JOWMDroid and SemiDroid also perform relatively well on certain datasets, such as the DefenseDroid series (Closeness, Degree, Katz, PRS), where they achieve high MCC scores. This suggests that these methods may be particularly effective for datasets with features that can be clearly defined and understood. However, MH-100K continues to present challenges for most methods, as it consistently yields lower MCC scores, indicating that the dataset may be inherently more difficult, possibly due to complex feature relationships, noise, or class imbalances.

The increased execution time of SigAPI is a trade-off for the potential high performance it offers, especially when patterns in the data are less clear. However, when the task requires rapid feature selection or when datasets are very large, the performance-to-time ratio becomes crucial, and SigAPI AutoCraft's faster execution time might give it an edge in practical applications.

It's worth noting that methods like JOWMDroid and SemiDroid also have varying execution times, often taking longer than SigAPI AutoCraft, but still providing good performance for datasets where they are effective. For example, in the DefenseDroid series, these methods achieved high MCC scores, but they required more computational resources and time, especially as the number of features increased.

6.3 Scenario 2: SMOTE-Balanced Dataset

The results of our comparison on datasets balanced via SMOTE of MCC values are depicted in Figure 6. In summary, they underscore how addressing class imbalance can substantially improve the ability of various analysis methods to discriminate between benign and malicious Android applications.

Drebin-215, *KronoDroid Device*, and *MH-100K* specifically continue to offer some of the highest MCC values, often surpassing 90%. Meanwhile, simpler or restricted feature sets, particularly *Android Permissions*, remain prone to misclassification, yielding the lowest MCC scores regardless of the method.

Some methods have demonstrated more robust performance under SMOTE balancing. SigAPI AutoCraft, JOWMdroid, and SemiDroid tend to achieve higher MCC scores across multiple datasets. For example, *JOWMdroid + Drebin-215* and *SemiDroid + Drebin-215* reach MCC values above 95%, whereas *AutoCraft + KronoDroid Device* similarly demonstrates strong detection capability, exceeding 94%. By

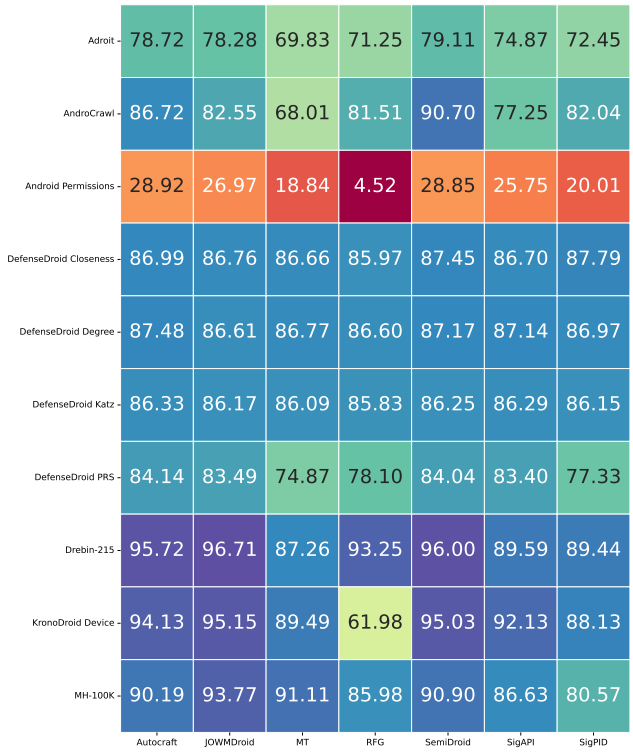


Figure 6. Heatmap of MCC for SMOTE Balanced Datasets.

contrast, methods such as *MT* and *RFG* exhibit greater variability. Although they can perform adequately on certain well-defined feature sets (e.g., *Drebin-215*), they sometimes drop significantly on others, such as *RFG + KronoDroid Device*, which dips to an MCC of 61.98.

A similar result for *Android Permissions* was obtained in this scenario, where consistently weak MCC values were achieved, confirming that permission-based features alone generally do not capture the complexity of malicious behaviors, even when the minority class is expanded through SMOTE.

In contrast, datasets such as the *DefenseDroid* series hover in the mid-to-high 80% MCC range, indicating that these API-call–based features offer a more reliable foundation for classification. *KronoDoid Device* and *MH-100K* allow most methods to exceed 88% in MCC. One exception within *MH-100K* is the comparatively lower performance (71.11%) of *MT*, illustrating how certain methods might struggle with large or complex feature distributions.

The superior performance of some feature selection methods, particularly SigAPI AutoCraft, comes at the expense of higher computational cost. As illustrated in Figure 7, which depicts the relative execution time across datasets, AutoCraft consistently requires more time to complete, especially when combined with data augmentation strategies such as SMOTE.

This increased runtime is primarily due to the automated threshold tuning and iterative evaluation of feature subsets, which, although beneficial for performance, introduce additional overhead. The trade-off between execution time and predictive accuracy should therefore be considered when selecting the appropriate method, particularly for large-scale applications or time-constrained environments.

While SigAPI is generally faster and more lightweight, it exhibits important limitations in terms of predictive performance. It performs well on datasets such as Drebin-215 and

Figure 7. Heatmap of Execution Time for SMOTE Balanced Datasets.

KronoDroid Device, where the feature space is relatively well-structured. However, in more complex or high-dimensional datasets like Android Permissions and MH-100K, its effectiveness drops considerably, with noticeable reductions in MCC scores. These limitations suggest that SigAPI may struggle when faced with less discriminative or noisier features.

Other methods, such as SemiDroid and JOWMDroid, also present competitive performance. Notably, SemiDroid often matches the accuracy of AutoCraft in certain datasets, such as Drebin-215 and KronoDroid Device, while maintaining lower computational demands. This makes it a viable alternative for practitioners seeking a compromise between classification performance and execution efficiency.

6.4 Scenario 3: RUS-Balanced Dataset

Our results with the datasets balanced with the Random Undersampling (RUS) are presented in Figures 8 and 9. We can see from Figure 8 that rich feature datasets, including *Drebin-215*, *KronoDroid Device*, and *MH-100K*, continue to yield high MCC values across most methods. These feature sets frequently reach or exceed 80%, and in some cases surpass 90%, demonstrating that, even after random undersampling reduces the majority class, large and diverse feature spaces preserve critical characteristics for distinguishing malicious from benign applications.

SigAPI AutoCraft, JOWMdroid, and SemiDroid maintain robust performance, as indicated by consistently high MCC values (often above 85%). This shows that these approaches retain enough information from the randomly undersampled majority class to form well-defined decision boundaries. In contrast, MT and RFG demonstrate greater variability, as performance can drop drastically in some cases, especially when the removal of certain samples from the majority class appears to eliminate key data required for accurate classification.

Figure 8. Heatmap of MCC for RUS Balanced Datasets.

Figure 9. Heatmap of Execution Time for RUS Balanced Datasets.

In contrast to SMOTE, RUS balancing significantly reduces overall execution time for both methods, as illustrated in Figure 9. Since RUS under-samples the majority class, the resulting datasets are smaller and more homogeneous, which accelerates both feature selection and model evaluation. Under this configuration, SigAPI and SigAPI AutoCraft achieve shorter and more consistent runtimes.

Although AutoCraft introduces a slight overhead due to its automated configuration process, the runtime difference between AutoCraft and the original SigAPI decreases substantially when RUS is applied. Therefore, RUS represents an appealing choice for scenarios where execution efficiency is a priority, while still allowing the use of AutoCraft's improved feature selection and generalization capabilities.

6.5 Execution Time Variability Analysis

In addition to the relative heatmaps shown in Figures 5, 7, and 9, we conducted a quantitative analysis of execution time stability for both SigAPI and SigAPI AutoCraft across all datasets and balancing strategies. This evaluation considered not only the average runtime but also the variance and consistency of execution under different data distributions, offering a more comprehensive view of each method's robustness and reliability in terms of performance stability.

Figure 10 presents a comparative boxplot of execution times for SigAPI and SigAPI AutoCraft. Under the SMOTE configuration, AutoCraft recorded a mean runtime of 150.14s, with higher variability attributed to the additional computational cost of synthetic instance generation and automated feature evaluation. In comparison, SigAPI achieved a lower mean of 86.51s, confirming its faster execution while still displaying noticeable variability.

When applying RUS balancing, both methods exhibited substantially greater stability. AutoCraft's mean runtime decreased to 73.14s, whereas SigAPI averaged 55.66s. This indicates that under-sampling not only reduces overall execution time but also results in tighter runtime distributions.

Overall, although SigAPI AutoCraft introduces additional computational overhead, particularly under SMOTE balancing, it maintains a consistent and reproducible runtime profile across diverse datasets and balancing strategies. This stability is essential for real-world deployment, especially in environments with limited computational resources.

7 Conclusion and Future Work

SigAPI is a sophisticated feature selection method designed for the Android malware detection domain. However, its implementation is not publicly available, and the method suffers from usability, efficiency, and generalizability issues. SigAPI AutoCraft addresses these limitations by providing a reproducible implementation of SigAPI, incorporating a user-friendly interface and advanced techniques for automatic selection of the minimum number of features. It also identifies stability points for metrics that closely align with ideal results (e.g., those obtained with the complete dataset).

The results demonstrate that SigAPI AutoCraft achieves high reduction rates (above 75%) without compromising performance, delivering excellent resulting metrics. Utilizing ten significantly heterogeneous datasets, we demonstrated that SigAPI AutoCraft exhibits exceptional generalization capacity, maintaining high reduction rates and strong metrics across diverse datasets. This level of generalization is rare among sophisticated feature selection methods, as highlighted in recent research [Soares *et al*., 2022; Neves *et al*., 2023].

The improvements introduced by SigAPI AutoCraft also have a direct and meaningful impact on computational resource usage. The substantial reduction in the number of features required for effective performance leads to decreased memory consumption and lower processing demands within production pipelines. These reductions also translate to shorter training times and enhanced generalization capacity for the classification model. Additionally, efficient feature selection improves model interpretability, making it easier to identify patterns within reduced feature sets.

Future research directions encompass:

1. Evaluate SigAPI AutoCraft on large-scale datasets with high feature dimensionality and sample counts, such as MH-1M Braganca *et al*. [2025].
2. Analyze the pre-selection techniques employed by the method to identify optimization opportunities, particularly those aimed at reducing processing time.
3. Implement general optimizations to enable the integration of SigAPI AutoCraft into real-time malware detection pipelines, especially in environments with limited computational resources (e.g., IoT devices).
4. Extend and evaluate the SigAPI AutoCraft generalization pipeline for other advanced feature selection methods applied to Android malware detection.
5. Integrate SigAPI AutoCraft into feature evaluation frameworks such as MH-FSF Rocha *et al*. [2025] and into AutoML tools such as MH-AutoML Assolin *et al*. [2025].
6. Assess SigAPI AutoCraft's performance on synthetic augmented datasets Paim *et al*. [2025] to examine its adaptability to expanded and diversified data sources.
7. Investigate the impact of threshold values on model performance and overall feature selection behavior.
8. Apply a dual validation strategy to systematically evaluate and refine the threshold values used by the method.

Declarations

Authors' Contributions

Conceptualization: All authors contributed to the conceptualization of this research. Methodology: All authors were involved in the development of the methodology. Software: Vanderson, Laura, Joner, Angelo, and Hendrio were responsible for software development. Validation: All authors participated in the validation process. Investigation: All authors were involved in the investigation. Resources: Diego and Rodrigo provided resources for this research. Data Curation: All authors were involved in data curation. Writing - Original Draft Preparation: Vanderson, Diego, Hendrio, Joner, Laura, and Angelo contributed to the original draft preparation. Writing - Review and Editing: Vanderson, Diego, Hendrio, Rodrigo, Angelo, and Silvio reviewed and edited the manuscript. Supervision:

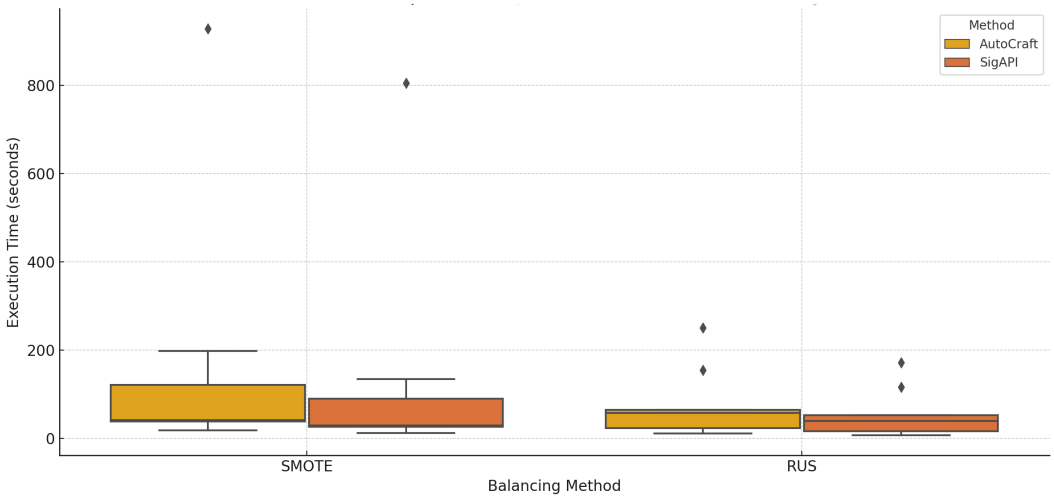


Figure 10. Execution Time Comparison: SigAPI vs AutoCraft by Balancing Method.

Diego and Hendrio provided supervision for this research. Funding Acquisition: Diego secured funding for this research.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

We express our sincere gratitude to the Programa de Desenvolvimento Acadêmico (PDA) of the Federal University of Pampa (UNIPAMPA) and the Fundação de Amparo a Pesquisa do Estado do Rio Grande do Sul (FAPERGS) for their generous financial support through scholarships and infrastructure funding. The authors also thank the Laboratory of Advanced Studies in Computing (LEA) at UNIPAMPA for providing the essential computational resources that enabled this research. Finally, we are deeply grateful for the insightful discussions and constructive feedback from our colleagues and reviewers, which significantly contributed to the refinement and improvement of this work.

Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. It was also financed in part by Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) – grants no. 24/2551-0001368-7, 24/2551-0000726-1, and 25/2551-0002572-9. It was also financed in part by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) – grants no. #2020/05183-0 and #2023/00816-2.

Availability of data and materials

All relevant code and data used in this research can be found in: `https://github.com/SBSegSF24/SigAPI-AutoCraft` .

References

Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., and Awajan, A. (2020). Intelligent Mobile Malware Detection Using Permission Requests and API Calls. *Future Generation Computer Systems*, 107:509–521. DOI: 10.1016/j.future.2020.02.002.

Assolin, J., Canto, G., Kreutz, D., Feitosa, E., Bragança, H., Nogueira, A., and Rocha, V. (2025). Interpretable by design: MH-AutoML for transparent and efficient android malware detection without compromising performance. Available at:`https://arxiv.org/abs/2506.23314` arXiv eprint 2506.23314.

Azad, M. A., Riaz, F., Aftab, A., Rizvi, S. K. J., Arshad, J., and Atlam, H. F. (2022). Deepsel: A novel feature selection for early identification of malware in mobile applications. *Future Generation Computer Systems*, 129:54–63. DOI: 10.1016/j.future.2021.10.029.

Bolón-Canedo, V. and Alonso-Betanzos, A. (2019). Ensembles for feature selection: A review and future trends. *Information fusion*, 52:1–12. DOI: 10.1016/j.inffus.2018.11.008.

Braganca, H., Kreutz, D., Rocha, V., Assolin, J., , and Feitosa, E. (2025). MH-1M: A 1.34 million-sample comprehensive multi-feature android malware dataset for machine learning, deep learning, large language models, and threat intelligence research. Available at:`https://arxiv.org/abs/2511.00342` arXiv eprint 2511.00342.

Cai, L., Li, Y., and Xiong, Z. (2021). Jowmdroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Computers & Security*, 100:102086. DOI: 10.1016/j.cose.2020.102086.

Cohen, I. et. al. (2009). Pearson Correlation Coefficient. *Noise reduction in speech processing*. DOI: 10.1007/978−1-4020−5614-7_2569.

Costa, E., Kreutz, D., Rocha, V., Leão, L., Sabóia, S., Neves, N., and Feitosa, E. (2022). FS3E: Uma Ferramenta Para Execução e Avaliação de Métodos de Seleção de Características Para Detecção de Malwares Android. In *XXII SBSeg*. SBC. DOI: 10.5753/sbseg_estendido.2022.227041.

Feizollah, A., Anuar, N. B., Salleh, R., and Wahab, A. W. A. (2015). A review on feature selection in mobile malware detection. *Digital investigation*, 13:22–37. DOI:

10.1016/j.diin.2015.02.001.

Galib, A. H. and Hossain, B. M. M. (2020). Significant API Calls in Android Malware Detection (Using Feature Selection Techniques and Correlation Based Feature Elimination). In *The 32nd SEKE*. DOI: 10.18293/SEKE2020-143.

Golrang, A., Yayilgan, S. Y., and Elezaj, O. (2021). The Multi-Objective Feature Selection in Android Malware Detection System. In *Intelligent Tech. and Applications*, page 311. DOI: 10.1007/978−3-030−71711-7_26.

Hammouri, A. I., Awadallah, M. A., Braik, M. S., Al-Betar, M. A., and Beseiso, M. (2024). Improved dwarf mongoose optimization algorithm for feature selection: Application in software fault prediction datasets. *Journal of Bionic Engineering*, 21(4):2000–2033. DOI: 10.1007/s42235-024-00524-4.

Kim, J., Ban, Y., Ko, E., Cho, H., and Yi, J. H. (2022). Mapas: a practical deep learning-based android malware detection system. *International Journal of Information Security*, 21(4):725–738. DOI: 10.1007/s10207-022-00579-6.

Mahindru, A. and Sangal, A. L. (2021). FSDroid: A Feature Selection Technique to Detect Malware from Android Using Machine Learning Techniques. *Multimedia Tools and Applications*, 80:13271–13323. DOI: 10.1007/s11042-020-10367-w.

Maniriho, P., Mahmood, A. N., and Chowdhury, M. J. M. (2023). API-MalDetect: Automated Malware Detection Framework for Windows Based on API Calls and Deep Learning Techniques. *JNCA*, 218:103704. DOI: 10.1016/j.jnca.2023.103704.

Neves, N., Rocha, V., Kreutz, D., Bragança, H., and Feitosa, E. (2023). Avaliação de Métodos de Seleção de Características de Amostras Android com a Ferramenta FS3E (v2). In *Anais da XX ERRC*. SBC. DOI: 10.5753/errc.2023.928.

Paim, K. O., Nogueira, A. G. D., Kreutz, D., Cordeiro, W., and Mansilha, R. B. (2025). MalDataGen: A modular framework for synthetic tabular data generation in malware detection. In *Anais Estendidos do XXV Simpósio Brasileiro de Cibersegurança (SBSeg 2025)*, SBSeg Estendido 2025, page 38–47. Sociedade Brasileira de Computação - SBC. DOI: 10.5753/sbseg_estendido.2025.12113.

Qazi, N. and Raza, K. (2012). Effect of Feature Selection, SMOTE and under Sampling on Class Imbalance Classification. In *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, pages 145–150. DOI: 10.1109/UKSim.2012.116.

Qiu, J., Han, Q.-L., Luo, W., Pan, L., Nepal, S., Zhang, J., and Xiang, Y. (2023). Cyber Code Intelligence for Android Malware Detection. *IEEE Transactions on Cybernetics*, 53(1):617–627. DOI: 10.1109/TCYB.2022.3164625.

Rocha, V., Kreutz, D., Canto, G., Bragança, H., and Feitosa, E. (2025). MH-FSF: A unified framework for overcoming benchmarking and reproducibility limitations in feature selection evaluation. Available at:`https://arxiv.org/abs/2507.10591`.

Şahin, D. Ö., Kural, O. E., Akleylek, S., and Kılıç, E. (2023). A novel android malware detection system: adaption of filter-based feature selection methods. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15. DOI: 10.1007/s12652-021-03376-6.

Salah, A., Shalabi, E., and Khedr, W. (2020). A lightweight android malware classifier using novel feature selection methods. *Symmetry*, 12(5). DOI: 10.3390/sym12050858.

Singh, A. K., Jaidhar, C., and Kumara, M. A. (2019). Experimental analysis of android malware detection based on combinations of permissions and api-calls. *Journal of Computer Virology and Hacking Techniques*, 15:209–218. DOI: 10.1007/s11416-019-00332-z.

Soares, T., Kreutz, D., Rocha, V., Costa, E., Leão, L., Pontes, J., Assolin, J., Rodrigues, G., and Feitosa, E. (2022). Uma Análise de Métodos de Seleção de Características Aplicados à Detecção de Malwares Android. In *Anais do XXII SBSeg*. SBC. DOI: 10.5753/sbseg.2022.225321.

Soi, D., Sanna, A., Maiorca, D., and Giacinto, G. (2024). Enhancing android malware detection explainability through function call graph apis. *Journal of Information Security and Applications*, 80:103691. DOI: 10.1016/j.jisa.2023.103691.

Song, F., Guo, Z., and Mei, D. (2010). Feature Selection Using Principal Component Analysis. In *2010 International Conference on System Science, Engineering Design and Manufacturing Informatization*, volume 1, pages 27–30. DOI: 10.1109/ICSEM.2010.14.

Tschiedel, L., Rocha, V., Kreutz, D., Bragança, H., Quincozes, S., Nogueira, A., and Assolin, J. (2024). SigAPI AutoCraft: Uma Ferramenta de Seleção de Características com Capacidade de Generalização. In *Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 169–176, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg_estendido.2024.243361.

Venkatesh, B. and Anuradha, J. (2019). A Review of Feature Selection and Its Methods. *Cybernetics and Information Technologies*, 19(1):3–26. DOI: doi:10.2478/cait-2019-0001.

Wang, L., Gao, Y., Gao, S., and Yong, X. (2021). A new feature selection method based on a self-variant genetic algorithm applied to android malware detection. *Symmetry*, 13(7). DOI: 10.3390/sym13071290.

Wei, G., Zhao, J., Feng, Y., He, A., and Yu, J. (2020). A Novel Hybrid Feature Selection Method Based on Dynamic Feature Importance. *Applied Soft Computing*, 93:106337. DOI: 10.1016/j.asoc.2020.106337.

Wu, Y., Li, M., Zeng, Q., Yang, T., Wang, J., Fang, Z., and Cheng, L. (2023). DroidRL: Feature Selection for Android Malware Detection with Reinforcement Learning. *Computers & Security*, 128:103126. DOI: 10.1016/j.cose.2023.103126.

Yang, H., Wang, Y., Zhang, L., Cheng, X., and Hu, Z. (2024). A novel android malware detection method with api semantics extraction. *Computers & Security*, 137:103651. DOI: 10.1016/j.cose.2023.103651.

Zou, D., Wu, Y., Yang, S., Chauhan, A., Yang, W., Zhong, J., Dou, S., and Jin, H. (2021). Intdroid: Android malware detection based on api intimacy analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(3):1–32. DOI: 10.1145/3442588.