Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5655

Synthetic Data: AI's New Weapon Against Android Malware

Angelo Gaspar Diniz Nogueira [**Universidade Federal do Pampa (UNIPAMPA)** | ***angelonogueira.aluno@unipampa.edu.br***]
Kayua Oleques Paim [**Universidade Federal do Rio Grande do Sul (UFRGS)** | ***kopaim@inf.ufrgs.br***]
Hendrio Bragança [**Universidade Federal do Amazonas (UFAM)** | ***hendrio.luis@icomp.ufam.edu.br***]
Rodrigo Brandão Mansilha [**Universidade Federal do Pampa (UNIPAMPA)**| ***rodrigomansilha@unipampa.edu.br***]
Diego Kreutz [**Universidade Federal do Pampa (UNIPAMPA)**| ***diegokreutz@unipampa.edu.br***]

Received: 07 March 2025 • **Accepted:** 15 July 2025 • **Published:** 28 April 2026

Abstract The ever-increasing number of Android devices and the accelerated evolution of malware, reaching over 35 million samples by 2024, highlight the critical importance of effective detection methods. Attackers are now using Artificial Intelligence to create sophisticated malware variations that can easily evade traditional detection techniques. Although machine learning has shown promise in malware classification, its success relies heavily on the availability of up-to-date, high-quality datasets. The scarcity and high cost of obtaining and labeling real malware samples presents significant challenges in developing robust detection models. In this paper, we propose MalSynGen, a Malware Synthetic Data Generation methodology that uses a conditional Generative Adversarial Network (cGAN) to generate synthetic tabular data. This data preserves the statistical properties of real-world data and improves the performance of Android malware classifiers. We evaluated the effectiveness of this approach using various datasets and metrics that assess the fidelity of the generated data, its utility in classification, and the computational efficiency of the process. Our experiments demonstrate that MalSynGen can generalize across different datasets, providing a viable solution to address the issues of obsolescence and low quality data in malware detection.

Keywords: Android Malware, Machine learning (ML), Artificial Inteligence (AI), Conditional Generative Adversarial Networks (cGANs), Synthetic data

1 Introduction

With approximately three billion Android devices in operation worldwide [Gartenberg, 2021], the mobile cybersecurity landscape faces formidable challenges. In 2024 alone, Kaspersky reported over 33.3 million cyberattacks targeting smartphone users globally, encompassing diverse forms of malware and unwanted software [Kaspersky Lab, 2024]. Adding to this problem, attackers are using Artificial Intelligence (AI) to rapidly generate new malware variants by exploiting patterns learned from existing malware [Renjith *et al*., 2022]. Furthermore, AI generators can potentially learn from benign applications, enabling malware to evade advanced antivirus solutions. Consequently, there is a pressing need for innovative techniques that can not only detect current threats, but also anticipate future ones.

Recent research demonstrates the effectiveness of supervised learning techniques in detecting and mitigating threats to Android malware, achieving high detection rates when combined with robust feature engineering [Kouliaridis and Kambourakis, 2021; Meijin *et al*., 2022]. However, the performance of these methods depends on: (1) the quantity of training samples, ensuring a sufficiently representative dataset; (2) the representativeness of attributes, encompassing the diversity and volume of features; and (3) the currency of the dataset, reflecting the latest trends and technological advances of malware [Botacin *et al*., 2021; Paullada, A. et. al., 2021].

The scarcity of high-quality training data is a primary factor that contributes to the failure of 80% of AI projects [AI & Data Today, 2023]. Specifically, sample size, feature representativeness, and data recency directly influence the performance of Android malware detection models [Miranda *et al*., 2022; Kouliaridis *et al*., 2020; Wang *et al*., 2019]. This challenge is exacerbated by the increasing use of generative AI technologies, such as Generative Adversarial Networks (GANs), by attackers to rapidly and continuously mutate malware [Xiao *et al*., 2018; Chauhan *et al*., 2021; Hu and Tan, 2022].

Obtaining sufficient, representative, and up-to-date real Android malware samples is a complex, costly, and time-consuming undertaking [Vilanova *et al*., 2022; Siqueira *et al*., 2021, 2022; Assolin *et al*., 2022; Rocha V. et. al, 2023]. Limitations such as the difficulty of acquiring malicious samples and the labor intensive labeling process, exemplified by the restricted labeling rate of services such as VirusTotal[1], render the continuous creation of updated datasets for model training and validation technically impractical.

Synthetic data generation has emerged as a valuable solution to address data scarcity across various domains. While numerous generic tools exist for generating synthetic tabular data [Xu *et al*., 2019; Rajabi and Garibay, 2022], specific applications within the Android malware domain remain unexplored. This domain presents two key challenges: hyperparameter selection and synthetic data evaluation. Firstly, the effectiveness of machine learning algorithms, particularly deep learning models like cGANs, is highly sensitive to hyperparameter tuning [Weerts *et al*., 2020; Kurach *et al*., 2019; Sabiri *et al*., 2022]. Secondly, there is a lack of stan-

[1]VirusTotal, a leading API service for metadata labeling, limits processing to 250 labels per day: `https://developers.virustotal.com/reference/overview`.

dardized metrics for evaluating synthetic data, both generally [Platzer and Reutterer, 2021] and within the specific context of Android malware. Ideally, synthetic data should maintain statistical fidelity to the original data while enhancing its utility for training supervised learning models. Furthermore, the generation process must be computationally efficient, offering a cost-effective alternative to acquiring real data.

This paper introduces MalSynGen (Malware Synthetic Data Generation), a comprehensive methodology and publicly available framework for generating and evaluating synthetic tabular data tailored for Android malware detection. We utilize a conditional Generative Adversarial Network (cGAN) model, inspired by [Mirza and Osindero, 2014], to generate synthetic data. Our evaluation framework assesses: (i) the *fidelity* of the generated synthetic data compared to real data, and (ii) the *utility* of the synthetic data in Android malware classification using a variety of classifiers. This work significantly expands upon our prior research by providing a more detailed methodology, a broader evaluation across diverse Android malware datasets, and a thorough analysis of computational resource consumption.

The key contributions of this expanded research are:

1. A cGAN model designed to generate synthetic tabular data that effectively supports Android malware classification.
2. A systematic methodology for training and evaluating the proposed cGAN model.
3. A comprehensive set of metrics for assessing both the *fidelity* and utility of the generated synthetic data in Android malware classification.
4. An extensive evaluation across multiple established Android malware datasets, demonstrating the generalization capabilities of our methodology.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 details the conceptual methodology and cGAN model. Section 4 outlines the evaluation process, including implementation and deployment details. Section 5 presents and discusses the evaluation results. Finally, Section 6 summarizes the main conclusions and outlines future research directions.

2 Related Work

In Table 1, we present the main related works in the context of tabular data augmentation. We list the techniques, metrics, domain, and datasets used in each work. As it can be seen, GANs are frequently used to generate synthetic tabular data. However, other techniques are also used in the data augmentation process, such as the use of large language models (LLMs) [Borisov *et al*., 2022] and diffusion models [Villaizán-Vallelado *et al*., 2025], due to their ability to effectively capture and generate complex patterns.

Most solutions for tabular data seek to capture the particularities of a specific context, such as healthcare [Choi, E. et. al., 2017], demographics [Rajabi and Garibay, 2022] and malware VBA [Mimura, 2020]. We can also observe that three solutions are specific to the context of malware Android, where we can see a predominance of cGANs [Amin *et al*., 2022; Li *et al*., 2024].

While prior solutions largely evaluate synthetic data utility through supervised learning model performance, they predominantly rely on standard binary classification metrics like precision, accuracy, recall, and F1-score. Furthermore, they typically utilize synthetic data in either the training or evaluation phase, but not both. This approach presents challenges, as high performance could stem from mere data replication rather than genuine novelty, while entirely novel data might yield poor classification if lacking inherent structure.

To overcome these limitations, we expand upon existing metrics by proposing two distinct categories: *utility* and *fidelity*. Utility metrics align with those commonly used in related works, whereas *fidelity* metrics, as emphasized in recent research [Canbek *et al*., 2021; Rainio *et al*., 2024], are specifically designed for assessing generative models and synthetic data quality. Additionally, we integrate synthetic data into complementary evaluation methodologies to enhance robustness. Unlike previous works that primarily use the Training on Synthetic, Testing on Real (TSTR) method, we adopt a dual approach: Training on Synthetic, Testing on Real (TSTR) and Training on Real, Testing on Synthetic (TRTS). This comprehensive strategy ensures a more thorough evaluation.

3 Malware Synthetic Data Generation (MalSynGen)

This section details the conceptual components of the MalSynGen framework, encompassing both the overall methodology and the underlying cGAN model. We begin by outlining the methodological process and subsequently describe the generative model.

3.1 Methodology

We illustrate the execution flow of the MalSynGen framework in Figure 1. The proposed flow consists of three main steps: selection and manipulation of the original dataset, training of classifiers, training of the conditional Generative Adversarial Network (cGAN), and evaluation of results.

In the first stage, **selection**, we choose a real dataset and perform balancing by the class (benign or malignant) with the fewest samples between the two categories. The balancing of the benign and malignant samples of the dataset is accomplished through the use of subsampling techniques. We then prepare the *dataset* for k-folds cross-validation. The balanced real *dataset* is divided into *k* equally sized subsets, and at each iteration, one part is chosen as the evaluation subset (*Dataset* r) and the remaining *(k-1)* subsets are used for training (*Dataset* R).

In the **training** step, the framework receives as input the cGAN training hyperparameters[2] and the hyperparameters of the classification algorithms. In addition, for each fold, the respective training (R) and evaluation (r) subsets are used. The training dataset (R) and part of the hyperparameters are

[2]In this work, we implement our own cGAN model.

Table 1. Related works on the context of synthetic tabular data. We list the Technique, metrics domain and datasets

Publication	Technique	Metrics	Evaluation Method	Datasets
Choi, E. et. al. [2017]	GAN	Precision, F1 score, recall, Bernoulli distribution	TSTR	2 medical
Xu and Veeramachaneni [2018]	GAN	F1 score, accuracy, mean squared error, and absolute squared error	TSTR	3 general purpose
Park, N. et. al [2018]	GAN	Cumulative distribution, F1 score, mean relative error, Euclidean distance, and AUC	TSTR	4 general purpose
Xu *et al*. [2019]	cGAN	Accuracy, F1 score, R^2, and log-likelihood	TSTR	7 general purpose
Mimura [2020]	GAN	Accuracy, F1 score, and Recall	TSTR	1 VBA malware
Rajabi and Garibay [2022]	GAN	Accuracy, F1 score, and measure of discrimination	TSTR	5 demographic data
Amin *et al*. [2022]	GAN	F1 score, recall, accuracy, precision, AUC, FPR, and coverage	TSTR	2 Android malware
Borisov *et al*. [2022]	LLM	Machine learning efficiency, accuracy, distance to the closest registries, measure of discrimination, and bivariate joint distribution chart	TSTR	6 general purpose
Casola *et al*. [2023]	cGAN	F1 score, recall, accuracy, precision, KL divergence, maximum mean discrepancy, and mean squared error	TSTR	1 Android malware
Li *et al*. [2024]	cGAN	F1 score, recall, accuracy, and precision	TSTR	2 Android malware
Zhao *et al*. [2024]	cGAN	Accuracy, F1 score, AUC, mean absolute error, R^2 score, and variance score	TSTR	6 general purpose
Villaizán-Vallelado *et al*. [2025]	Diffusion Model	Average Wasserstein distance, Mean Jensen-Shannon distance, Mean L2 distance of the correlation matrices, Machine learning efficiency, and F1 score	TSTR	10 general purpose
This work	cGAN	F1 score, recall, accuracy, precision, AUC, cosine similarity, squared error, *p*-value, mean discrepancy, and Euclidean distance	TSTR/ TRTS	7 Android malware

used to train the cGAN generative neural network. The trained cGAN instance is then used to generate a synthetic training dataset (S). In sequence, the same cGAN instance is used to generate a synthetic evaluation dataset (s) by performing a transformation over the real evaluation subset dataset (r). It is important to note that the evaluation data and synthetic evaluation data are not used at any point to train or fine-tune the cGAN generative model.

In addition to training and using cGANs to generate synthetic data, we also need to select, tune, and train classifiers (e.g., Support Vector Machine, Decision Tree) to subsequently assess the utility of the generated synthetic data. We implement the two methods proposed in [Esteban *et al*., 2017]: training on real data (R) and test the classifier with synthetic data (s) (TRTS) and; training on synthetic data (S) and test with real data (r) (TSTR).

In the **evaluation** stage, we execute the classifiers and compute the primary metrics, *utility* and *fidelity*.

For each trained classifier and corresponding evaluation subset, we extract the binary classification test results, specifically the counts of true positives, false positives, true negatives, and false negatives.

From these binary classification results, we construct confusion matrices and calculate standard binary classification metrics, including Accuracy, Precision, Recall, AUC, and F1-Score. Given that false negatives are more detrimental than false positives in malware detection, we prioritize Recall over Precision.

Upon completing the k-fold cross-validation, we proceed to the *utility* evaluation. This involves calculating the mean and standard deviation of each primary metric for each classifier, using both the Training on Synthetic, Testing on Real (TSTR) and Training on Real, Testing on Synthetic (TRTS) methods.

Subsequently, we conduct a *fidelity* assessment to determine if the synthetic evaluation dataset (s) accurately reflects the statistical properties of the real evaluation dataset (r). This assessment employs metrics such as cosine similarity, Euclidean distance, and squared error, which evaluate both

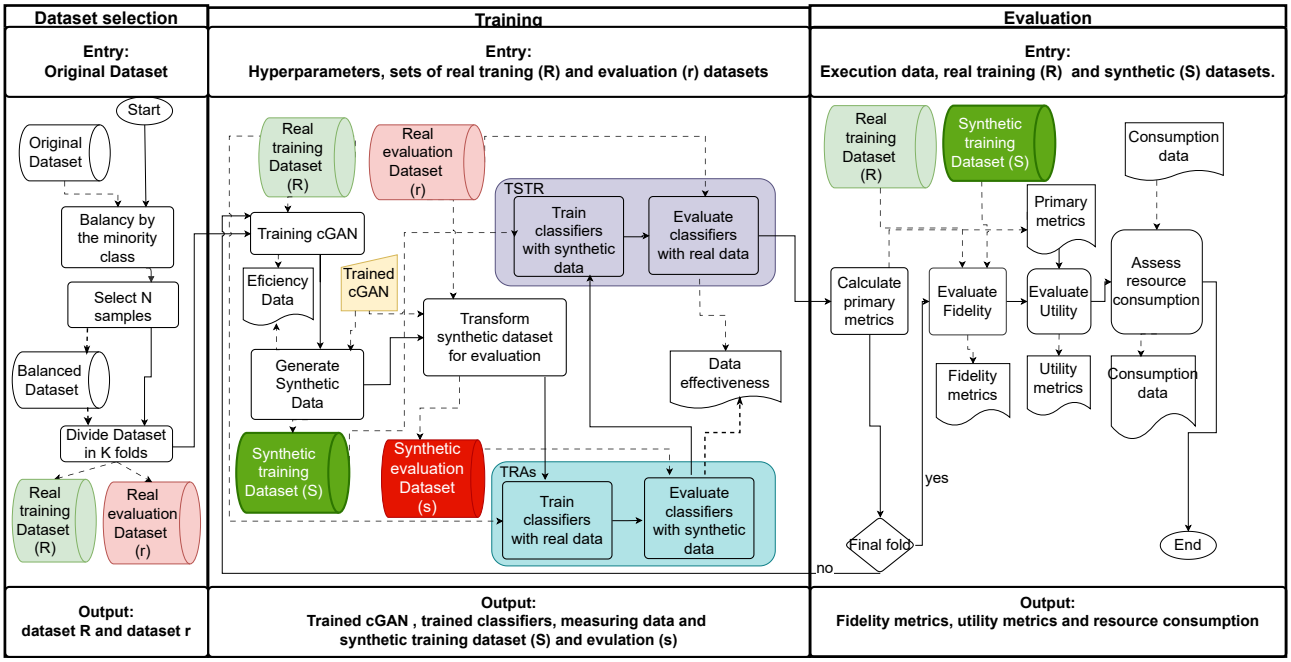


Figure 1. Selection, training, and evaluation of cGANs using MalSynGen. Solid arrows denote the sequential order of processes, while dotted arrows indicate the flow of artifact creation or utilization (datasets, models, raw data, and performance metrics). Certain artifacts are depicted multiple times to streamline the visualization of arrow intersections.

the individual feature distributions and the multivariate correlations between features.

Finally, we assess the faithfulness of the synthetic data's utility to the real data's utility using the Wilcoxon signed-rank test [Wilcoxon, 1945]. This test operates on paired samples, with two hypotheses: H_0, indicating no statistically significant difference between the samples, and H_1, indicating a statistically significant difference. The test calculates the absolute differences between paired samples, ranks these differences, and computes a concordance coefficient. The resulting p-value is compared to a threshold to determine whether to reject or accept H_0. To provide a comprehensive evaluation of classifier parameters, we calculate the average of the p-values obtained from the *utility* metrics for these hypothesis tests.

3.2 The Generative Model

The core intelligence of MalSynGen lies in its neural network model. Our approach is based on the Conditional Generative Adversarial Networks (cGANs) architecture [Mirza and Osindero, 2014], an extension of GANs that incorporates conditional variables to guide both the generation and discrimination of synthetic samples. The model consists of a generator, responsible for synthesizing samples, and a discriminator, tasked with distinguishing between real and generated instances. Both components are trained in an adversarial manner, optimizing their cost functions concurrently to reach an equilibrium where the generator produces samples indistinguishable from real ones.

In the proposed model, the sample labels (benign or malicious) are incorporated into the generator and discriminator through an *embedding* layer. This hidden layer projects that information into a latent space compatible with the model inputs. This mechanism explicitly conditions the generation of samples of the desired class, encouraging the modeling of the underlying probability distribution of each data category.

The generator and discriminator sub-models are composed of sequences of densely connected layers, with variable numbers of neurons, interspersed with dropout layers [Creswell *et al.*, 2018], used to mitigate *overfitting*. The use of dropout aims to reduce the discriminator's excessive dependence on specific patterns of the training set [Kim and Park, 2023], preventing the generator from directly replicating the learned examples instead of capturing the latent distribution of the data. Different values were adopted for the dropout rates of both the generator and the discriminator, depending on the dataset considered.

The nonlinear activation applied to the hidden layers is the Leaky Rectified Linear Unit (LeakyReLU) [Maas *et al.*, 2013]. This activation function was chosen for its superior training performance compared to other ReLU functions [Radford, 2015]. In the output layers, we use the sigmoid function, ensuring that both the generator outputs and the discriminator predictions remain within the range $[0, 1]$, consistent with the normalization of the input data. We decided not to include normalization layers, a common practice in deep networks, in the final implementation based on empirical experiments. This result is justified by the reduced depth of the architecture and the prior normalization of the input data to the range $[0, 1]$, which minimizes the occurrence of problems such as dissipation or explosion of the gradient.

For the training process, the loss function Binary Cross-Entropy was used, which is suitable for problems involving binary classification, such as the problem of distinguishing between real and generated samples. This function is widely used for training adversarial networks.

Figure 2 illustrates the general architecture of a cGAN,

where both the generator and the discriminator are fed labels. The generator uses these labels and latent noise to generate synthetic data, while the discriminator is trained with real data. The discriminator then tries to distinguish between real and synthetic data. Based on the results of the data classification, the loss value of the network is assigned, which is then used to adjust the weights and parameters of the network.

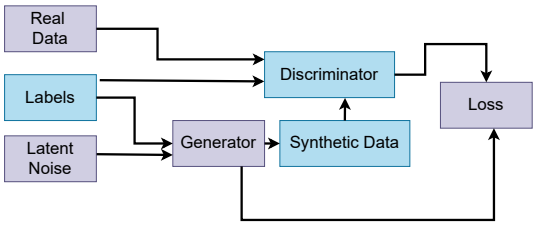


Figure 2. MalSynGen is based on the architecture of Conditional Generative Adversarial Networks (cGANs).

Figures 3 and 4 present the detailed architecture of the cGAN components shown in Figure 2: the discriminator and the generator. These components have similar structures, with the main difference being their input and output. Initially, an embedding layer is used to concatenate the latent noise and labels, transforming them into values in the latent space. These values are then processed through intermediate blocks, which consist of dense layers, activation functions, and dropout layers, all aimed at learning the data distribution and performing classification. Finally, dense layers, along with their activation functions, are responsible for normalizing the output and defining the format of the generated data.

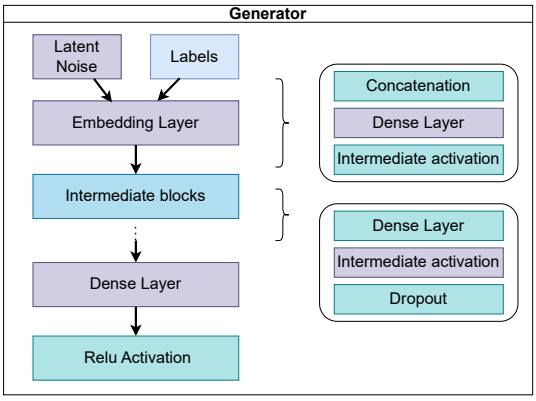


Figure 3. Generator of the cGAN architecture used in MalSynGen.

4 Evaluation

In this section, we present the results of utilizing the MalSynGen framework to generate synthetic datasets. We evaluate two scenarios:

1. whether the synthetic dataset preserves the essential characteristics of the original dataset (i.e., fidelity), while effectively extending and diversifying it.
2. whether the synthetic data can be used with classifiers (i.e., utility).

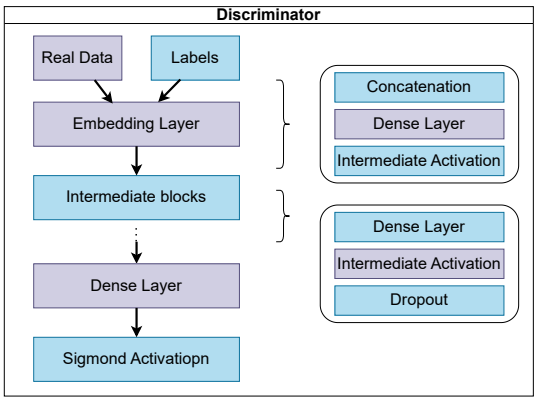


Figure 4. Discriminator of the cGAN architecture used in MalSynGen.

Table 2 summarizes the hyperparameters used in the instantiation and training of our cGAN model. The evaluation was performed using 10-fold cross-validation (k-fold = 10) with a batch size of 256 and the LeakyReLU activation function.

The range of epochs was chosen based on the range suggested by Antunes *et al*. [2023], and the density of neurons per layer was determined based on the ranges explored in Fristiana *et al*. [2024], as well as the dropout rates with the supporting middle and upper range from Seybold *et al*. [2018], which suggests rates from 0.2 to 0.4 for the dropout rate.

4.1 Android Malware Datasets

Table 3 presents the specifications of the datasets used in this study, obtained from the Malware-Hunter project public repository[3]. The table provides details on the number of features, as well as the number of malignant and benign samples, including the total number of samples. According to the repository, the datasets include up to 200 features selected using the chi-square method, and each dataset has a balanced total of up to 10,000 samples per class.

In order to further analyze the datasets used, we present in Figure 5 a comparative clustering analysis of malware samples, visualized using PCA-reduced 2D projections with K-means clustering. While clusters are distinguished by color within each plot for visual identification, it is important to note that identical colors across datasets do not indicate cluster correspondence.

The datasets exhibit distinct clustering patterns that indicate the potential existence of malware families, defined as groups of samples that share similar behavioral or structural characteristics. Higher-resolution versions of the clustering visualizations for each dataset are available in the GitHub repository[4] . Each dataset presents unique clustering dynamics, highlighting variations in malware distribution and feature representation across different data sources.

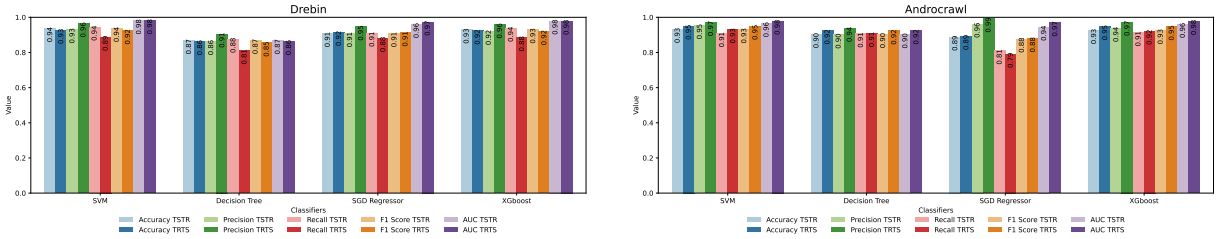
The Drebin dataset (Figure 5a) exhibits four clusters. Three of these clusters are tightly grouped in the leftmost corner of the figure. The third cluster (light green) represents the majority of samples (3411), while cluster 2 contains a single outlier sample.

[3] `https://github.com/MalwareDataLab/MalSynGen` on the folder Experiment_results, accessible in the jbcs tag: git checkout tags/jbcs

[4] `https://github.com/MalwareDataLab/MalSynGen` on the folder Datasets,accessible in the jbcs tag: git checkout tags/jbcs

Table 2. Hyperparameters used in the instantiation and training of our cGAN model.

Parameter	Androcrawl	Drebin	Adroit	Android P	Kronodroid E	Kronodroid R
Epochs	2,000	5,000	5,000	1,000	5,000	5,000
Neurons per layer (G)	2,048	2,048	64	1,024	512	512
Neurons per layer (D)	512	1,024	32	512	256	256
Dropout rate generator	0.2	0.2	0.05	0.2	0.025	0.025
Dropout rate discriminator	0.4	0.4	0.1	0.4	0.05	0.05
Initializer deviation	0.5	0.5	0.5	0.5	0.4	0.4



(a) Drebin dataset (b) Androcrawl dataset

Figure 5. Clustering of malware samples within datasets.

Table 3. Datasets considered for this study

Dataset	Features	Malware	Benign	Total
Adroit	118	3,418	3,418	6,836
Androcrawl	136	10,170	10,170	20,340
Android Permissions	148	9,077	9,077	18,154
Drebin215	200	5,555	5,555	11,110
KronoDroid Real Device	200	10,000	10,000	20,000
KronoDroid Emulator	200	10,000	10,000	20,000

In contrast, the Androcrawl dataset (Figure 5b) shows more dispersed clustering with five distinct groups. Clusters 0, 2, and 3 maintain somewhat comparable sizes (ranging from 1,500 to 2,300 samples). Cluster 1 (blue) appears to be composed of scattered samples (286), and cluster 4 (light green) emerges as the predominant group (4,176).

Additional plots of malware sample clusters for all other datasets can be found in Appendix B.

4.2 Fidelity Metrics

In this study, *fidelity* metrics are used to measure the similarity between real and synthetic data. We consider the following *fidelity* metrics: mean squared error, cosine similarity, and squared Euclidean distance, as shown in Table 4, where n represents the total number of samples; x and y are, respectively, the synthetic and real values of each sample.

These metrics enable us to verify whether the synthetic dataset reproduces the statistical characteristics of the original dataset. This includes not only the distribution of individual features but also the multivariate correlations between them, ensuring that the synthetic set mirrors the same population as the original. Additionally, we evaluate positive and negative samples separately to ensure that each class closely resembles the original.

Specifically, the Mean Squared Error is a metric used to assess the quality of a regression model. It is measured by calculating the mean of the squared differences between the predicted values (x_i) and the real values (y_i). A lower value indicates that the predicted values are closer to the real values. However, a value of zero is not ideal in our case, as it indicates that the model generated identical values to the real values.

Table 4. Fidelity metric formulas used to evaluate the similarity between real data (x) and synthetic data (y); n represents the total number of samples.

Metric	Formula
Cosine similarity	$\frac{x \cdot y}{\lvert x\rvert\lvert y\rvert}$
Squared Euclidean distance	$\delta \cdot \delta^T$, where $\delta = \mu_{\text{real}} - \mu_{\text{synth}}$, $\mu_{\text{real}} = \frac{1}{n}\sum_{i=1}^{n} x_i$, and $\mu_{\text{synth}} = \frac{1}{m}\sum_{j=1}^{m} y_j$
Mean squared error	$\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$

Cosine similarity is a metric used to measure the similarity between two vectors. In our case, it is used to compare the synthetic and real values (x and y, respectively). The similarity value ranges between 0 and 1, with values closer to 1 indicating higher similarity between the vectors. However, a value of 1 is not desirable, as it indicates that both sets are identical.

The squared Euclidean distance, similarly to the mean squared error, is a metric used to measure the difference between two sets. It returns the squared difference between the means of the real and synthetic samples. A smaller value indicates that the predicted values are closer to the real values. However, a value of zero indicates that both sets are identical; therefore, an ideal value approaches zero but is not equal to zero.

4.3 Utility Metrics

We also consider *utility* metrics to assess the performance of machine learning classifiers trained with synthetic data, considering both TRTS (train on real, test on synthetic) and TSTR (train on synthetic, test on real) evaluation methods. To evaluate the utility of the generated data, we considered four

classifiers: *SupportVectorMachine* (SVM), *Decision Tree* (DT), *XGBoost*, and *Stochastic Gradient Descent Regressor* (SGDR).

In each fold, we applied the classifiers to both the real and synthetic datasets generated by each cGAN configuration. Based on the results, we generated confusion matrices and ROC curves, and extracted the previously defined *utility* metrics: accuracy, precision, recall, F1-score, and AUC (Area Under the ROC Curve).

While accuracy, precision, recall, and F1-score are commonly used metrics, we also focus on the AUC, a statistical metric used to evaluate the performance of a machine learning model, particularly in binary classification tasks. The AUC graphically represents the relationship between the true positive rate (TPR) and the false positive rate (FPR) across different decision thresholds. A value of 1 indicates a model with perfect predictions, 0.5 indicates a model that makes random decisions, and a value of 0 suggests the model makes all predictions incorrectly. The formula for calculating the AUC is as follows:

$$AUC = TPR - (1 - \frac{TN}{FP + TN}) \cdot FPR \qquad (1)$$

To verify the statistical significance of the metrics, we utilize the Mann-Whitney U test. The Mann-Whitney U test operates on two independent samples with two hypotheses: H_0, which states that there is no statistically significant difference between the distributions of the samples, and H_1, which states that there is a statistically significant difference. This is determined by calculating the U statistic based on the sum of the ranks of ordered observations from the same sample. This calculation results in a p-value, which is compared with a threshold to determine whether to reject or accept H_0.

To complement and validate the utility metrics, we calculate the average of the p-values obtained from the *utility* metrics for our null hypothesis (H_0), which posits that there is no statistically significant difference between the performance of the two classifier sets, thereby demonstrating comparable utility. This averaging provides a comprehensive assessment of the classifier parameters. Furthermore, we adopt a threshold of 0.05 for the p-value, a standard value widely accepted in scientific literature, to conduct our experiments.

In addition to the utility metrics, we also collect data on the consumption of computational resources at the end of the evaluation. This includes the percentage of CPU and memory usage during each iteration of the training stage, providing insights into the computational demands of our approach.

4.4 Environment and Technologies

Four machines with Debian GNU/Linux 12 distributions (kernel 6.1.0-27-amd64), x86-64 architectures, Intel(R) Core(TM) i7-9700 CPUs, and 16 GB of RAM were used to perform the experiments in this study.

To implement and execute the core architecture of our cGAN, we used Python (version 3.8) and the following main libraries:

- *NumPy* 1.21.5: Used for various vector and matrix operations involving the labels and samples generated by the network.
- *Keras* 2.9.0: Used to define the conditional models of the generator and discriminator.
- *TensorFlow* 2.9.1: Applied in conjunction with Keras to define the network's loss functions.
- *Pandas* 1.4.4: Responsible for processing and handling input and output data in CSV format.
- *scikit-learn* 1.1.1: Used to stratify folds during executions and to calculate evaluation metrics.
- *MLflow*[5]: To track computational resource consumption metrics.

Furthermore, more details on the cGAN implementation, resources used, and results of all executions can be found in the GitHub repository[6].

5 Results and discussion

5.1 Utility metrics

Figure 6 presents the average 10-fold cross-validation results for the SVM classifier across all six datasets, visualized through a performance heatmap. In this visualization, darker shading indicates better performance (values closer to 1.0), while lighter shading signifies poorer performance (values closer to 0.0). For each of the six datasets, the heatmap organizes results by five key evaluation metrics (accuracy, precision, F1-score, AUC, and recall), with each metric showing paired rows for both TSTR (Train on Synthetic, Test on Real) and TRTS (Train on Real, Test on Synthetic) scenarios.

With the sole exception of the Android Permission dataset, the remaining datasets consistently demonstrated high utility metrics, with values ranging from 0.79 to 0.98 across all evaluated metrics in both TSTR and TRTS scenarios. Particularly noteworthy are the recall values, which ranged from 0.84 to 0.97. This is a crucial metric for malware classification, as false negatives carry significant security implications. These robust results strongly suggest that the generated synthetic data is highly useful for malware classification across multiple diverse datasets. Additionally, the diminished performance observed for the Android Permission dataset (with metrics ranging between 0.55 and 0.61) likely reflects inherent constraints in its source data. This is further supported by prior experiments using real-world data for both evaluation and training, which yielded a similarly low performance for this specific dataset.

A complete list of all classifiers used in this study, along with their corresponding utility metrics, can be found in Appendix A.

In Table 5, we present the p-values obtained from the Wilcoxon test for accuracy, precision, F1-score, recall, and AUC for each classifier. Each p-value indicates the statistical significance of the differences between the classifier sets.

The results indicate that in most cases, there is no statistically significant difference between the classifier sets, regardless of the classifier used. The only exception was observed with the XGBoost classifier for the Kronodroid Emulator dataset, which had a p-value of 0.0348. Although this value

[5] `https://mlflow.org/`

[6] `https://github.com/MalwareDataLab/MalSynGen`

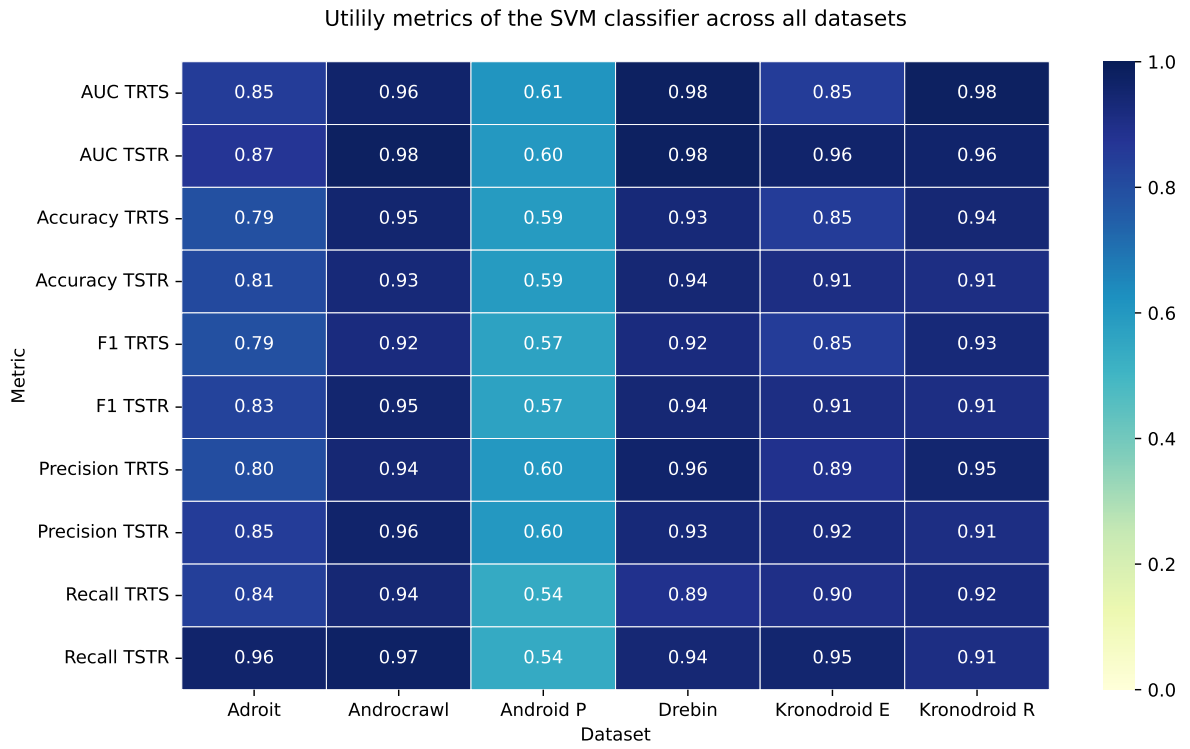


Figure 6. Performance heatmap of SVM classifier utility metrics across all datasets for TSTR and TRTS scenarios.

Table 5. *p*-values of the classifier metrics

Dataset	SVM	Decision Tree	SGD Regressor	XGBoost
Adroit	0.0668	0.3906	0.2254	0.1563
Androcrawl	0.0570	0.1566	0.4945	0.1281
Android Permissions	0.4287	0.3020	0.5871	0.7745
Drebin	0.1801	0.3223	0.2164	0.3219
Kronodroid Emulator	0.1134	0.0684	0.1062	0.0348
Kronodroid Real	0.1109	0.3641	0.0602	0.1867

is below the defined threshold, it is close to the limit. Even so, the model's performance metrics remain high, with accuracy ranging from 0.86 to 0.88, recall between 0.85 and 0.88, and a high AUC, ranging from 0.93 to 0.95. Furthermore, an anomalous behavior is observed with the Android Permission dataset, where the *p*-values are high, between 0.3020 and 0.7745, indicating high similarity between the performance of the two classifier sets, despite the low utility metrics.

In Figures 7 and 8 we present a comparative analysis of MalSynGen against SDV's CTGAN [Xu *et al.*, 2019]. This comparison is conducted through 10-fold cross-validation using SVM classifiers on the Drebin and Androcrawl datasets.

For both MalSynGen and CTGAN, identical hyperparameters are maintained for the AndroCrawl and Drebin datasets, as specified in Table 2. The only exceptions are a reduced layer configuration (256G/64D layers) and the application of 100 training epochs across both models. The evaluation comprehensively implements both TSTR (Train on Synthetic, Test on Real) and TRTS (Train on Real, Test on Synthetic) methodologies for each model.

The results indicate that MalSynGen generates synthetic data with superior utility metrics across both evaluation paradigms (TSTR and TRTS), demonstrating an average improvement of 0.18375 on the Androcrawl dataset and 0.1025 on the Drebin dataset, compared to CTGAN. Although a slight exception is observed in the TSTR recall value for the Drebin dataset (0.55 for MalSynGen versus 0.61 for CTGAN), the overall findings suggest that MalSynGen consistently outperforms CTGAN in the context of Android malware classification for this specific scenario.

5.2 Fidelity metrics

In Table 6 and Table 7, we present the results obtained for the *fidelity* metrics (cosine similarity, squared Euclidean distance, and mean squared error) for the datasets considered in this study.

Table 6. Fidelity metric values for positive samples (malware).

Positive Samples			
Dataset	**Cosine**	**Euclidean Distance**	**Squared Error**
Adroit	0.697	0.097	0.052
Androcrawl	0.602	0.151	0.116
Android P	0.321	0.017	0.035
Drebin	0.370	0.398	0.138
Kronodroid E	0.632	0.352	0.141
Kronodroid R	0.626	0.277	0.167

Table 7. Fidelity metric values for negative samples (benign).

Negative Samples			
Dataset	**Cosine**	**Euclidean Distance**	**Squared Error**
Adroit	0.589	0.127	0.045
Androcrawl	0.523	0.050	0.088
Android P	0.359	0.016	0.037
Drebin	0.540	0.262	0.169
Kronodroid E	0.676	0.110	0.098
Kronodroid R	0.641	0.102	0.104

The cosine similarity results indicate that the synthetic data

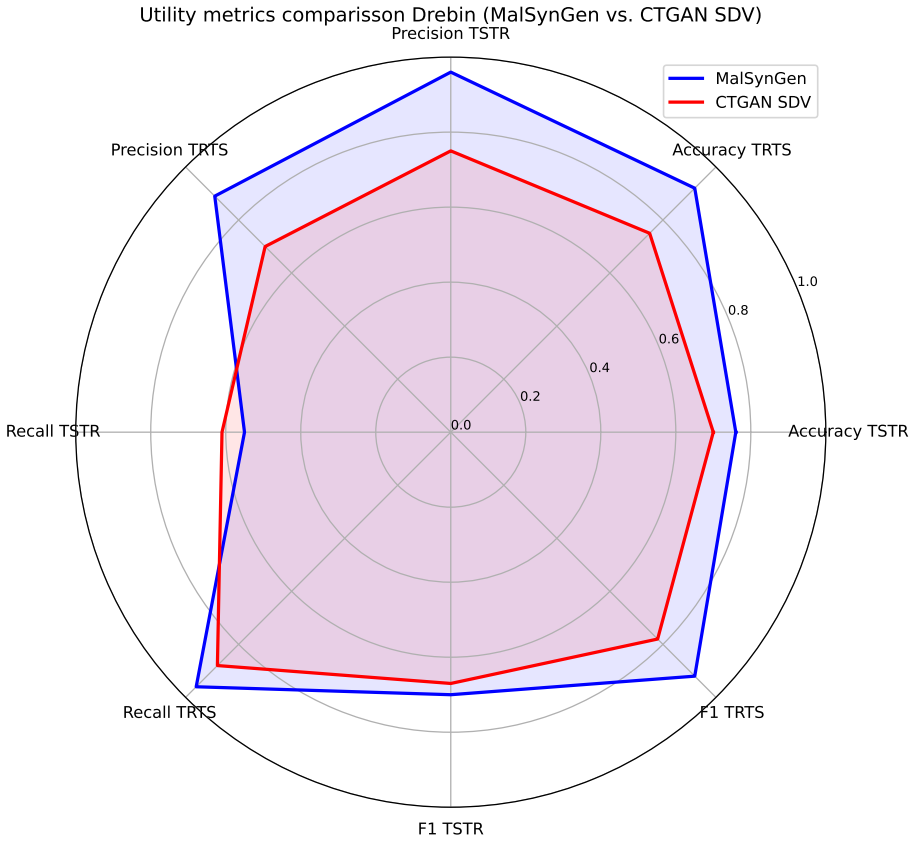


Figure 7. Comparative utility metrics for MalSynGen and CTGAN SVM performance on Drebin dataset

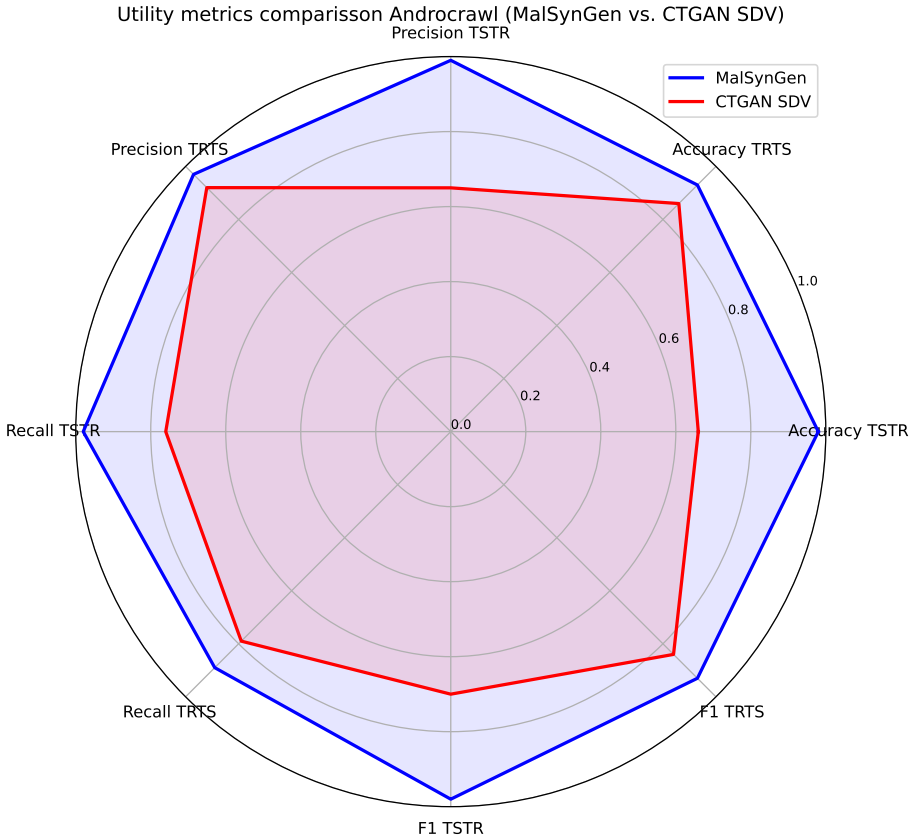


Figure 8. Comparative utility metrics for MalSynGen and CTGAN SVM performance on Androcrawl dataset

is similar, but not identical, to the original data, as the values approach 1, except for the Android P and Drebin datasets. However, as previously emphasized, the metrics obtained from classifiers trained with Android P are inherently low, leading to the synthetic data exhibiting similar behavior. Regarding Drebin, despite having a cosine similarity of 0.370 for positive samples, which is low compared to other results, its other metrics and the value for negative samples indicate that the data still capture the data patterns.

The mean squared error values indicate high similarity between the synthetic and original data for all datasets, as they are close to zero, though not exactly zero. Furthermore, the Euclidean distance values corroborate the squared error results. The only exceptions are the Drebin, Kronodroid E, and Kronodroid R datasets, which show slightly higher values. Nevertheless, these values remain within an acceptable threshold, and the other metrics demonstrate positive results.

5.3 Computational Resources

In Table 8, we present the results obtained by tracking the consumption of computational resources (CPU and memory), along with the total time used to run all experiments (i.e., 10 runs for TRTS and TSTR each). The experiments were conducted in the environment described in Section 4.4.

Table 8. Average consumption of computational resources.

Dataset	CPU (%)	Memory (%)	Execution time
Adroit	44.09%	15.51%	0.731h
Androcrawl	75.49%	17.18%	2.8h
Android P	66.04%	23.87%	2.2h
Drebin	78.40%	18.10%	4.1h
Kronodroid E	50.97%	18.33%	3.4h
Kronodroid R	51.97%	18.10%	3.3h

The results suggest that configurations with higher layer density (Androcrawl and Drebin) exhibit the highest CPU usage, while those with a greater number of epochs (Adroit, Kronodroid E, and Kronodroid R) show the highest memory consumption and execution time. This indicates a correlation between hyperparameters and computational resource consumption. Additionally, the results demonstrate that MalSynGen is CPU-intensive but does not consume excessive memory during execution. The variations in computational resource consumption are primarily influenced by dataset size and hyperparameter values.

6 Final Remarks

MalSynGen is a publicly available framework designed for training and evaluating cGAN generative networks, specifically for generating synthetic tabular datasets in the context of Android malware detection. It employs a methodology that rigorously assesses both the utility and fidelity of the generated data.

Key findings from experiments conducted on six Android malware datasets demonstrate that the synthetic datasets produced by MalSynGen are:

- **Useful**: Exhibiting high utility metrics, with values ranging from 0.75 to 0.98 in five datasets, and high AUC scores (0.80 to 0.99), confirming consistent and positive performance across various classifiers.
- **Faithful**: Maintaining fidelity to the original data, as evidenced by cosine similarity approaching 1 in four datasets, mean squared error values close to zero, and Euclidean distance values within acceptable thresholds.

Analysis of computational resource consumption indicates that MalSynGen is CPU-intensive but does not require excessive memory. Furthermore:

- Configurations with higher layer densities increase CPU usage.
- Configurations with a higher number of epochs demand more memory and execution time.

Future research directions may include:

- Expanding the evaluation protocol to incorporate additional metrics of utility, statistical fidelity, privacy risk, and computational performance, as well as assessing the impact of different machine learning classifiers.
- Conducting a systematic comparative analysis with general-purpose synthetic data generation tools under controlled experimental conditions and using identical datasets.
- Extending the methodology to support alternative neural architectures for synthetic tabular data generation, including recent generative models and hybrid approaches.
- Evaluating the framework across a broader range of cybersecurity scenarios and large-scale datasets, including recent benchmarks such as MH-1M Bragança *et al*. [2025] and other emerging datasets.

Braganca2025

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: Kayua Oleques Paim, Hendrio Bragança, and Angelo Gaspar were responsible for software development. Validation: All authors participated in the validation process. Formal Analysis: All authors contributed to the formal analysis of the results. Investigation: All authors were involved in the investigation. Resources: Diego Kreutz and Rodrigo Mansilha provided resources for this research. Data Curation: All authors were involved in data curation. Writing - Original Draft Preparation: All authors contributed to the original draft preparation. Writing - Review and Editing: All authors reviewed and edited the manuscript. Supervision: Diego Kreutz and Rodrigo Mansilha provided supervision for this research. Project Administration: Diego Kreutz and Rodrigo Mansilha administered the project. Funding Acquisition: Diego Kreutz and Rodrigo Mansilha secured funding for this research.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The authors extend their gratitude to Anna Luiza Gomes da Silva and Lucas Ferreira Areias de Oliveira for their invaluable contributions to hyperparameter configuration and resource collection. The authors also express their sincere appreciation to the anonymous SBSeg 2024 reviewers for their insightful suggestions and feedback.

Funding

This research was partially supported by the National Education and Research Network (RNP) through the "Programa Hackers do Bem" initiative and the GT Malware DataLab. t also received support from the 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

Code: `https://github.com/MalwareDataLab/MalSynGen`
Datasets: The datasets can be found in the `datasets` directory located inside the GitHub repository.

References

AI & Data Today (2023). Top 10 reasons why ai projects fail. `https://t.ly/wMBj5`.

Amin, M., Shah, B., Sharif, A., Ali, T., Kim, K.-I., and Anwar, S. (2022). Android malware detection through generative adversarial networks. *Transactions on Emerging Telecommunications Technologies*, 33(2). DOI: 10.1002/ett.3675.

Antunes, A., Ferreira, B., Marques, N., and Carriço, N. (2023). Hyperparameter optimization of a convolutional neural network model for pipe burst location in water distribution networks. *Journal of Imaging*, 9(3):68. DOI: 10.3390/jimaging9030068.

Assolin, J., Kreutz, D., Siqueira, G., Rocha, V., Miers, C., Mansilha, R., and Feitosa, E. (2022). DroidAutoML: uma ferramenta de automl para o domínio de detecção de malwares android. In *Anais Estendidos do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 135–142, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg_estendido.2022.227037.

Borisov, V., Seßler, K., Leemann, T., Pawelczyk, M., and Kasneci, G. (2022). Language models are realistic tabular data generators. *arXiv preprint arXiv:2210.06280*. DOI: 10.48550/arXiv.2210.06280.

Botacin, M., Ceschin, F., Sun, R., Oliveira, D., and Grégio, A. (2021). Challenges and pitfalls in malware research. *Computers & Security*, 106:102287. DOI: 10.1016/j.cose.2021.102287.

Bragança, H., Kreutz, D., Rocha, V., Assolin, J., and Feitosa, E. (2025). MH-1M: A 1.34 million-sample multi-feature android malware dataset with rich metadata. *Scientific Data*, 13(1):153. DOI: 10.1038/s41597-025-06469-5.

Canbek, G., Taskaya Temizel, T., and Sagiroglu, S. (2021). BenchMetrics: A systematic benchmarking method for binary classification performance metrics. *Neural Computing and Applications*, 33(21). DOI: 10.1007/s00521-021-06103-6.

Casola, K., Paim, K., Mansilha, R., and Kreutz, D. (2023). Droidaugmentor: uma ferramenta de treinamento e avaliação de cgans para geração de dados sintéticos. In *Anais Estendidos do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 57–64, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg_estendido.2023.235793.

Chauhan, R., Sabeel, U., Izaddoost, A., and Shah Heydari, S. (2021). Polymorphic adversarial cyberattacks using wgan. *Journal of Cybersecurity and Privacy*, 1(4):767–792. DOI: 10.3390/jcp1040037.

Choi, E. et. al. (2017). Generating multi-label discrete patient records using generative adversarial networks. In *Machine learning for healthcare conference*, pages 286–305. DOI: 10.48550/arXiv.1703.06490.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., and Bharath, A. A. (2018). Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65. DOI: 10.1109/MSP.2017.2765202.

Esteban, C., Hyland, S. L., and Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional GANs. *arXiv preprint arXiv:1706.02633*. DOI: 10.48550/arXiv.1706.02633.

Fristiana, A. H., Alfarozi, S. A. I., Permanasari, A. E., Pratama, M., and Wibirama, S. (2024). A survey on hyperparameters optimization of deep learning for time series classification. *IEEE Access*, 12:191162–191198. DOI: 10.1109/ACCESS.2024.3516198.

Gartenberg, C. (2021). Google says there are now over 3 billion active android devices. Available at:`https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021`. Accessed: 2025-01-06.

Hu, W. and Tan, Y. (2022). Generating adversarial malware examples for black-box attacks based on GAN. In *International Conference on Data Mining and Big Data*, pages 409–423. Springer. DOI: 10.1007/978-981-19-8991-9_29.

Kaspersky Lab (2024). Banking data theft: Attacks on smartphones triple in 2024, kaspersky reports. Available at:`https://www.kaspersky.com/about/press-releases/banking-data-theft-attacks-on-smartphones-triple-in-2024-kaspersky-reports`. Accessed: 2025-05-18.

Kim, J. and Park, H. (2023). Limited discriminator gan using explainable ai model for overfitting problem. *ICT Express*, 9(2):241–246. DOI: 10.1016/j.icte.2021.12.014.

Kouliaridis, V. and Kambourakis, G. (2021). A comprehensive survey on machine learning techniques for Android malware detection. *Information*, 12(5):185. DOI: 10.3390/info12050185.

Kouliaridis, V., Kambourakis, G., and Peng, T. (2020). Feature importance in android malware detection. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1449–1454. IEEE. DOI: 10.1109/TrustCom50675.2020.00195.

Kurach, K., Lučić, M., Zhai, X., Michalski, M., and Gelly, S. (2019). A large-scale study on regularization and normalization in gans. In *International conference on machine learning*, pages 3581–3590. PMLR. DOI: 10.48550/arXiv.1807.04720.

Li, J., He, J., Li, W., Fang, W., Yang, G., and Li, T. (2024). SynDroid: An adaptive enhanced Android malware classification method based on CTGAN-SVM. *Computers & Security*, 137:103604. DOI: 10.1016/j.cose.2023.103604.

Maas, A. L., Hannun, A. Y., Ng, A. Y., *et al*. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA. Available at:`https://www.semanticscholar.org/paper/Rectifier-Nonlinearities-Improve-Neural-Network-Maas/367f2c63a6f6a10b3b64b8729d601e69337ee3cc`.

Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., Yinwei, W., and Jiaxuan, G. (2022). A systematic overview of android malware detection. *Applied Artificial Intelligence*, 36(1):2007327. DOI: 10.1080/08839514.2021.2007327.

Mimura, M. (2020). Using fake text vectors to improve the sensitivity of minority class for macro malware detection. *JISA*, 54:102600. DOI: 10.1016/j.jisa.2020.102600.

Miranda, T. C., Gimenez, P.-F., Lalande, J.-F., Tong, V. V. T., and Wilke, P. (2022). Debiasing android malware datasets: How can i trust your results if your dataset is biased? *IEEE Transactions on Information Forensics and Security*, 17:2182–2197. DOI: 10.1109/TIFS.2022.3180184.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*. DOI: 10.48550/arXiv.1411.1784.

Park, N. et. al (2018). Data synthesis based on Generative Adversarial Networks. *arXiv preprint arXiv:1806.03384*. DOI: 10.48550/arXiv.1806.03384.

Paullada, A. et. al. (2021). Data and its (dis) contents: A survey of dataset development and use in machine learning research. *Patterns*, 2(11). DOI: 10.1016/j.patter.2021.100336.

Platzer, M. and Reutterer, T. (2021). Holdout-Based Empirical Assessment of Mixed-Type Synthetic Data. *Frontier in Big Data*. DOI: 10.3389/fdata.2021.679939.

Radford, A. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*. DOI: 10.48550/arXiv.1511.06434.

Rainio, O., Teuho, J., and Klén, R. (2024). Evaluation metrics and statistical tests for machine learning. *Scientific Reports*, 14(1):6086. DOI: 10.1038/s41598-024-56706-x.

Rajabi, A. and Garibay, O. O. (2022). TabfairGAN: : Fair Tabular Data Generation with Generative Adversarial Networks. *ML and Knowledge Extraction*, 4(2):488. DOI: 10.3390/make4020022.

Renjith, G., Laudanna, S., Aji, S., Visaggio, C. A., and Vinod, P. (2022). GANG-MAM: GAN based engine for modifying Android malware. *SoftwareX*, 18:100977. DOI: 10.1016/j.softx.2022.100977.

Rocha V. et. al (2023). AMGenerator e AMExplorer: Geração de metadados e construção de datasets android. In *Anais Estendidos do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 41–48. SBC. DOI: 10.5753/sbseg_estendido.2023.235801.

Sabiri, B., El Asri, B., and Rhanoui, M. (2022). Effect of convulsion layers and hyper-parameters on the behavior of adversarial neural networks. In *International Conference on Enterprise Information Systems*, pages 222–245. Springer. DOI: 10.1007/978-3-031-39386-0_11.

Seybold, C. *et al*. (2018). Dropout-gan: Learning from a dynamic ensemble of discriminators. *arXiv preprint arXiv:1807.11346*. DOI: 10.48550/arXiv.1807.11346.

Siqueira, G., Kreutz, D., Assolin, J., Costa, E., Miers, C., Mansilha, R., Pontes, J., and Feitosa, E. (2022). Avaliaçao de ferramentas de automl em datasets de detecçao de malwares android. In *Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 302–315. SBC. DOI: 10.5753/sbseg.2022.225317.

Siqueira, G., Rodrigues, G., Feitosa, E., and Kreutz, D. (2021). QuickAutoML: Uma ferramenta para treinamento automatizado de modelos de aprendizado de máquina. In *Anais da XIX Escola Regional de Redes de Computadores*, pages 85–90, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/errc.2021.18547.

Vilanova, L., Kreutz, D., Assolin, J., Quincozes, V., Miers, C., Mansilha, R., and Feitosa, E. (2022). ADBuilder: uma ferramenta de construção de datasets para detecção de malwares android. In *Anais Estendidos do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 143–150, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg_estendido.2022.227038.

Villaizán-Vallelado, M., Salvatori, M., Segura, C., and Arapakis, I. (2025). Diffusion models for tabular data imputation and synthetic data generation. *ACM Transactions on Knowledge Discovery from Data*, 19(6). DOI: 10.1145/3742435.

Wang, H., Si, J., Li, H., and Guo, Y. (2019). RmvDroid: Towards a reliable android malware dataset with app metadata. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 404–408. DOI: 10.1109/MSR.2019.00067.

Weerts, H. J., Mueller, A. C., and Vanschoren, J. (2020). Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*. DOI: 10.48550/arXiv.2007.07588.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. biom. bull., 1, 80. Available at:`https://www.jstor.org/stable/3001968`.

Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M., and Song, D. (2018). Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610*. DOI: 10.48550/arXiv.1801.02610.

Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. (2019). Modeling Tabular Data Using Conditional

GAN. *Advances in neural information processing systems*, 32. DOI: 10.5555/3454287.3454946.

Xu, L. and Veeramachaneni, K. (2018). Synthesizing Tabular Data Using Generative Adversarial Networks. *arXiv preprint arXiv:1811.11264*. DOI: 10.48550/arXiv.1811.11264.

Zhao, Z., Kunar, A., Birke, R., Van der Scheer, H., and Chen, L. Y. (2024). Ctab-gan+: Enhancing tabular data synthesis. *Frontiers in big Data*, 6:1296508. DOI: 10.48550/arXiv.2204.00401.

A Classifier Utility Metrics Across All Datasets

In this appendix, we present the set of utility metrics for all classifiers across each evaluated dataset. We detail the mean values of the utility metrics (AUC, precision, recall, accuracy, and F1-score) obtained for MalSynGen through a 10-fold evaluation process, employing both the TSTR (Train on Synthetic, Test on Real) and TRTS (Train on Real, Test on Synthetic) methods.

The detailed results are visualized in:

- Figure 9 (Androcrawl) [7]
- Figure 10 (Drebin)
- Figure 11 (Adroit)
- Figure 12 (Kronodroid Emulator)
- Figure 13 (Android Permission)
- Figure 14 (Kronodroid Real Device)

For each classifier, these figures display five pairs of bars, with each pair representing one of the five metrics (accuracy, precision, F1-score, AUC, and recall). Within each pair, one bar illustrates the value for the TSTR scenario, and the other for the TRTS scenario.

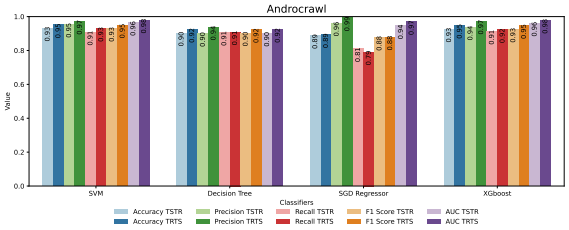


Figure 9. Utility metrics AndroCrawl dataset

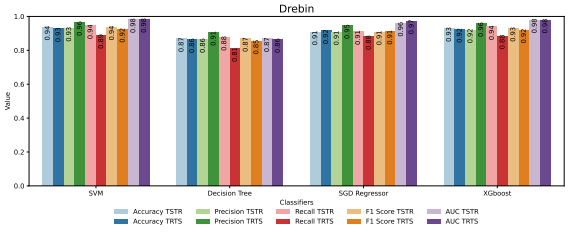


Figure 10. Utility metrics Drebin dataset

[7] Higher resolution of the images can be found at `https://github.com/MalwareDataLab/Datasets-JBCS/tree/b53e8a184003d775e02c2dc3e9b953ffcb8d6a9c/Figures`

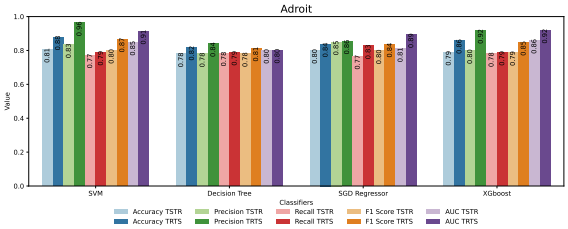


Figure 11. Utility metrics Adroit dataset

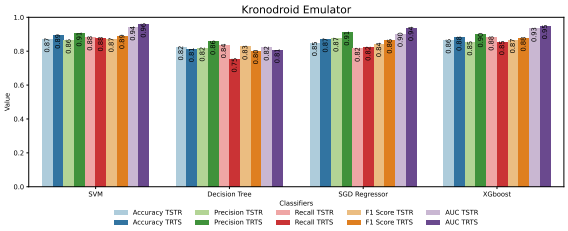


Figure 12. Utility metrics Kronodroid E dataset

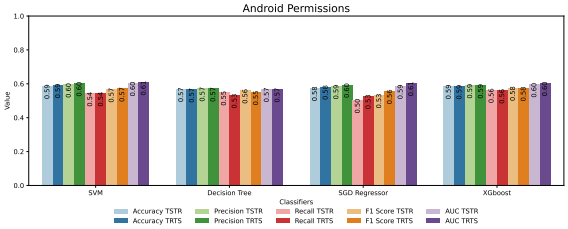


Figure 13. Utility metrics Android P dataset

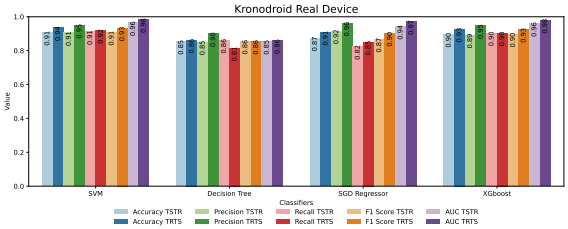


Figure 14. Utility metrics Kronodroid R dataset

Notably, all classifiers consistently exhibited high utility (0.75–0.9), particularly in AUC, precision, and F1-score, across most datasets. As previously discussed, the Android Permissions dataset represented an exception. These consistent results further confirm the practical utility of our synthetic data for malware classification. We also observed a stable performance hierarchy among the classifiers: Support Vector Machines (SVM) consistently delivered the highest utility, followed by XGboost, SGD Regressor, and Decision Tree models.

B Malware Sample Clustering Across All Datasets

In this appendix, we present the clustering results for the other datasets considered in this study, visualized using PCA-reduced 2D projections with K-means clustering. While we distinguish clusters by color within each plot for visual identification, we note that identical colors across datasets do not indicate cluster correspondence.

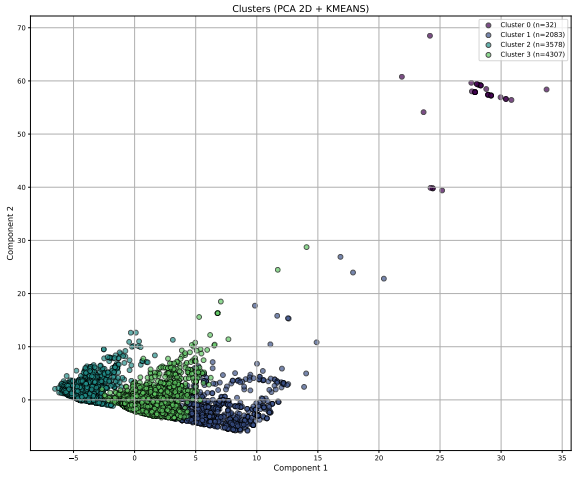


Figure 15. Clustering of malware samples in Kronodroid E dataset.

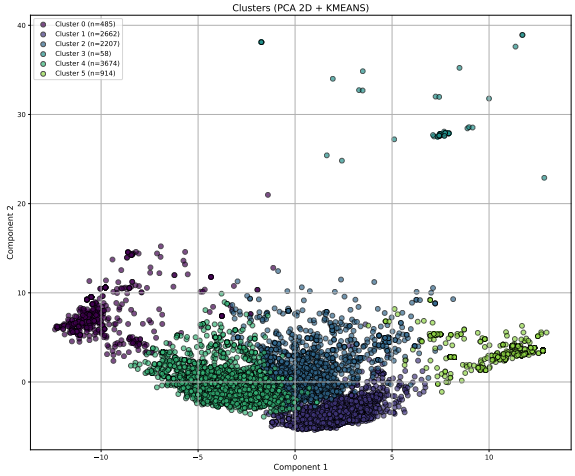


Figure 16. Clustering of malware samples in Kronodroid R dataset.

Analyzing the figures, we observed two distinct clustering patterns, similar to those presented in Figure 5. In the first

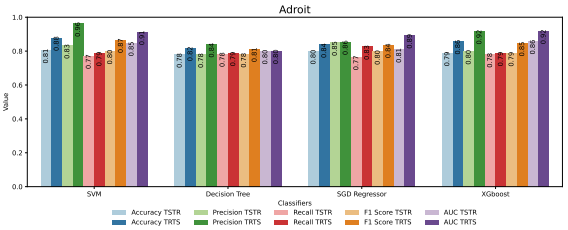


Figure 17. Cluster analysis of malware samples in Adroit dataset.

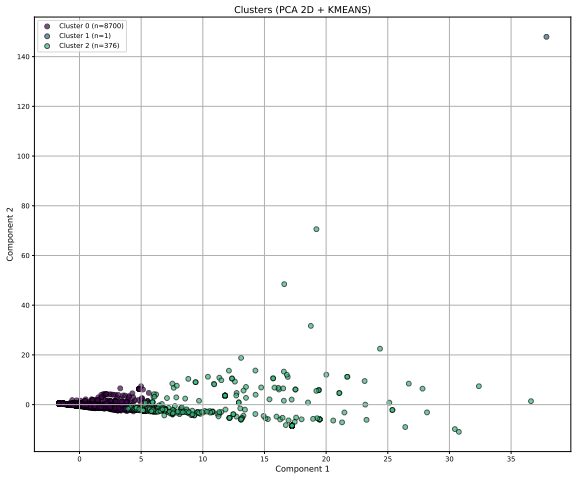


Figure 18. Cluster analysis of malware samples in Android Permission dataset.

pattern, Figures 15 and 16 exhibit distinct groupings that resemble Figure 5(a) (Androcrawl). These are characterized by one dominant cluster containing the majority of samples (e.g., Cluster 3, 4,307 in Kronodroid E; Cluster 4, 3,674 in Kronodroid R), alongside several secondary clusters of comparable size. These figures also include small outlier clusters (e.g., Cluster 0, 32 in Kronodroid E; Cluster 3, 58 in Kronodroid R).

The second pattern, observed in the Adroit and Android Permission datasets, is more similar to Figure 5(b) (Drebin). In these cases, samples and clusters are grouped more tightly together, with a very small number of anomalous samples forming their own distinct clusters (e.g., Cluster 1, 1 sample in Android Permission; Cluster 1, 3 samples in Adroit dataset).