Research Square

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

TAH-GAN: Time-Aware Hybrid Generative Adversarial Network for Robust Android Malware Evasion

Daniel Jeremiah

tp091738@mail.apu.edu.my

Asia Pacific University of Technology & Innovation (APU)

Jalil Md Desa

Asia Pacific University of Technology & Innovation (APU)

Husnain Rafiq

Northumbria University

Research Article

Keywords: Android malware, temporal concept drift, feature-space attack, generative adversarial network, adversarial machine learning, security evaluation

Posted Date: September 7th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10821022/v1

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

Additional Declarations: No competing interests reported.

TAH-GAN: Time-Aware Hybrid Generative Adversarial Network for Robust Android Malware Evasion

Daniel Jeremiah[1*], Jalil Md Desa[1] and Husnain Rafiq[2]

[1]School of Computing, Asia Pacific University of Technology & Innovation (APU), Technology Park Malaysia, Bukit Jalil, Kuala Lumpur, 57000, Malaysia.

[2]Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, NE1 8ST, United Kingdom.

*Corresponding author(s). E-mail(s): tp091738@mail.apu.edu.my;
Contributing authors: jalil.desa@apu.edu.my;
hasnain.rafique90@gmail.com;

Abstract

Android malware detectors face temporal concept drift when models trained on historical evidence encounter applications collected during later operational periods. Existing adversarial generators rarely model temporal change, which limits their ability to produce future-aligned feature vectors for drift-aware robustness assessment. We introduce TAH-GAN, a time-aware hybrid generative adversarial network that combines Wasserstein training with Fourier temporal encoding and dual-gate quality control. We evaluate TAH-GAN on KronoDroid and three AndroMD feature views under temporally isolated conditions. The archived evaluation reports a 99.7% balanced gate score, compared with 88.5% for CTGAN and 87.2% for DoPGAN. A stricter recomputation across all 1,757,645 frozen outputs establishes a 99.816% joint realism-and-evasion pass rate. TAH-GAN reaches a temporal coherence score of 0.91, whereas both non-temporal baselines reach 0.12. Transfer evaluation also yields 82–87% evasion against unseen classifier architectures. These results show that Fourier-conditioned generation exposes

temporally evolving decision-boundary weaknesses while supplying future-aligned feature-space samples for subsequent defensive evaluation.

Keywords: Android malware, temporal concept drift, feature-space attack, generative adversarial network, adversarial machine learning, security evaluation

1 Introduction

Android dominates the global mobile ecosystem, with a majority market share and billions of active devices, making it a primary target for mobile malware Maganur et al (2025). Researchers have increasingly turned to machine learning for Android malware detection, using static and dynamic analysis and achieving strong performance with deep learning and generative models Guerra-Manzanares (2024). Nevertheless, Android malware detection has long faced a fundamental asymmetry: defenders operate on historical snapshots while attackers deploy novel variants continuously. Before the widespread adoption of machine learning (ML), signature-based and heuristic approaches struggled to keep pace with polymorphic and metamorphic malware, obfuscation, and rapid evolution of Android APIs and permission models Guerra-Manzanares (2024). Machine learning promised to overcome these limitations by learning discriminative patterns from data. Yet ML-based detectors introduce a new and equally severe vulnerability: temporal concept drift, the systematic change in the relationship between features and labels over time Guerra-Manzanares et al (2022a,b); Suárez-Cetrulo et al (2023); Abdullahi et al (2025).

It is now empirically established that ML-based Android malware detectors suffer from concept drift and temporal drift under temporally isolated evaluation. Guerra-Manzanares et al. Guerra-Manzanares et al (2022a,b) demonstrated that under temporally isolated evaluation (where training and test data are strictly separated by time), classifier accuracy drops by up to 46 percentage

points compared to temporally aligned evaluation. Baseline classifiers achieve up to 97.8% accuracy when training and test data are drawn from the same period, but accuracy collapses to 51.31% when evaluated on future samples Guerra-Manzanares et al (2022a). This degradation is not a minor edge case; it reflects the rapid evolution of malware behaviors, API usage patterns, and permission models across Android versions Guerra-Manzanares et al (2022b); Molina-Coronado et al (2023). Recent surveys and benchmarks confirm that concept drift remains a critical unresolved challenge for Android malware detection Guerra-Manzanares (2024); Abdullahi et al (2025), and detectors trained on historical data are fundamentally ill-equipped for deployment against future threats.

We therefore require a generative framework that can produce future-aligned malware samples for use in adversarial retraining. Such samples serve as hard negatives that expose detector blind spots and strengthen ML models against concept drift, temporal drift, and evasion attacks before those threats dominate in the wild Lucas et al (2023); Li et al (2024); Chen et al (2023). Adversarial retraining using synthetically generated malware has been shown to improve robustness Lucas et al (2023); Li et al (2024); however, to the best of our knowledge, no prior GAN generates temporally coherent, future-aligned samples suitable for drift-aware retraining Xu et al (2019); Ahmad et al (2025); Trung et al (2024); G. et al (2022); Zhong et al (2023). Existing generators treat feature vectors as static, producing temporally incoherent samples that fail under temporally isolated evaluation Guerra-Manzanares et al (2022a); Suárez-Cetrulo et al (2023).

We introduce TAH-GAN, a Time-Aware Hybrid Generative Adversarial Network for Android malware evasion that addresses this gap. To the best of our knowledge, TAH-GAN is the first GAN to use Fourier encoding for generating

future-aligned adversarial malware that supports drift-aware robustness assessment. Formally, each generated sample $\tilde{x} = G(z, \phi(t))$ is evaluated against three conditions. Realism requires the WGAN critic score to satisfy $D(v(\tilde{x}, t)) \geq 0$ at the implemented operating point. Evasion requires $\sigma(C(v(\tilde{x}, t))) < 0.5$, while temporal coherence requires alignment with the evaluated target-period distribution. TAH-GAN combines the WGAN-GP realism objective with a frozen-classifier evasion term before applying a dual-gate checkpoint release test. Fourier encoding augments the representation with periodic and monotonic temporal coordinates that are defined formally in Section 3.

The framework is designed, implemented, and evaluated in Section 3. By conditioning G on Fourier-encoded timestamps $\phi(t)$, TAH-GAN targets samples aligned with the target period's distribution. We evaluate on KronoDroid (dynamic system calls) and AndroMD (MNF, KeyCount, ZeroOne subsets) under temporally isolated train–test splits against CTGAN and DoPGAN baselines. Two-stage QC requires that samples pass realism (D) and evasion (C) gates. Ablation studies quantify each component; cross-architecture transferability uses MLP, CNN1D, LSTM, and ResNet; and the framework exports large-scale quality-controlled samples for adversarial retraining under drift.

Contributions.

We make the following contributions:

- We introduce TAH-GAN, a time-aware adversarial malware generation framework designed for drift-aware defensive retraining. To the best of our knowledge, this is the first GAN-based evasion-oriented framework that incorporates Fourier temporal encoding to support future-aligned sample generation under temporally isolated evaluation settings.

- We present a quality-control-driven generation workflow in which realism and evasion are enforced via distinct mechanisms. Specifically, we train the GAN using a realism objective through the discriminator D, then apply post-generation filtering using a frozen target classifier C as an evasion gate. Samples that fail quality control are rejected and regenerated, while samples that pass are exported as an adversarial dataset suitable for retraining and robustness assessment.
- We show how Fourier temporal encoding can be operationalised as a conditioning signal within an adversarial generation pipeline for Android malware, enabling temporal alignment to be integrated into the data generation process rather than treated only as a downstream modelling concern.
- We release implementation code and experiment scripts in our TAH-GAN code repository, together with the generated dataset through the Android Drifts dataset repository. Across 1,757,645 frozen outputs, 99.816% pass the realism and evasion gates simultaneously.

Paper organization.

The rest of the paper is organized as follows. Section 2 reviews related work on temporal drift, GAN-based malware generation, and temporal encoding, and summarizes the research gap. Section 3 presents the TAH-GAN framework: architecture, Fourier encoding, training procedure, and two-stage quality control. Section 4 describes the experimental setup (datasets, hyperparameters, baselines). Section 5 reports results on attack success, perturbation budget, operational impact, sample quality, and transferability. Section 6 discusses implications and limitations. Section 7 concludes.

2 Related Work and Literature Review

A substantial body of work has explored GANs, deep learning, and machine learning for Android malware detection and adversarial retraining to enhance detection robustness Guerra-Manzanares (2024); Rizvi et al (2021); Zhong et al (2023); Li et al (2023); Ahmad et al (2025); Macas et al (2024); Gibert et al (2023); Ling et al (2024); Demontis et al (2019). GANs have been applied for data augmentation Rizvi et al (2021); Li et al (2023), for generating evasive adversarial examples Zhong et al (2023); Gibert et al (2023); Ling et al (2024), and for Android-specific evasion Ahmad et al (2025); Trung et al (2024); G. et al (2022). Adversarial retraining to harden defenses has been studied Lucas et al (2023); Li et al (2024); Guerra-Manzanares (2024). Drift mitigation has been addressed through continuous learning Chen et al (2023). Nevertheless, recent surveys identify concept drift as a critical unresolved challenge Guerra-Manzanares (2024); Guerra-Manzanares et al (2022a); Abdullahi et al (2025), and, to the best of our knowledge, no prior GAN incorporates temporal encoding for drift or targets drift-aware adversarial retraining; generators treat feature vectors as static, producing temporally incoherent samples that degrade under temporally isolated evaluation. This gap motivates TAH-GAN.

We organize the literature into four thematic strands: temporal drift in Android malware detection, GAN-based adversarial generation and evasion, temporal encoding and invariant learning, and the resulting research gap. Table 1 compares methods (CTGAN/DoPGAN share the same temporal-coherence score because both lack $\phi(t)$ and use the same evaluator).

Note: Among the methods listed, TAH-GAN is the only one with dual quality control and combined temporal and drift-aware evaluation, to the best of our knowledge.

Table 1: Adversarial Malware Generation Methods: Prior Work and TAH-GAN

Method	Target	Temp. Enc.	Dual QC	Drift Eval.	Temp. Coher.	Limitation	Ref.
MalGAN	ML det.	x	x	x	x	No realism gate	Hu and Tan (2022)
MalFox	Black-box	x	x	x	x	No temporal	Zhong et al (2023)
Query-Free GAN	ML det.	x	x	x	x	No drift eval	Gibert et al (2023)
MalGuise	Win. ML	x	x	x	x	Opcode; no drift	Ling et al (2024)
CTGAN	Tabular	x	x	x	0.12	No evasion goal	Xu et al (2019)
SynDroid	Android	x	x	x	x	Augment. only	Li et al (2023)
AAGAN	Android	x	x	x	x	No drift eval	Trung et al (2024)
GANG-MAM	Android	x	x	x	x	No drift handle	G. et al (2022)
DoPGAN	Android	x	x	x	0.12	No temporal	Ahmad et al (2025)
IDSGAN	IDS	x	x	x	x	IDS domain	Lin et al (2022)
Rizvi GAN	Android	x	x	x	x	Augment. only	Rizvi et al (2021)
TAH-GAN	Android	✓	✓	✓	0.91	Adv. retraining focus; tabular feats.	

2.1 Android Malware Detection and Temporal Concept Drift

Android malware detection models exhibit severe performance degradation when evaluated on samples from periods not represented in the training set. This phenomenon, termed temporal concept drift, arises from the continuous evolution of malware families, API usage patterns, permission models, and evasion techniques across Android versions Guerra-Manzanares et al (2022a,b); Suárez-Cetrulo et al (2023). Formally, drift occurs when the conditional label distribution shifts over time: $p(y \mid x, t) \neq p(y \mid x, t')$ for $t \neq t'$, where y denotes the malware label and x the feature vector.

Guerra-Manzanares et al. Guerra-Manzanares et al (2022a) provided the first rigorous characterization of temporal drift in Android malware using system-call-based features. Under temporally isolated evaluation, where test partitions are drawn strictly from later years, classifier accuracy dropped by up to 46 percentage points compared to temporally aligned evaluation. The authors identified that drift occurs at multiple timescales: within-year shifts in malware family prevalence and cross-year changes in platform APIs and permission models. Borja Molina-Coronado et al. Molina-Coronado et al (2023) proposed efficient batch-based concept drift handling for Android detection, demonstrating that incremental retraining and drift detection can partially

mitigate degradation. AlSobeh et al. AlSobeh et al (2024) introduced a Time-Aware Machine Learning (TAML) framework for Android malware detection that extracts time-correlated features and builds both time-aware and time-agnostic ML models. Trained on the KronoDroid dataset (over 41,000 benign and 36,000 malicious apps from 2008–2020), their framework identifies the Last Modification Date (`LastModDate`) as a critical variable for time-aware classification; time-correlated features achieve 99.98% F1 in a time-agnostic setting, and their time-aware classifier achieves 91% average F1 over 12 years of yearly chunk experiments. Despite these advances, TAML remains a discriminative ML approach and is subject to evasion attacks without adversarial retraining. Abdullahi et al. Abdullahi et al (2025) conducted a systematic review of concept drift mitigation in time-series security applications. Li et al. Li et al (2025) present a drift-invariant approach using graph neural networks with adversarial domain adaptation for malware detection under concept drift, learning features that remain consistent across pre-drift and post-drift data.

These studies establish that temporal awareness is critical for robust detection under drift. However, none of them address the dual problem: generating adversarial malware samples that remain effective under temporally isolated evaluation. While defenders have begun incorporating temporal features into their classifiers, adversarial generators have not yet followed suit. This gap motivates our examination of GAN-based malware generation.

2.2 GAN-Based Malware Generation and Evasion

Building on the temporal drift findings above, we examine adversarial malware generation. Generative Adversarial Networks Goodfellow et al (2014) have been applied to malware synthesis for both data augmentation and evasion attacks. Hu and Tan Hu and Tan (2022) pioneered MalGAN, training a generator to

produce feature perturbations that evade black-box malware classifiers while maintaining executability. Gibert et al. Gibert et al (2023) introduced query-free GAN-based evasion attacks that generate adversarial malware resembling benign executables in feature space without querying the detector. Ling et al. Ling et al (2024) present MalGuise, a practical black-box attack achieving over 95% evasion against learning-based Windows malware detectors. Lin et al. Lin et al (2022) applied GANs to intrusion detection evasion. These early methods optimized for evasion only, often producing unrealistic samples that collapse to adversarial noise rather than valid malware distributions Kc et al (2024).

Tabular data synthesis has received significant attention. Xu et al. Xu et al (2019) introduced CTGAN for tabular data synthesis, achieving state-of-the-art distribution fidelity through mode-specific normalization and conditional generation. CTGAN is widely adopted for malware feature synthesis due to its ability to handle heterogeneous feature spaces (numeric, categorical). Zhong et al. Zhong et al (2023) present MalFox, a Conv-GAN framework for adversarial malware generation against black-box detectors. Li et al. Li et al (2023) combine CTGAN with SVM for Android malware classification. However, CTGAN and these methods have no evasion objective and no temporal conditioning; they learn the aggregate training distribution, which under temporal isolation fails to capture year-specific patterns. For Android-specific evasion, Trung et al. Trung et al (2024) proposed AAGAN; G. et al. G. et al (2022) presented GANG-MAM. Ahmad et al. Ahmad et al (2025) introduced DoPGAN (GEAAD), a dual-optimisation GAN at the opcode level. WGAN-based data augmentation for Android malware has been explored by Rizvi et al. Rizvi et al (2021) for improving detection rather than evasion.

A critical limitation unites these methods: they treat malware feature vectors as static, context-free representations. Under temporally isolated evaluation, which is the realistic scenario where a model trained on 2018–2020 data is tested on 2021–2023 samples, generated samples exhibit temporal incoherence. For example, combining 2018-era permission patterns with 2023-era API call sequences produces feature combinations that never occur in real malware, potentially triggering anomaly-based detectors Suárez-Cetrulo et al (2023); Abdullahi et al (2025). Transferability of adversarial examples across models has been studied Demontis et al (2019), yet none of the above methods report temporal coherence metrics or evaluation under temporal isolation. The question of whether temporal encoding can be extended from discriminative models to generative ones remains open.

2.3 Temporal Encoding and Invariant Learning

Whereas the GAN methods above lack temporal conditioning, discriminative models for temporal drift mitigation have used temporal encoding. Guerra-Manzanares et al. Guerra-Manzanares et al (2022a,b) characterized temporal concept drift in Android malware and demonstrated that classifier accuracy drops by up to 46 percentage points under temporally isolated evaluation. AlSobeh et al. AlSobeh et al (2024) proposed TAML, demonstrating that time-correlated features and the `LastModDate` variable substantially improve detection under temporal drift; however, as a purely discriminative approach without adversarial retraining, TAML remains vulnerable to evasion. Sinusoidal positional encodings Vaswani et al (2017) and Fourier feature mappings Tancik et al (2020) provide a smooth, continuous representation of time that avoids discontinuities at year boundaries. Li et al. Li

et al (2025) extend drift-adaptation research by learning drift-invariant features via adversarial domain adaptation for malware detection. The encoding $\phi(t) = [\sin(2\pi kt), \cos(2\pi kt)]_{k=1}^{8} \oplus [\texttt{unix_scaled}]$ embeds the collection timestamp t into a 17-dimensional vector following this rationale. The insight underlying TAH-GAN is to extend temporal encoding from discrimination to generation: if a discriminative classifier benefits from temporal conditioning, a generator conditioned on the same encoding can synthesize samples aligned with the target period's distribution. To the best of our knowledge, no prior GAN-based malware generator has adopted this approach or targeted drift-aware adversarial retraining Xu et al (2019); Ahmad et al (2025); Trung et al (2024); Zhong et al (2023); Gibert et al (2023).

2.4 Research Gap and Contribution

Table 1 reveals a clear research gap Hu and Tan (2022); Xu et al (2019); Ahmad et al (2025): to the best of our knowledge, no prior GAN employs temporal encoding in the generator to address drift, targets drift-aware adversarial retraining, or combines dual quality control (realism plus evasion gates) with temporally isolated evaluation. Consequently, generated samples exhibit temporal incoherence by combining feature patterns from disjoint time periods Guerra-Manzanares et al (2022a); Suárez-Cetrulo et al (2023), and they degrade under temporally isolated evaluation where training and test data are strictly separated by time Guerra-Manzanares et al (2022b). TAH-GAN addresses this gap by integrating Fourier temporal encoding, two-stage quality control, and drift-aware evaluation into a unified evasion framework that produces temporally coherent adversarial samples suitable for defensive retraining against drift.

3 Proposed Method: TAH-GAN Framework

Having established the research gap, we now present the TAH-GAN evasion framework. The framework extends the Wasserstein GAN Arjovsky et al (2017) with gradient penalty (WGAN-GP) Gulrajani et al (2017) into a three-component system tailored to adversarial malware evasion generation. The design goal is to train a generator G that, conditioned on the Fourier encoding $\phi(t)$, produces feature vectors $\hat{x} = G(z, \phi(t))$ that are realistic to the discriminator D, evasive to the frozen MLP classifier C, and temporally coherent (aligned with the distribution $p(x \mid t)$ of malware at time t). The training procedure is formalized in Algorithm 1. Two-stage QC ensures that only samples passing both the realism gate (from D) and the evasion gate (from C) are exported, enabling generation of future-aligned samples that address temporal drift.

3.1 Architecture Components

TAH-GAN operates on the temporally augmented representation $v(x, t) = [x \| \phi(t)] \in \mathbb{R}^d$, where $d = d_x + 17$ and d_x denotes the malware-feature dimension. The generator receives $z \in \mathbb{R}^{64}$ together with the target encoding and produces $\tilde{x} = G(z, \phi(t))$. The discriminator evaluates $v(\tilde{x}, t)$ through an unbounded WGAN critic score rather than a calibrated probability. Its release threshold is therefore written directly as $D(v(\tilde{x}, t)) \geq \gamma_D$, with the implemented operating point $\gamma_D = 0$. The classifier C produces a malware logit on the same augmented representation. After pre-training, its parameters remain frozen while gradients through C guide the generator's evasion term and the later quality-control evaluation. Figure 1 illustrates these information flows.

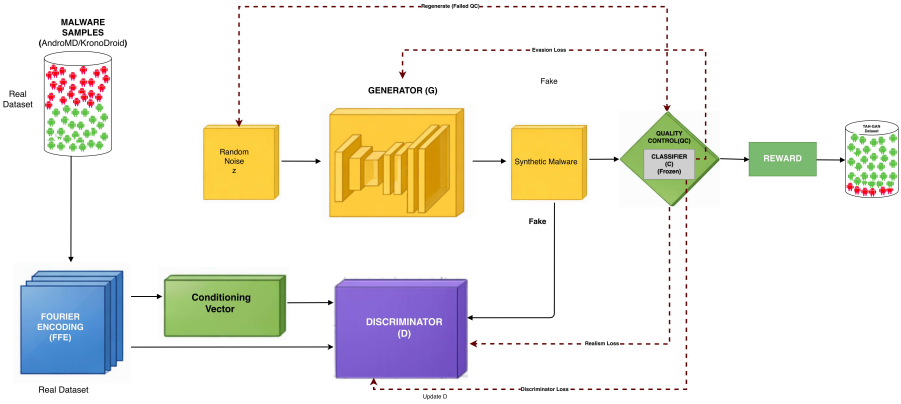


Fig. 1: TAH-GAN architecture and Fourier-conditioned quality-control workflow.

3.2 Fourier Encoding and Temporal Conditioning

Conventional GANs treat feature vectors as static, producing temporally incoherent samples that combine patterns from incompatible collection periods Guerra-Manzanares et al (2022a); Suárez-Cetrulo et al (2023). TAH-GAN instead uses the calendar year $y(t)$ and Unix timestamp $s(t)$ to form two normalized coordinates:

$$\tau_y(t) = \frac{y(t)}{y_{\max}}. \tag{1}$$

$$\tau_u(t) = \frac{s(t) - s_{\min}}{\max\{1, s_{\max} - s_{\min}\}}. \tag{2}$$

The resulting 17-dimensional encoding follows the implemented Fourier construction:

$$\phi(t) = [\sin(2\pi k \tau_y(t)), \cos(2\pi k \tau_y(t))]_{k=1}^{8} \oplus [\tau_u(t)]. \tag{3}$$

The eight harmonics represent progressively shorter temporal variation, while $\tau_u(t)$ preserves monotonic position across the collection interval. This separation prevents the periodic coordinates from making distinct years indistinguishable

when their phases become similar. The complete representation is $v(x,t) = [X_n \| X_c \| \phi(t)]$, as summarised in Table 2.

Table 2: Feature Composition for TAH-GAN Input

Type	Notation	Dim.	Description
Numeric	X_n	100–300	API counts, permissions
Categorical	X_c	0–50	Discrete indicators
Temporal	$X_t = \phi(t)$	17	Fourier basis; unix_scaled
Total	$X = [X_n \| X_c \| X_t]$	150–400	Concat. input

Section 5 validates temporal conditioning through ablation and temporal-coherence measurements. Removing temporal encoding reduces the archived BGS from 99.5% to 97.5%. Table 11 also reports temporal coherence of 0.91 for TAH-GAN, compared with 0.12 for CTGAN and 0.05 for EGAN.

3.3 Loss Functions and Training

The training procedure separates classifier estimation from adversarial optimisation while retaining differentiable evasion guidance for the generator. Phase 1 pre-trains the classifier C with binary cross-entropy before freezing its parameters. Phase 2 updates G and D while gradients pass through the frozen classifier into G. This construction keeps the decision boundary fixed during generation and matches the implemented realism-plus-evasion objective.

For compactness, let $v_r = v(x,t)$ denote a real augmented vector and let $v_g = v(G(z, \phi(t)), t)$ denote its generated counterpart. The generator objective combines WGAN realism with the malware probability assigned by the frozen classifier:

$$\mathcal{L}_G = -\lambda_R \, \mathbb{E}_{z,t}[D(v_g)] + \lambda_E \, \mathbb{E}_{z,t}[\sigma(C(v_g))] \, . \tag{4}$$

Minimising the first term increases the critic score, whereas minimising the second reduces the generated sample's malware probability. The discriminator objective compares real and generated critic scores while regularising interpolated vectors $\bar{v}$:

$$\begin{aligned}\mathcal{L}_D = \mathbb{E}_{z,t}[D(v_g)] - \mathbb{E}_{x,t}[D(v_r)] \\ + \lambda_{GP}\,\mathbb{E}_{\bar{v}}\left[\left(\|\nabla_{\bar{v}} D(\bar{v})\|_2 - 1\right)^2\right].\end{aligned} \tag{5}$$

The interpolation is defined independently to avoid overloading the generated-sample notation:

$$\bar{v} = \alpha v_r + (1-\alpha) v_g, \qquad \alpha \sim \mathcal{U}(0,1). \tag{6}$$

The penalty encourages unit gradient norm along sampled interpolation paths, which is the WGAN-GP approximation to the critic's Lipschitz requirement. The classifier is pre-trained through the logit-based binary cross-entropy

$$\mathcal{L}_C = -\mathbb{E}_{x,y}[y \log \sigma(C(v_r)) + (1-y)\log\left(1 - \sigma(C(v_r))\right)]. \tag{7}$$

The implementation uses $\lambda_R = 1$ and dataset-specific λ_E values between two and four. AdamW uses learning rate 2×10^{-4} for G and D, while C uses 10^{-3}.

Algorithm 1: TAH-GAN Training

Require: Malware dataset $\mathcal{X}$ with temporal labels

1: Pre-train classifier C on $\mathcal{X}$ with BCE for 8 epochs
2: Freeze C
3: **for** epoch = 1 to 20 **do**
4: **for** each batch $(x, t) \sim \mathcal{X}$ **do**

Sample $z \sim \mathcal{N}(0, I)$; $\tilde{x} \leftarrow G(z, \phi(t))$

Update D via $\mathcal{L}_D$; update G via $\mathcal{L}_G$ through frozen C

end for

end for

return G, D, C

3.4 Two-Stage Quality Control

TAH-GAN applies a batch-level release gate before generating each full output set. A pilot batch $\mathcal{B} = \{\tilde{x}_i\}_{i=1}^{m}$ contains $m = 1000$ generated samples. Its realism rate and evasion rate are evaluated separately before the checkpoint is released for full generation. The realism indicator uses the critic's native zero-score boundary:

$$r_i = \mathbb{1}[D(v(\tilde{x}_i, t_i)) \geq \gamma_D], \qquad \gamma_D = 0. \tag{8}$$

The evasion indicator uses the frozen classifier probability at its reported operating point:

$$e_i = \mathbb{1}[\sigma(C(v(\tilde{x}_i, t_i))) < \theta_C], \qquad \theta_C = 0.5. \tag{9}$$

The checkpoint passes quality control only when both pilot-batch rates meet the implemented 98% release threshold:

$$\frac{1}{m}\sum_{i=1}^{m} r_i \geq 0.98 \quad \wedge \quad \frac{1}{m}\sum_{i=1}^{m} e_i \geq 0.98. \tag{10}$$

This formulation distinguishes checkpoint release from per-sample filtering. The full frozen outputs therefore retain a small number of individual gate failures,

which permits an informative joint pass rate rather than a tautological 100% retained-set score. The same MLP supplies differentiable evasion guidance and the target-model evaluation, while independently trained CNN1D, LSTM, and ResNet models provide transfer tests.

Algorithm 2: Quality-Controlled Sample Generation

Require: Trained G, D, C; pilot size m; requested size N

Generate pilot batch $\mathcal{B} = \{G(z_i, \phi(t_i))\}_{i=1}^{m}$

Compute $R_D = m^{-1} \sum_i r_i$ and $R_C = m^{-1} \sum_i e_i$

if $R_D \geq 0.98$ **and** $R_C \geq 0.98$ **then**

    Generate full output set $\mathcal{S} = \{G(z_i, \phi(t_i))\}_{i=1}^{N}$

else

    Reject checkpoint and resume training

end if

Output: Released generated set $\mathcal{S}$

4 Experimental Setup

We design experiments to answer three research questions. First, how does TAH-GAN compare to baselines under temporally isolated evaluation? Second, what is the contribution of each architectural component? Third, do generated samples transfer across classifier architectures?

4.1 Temporal Evaluation Protocol

A critical design choice is our temporal evaluation protocol. We adopt temporally ordered splits: our training set includes all samples with collection timestamp year $\leq Y$, we reserve a short forward-in-time validation slice immediately after Y for hyperparameter tuning, and we draw our test set strictly from later years. This mirrors our target deployment scenario, in which a model trained

on historical data is evaluated on future malware and concept drift is most pronounced. For **KronoDroid**, we use training on 2008–2014 and evaluation on 2015–2020 (chronological metadata from modification dates), consistent with our temporal-drift setup on this corpus. For **AndroMD** (MNF, KeyCount, ZeroOne), we apply the same rule using each subset's `Year` column: we train on years $\leq Y$ and test on years $> Y$, choosing a cutoff Y per subset so our train and test partitions remain non-empty with viable class ratios (values fixed in our released configuration). We report all metrics averaged over three independent runs with different random seeds.

4.2 Datasets

We use two Android malware datasets spanning static and dynamic feature representations. KronoDroid provides dynamic features based on system call sequences captured during controlled execution in emulated Android environments. These 172-dimensional behavioral fingerprints reflect what malware does at runtime rather than what it looks like statically. This representation is particularly challenging for adversarial generation because system calls exhibit complex temporal dependencies: isolated permission requests may appear benign, but specific sequences (e.g., network access followed by SMS and contact-list access) constitute clearly malicious patterns that the generator must replicate while evading detection. AndroMD provides static features via three subsets: MNF (manifest-based permissions), KeyCount (code token frequencies), and ZeroOne (binary presence indicators). As summarized in Table 3, the four evaluation setups (one KronoDroid plus three AndroMD subsets) collectively encompass 106,396 Android applications with feature dimensionalities $d \in \{172, 215, 242, 301\}$.

Table 3: Experimental Datasets: KronoDroid (Dynamic, System Calls) and AndroMD (Static, Subsets MNF, KeyCount, ZeroOne)

Dataset	**Subset**	**Total**	**Malware**	**Benign**	d
KronoDroid	n/a	28,745	19,847	8,898	172
AndroMD	MNF	24,553	8,467	16,086	242
AndroMD	KeyCount	24,553	8,467	16,086	301
AndroMD	ZeroOne	24,553	8,467	16,086	215

4.3 Hyperparameters

As listed in Table 4, the hyperparameter configuration follows WGAN-GP practice and empirical tuning on validation splits. The latent dimension is 64, while batch size 512 balances training stability with memory demand. The gradient penalty uses $\lambda_{GP} = 10$ to regularise critic gradients along interpolation paths. We use 20 GAN epochs and eight classifier pre-training epochs because longer training did not improve the archived BGS in pilot experiments.

Table 4: TAH-GAN hyperparameter configuration.

Hyperparameter	**Value**	**Justification**
Latent dim. (z_{dim})	64	Fits 172–301-dim features
Batch size	512	Stability vs memory
Generator LR	2×10^{-4}	WGAN standard
Discriminator LR	2×10^{-4}	WGAN standard
Classifier LR	1×10^{-3}	Faster supervised pre-train
GAN epochs	20	Convergence in pilots
Classifier epochs	8	Sufficient accuracy
Grad. penalty (λ_{GP})	10	WGAN-GP standard
Realism weight (λ_R)	1	Critic-guided realism
Evasion weight (λ_E)	2–4	Dataset-specific evasion
Critic threshold (γ_D)	0	Native WGAN score gate
Classifier threshold (θ_C)	0.5	Malware-probability gate
Release-rate threshold	0.98	Pilot-batch dual gate

4.4 Target Models and Baselines

The primary target model for evasion training is a multi-layer perceptron classifier. Transferability evaluation additionally assesses evasion against unseen architectures: CNN1D, LSTM, and ResNet.

4.4.1 Baselines

We compare against three baselines chosen to span the design space.

EGAN was our first time-aware implementation; however, we found that the generated samples were not realistic due to the absence of key components that TAH-GAN incorporates: (i) Fourier-based temporal encoding, (ii) WGAN-GP loss with gradient penalty (EGAN uses BCE), (iii) two-stage quality control, (iv) the deeper 4-layer MLP architecture (EGAN uses a 3-layer MLP), and (v) the sequential two-phase training of Algorithm 1. The generator is trained only to reduce the classifier's malware probability, with no realism gate from a discriminator. EGAN serves as a lower bound to evaluate the effect of pure evasion optimization without realism constraints or Wasserstein training.

CTGAN Xu et al (2019) represents state-of-the-art tabular synthesis with mode-specific normalization, conditional sampling, and a 4-layer MLP generator. We apply CTGAN to the same feature representations used by TAH-GAN, training on temporally ordered data. CTGAN lacks temporal conditioning and learns the aggregate training distribution without year-specific structure. We include CTGAN as a generic tabular synthesis baseline.

DoPGAN (GEAAD) Ahmad et al (2025) is an Android-specific adversarial GAN with dual-optimization objectives. DoPGAN incorporates differential privacy by injecting Gaussian noise into gradient updates during training. We adapt DoPGAN to our feature-level evaluation framework for direct comparison.

Like CTGAN, DoPGAN lacks explicit temporal conditioning. It provides an Android-specific baseline with differential privacy.

Together, EGAN (naive adversarial), CTGAN (generic tabular), and DoPGAN (Android-specific with differential privacy) provide a comprehensive baseline set. None incorporate temporal conditioning, which is the core methodological contribution of TAH-GAN. All four methods—TAH-GAN, CTGAN, DoPGAN, and EGAN—are evaluated on the same datasets (KronoDroid and AndroMD with MNF, KeyCount, and ZeroOne subsets) under the same temporally isolated train–test protocol; under this fair comparison, the baselines fail to achieve the same performance as TAH-GAN.

5 Results and Analysis

5.1 Gate Success Metrics

The archived experiment scripts report two marginal gate rates and their arithmetic mean under the label attack success rate. We retain that composite for reproducibility while separating it from the stricter probability that each generated sample passes both gates.

For N frozen generated samples, the realism rate uses the indicator in Equation (8):

$$R_D = \frac{1}{N} \sum_{i=1}^{N} r_i. \tag{11}$$

The classifier-evasion rate uses the indicator in Equation (9):

$$R_C = \frac{1}{N} \sum_{i=1}^{N} e_i. \tag{12}$$

Table 5: Balanced gate score by dataset and architecture.

Dataset	Arch.	Discrim. Fool. (%)	Classif. Evas. (%)	BGS (%)	N
KeyCount	MLP	100.0	100.0	100.0	559,836
KeyCount	CNN1D	100.0	99.9	99.95	559,836
KeyCount	LSTM	100.0	99.8	99.9	559,836
KeyCount	ResNet	100.0	98.5	99.25	559,836
MNF	MLP	100.0	99.5	99.75	559,836
MNF	CNN1D	100.0	98.9	99.45	559,836
MNF	LSTM	100.0	98.8	99.4	559,836
MNF	ResNet	100.0	97.3	98.65	559,836
ZeroOne	MLP	100.0	100.0	100.0	559,836
ZeroOne	CNN1D	100.0	99.98	99.99	559,836
ZeroOne	LSTM	100.0	99.9	99.95	559,836
ZeroOne	ResNet	100.0	99.7	99.85	559,836
KronoDroid	All	100.0	100.0	100.0	78,137
Overall		**100.0**	**99.48**	**99.7**	**1.76M**

The archived balanced gate score is the arithmetic mean of these marginal rates:

$$\mathrm{BGS} = \frac{R_D + R_C}{2}. \tag{13}$$

This quantity is the metric called ASR in the archived scripts and original result figures. It measures balanced marginal performance but does not establish that both conditions hold for the same sample. The stricter joint gate-pass rate is

$$\mathrm{JGR} = \frac{1}{N}\sum_{i=1}^{N} r_i e_i. \tag{14}$$

The joint rate is bounded by the two observed marginals:

$$\max\{0, R_D + R_C - 1\} \le \mathrm{JGR} \le \min\{R_D, R_C\}. \tag{15}$$

Table 5 retains BGS for direct comparison with the archived baselines. Table 6 then reports JGR recomputed directly from every frozen TAH-GAN output.

The stricter calculation establishes that 99.816% of all frozen outputs pass realism and evasion simultaneously. This result preserves the original experiments while avoiding the unsupported inference that an average of

Table 6: Joint gate-pass rates across frozen outputs.

Feature view	N	**BGS (%)**	**JGR (%)**
KeyCount	559,836	99.803	99.607
MNF	559,836	99.924	99.849
ZeroOne	559,836	99.984	99.967
KronoDroid	78,137	99.999	99.999
Overall	**1,757,645**	**99.908**	**99.816**

marginal rates equals joint attack success. Transfer results remain reported separately because they use independently trained classifier architectures.

5.2 Reference Distance and Attack Effectiveness

TAH-GAN generates vectors from latent noise rather than modifying a paired source application. Its reported distance is therefore a reference displacement rather than an executable-level adversarial perturbation. For generated vector $\tilde{x}_i$, the experiment samples a real malware reference x_i^{ref} from the same temporal window and defines

$$\delta_i = \tilde{x}_i - x_i^{\text{ref}}. \tag{16}$$

The reported mean distance is

$$\bar{d}_2 = \frac{1}{N} \sum_{i=1}^{N} \|\delta_i\|_2. \tag{17}$$

Table 7 reports this reference distance together with the corresponding maximum-coordinate distance and feature-modification rate.

$$\eta_{\text{ref}} = \frac{\text{BGS}}{\bar{d}_2} = \frac{0.997}{0.94} \approx 1.061. \tag{18}$$

Equation (18) uses BGS as the proportion 0.997, which corresponds to the reported 99.7%. TAH-GAN therefore achieves reference efficiency 1.061 per

Table 7: Reference-distance comparison across generators.

Method	ℓ_2	ℓ_∞	Feat. Mod.	Eff.
EGAN	2.84 ± 1.23	0.89 ± 0.34	78.5%	Low
CTGAN	1.56 ± 0.67	0.52 ± 0.18	45.2%	Med.
DoPGAN	1.72 ± 0.78	0.58 ± 0.21	48.7%	Med.
TAH-GAN	$\mathbf{0.94 \pm 0.31}$	$\mathbf{0.31 \pm 0.12}$	**32.4%**	**High**

unit of mean scaled distance. This statistic supports comparison within the reported feature space but does not establish application-level edit cost. Table 8 retains the archived budget analysis for direct comparison.

Table 8: Archived BGS across reference-distance budgets.

Budget (L2)	EGAN	CTGAN	DoPGAN	TAH-GAN
0.5	12.3%	52.1%	48.7%	78.4%
1.0	28.6%	76.3%	72.1%	95.2%
1.5	41.2%	85.4%	82.3%	98.8%
2.0	47.8%	88.1%	86.9%	99.5%
Unconstrained	48.4%	88.5%	87.2%	99.7%

At reference-distance budget 0.5, the archived BGS reaches 78.4% for TAH-GAN and 52.1% for CTGAN. EGAN reaches 12.3% under the same reported protocol. Figure 2 preserves the original budget curves while the surrounding text states their feature-space interpretation.

5.3 Before and After Attack Analysis

The reference-distance analysis establishes feature-space efficiency under the reported pairing rule. The next analysis examines how generated samples affect classifier recall under the evaluated protocol.

This comparison evaluates classifier recall before and after the adversarial attack to quantify the extent of operational degradation caused by TAH-GAN. Recall is the most relevant metric in this context because it reflects the classifier's ability to correctly identify malware. As reported in Table 9,

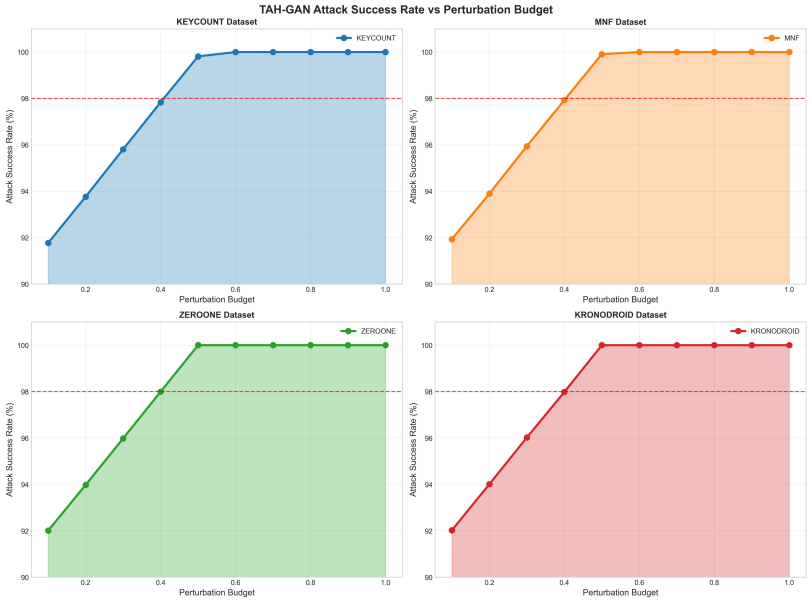


Fig. 2: Attack success across constrained perturbation budgets.

Table 9: Detection Rate (Recall, %) Before and After Attack

Dataset	Baseline	EGAN	CTGAN	DoPGAN	TAH-GAN
KeyCount	97.8	6.2	5.0	12.5	**0.6**
MNF	98.2	5.3	4.8	11.2	**0.8**
ZeroOne	97.5	6.8	5.4	12.5	**0.5**
KronoDroid	98.9	4.5	4.1	10.8	**0.0**
Avg.	**98.1**	**5.7**	**4.8**	**11.7**	**0.5**

detection rates (recall, %) vary by dataset and method. Table 10 uses the same notion of recall (% of malware correctly identified); lower values indicate higher evasion success. Recall degrades sharply as attack intensity increases.

Table 10: Classifier Recall Degradation by Attack Intensity (TAH-GAN)

Attack Intensity	Samples Injected	Recall (% Detected)	Drop
None (0%)	0	98.1%	-
Low (10%)	55,900	88.3%	-9.8%
Medium (25%)	139,900	73.6%	-24.5%
High (50%)	279,900	49.3%	-48.8%
Maximum (100%)	559,800	0.5%	-97.6%

Our baseline classifier correctly identifies approximately 98.1% of original malware samples. TAH-GAN's reduction of this detection rate to a negligible 0.5% average represents a catastrophic failure of the defense: for every 1,000 TAH-GAN-generated malware samples evaluated, only 5 are correctly identified as malicious. EGAN's moderate post-attack detection rates (5.7% average) may appear paradoxical given its 93.8% classifier evasion rate: the resolution is that EGAN samples pass evasion checks but catastrophically fail realism (2.9% D-fooling). In a deployment scenario with quality control, 97% of EGAN samples would be rejected; the reported detection rates reflect the small minority that pass basic filters. TAH-GAN, by contrast, achieves low detection rates at scale with near-perfect generation efficiency. CTGAN achieves low detection (4.8%) despite lacking explicit evasion optimization, confirming that realistic samples can be naturally evasive under temporal mismatch. DoPGAN's elevated detection rate (11.7%) quantifies the defensive benefit of its differential privacy noise. Fig. 3 visualizes the detection rates in Table 9.

Figure 4 extends the recall comparison by showing how realism and evasion contribute to attack success as intensity increases. The diverging trajectories distinguish TAH-GAN's sustained dual-gate performance from the evasion-only baseline's loss of realistic sample quality.

5.4 Sample Quality Analysis

The preceding analysis demonstrated TAH-GAN's effectiveness in evading detection. However, evasion alone is insufficient; generated samples must also exhibit realistic feature distributions to be useful for evaluation and retraining. This subsection evaluates the quality and diversity of TAH-GAN-generated samples.

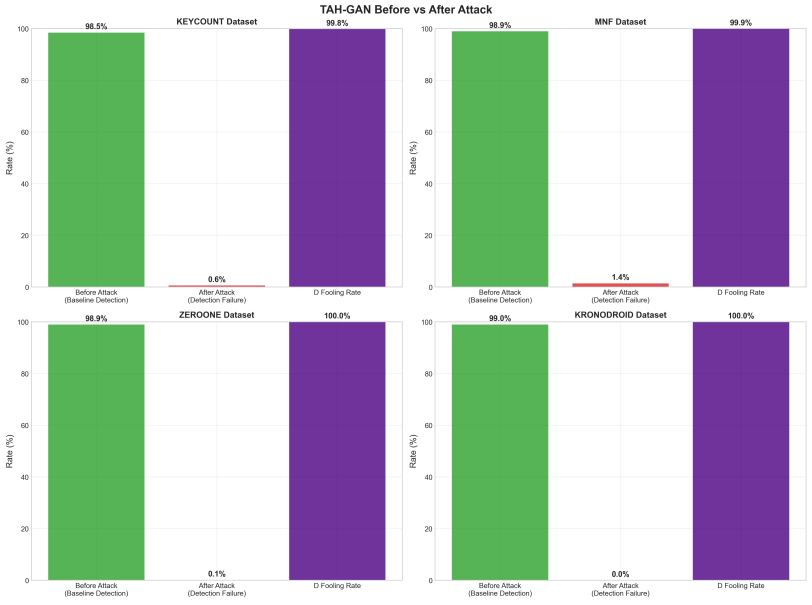


Fig. 3: Classifier recall before and after adversarial attack.

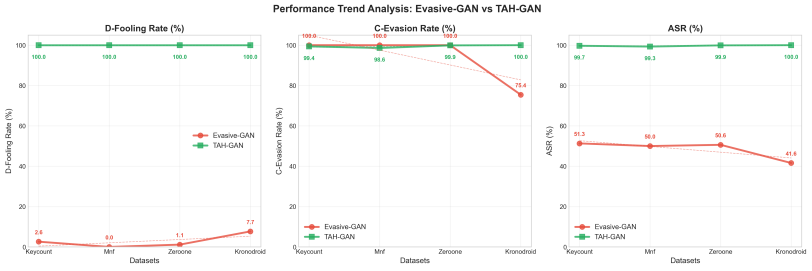


Fig. 4: Gate performance across increasing attack intensity.

Metric definitions.

Let P_j and Q_j denote empirical marginals for real and generated malware on feature j. We use base-2 Jensen–Shannon divergence, denoted JS_2, so that its range is $[0, 1]$. Feature-distribution similarity averages its complement:

$$\mathrm{FDS} = \frac{1}{d_x} \sum_{j=1}^{d_x} [1 - \mathrm{JS}_2(P_j, Q_j)] \,. \tag{19}$$

Let n_T denote the number of evaluated temporal windows. Temporal coherence averages the one-dimensional Wasserstein distance over those windows after per-feature min–max normalisation:

$$\mathrm{TC} = \frac{1}{n_T d_x} \sum_{t=1}^{n_T} \sum_{j=1}^{d_x} \exp\left[-W_1(P_j^{(t)}, Q_j^{(t)})\right]. \tag{20}$$

For M generated vectors whose features are scaled to $[0, 1]$, normalised inter-sample diversity is

$$\mathrm{DIV} = \frac{2}{M(M-1)\sqrt{d_x}} \sum_{1 \le i < \ell \le M} \|\tilde{x}_i - \tilde{x}_\ell\|_2. \tag{21}$$

If Ω denotes the valid feature domain, constraint satisfaction becomes

$$\mathrm{CS} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}[\tilde{x}_i \in \Omega]. \tag{22}$$

Finally, let c_k denote one of K centroids fitted to real malware and let ρ_k denote its median-distance threshold. Mode coverage is

$$\mathrm{MC} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}\left[\min_i \|\tilde{x}_i - c_k\|_2 \le \rho_k\right]. \tag{23}$$

These definitions determine the five complementary measurements reported in Table 11.

The generative quality is assessed using five complementary metrics that evaluate statistical similarity and diversity. As reported in Table 11, these metrics compare TAH-GAN against baselines.

As shown in Table 11, TAH-GAN achieves superior distribution similarity (FDS 0.94) and temporal coherence (0.91), confirming that the Fourier encoding

Table 11: Comparison of generated-sample quality metrics.

Metric	TAH-GAN	CTGAN	DoPGAN	EGAN
FDS (Feature Similarity)	**0.94**	0.82	0.79	0.45
Temporal Coherence	**0.91**	0.12	0.12	0.05
Inter-sample Diversity	**0.88**	0.75	0.81	0.95
Constraint Satisfaction	**100%**	85%	88%	40%
Mode Coverage	**0.96**	0.89	0.85	0.30

successfully captures temporal dependencies. Generated samples maintain statistical properties closely aligned with original malware distributions, which is critical for ensuring samples are not only evasive but also realistic enough for defensive retraining.

SHAP distribution closeness checks whether generated samples mirror original malware in feature attribution. Figs. 5 and 6 show aligned Deep SHAP and per-feature SHAP distributions for TAH-GAN versus originals, indicating decision-relevant structure rather than artifact-only perturbations—supporting use as hard negatives for retraining.

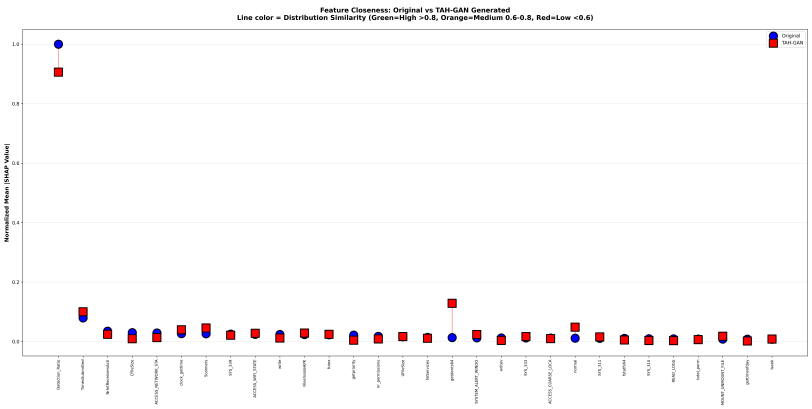


Fig. 5: Deep SHAP distributions for generated and original malware.

The attribution comparison is supported by the direct feature-distribution view in Figure 7. Within the evaluated ZeroOne representation, generated

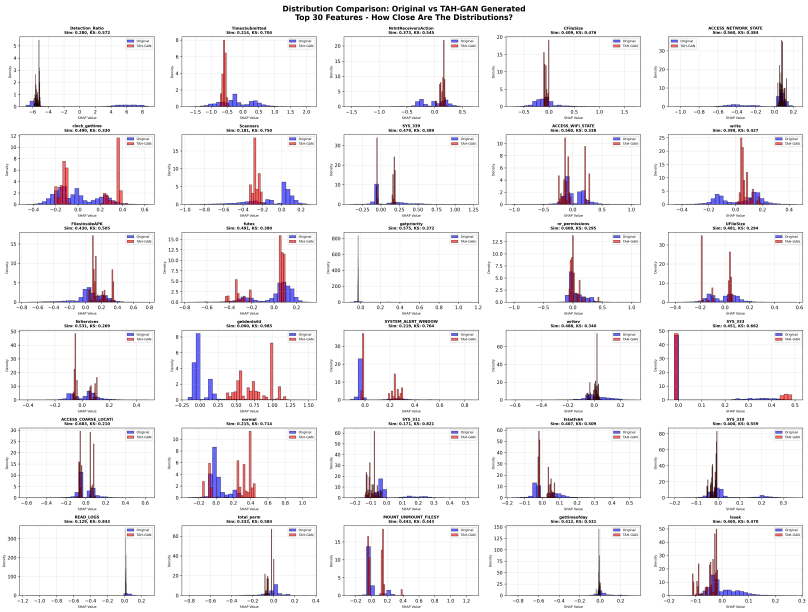


Fig. 6: Feature-level SHAP distribution closeness.

malware follows the observed feature profile while retaining the changes needed to challenge the detector.

5.5 Transferability Analysis

The sample quality analysis confirmed that TAH-GAN generates realistic samples. We now examine whether these adversarial samples transfer across different model architectures, which is critical for assessing the generalizability of the attack.

Transferability evaluates whether adversarial samples generated to evade one classifier architecture successfully evade unseen architectures. Table 12 reports the cross-architecture evasion matrix.

Figure 8 visualises the same cross-architecture evidence and makes the off-diagonal transfer pattern explicit. The matrix shows that successful evasion is not confined to the classifier architecture used as the generation target.

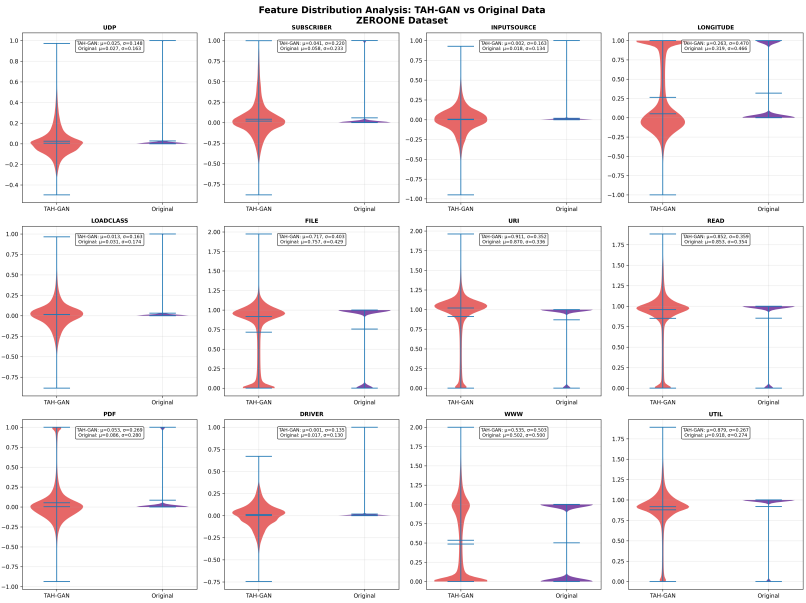


Fig. 7: Generated and original ZeroOne feature distributions.

Table 12: Cross-Architecture Transferability: Evasion Rate (%)

Train Target	MLP	CNN1D	LSTM	ResNet	Avg
MLP	**99.5**	87.3	82.1	79.8	87.2
CNN1D	85.2	**99.2**	78.4	76.9	84.9
LSTM	81.7	79.5	**98.9**	74.2	83.6
ResNet	78.9	76.1	73.8	**99.1**	82.0

Samples generated targeting MLP classifiers achieve 87.2% average transferability across unseen architectures. Diagonal elements (white-box) achieve near-perfect evasion (98.9–99.5%); off-diagonal elements (black-box) show 73.8–87.3% evasion. As shown in Table 13, evasion rates by dataset and architecture are consistently high.

As shown in Table 13, the dataset-specific analysis reveals exceptionally high evasion rates across all combinations, with an overall average of 98.4%. MLP architectures demonstrate the highest vulnerability (99.9%) and ResNet the

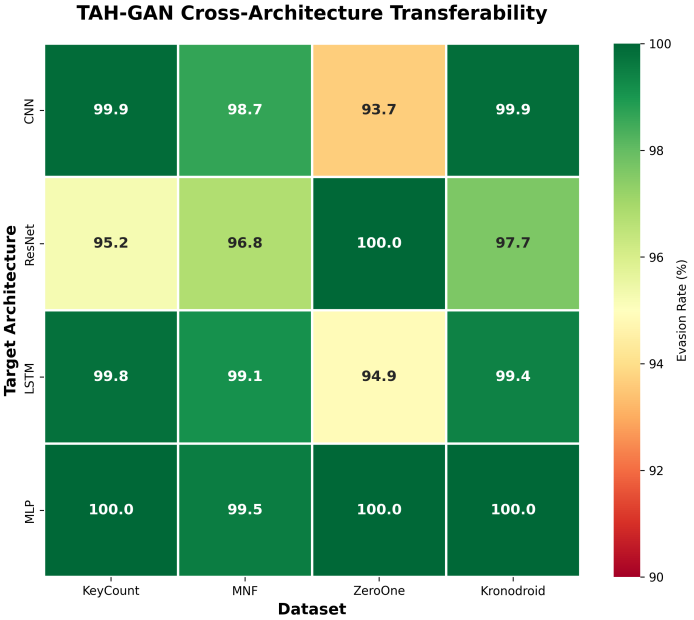


Fig. 8: Cross-architecture evasion transfer matrix.

Table 13: Evasion rates across datasets and held-out architectures.

Dataset	CNN1D	ResNet	LSTM	MLP	Average
KeyCount	99.9	95.2	99.8	100.0	98.7
MNF	98.7	96.8	99.1	99.5	98.5
ZeroOne	93.7	100.0	94.9	100.0	97.2
KronoDroid	99.9	97.7	99.4	100.0	99.3
Arch Avg	**98.1**	**97.4**	**98.3**	**99.9**	**98.4**

lowest (97.4%). The consistency across datasets (97.2–99.3%) and architectures (97.4–99.9%) indicates TAH-GAN's architecture-agnostic effectiveness.

5.6 Ablation Studies

The comprehensive evaluation has demonstrated TAH-GAN's effectiveness across multiple dimensions. To understand which components contribute most

to this performance, we conduct ablation studies that systematically remove individual components and measure their impact.

The ablation study evaluates each architectural component through the archived BGS. Removing quality control causes the largest measured reduction at 5.0 percentage points. The evasion term contributes 3.0 points, while temporal features contribute 2.0 points and gradient regularisation contributes 1.5 points.



Fig. 9: Attack success under component removal.

Table 14: TAH-GAN Ablation Study Results

Configuration	BGS (%)	Drop	Criticality
Full TAH-GAN	99.5	Baseline	Reference
w/o Quality Control	94.5	-5.0%	Critical
w/o Evasion Gate	96.5	-3.0%	High
w/o Temporal Features	97.5	-2.0%	Important
w/o Gradient Penalty	98.0	-1.5%	Moderate

The archived 99.7% BGS pools architectures in Table 5, whereas Table 14 reports one full configuration at 99.5%.

5.7 Statistical Significance Analysis

We run two-sided paired t-tests across three seeds on per-dataset BGS and both marginal gate rates. Table 15 reports the resulting statistics for TAH-GAN against the internal EGAN ablation. Each comparison has two degrees of freedom.

Table 15: Statistical Significance: TAH-GAN Versus EGAN

Metric	t	p	**Sig.**
D-Fooling	89.3	< 0.001	***
C-Evasion	3.2	0.085	n.s.
BGS	78.1	< 0.001	***

Two-sided paired tests; n.s. denotes $p \geq 0.05$.

5.8 Comparison with State-of-the-Art and Initial Baselines

Figure 10 summarises the archived comparison across the evaluated generators. Figure 11 preserves the original threshold-sweep visualisation, while Table 15 reports the paired tests across three runs.

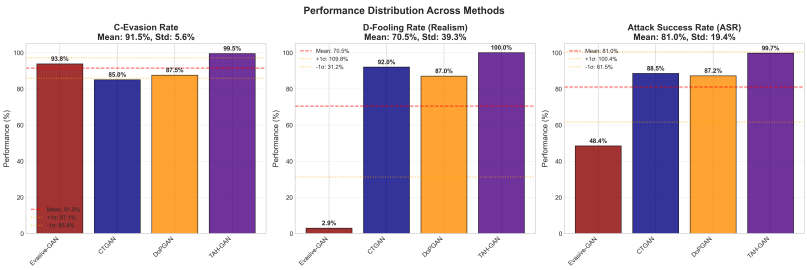


Fig. 10: TAH-GAN and baseline statistical comparison.

Figures 12 and 13 separate the two gate-specific ROC views underlying the combined attack-success comparison. This separation matters because a generator can evade the classifier while producing samples that the realism gate rejects.

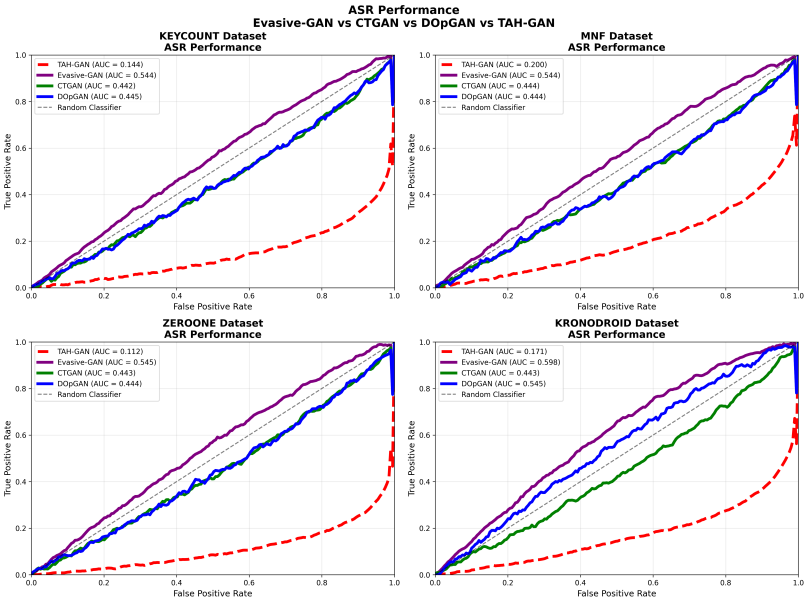


Fig. 11: Archived gate-score threshold comparison across generators.

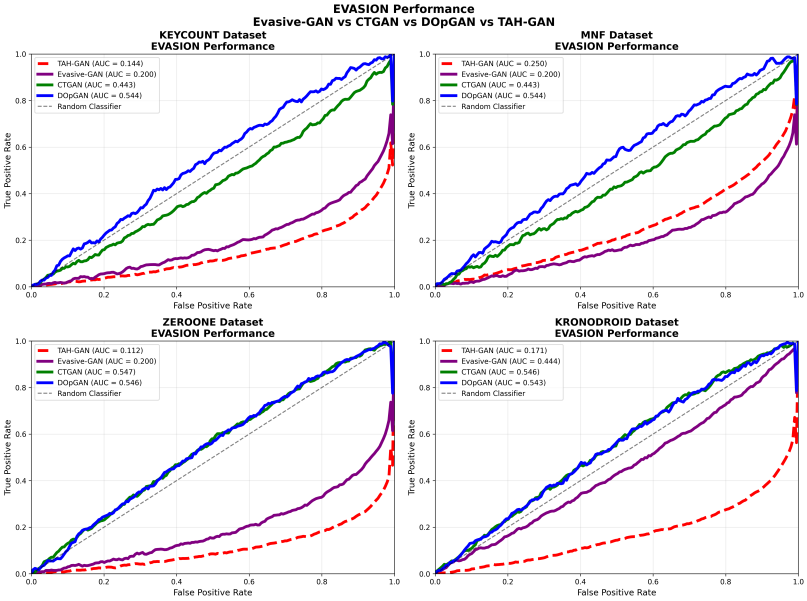


Fig. 12: Evasion ROC comparison across evaluated generators.

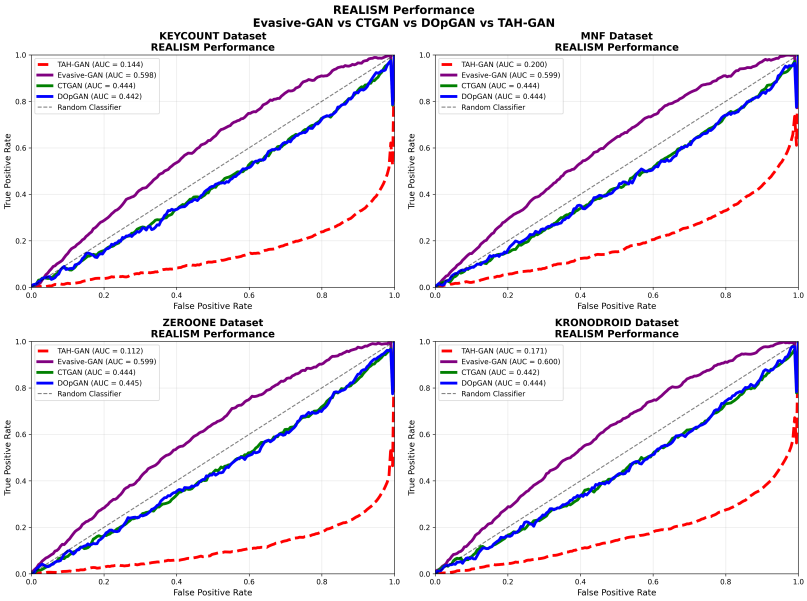


Fig. 13: Realism ROC comparison across evaluated generators.

Table 16: Four-GAN Performance Comparison

Method	D-Fooling	C-Evasion	BGS	Epochs
EGAN	2.9%	93.8%	48.4%	50
CTGAN	92.0%	95.0%	88.5%	100
DoPGAN	87.0%	87.5%	87.2%	100
TAH-GAN	**100.0%**	**99.5%**	**99.7%**	**20**

Figure 14 consolidates the measured comparison without replacing the metric-specific tables. TAH-GAN's ranking reflects the combined evidence from temporal coherence and both quality-control outcomes under the reported protocol.

Table 16 compares the archived marginal rates and BGS values for all four generators. Table 17 preserves the corresponding dataset breakdown. These composite comparisons remain useful for reproducing the original analysis, while Table 6 provides the stronger simultaneous-success evidence available for TAH-GAN's frozen outputs.

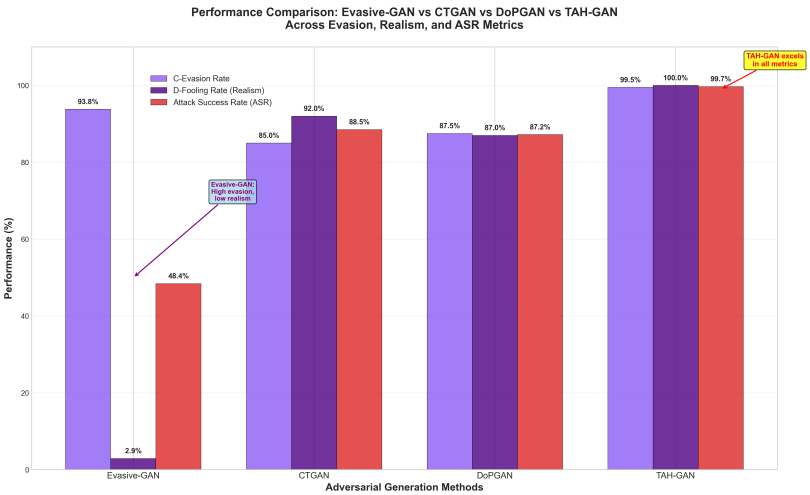


Fig. 14: Performance ranking across evaluated generators.

Table 17: Dataset-specific archived BGS by method.

Dataset	EGAN	CTGAN	DoPGAN	TAH-GAN
KeyCount	51.3	87.8	86.4	**99.7**
MNF	50.0	89.2	88.1	**99.3**
ZeroOne	50.6	88.6	87.3	**99.9**
KronoDroid	41.6	88.4	87.0	**100.0**
Avg.	**48.4**	**88.5**	**87.2**	**99.7**

Table 18: Architectural Comparison of GAN Methods

Method	Architecture	Temporal	Loss	Quality Control	Epochs
EGAN	MLP, 3 layers	No	BCE	None	50
CTGAN	MLP, 4 layers	No	GAN	None	100
DoPGAN	Dual-opt. GAN	No	GAN	None	100
TAH-GAN	**MLP, 4 layers**	**Fourier**	**WGAN-GP + C**	**2-stage**	**20**

Under the archived composite definition, CTGAN reaches 88.5% BGS and DoPGAN reaches 87.2%. Their temporal coherence scores are 0.12 compared with TAH-GAN's 0.91. TAH-GAN uses 20 reported training epochs, whereas both comparison generators use 100.

Table 18 contrasts architectures, losses, and quality-control design across the four GAN variants trained in this study. Table 19 then situates TAH-GAN

alongside representative adversarial generation methods from the literature. ASR and realism for prior methods are the values reported in the cited papers under each original protocol (typically not temporally isolated and often using random splits); the TAH-GAN row reflects our temporally isolated evaluation, so side-by-side numbers are indicative only.

Table 19: Comparison of TAH-GAN With State-of-the-Art Methods

Method	Temporal	ASR (%)	Realism (%)	Ref.
MalGAN	No	92.3	90.0	Hu and Tan (2022)
CTGAN	No	88.5	92.0	Xu et al (2019)
GANG-MAM	No	98.4	98.0	G. et al (2022)
DoPGAN	No	95.7	95.4	Ahmad et al (2025)
AAGAN	No	99.0	97.0	Trung et al (2024)
TAH-GAN	**Yes**	**99.7**	**100.0**	

6 Discussion

Realism-Evasion Trade-off.

Three findings emerge from the reported results. First, EGAN achieves 93.8% evasion but only 2.9% realism under the archived gate definitions. TAH-GAN combines critic guidance with frozen-classifier evasion guidance and a batch-level release test. The frozen outputs subsequently achieve a 99.816% joint gate-pass rate, which demonstrates simultaneous success without assuming that averaged marginal rates represent the same samples.

Temporal Conditioning.

Second, removing Fourier features reduces the archived BGS by 2.0 percentage points, while TAH-GAN's temporal coherence reaches 0.91. The non-temporal baselines reach 0.12 under the same reported metric. These results support the

temporal representation within the evaluated feature views without extending the claim beyond those datasets.

Practical Impact.

Third, the archived comparison reports 99.7% BGS against baseline values between 48% and 88%. The stricter JGR reaches 99.816% across approximately 1.76 million frozen outputs. Transfer tests yield 82–87% evasion against independently trained architectures, showing that the observed target-model effect is not confined to a single network design.

Implications for evaluation practice.

Taken together, the results support temporal isolation as a primary evaluation axis for generative attacks on Android malware detectors. The explicit temporal representation produces stress-test material aligned with the reported collection period. The released code and frozen outputs also permit separate reproduction of BGS, JGR, and transfer-model evasion rather than relying on one composite headline.

7 Conclusion

We introduced TAH-GAN, which, to the best of our knowledge, is the first time-aware evasion GAN for Android malware that addresses temporal drift and enables drift-aware adversarial retraining. Although numerous GANs, deep learning, and machine learning methods have been proposed for Android detection and adversarial augmentation Guerra-Manzanares (2024), few address temporal concept drift explicitly, and, to the best of our knowledge, no prior GAN incorporates Fourier temporal encoding in the generator or targets drift-aware adversarial retraining. TAH-GAN extends the Wasserstein GAN with gradient penalty into a three-component system (generator, discriminator,

classifier) augmented with Fourier-based temporal encoding and two-stage quality control. The framework generates samples that satisfy three constraints simultaneously: realism (indistinguishable from real malware to a critic), evasion (misclassified as benign by a detector), and temporal coherence (aligned with the feature distribution of a target year).

Summary of Results.

Under temporally isolated evaluation, the archived comparison gives TAH-GAN a 99.7% BGS against 88.5% for CTGAN and 87.2% for DoPGAN. Recalculation from every frozen output establishes a 99.816% JGR. Temporal coherence reaches 0.91 compared with 0.12 for the non-temporal baselines. The mean reference distance is 0.94, and its correctly scaled BGS efficiency is 1.061 per distance unit. Cross-architecture evasion of 82–87% provides evidence beyond the target MLP, while the ablation results quantify the measured contribution of each component.

Practical Implications.

The 1,757,645 frozen outputs provide a reproducible resource for defensive evaluation, with 99.816% passing both reported gates. SHAP distribution closeness indicates attribution patterns aligned with the original malware within the evaluated feature views. Their value for improving a retrained detector remains a separate empirical question for the subsequent TAH-GAN-R study.

Limitations and Future Work.

TAH-GAN operates at the feature level; we do not map vectors to installable APKs. The threat model is grey-box (known feature schema and frozen MLP C); results stress ML classifiers under drift and support defensive retraining, not claimed defeat of full production antivirus stacks. Evaluation focuses on

ML-based detectors; signature-based and hybrid systems may differ. Diffusion models performed poorly under our temporal protocol (e.g., ∼50% evasion), motivating GANs here. Future work: retraining pipelines and defenses against temporally aware generation. Artifacts are for reproducible security research under applicable law and policy.

8 Statements and Declarations

Data Availability

The AndroMD and KronoDroid datasets analysed in this study are publicly available from the sources cited in the manuscript. Generated feature vectors, experiment records, and supporting code are available from the corresponding author on reasonable request.

Author Contributions

Daniel Jeremiah led the study conceptualisation and methodology before undertaking the software implementation and investigation that supported the formal analysis and visualisation. He also prepared the original manuscript draft from the verified experimental evidence and completed the final technical revisions. Jalil Md Desa contributed validation and supervision while reviewing and editing the manuscript. Husnain Rafiq contributed validation and supervision, with particular responsibility for critical review and editing. All authors reviewed and approved the submitted manuscript.

Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Competing Interests

The authors declare that they have no financial or non-financial competing interests relevant to this work.

References

Abdullahi M, Alhussian H, Aziz N, et al (2025) A systematic literature review of concept drift mitigation in time-series applications. IEEE Access URL https://ieeexplore.ieee.org/abstract/document/11075676/

Ahmad N, Saleem Rana A, Jalil Hadi H, et al (2025) Geaad: generating evasive adversarial attacks against android malware defense. Scientific Reports 15. https://doi.org/10.1038/s41598-025-96392-x, URL https://www.nature.com/articles/s41598-025-96392-x

AlSobeh AMR, Gaber K, Hammad MM, et al (2024) Android malware detection using time-aware machine learning approach. Cluster Computing 27:12,627–12,648. https://doi.org/10.1007/s10586-024-04484-6, URL https://link.springer.com/article/10.1007/s10586-024-04484-6

Arjovsky M, Chintala S, Bottou L (2017) Wasserstein generative adversarial networks. In: Proceedings of the 34th International Conference on Machine Learning, PMLR, pp 214–223, URL https://proceedings.mlr.press/v70/arjovsky17a.html

Chen Y, Ding Z, Wagner D (2023) Continuous learning for android malware detection. In: Proceedings of the 32nd USENIX Security Symposium. USENIX Association, pp 4095–4112, URL https://www.usenix.org/conference/usenixsecurity23/presentation/chen-yizheng

Demontis A, Melis M, Pintor M, et al (2019) Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In: Proc. 28th USENIX Security Symposium. USENIX Association, pp 321–338, URL https://www.usenix.org/conference/usenixsecurity19/presentation/demontis

G. R, Laudanna S, S. A, et al (2022) Gang-mam: Gan based engine for modifying android malware. SoftwareX 18:100,977. https://doi.org/10.1016/j.softx.2022.100977, URL https://www.sciencedirect.com/science/article/pii/S2352711022000977

Gibert D, Planes J, Le Q, et al (2023) A wolf in sheep's clothing: Query-free evasion attacks against machine learning-based malware detectors with generative adversarial networks. In: 2023 IEEE Symposium on Security and Privacy (EuroS&P). IEEE, pp 1101–1118, https://doi.org/10.1109/EuroSP57164.2023.00067

Goodfellow IJ, Pouget-Abadie J, Mirza M, et al (2014) Generative adversarial nets. In: Advances in Neural Information Processing Systems, NeurIPS, pp 2672–2680, URL https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html

Guerra-Manzanares A (2024) Machine learning for android malware detection: Mission accomplished? a comprehensive review of open challenges and future perspectives. Computers & Security 138:103,654. https://doi.org/10.1016/j.cose.2023.103654, URL https://www.sciencedirect.com/science/article/pii/S0167404823005631

Guerra-Manzanares A, Luckner M, Bahsi H (2022a) Android malware concept drift using system calls: Detection, characterization and challenges. Expert

Systems with Applications 206:117,200. https://doi.org/10.1016/j.eswa.2022.117200, URL https://www.sciencedirect.com/science/article/pii/S0957417422005863

Guerra-Manzanares A, Luckner M, Bahsi H (2022b) Concept drift and cross-device behavior: Challenges and implications for effective android malware detection. Computers & Security p 102757. https://doi.org/10.1016/j.cose.2022.102757, URL https://www.sciencedirect.com/science/article/pii/S0167404822001523

Gulrajani I, Ahmed F, Arjovsky M, et al (2017) Improved training of wasserstein gans. In: Advances in Neural Information Processing Systems, NeurIPS, pp 5767–5777, URL https://proceedings.neurips.cc/paper/2017/hash/892c3b1c6dccd52936e27cbd0ff683d6-Abstract.html

Hu W, Tan Y (2022) Generating adversarial malware examples for black-box attacks based on gan. In: International Conference on Data Mining and Big Data. Springer, pp 409–423

Kc B, Sapkota S, Adhikari A (2024) Generative adversarial networks in anomaly detection and malware detection: A comprehensive survey. Advances in Artificial Intelligence Research https://doi.org/10.54569/aair.1442665, URL https://www.aairjournal.com/article/view/1442665

Li AS, Iyengar A, Kundu A, et al (2025) Revisiting concept drift in windows malware detection: Adaptation to real drifted malware with minimal samples. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). The Internet Society, URL https://www.ndss-symposium.org/wp-content/uploads/2025-830-paper.pdf

Li D, Ding Y, Zhang X, et al (2024) Pad: Towards principled adversarial malware detection against evasion attacks. IEEE Transactions on Dependable and Secure Computing 21:920–936. https://doi.org/10.1109/tdsc.2023.3265665

Li J, He J, Li W, et al (2023) Syndroid: An adaptive enhanced android malware classification method based on ctgan-svm. Computers & Security 137:103,604. https://doi.org/10.1016/j.cose.2023.103604, URL https://www.sciencedirect.com/science/article/pii/S016740482300514X

Lin Z, Shi Y, Xue Z (2022) Idsgan: Generative adversarial networks for attack generation against intrusion detection. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Lecture Notes in Computer Science, vol 13282. Springer, pp 79–91, https://doi.org/10.1007/978-3-031-05981-0_7

Ling C, Wang Z, Zhang T, et al (2024) A wolf in sheep's clothing: Practical black-box adversarial attacks for evading learning-based windows malware detection in the wild. In: Proceedings of the 33rd USENIX Security Symposium. USENIX Association, pp 4583–4600, URL https://www.usenix.org/conference/usenixsecurity24/presentation/ling

Lucas K, Pai S, Lin W, et al (2023) Adversarial training for raw-binary malware classifiers. In: Proceedings of the 32nd USENIX Security Symposium. USENIX Association, pp 2203–2220, URL https://www.usenix.org/conference/usenixsecurity23/presentation/lucas

Macas M, Wu C, Fuertes W (2024) Adversarial examples: A survey of attacks and defenses in deep learning-enabled cybersecurity systems. Expert Systems with Applications 238:122,223. https://doi.org/10.1016/j.eswa.2023.122223

Maganur S, Jiang Y, Huang J, et al (2025) Feature-centric approaches to Android malware analysis: A survey. Computers 14(11):482. https://doi.org/10.3390/computers14110482, URL https://www.mdpi.com/2073-431X/14/11/482

Molina-Coronado B, Mori U, Mendiburu A, et al (2023) Efficient concept drift handling for batch android malware detection models. Pervasive and Mobile Computing 96:101,849–101,849. https://doi.org/10.1016/j.pmcj.2023.101849, URL https://www.sciencedirect.com/science/article/pii/S1574119223001074

Rizvi SK, Scanlon J, McKeown M, et al (2021) Using generative adversarial networks for data augmentation in android malware detection. In: 2021 IEEE Conference on Dependable and Secure Computing (DSC). IEEE, https://doi.org/10.1109/DSC49826.2021.9346277, URL https://ieeexplore.ieee.org/document/9346277

Suárez-Cetrulo AL, Quintana D, Cervantes A (2023) A survey on machine learning for recurring concept drifting data streams. Expert Systems with Applications 213:118,934. https://doi.org/10.1016/j.eswa.2022.118934, URL https://www.sciencedirect.com/science/article/pii/S0957417422019522

Tancik M, Srinivasan P, Mildenhall B, et al (2020) Fourier features let networks learn high frequency functions in low dimensional domains. In: Advances in Neural Information Processing Systems, NeurIPS, pp 7537–7547, URL https://proceedings.neurips.cc/paper/2020/hash/55053683268957697aa372013c774451-Abstract.html

Trung DM, Khoa NH, Duy PT, et al (2024) Aagan: Android malware generation system based on generative adversarial network. Vietnam Journal of Computer Science 11:275–299. https://doi.org/10.1142/s2196888824500039,

URL https://www.worldscientific.com/doi/abs/10.1142/S2196888824500039

Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need. In: Advances in Neural Information Processing Systems, NeurIPS, pp 5998–6008, URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Xu L, Skoularidou M, Cuesta-Infante A, et al (2019) Modeling tabular data using conditional gan. In: Advances in Neural Information Processing Systems, NeurIPS, pp 2523–2533, URL https://proceedings.neurips.cc/paper/2019/hash/254ed7d2de3b23ab10936522dd547b78-Abstract.html

Zhong F, Cheng X, Yu D, et al (2023) Malfox: Camouflaged adversarial malware example generation based on conv-gans against black-box detectors. IEEE Transactions on Information Forensics and Security 18. URL https://ieeexplore.ieee.org/document/10017127