Evolutionary Malware Data Augmentation

Callum Musselwhite*, Bingle Stegmann Kruger*†, Geoff Nitschke*
*Department of Computer Science, University of Cape Town, Cape Town, South Africa
{MSSCAL002, KRGBIN001}@myuct.ac.za, gnitschke@cs.uct.ac.za
†Frankfurt School of Finance & Management, 60322 Frankfurt am Main, Germany

***Abstract*—Machine learning benefits malware detection and classification, yet scarce labelled data and rapidly evolving threats hinders progress. We investigate generative, evolutionary, and hybrid data augmentation in static malware detection given data scarcity and distributional shifts. We evaluate a generative adversarial network, evolutionary method, and hybrid method, with static features and Random Forest classifiers. Experiments across random scarcity, temporal holdout, and leave-one-family-out regimes (dataset treatments) indicate evolutionary augmentation provides consistent improvements if positive samples are extremely scarce and under temporal drift. Results also indicate the effects of leave-one-family-out evaluation are family dependent and as real data availability increases, augmentation benefits diminish, where training on real data alone becomes sufficient.**

I. Introduction

Malware (malicious software) is any program created to cause harm or infiltrate computing systems when executed [1], and manifests in various forms, including viruses, worms, Trojans, and spyware [2]. Malware is continually adapted, resulting in an arms race between attackers and defenders [3]. Predictive machine learning [4] has been used for learning both malware detectors and classifiers [1], [5]. However, various challenges persistent. First, labelled malware data are often scarce, imbalanced, or outdated due to data collection and annotation problems [6], [7]. Moreover, copyright restrictions on benign binaries and commercial interests limit access to data [1], resulting in incomplete, class-imbalanced, or outdated datasets, particularly for rare or emerging malware families. This limitation undermines model accuracy, since classifiers require sufficient samples and intra-class diversity to learn reliable decision boundaries and avoid over-fitting in high-dimensional feature spaces [8]. Second, concept drift [9] arises as malware distributions evolve with emergent new families, code modifications, and obfuscation techniques such as polymorphism [10] and metamorphism [11]. Classifiers trained on historical data degrade substantially when applied to future samples. This limits their robustness and generalizability [12].

Data augmentation addresses data scarcity by expanding training sample diversity in domains such as computer vision and natural language processing [13]. However, data augmentation in anti-malware applications is problematic since program representations are structured and semantics-bearing: small perturbations invalidate samples or create detectable artifacts [3], [14]. Thus, *Generative Adversarial Networks* (GANs) [15] are proposed as an alternate synthetic data generation approach. GANs model high-dimensional malware feature distributions and generate synthetic samples to improve classifiers under scarcity [16], [17]. GANs for data augmentation improve detection accuracy, especially for Android malware and zero-day settings [17]–[19], with gains strongest when synthetic data is carefully balanced against limited real samples [19], [20]. However, GAN training remains unstable and susceptible to mode collapse, which limits coverage of rare but decision-critical behaviors [21], [22]. Also, most studies are small-scale, Android-centric, use *Independent and Identically Distributed* (IID) splits, and rarely sweep scarcity levels, evaluate under temporally ordered or leave-one-family-out (LOFO) splits, synthetic-to-real ratios, multi-seed stability, and shift-sensitive metrics, leaving robustness under drift largely unexplored [16]–[20].

Previous data augmentation for malware detection and classification work [17], [20], [23] only evaluates augmentation in contrived experiments or report aggregate gains without examining effectiveness across varying degrees of data scarcity. Evaluations often omit realistic distribution shifts, such as temporal drift or malware family exclusion. Thus, the augmentation threshold (the point at which additional synthetic samples no longer yield measurable performance improvements) remains largely unexplored. Ascertaining when augmentation yields gains and when it does not enables more effective labelling and computational resource allocation in response to adapting malware threats.

We evaluate if generative versus evolutionary versus hybrid augmentation improve static malware detection for extreme data scarcity and distribution shift. We use GAN-based augmentation (GAN), feature-space evolutionary synthesis (EVO) and hybridize both techniques (EvoGAN), using static Windows *Portable Executable* (PE) features [6]. To isolate augmentation impact, we keep feature representation and downstream classifier (random forest [24]) fixed for all conditions, evaluating performance under random sub-sampling for IID scarcity, temporal holdout to capture concept drift, and leave-one-family-out testing for unseen malware family generalization. We also test against non-generative benchmarks: *Random Over-Sampling* (ROS), SMOTE [25], and real-only training to address two research questions. First, given temporal distribution shift, do generative, evolutionary-based, or hybrid augmentation methods improve detection performance versus non-generative benchmarks across varying levels of data scarcity? Second, given malware family exclusion (during training), do our augmentation methods improve generalization to unseen families relative to the same baselines?

II. METHODS

We consider three malware only augmentation variants in feature space: Wasserstein GAN [26] with gradient penalty, a feature space evolutionary synthesizer, and a hybrid refining GAN samples with evolutionary operators. We apply the same methods and filtering protocol across multiple Windows PE feature datasets: BODMAS, EMBER, and SOREL [6], [7], [27], which are established benchmark datasets for static Windows PE malware detection, widely used to evaluate machine learning methods on large-scale, labeled malware corpora [6], [7], [27]. Dataset details, including split definitions and feature dimensionality, are described in the Experiments section III.

A. Wasserstein GAN with Gradient Penalty

We synthesize malware feature vectors using a Wasserstein GAN with gradient penalty (WGAN-GP) [26], [28]. Generators (G) and discriminators (D) are fully connected *Multi-Layer Perceptrons* (MLPs) with two hidden layers, with latent $z \in \mathbb{R}^{d_z}$. The GAN is trained on the malware only rows and benign samples are not used to fit G.

a) Architecture: The WGAN-GP generator applies Linear, ReLU, Linear, ReLU, and Linear activations to the d dimensional feature space. The discriminator applies Linear, LeakyReLU(η), Linear, LeakyReLU(η), and Linear activations to a scalar discriminator score. For preprocessing and inverse transform, $X_{\text{mal}} \subset \mathbb{R}^d$ are malware dataset rows, where we fit a standardizer on X_{mal} as defined in Eq. (1),

$$\hat{x} = \frac{x - \mu}{\sigma}, \quad \mu = \mathbb{E}[x], \quad \sigma = \sqrt{\mathbb{V}[x]}. \tag{1}$$

We train on $\hat{X}_{\text{mal}}$, where at sampling time we invert the transform with the stored (μ, σ) to obtain x_{syn} in the original feature space prior to shared filtering.

b) Objectives: Given p_r (malware data distribution) and $p_z = \mathcal{N}(0, I)$, the WGAN-GP discriminator maximizes the Wasserstein objective with gradient penalty given in Eq. (2).

$$L_D = \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))] + \lambda_{gp}\, \mathbb{E}_{\hat{x}} \left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1 \right)^2 \tag{2}$$

Where, $\hat{x} = \epsilon x + (1 - \epsilon) G(z)$, $\epsilon \sim U(0, 1)$. The generator minimizes the discriminator score on fakes using loss (Eq. 3).

$$L_G = -\mathbb{E}_{z \sim p_z}[D(G(z))]. \tag{3}$$

c) Training and Hyper-parameters: We alternate n_{critic} discriminator steps per generator step and update D by descending ∇L_D, updating G by descending ∇L_G. We use Adam [29] with learning rate λ_{lr} and (β_1, β_2), where gradient penalty coefficient λ_{gp}, batch size B, n_{critic}, epochs, and other hyper-parameter values are reported in Section III.

d) Saved Artifacts and Usage.: For each TRAIN split or cut-off (Table I) we use generator θ_G and malware only scaler parameters (μ, σ), where augmentation draws $z \sim \mathcal{N}(0, I)$, compute $G(z)$, inverts the standardization using Eq. (1), and applies the shared budget and quality filtering (Table I).

B. EVO Feature Space Evolutionary Synthesizer

EVO generates malware candidates in the original feature space without standardization or inverse transforms. Starting from a like pool X^+ of real malware feature vectors, EVO forms offspring by a convex mix of two parents followed by a small feature wise Gaussian mutation as defined in Eq. (4).

$$x_{\text{child}} = a p_1 + (1 - a) p_2 + \epsilon \odot s, \quad p_1, p_2 \in X^+. \tag{4}$$

Where, $a \sim \text{Beta}(\alpha_{cx}, \alpha_{cx})$ controls the convex combination, while $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$ injects diversity. The element wise scale s is set from X^+ using the interquartile range, with a standard deviation fallback per coordinate as defined in Eq. (5).

$$s_j = \begin{cases} \text{IQR}_j(X^+), & \text{IQR}_j > \varepsilon, \\ \text{Std}_j(X^+), & \text{Std}_j > \varepsilon, \\ 1.0, & \text{otherwise}, \end{cases} \tag{5}$$

Where, ε is a small positive constant. To avoid implausible extremes for small sample sizes, each coordinate is clamped to robust quantile bounds estimated from X^+ using Eq. (6).

$$x_{\text{child},j} \leftarrow \min\left(\max(x_{\text{child},j}, q^+_{low,j}), q^+_{high,j} \right), \tag{6}$$

Where, quantile bounds are defined in Eq. (7).

$$(q^+_{low}, q^+_{high}) = \left(\text{Quantile}_{\tau_{low}}(X^+), \text{Quantile}_{\tau_{high}}(X^+) \right). \tag{7}$$

Quantile levels τ_{low} and τ_{high} define a clamping range to prevent unrealistic feature values under extreme scarcity. EVO uses a candidate batch larger than required and then reuses the shared budget and quality gates. We generate q_m times more candidates than needed and keep those that are on manifold, meaning the nearest neighbour distance to X^+ is below a median based threshold. When the boundary filter is enabled, candidates must be near but outside the negative region by constraining their distance to X^- within a fixed quantile band.

C. EvoGAN Evolution of GAN Samples

EvoGAN operates in the original feature space by drawing candidates from the trained generator G, inverse transforms to the raw space, and then applies one evolutionary refinement step to those candidates before the shared quality gate. Positive malware samples are denoted by $\hat{x}_i = G(z_i)$, $z_i \sim \mathcal{N}(0, I)$ and X^+, where we define the parent pool P as in Eq. (8),

$$P = \begin{cases} \{\hat{x}_1, \ldots, \hat{x}_m\}, & \text{parent source is GAN}, \\ X^+, & \text{parent source is real}, \\ \{\hat{x}_1, \ldots, \hat{x}_m\} \cup X^+, & \text{parent source is both}. \end{cases} \tag{8}$$

We then generate children using the same convex mix and mutation operator as EVO, with per dimension scale from Eq. (5), robust clamping from Eq. (6), bounds from Eq. (7). As with EVO, we oversample by a factor q_m and then retain only candidates that pass the on manifold and near boundary filters. The augmentation budget is identical to other methods and it is capped at a fixed number of synthetic positives per real positive, subject to the global training size cap (Table I).

TABLE I
HYPER-PARAMETERS AND FLAGS USED ACROSS ALL EXPERIMENTS (EXACT VALUES). THIS TABLE COMPLEMENTS TABLE II.

Component	Setting	Exact value(s)
Shared budget, filtering	TRAIN (reference sets)	Per experiment regime (IID, temporal, LOFO), TRAIN is regime-specific training partition before augmentation. Gate reference sets drawn from TRAIN: X^+=all malware positives from TRAIN, X^-=benign samples from TRAIN (when boundary gate enabled).
	Synth per real cap	$\alpha = 40$: $n_{\text{syn}} \leq \alpha\, n_{\text{real,pos}}$.
	Global train cap	$N_{\text{cap}} = 20000$ (cap, not a target; not binding at our fractions).
	Candidate oversampling	$q_m = 5$ candidates (applies to GAN, EVO, EvoGAN).
	Like set	Positives from full TRAIN.
	Quality gate (nn)	Nearest-neighbour manifold gate. For each candidate x, compute $d^+(x) = \min_{u \in X^+} \|x - u\|_2$. Let $t^+ = \text{median}(\{d^+(x)\})$ over the candidate batch; accept candidates with $d^+(x) \leq t^+$ and then keep the top n_{syn} by smallest $d^+(x)$.
	Quality gate (nn_boundary)	Compute $d^-(x)$=$\min_{v \in X^-} \|x - v\|_2$, require $d^-(x)$ to lie in quantile band $[0.20, 0.60]$ (k=5 as per neighbourhood operations). If disabled, only the nn manifold gate is applied.
	Seeds	$\{42, 1337, 2025\}$ for all runs.
GAN (WGAN GP)	Arch	MLP (2 hidden layers): G, D. Latent dim 128.
	Optimiser	Adam. LR 1×10^{-4}. $(\beta_1, \beta_2) = (0.0, 0.9)$.
	GP coeff	$\lambda_{gp} = 10$.
	Critic steps	$n_{\text{critic}} = 5$ (IID, LOFO), 4 (temporal).
	Batch size	128.
	Epochs	30 (IID, LOFO), 80 (temporal).
EVO	Parent pool	X^+ (real positives from TRAIN).
	Mix and mutation	Convex mix with $\alpha_{cx} = 2.0$. Gaussian mutation $\sigma = 0.10$ scaled by robust per feature spread. Clamp to $(q_{\text{low}}, q_{\text{high}}) = (0.01, 0.99)$.
	Boundary band	As shared: $[0.20, 0.60]$, $k = 5$ (when boundary gate enabled).
	Oversampling	$q_m = 5$.
	Budget	Uses shared α and N_{cap}.
EvoGAN	Parent source	parent_source=gan (GAN samples), then EVO step in raw space.
	Mix and mutation	Same as EVO.
	Clamp and band	Same as EVO and shared.
	LOFO ablation	Wider search: $\sigma = 0.15$, boundary band $[0.15, 0.70]$.
	Budget	Uses shared α and N_{cap}.
ROS	Mechanism	Duplicate minority ($y = 1$) with replacement (subject to N_{cap}).
SMOTE	Mechanism	Numeric only interpolation between a positive and one of its $k = 5$ positive neighbours. Subject to N_{cap}.
	Fallback	If $\|X^+\| < 2$ or invalid neighbourhoods, fall back to ROS.
RF and validation	Classifier	Random Forest: $n\_estimators$=400, max_depth=20, $class\_weight$=None, n_jobs=-1.
	Thresholding	$\tau^\star$ maximises Balanced Accuracy on a validation split from TRAIN (per split, cut-off).
	Temporal thresholds	$\tau^\star$ frozen per cutoff and reused for all windows.
	Metrics	ROC AUC, PR AUC, Accuracy, Precision, Recall, Specificity, Balanced Accuracy, F1, MCC, plus raw counts (TP, FP, TN, FN).

D. Data Augmentation Configuration

Given $n_{\text{real,pos}}$ denotes the number of real positives in the scarce TRAIN subset, we cap the number of synthetic positives (augmentation variants) by a per real ratio α as in Eq. (9).

$$n_{\text{syn}} \leq \alpha\, n_{\text{real,pos}} \tag{9}$$

Where, we enforce a ceiling on the total training size N_{const}, where if $n_{\text{real}} + n_{\text{syn}} < N_{\text{const}}$, we do not pad further. To give the filter headroom, each method uses an over-complete set of candidates and then keeps exactly n_{syn} after filtering. We generate the candidate count as in Eq. (10).

$$n_{\text{gen}} = q_m\, n_{\text{syn}} \tag{10}$$

Where, candidates apply the quality gate, retaining the top n_{syn} survivors. Distances are computed for two fixed reference sets drawn from TRAIN. These are X^+: malware positives, representing the like manifold, and X^-: benign samples, supporting boundary proximity checks. Each configuration is evaluated using the following seeds: $\{42, 1337, 2025\}$. Unless stated otherwise, the like set X^+ uses all positives in the full TRAIN partition. The boundary aware variant uses a fixed neighbour count k, accepting candidates whose $d^-(x)$ is within a quantile band $[q_\ell, q_u]$. For a candidate x, nearest neighbour distances are as in Eq. (11).

$$d^+(x) = \min_{u \in X^+} \|x - u\|_2, \quad d^-(x) = \min_{v \in X^-} \|x - v\|_2. \tag{11}$$

The default gate (Nearest Neighbour) accepts candidates near the positive manifold by requiring $d^+(x)$ to fall below a robust median based threshold (computed over the candidate batch). Evaluation is fixed for all methods, so the test partition and rolling windows are identical for real only, ROS, SMOTE, GAN, EVO, and EvoGAN. All variants train the same classifier with the same validation and thresholding procedure. Differences in ROC AUC, PR AUC, Balanced Accuracy, F1, or MCC arise from training distribution changes (augmentation method and budget). Scarcity floors on the real training set are enforced prior to augmentation. Section III reports settings for α, q_m, N_{const}, neighbourhood size and quantile bands.

E. Benchmarks

We evaluate generative variants versus three non-generative baselines under identical splits, sampling, held out test par-

tition, and rolling windows (Table I). Real-only, ROS, and SMOTE were selected as standard non-generative reference points for scarcity and class imbalance. Real-only provides the no-augmentation baseline, ROS represents the simplest resampling strategy by increasing minority frequency without introducing new feature values, and SMOTE represents the standard interpolation-based synthetic oversampling baseline for tabular data [25]. These baselines therefore span increasing augmentation complexity while remaining widely used and reproducible. To keep the comparison controlled, all benchmark methods use the same scarce TRAIN partition, downstream classifier, validation protocol, and global training-size cap. Their parameters were chosen as simple standard settings rather than being separately tuned per dataset, so that performance differences more directly reflect the augmentation mechanism itself rather than benchmark-specific optimisation.

a) Real Only: As a reference for the effect of scarcity, the detector is trained on the scarce training subset (TRAIN), as drawn for each fraction or window, with no synthetic data.

b) Random Over-sampling: We increase the number of malware positives by sampling minority examples with replacement from the training set, without creating new feature values. The scarce training is defined as in Eq. (12).

$$D_{\text{scarce,train}} = (X, y), \tag{12}$$

Where, the positive training vector set is defined in Eq. (13)

$$X^{+}_{\text{train}} = \{x_i : y_i = 1\}. \tag{13}$$

Duplicates are uniformly drawn (positive set) as in Eq. (14), and appended (n_{add}) to the training set.

$$x_i \sim \text{Unif}\left(X^{+}_{\text{train}}\right), \tag{14}$$

Where, the training size limit is enforced as in Eq. (15).

$$n_{\text{add}} \leq N_{\text{const}} - |X|. \tag{15}$$

c) SMOTE: We synthesize minority examples by interpolating between a positive $x \in X^{+}_{\text{train}}$ and a k nearest positive neighbour in feature space [25]. With neighbour $x^{(j)} \in N_k(x)$ and $\lambda \sim U(0,1)$, we generate a synthetic point as in Eq. (16),

$$x_{\text{new}} = x + \lambda \left(x^{(j)} - x\right). \tag{16}$$

If there are too few positives to form a valid neighbourhood, we set $k_{\text{eff}} = \min\big(k, \max(1, |X^{+}_{\text{train}}| - 1)\big)$, and if $|X^{+}_{\text{train}}| < 2$ we fall back to ROS. We enforce the limit: $n_{\text{add}} \leq N_{\text{const}} - |X|$ (Eq. 15). When the number of positives is too small to form valid neighbourhoods, the effective k is reduced.

III. Experiments

Experiments address our research questions (Section I), via evaluating GAN versus evolution-based synthetic data augmentation for malware detection given scarce positive samples. We also test classifier efficacy when distributions shift over time and across malware families. We evaluate the: real only, ROS, SMOTE, WGAN-GP, EVO, and EvoGAN methods under identical dataset splits, shared ratio capped budget and nearest neighbour filtering (Section II, Table I).

We execute three experiment regimes. First, IID scarcity calibration, sub-sampling malware positives in the training set to small fractions $f \in \{0.0005, 0.001, 0.0015, \ldots, 0.01\}$. We enforce class count floors to avoid degenerate fits (at least 50 positives, 50 negatives per scarce training subset). Second, temporal holdout, where we train strictly before a cut-off date and evaluate on future samples in fixed rolling windows to expose drift while simulating scarcity. Third, a *leave one family out* regime, holding out one malware family at a time to test cross family generalisation. Across regimes we compare the same training conditions under identical sampling and validation: real only, ROS, SMOTE, WGAN GP with nearest neighbour filtering, EVO, and EvoGAN. We apply the same overall pipeline across multiple Windows PE feature datasets, namely BODMAS, EMBER, and SOREL [6], [7], [27].

A. Datasets

All datasets are represented as fixed length numeric feature vectors extracted from Windows Portable Executable files [6], [7], [27]. No binaries are executed and no dynamic traces are collected. Each sample has a label $y \in \{0, 1\}$ where malware is 1 and benign is 0. For BODMAS, we use the static PE feature vectors and associated metadata [27]. Each sample is a tabular numeric vector with $d = 2381$ dimensions. Per sample metadata includes a timestamp and a malware family label. Blank or missing family labels are treated as benign for the purposes of family based splitting. For EMBER, we use the released feature vectors and labels and retain the original fixed length representation [6]. We use the labelled subset for training and evaluation. For SOREL, we follow the same integration approach as for EMBER. We load the provided feature representation and labels and convert them into the same NPZ plus metadata interface used for BODMAS and EMBER [7]. Where timestamps and other metadata are available, we retain them in the metadata file so that temporal evaluation can be run in the same way as on BODMAS.

B. Data Preparation

We standardize dataset access by converting each dataset into a common on-disk format. Each dataset is stored as an `.npz` archive containing the feature matrix $X \in \mathbb{R}^{n \times d}$ and label vector $y \in \{0, 1\}^n$, plus a metadata file that stores any available timestamp and family information, and any fixed split indices when provided. Where required, we also store a dataset identifier per row so that split indices and windowing operations remain reproducible after conversion.

a) BODMAS: We load the feature vectors and metadata and store them in the common format. Timestamps are parsed and normalised to UTC and retained for temporal splitting. Family labels are retained for leave-one-family-out splitting; blank or missing family entries are normalised to benign.

b) EMBER: We load the released feature vectors and labels from the labelled subset and retain the original fixed-length representation. We apply pre-processing for compatibility with the shared pipeline to ensure consistent dtypes and

array shapes, remove or impute non-finite values if present, and store the resulting X and y in the common format. We preserve train-test partitioning in the dataset with split indices, applying scarcity sub-sampling within the training partition.

c) SOREL: We load the provided feature representation and labels and convert them into the same `.npz` plus metadata interface used for BODMAS and EMBER. We store X and y in `.npz` and retain meta-data fields needed for temporal or family evaluation (e.g., timestamps and family identifiers) when provided by the dataset. If the dataset provides an official split, we preserve it by materialising split indices. Otherwise, we construct splits using the same rules as the BODMAS regimes when the required metadata is available.

C. Sampling and Data Split

a) IID scarcity (calibration): We quantify augmentation under pure scarcity using a fixed split into training and test data. We then sub-sample the training data to fractions $f \in \{0.0005, 0.001, 0.0015, \ldots, 0.01\}$ while enforcing floors of at least 50 positives and 50 negatives. The held out test set is identical across methods, so differences in ROC AUC, PR AUC, Balanced Accuracy, F1, and MCC reflect training set composition only. Each configuration is repeated over seeds $\{42, 1337, 2025\}$ and we report mean and standard deviation.

b) Temporal holdout and rolling windows: To evaluate robustness under temporal drift, we construct month anchored temporal splits using a cut-off timestamp c. Training uses samples with timestamps $t_i < c$ and testing uses samples with timestamps $t_i \geq c$. We use split indices (JSON) and verify there is no leakage by checking that the latest training timestamp is earlier than the earliest test timestamp. Per cut-off, we evaluate the fixed test partition in rolling windows (length W, stride S), where W=60 days, S=30 days, and report the first N=6 windows per cut-off. Augmentation is generated once per cut-off from the training partition (not regenerated per window). The model and decision threshold are fixed per cut-off and reused for all windows per cut-off.

c) Leave one family out: To evaluate cross family generalisation, we hold out one malware family at a time. For family f, training uses all samples whose family is not f, while testing uses the samples whose family is f. Benign samples are used for training and testing. We simulate scarcity by sub-sampling the training partition to the same fractions as in the IID regime, with the same class count floors. Operating thresholds are selected on a small training only validation split and then applied to the held out family test set.

D. Classifier and Validation Protocol

All methods use the same downstream classifier, a Random Forest trained on the raw tabular features. Hyper-parameters are fixed across dataset regimes and are reported in Table I. Metrics with thresholds are chosen on a small stratified validation split carved from the training partition for the current split or cut-off. The selection criterion is Balanced Accuracy. For temporal experiments, the threshold selected at a cut-off is reused for all rolling windows from that cut-off.

E. Evaluation Metrics, Aggregation, Reproducibility

We report ROC AUC and PR AUC as threshold free metrics, and thresholded metrics including Balanced Accuracy, F1, and MCC, since the datasets are class imbalanced and no single metric fully captures performance under this setting. ROC AUC and PR AUC summarize ranking quality independently of a fixed operating threshold, while Balanced Accuracy, F1, and MCC provide complementary views of performance once a decision threshold is chosen. For these metrics, we compute predictions using a fixed threshold selected on the training only validation split. For each configuration we run multiple seeds: $\{42, 1337, 2025\}$. We report mean and standard deviation across seeds. For temporal evaluation, we report per window trajectories. If a single scalar is needed we aggregate across windows using macro averaging of seed means.

IV. Results and Discussion

We report results for three evaluation regimes (Table II), given BODMAS, EMBER, and SOREL datasets. In all experiments, the held-out test slices and RF/validation protocol are fixed (Table I). Task performance differences reflect *training-set composition* (scarcity versus augmentation). We report means over seeds $\{42, 1337, 2025\}$ with ± 1 SD bands, and treat ROC AUC as the primary metric.

A. Random Scarcity (IID)

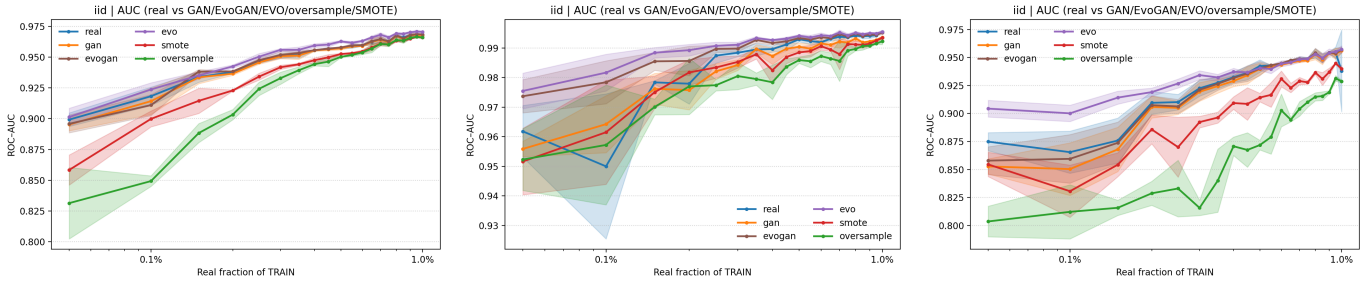
On IID splits, we vary the fraction f of real training data and compare Real-only, ROS, SMOTE, GAN, EVO and EvoGAN across all three datasets. Figure 1 shows AUC versus f (log-scaled x), aggregated over seeds with the test set fixed for all methods at each f. Across all datasets, AUC increases monotonically with f. At the smallest fractions, EVO consistently improves over the real-only baseline, and EvoGAN is often also above baseline but less reliably (e.g. SOREL). GAN closely follows the real-only baseline while SMOTE and ROS tends to lag. As f grows, variability across seeds shrinks, the curves converge, with all methods within a narrow AUC band at the largest f. Thus, in this calibration (no distribution shift), generative augmentation is most useful under extreme scarcity, with EvoGAN/EVO giving the clearest gains at the smallest f. The gains decrease as real data becomes available.

B. Temporal Holdout with Rolling Windows

We evaluate temporal robustness by fixing a cut-off c, training on $t < c$, and testing in rolling windows on $t \geq c$ (Section III). Windowed analyses use f=0.0005 (extreme scarcity) and report ROC AUC averaged over seeds with ± 1 SD bands. For the aggregated trend (Figure 2), AUC is averaged over windows in each (prefix, seed) and then averaged across prefixes and seeds. For EMBER, temporal experiments report only four fractions whereas BODMAS and SOREL use full fraction sweeps. Figure 2 compresses the rolling-window evaluation into one curve (per method). AUC increases with more real training data, and ordering is stable. EVO is consistently best for BODMAS and SOREL, but becomes the best for EMBER at high f. EvoGAN and

TABLE II
STANDARDISED SPLITS, METHODS, AND SHARED CONTROLS USED IN ALL STUDIES.

ID	Regime	Train and test construction	Methods	Budget and filtering (shared)	Objective
E1	IID scarcity	75 and 25 stratified split (seed 42). Fractions $f \in \{0.0005, 0.001, \ldots, 0.01\}$. Fixed test set.	Real only, ROS, SMOTE, GAN, EVO, EvoGAN.	Ratio cap $n_{\text{syn}} \leq \alpha n_{\text{real,pos}}$ with $\alpha = 40$. Candidate oversampling factor $q_m = 5$. Like set uses positives from full TRAIN. Default gate is nn. Global train size cap $N_{\text{cap}} = 20000$.	Calibration.
E2	Temporal holdout and windows	Cut-offs c. Train $t < c$. Test $t \geq c$ in rolling windows with width $W = 60$d and stride $S = 30$d. First $N = 6$ windows. Fractions f as in E1. Model and threshold fixed per cut-off.	Same as E1.	Same as E1. Windowing slices test only. Augmentation generated once per cut-off from TRAIN.	RQ1: temporal drift.
E3	Family LOFO	For each family f: Train malware not equal to f with benign included. Test is family f with benign included. Fractions f as E1 (no family overlap).	Same as E1.	Same as E1. Like set uses positives from LOFO TRAIN. EvoGAN LOFO ablation widens the EA search as in Table I.	RQ2: cross family.



(a) EMBER (b) BODMAS (c) SOREL

Fig. 1. IID AUC versus real fraction. Augmentation helps most when positives are extremely scarce and converges to the real-only baseline as data grows.

GAN track the real-only baseline, or are slightly below it. ROS is typically the lowest, followed by SMOTE (except for BODMAS). The gaps are largest at the smallest fractions and narrow as $f \to 1\%$, meaning augmentation helps most when positives are scarce. However, gaps are larger than in the IID regime, indicating greater variability in temporal experiments.

C. Family Generalisation (LOFO)

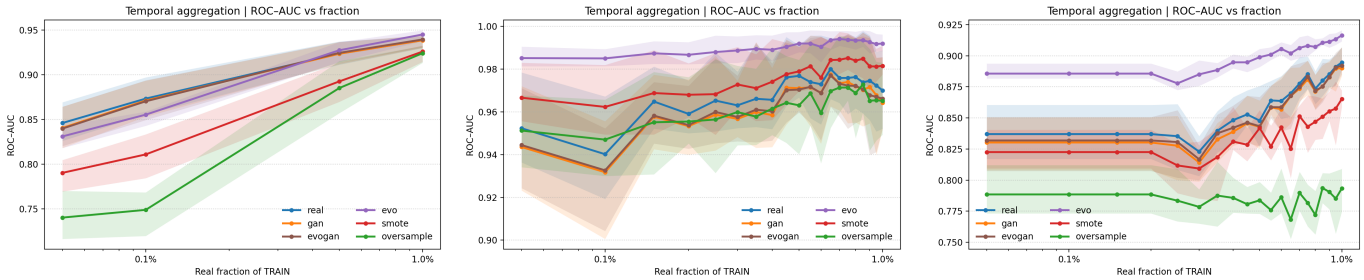
Per malware family f, all f-labelled malware is excluded from TRAIN (Table I) and reserved for testing, while benign remains available on both sides. We vary real-data fraction f used to create the scarce TRAIN subset and compare Real-only, ROS, SMOTE, GAN, EVO, and EvoGAN under the shared budget, filtering and validation protocol (Section III). Results are aggregated across families and seeds. Under family holdout, EVO again attains the strongest mean performance across most fractions, indicating reliable generalisation to unseen malware families. Variability is greatest at the smallest fractions and decreases as more real data becomes available.

To visualise variability at f=0.0005 across all datasets, Figure 3 shows the distribution of ROC AUC across evaluation slices pooled over datasets, where each point corresponds to a single (dataset, held-out family, seed) result. EVO's distribution is shifted upward and is tighter (smaller interquartile range), indicating more consistent performance on unseen families. SMOTE and ROS exhibit broader spread with more low outliers, while Real-only, GAN, and EvoGAN have similar central tendency but heavier lower tails than EVO. This complements the aggregate curve by showing the dispersion one can expect across held-out families under extreme scarcity.

Figure 4 plots ROC AUC versus f aggregated over families and seeds (mean with ± 1 SD bands). AUC rises with f for all methods. Across all fractions, except for low f with EMBER, EVO attains the highest mean ROC AUC with comparatively narrow dispersion. GAN and EvoGAN lie close to (but below) the real-only baseline. For BODMAS, SMOTE and ROS are between EVO and real-only and approach EVO as f increases. Conversely, for SOREL and EMBER, SMOTE and ROS remain the lowest. The variability is largest at extreme scarcity and shrinks as more real data becomes available.

D. Discussion

Across all three datasets and all evaluation regimes, EVO is the only augmentation method that consistently improves over the real-only baseline under the hardest settings. In the IID calibration regime, most methods converge as f increases, indicating diminishing returns once sufficient real positives are available. Under distribution shift (temporal windows) and generalisation stress (family LOFO), the separation between methods becomes clearer. EVO reliably transfers, while GAN and EvoGAN typically track the baseline and do not deliver the same shift-robust gains. Non-generative baselines behave



(a) EMBER (b) BODMAS (c) SOREL

Fig. 2. Temporal aggregation. Under time shift, EVO reliably transfers, yielding consistent gains at tiny fractions; gaps shrink with more real data.

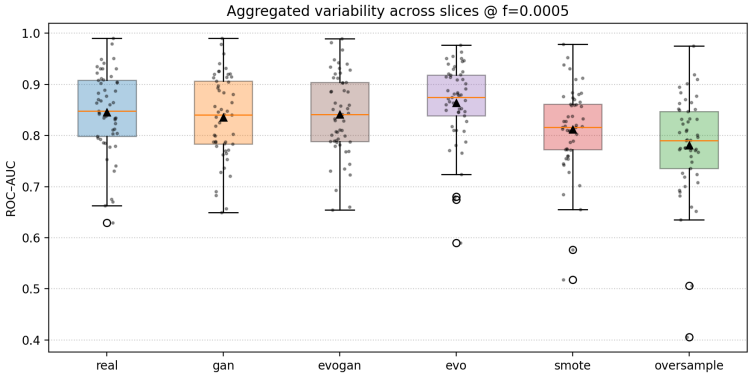
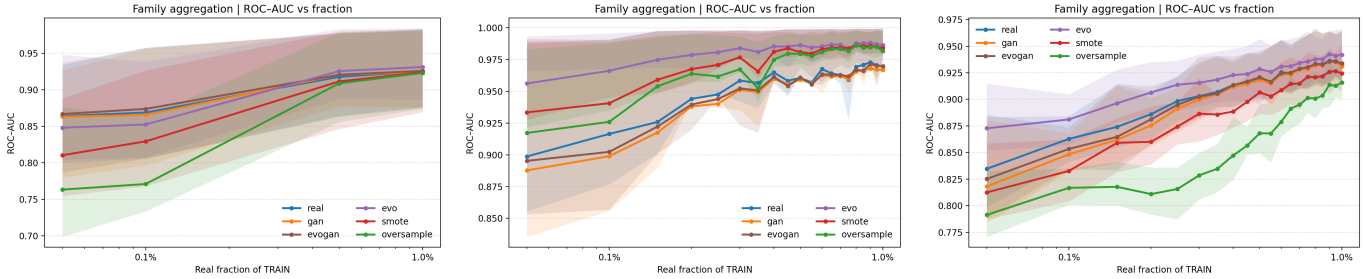


Fig. 3. AUC distribution across family×seed (f=0.0005). EVO's distribution is higher and tighter, showing more reliable performance on unseen families.



(a) EMBER (b) BODMAS (c) SOREL

Fig. 4. Family LOFO (aggregated). EVO attains the highest mean AUC across fractions, indicating better cross-family generalisation.

as expected. SMOTE effects are smaller and less stable, while ROS becomes detrimental as f grows.

Observed ordering is consistent with how each method modifies minority classes under scarcity. EVO performs local, scale-aware perturbations on real positives from TRAIN via convex mixing and Gaussian mutation (Table I), and filters candidates using the shared nearest-neighbour quality gate. This combination increases diversity, remaining close to the empirical malware manifold. This is crucial when $|X^+|$ is extremely small and test data are shifted across time or families. However, GAN-based sampling at tiny f is vulnerable to poor mode coverage, with few positives, the generator underrepresents rare but decision-relevant structures and subsequent filtering removes much of what is generated. This is why GAN and EvoGAN remain close to, or below, the real-only baseline under temporal shift and LOFO, even when appearing competitive in IID calibration. Aggregate curves also hide substantial heterogeneity across evaluation slices. In the temporal regime, windowed performance varies by cut-off and window index, with the hardest early post-cut-off windows exhibiting the largest gaps between methods. Gains are therefore not uniform across time. In the LOFO regime, per-family effects are heterogeneous. Some held-out families show clear improvements while others remain near zero, indicating that the benefit of augmentation depends on how much structure the unseen family shares with families

observed during training. Across regimes, improvements are most pronounced in the most challenging slices (extreme scarcity, strong shift), which is consistent with augmentation acting primarily as a support-expansion mechanism when the baseline training set is least representative.

The methods also differ in computational cost. ROS and SMOTE are the cheapest baselines, since they operate directly on the training data without fitting a generative model. EVO is more expensive than these simple resampling methods because it generates and filters candidate samples, but is substantially lighter than GAN-based approaches, which require iterative neural network training before sampling. EvoGAN is the most computationally demanding because it combines GAN training with an additional evolutionary refinement stage. Taken together with the performance results, this suggests that EVO offers the best trade-off between augmentation benefit and computational overhead under severe scarcity.

Overall, our results highlight that consistent augmentation matters most when the baseline training set is least representative, and and that under these conditions EVO is the most reliable method (considered in this study).

V. Conclusion

This study posited that synthetic data augmentation improves static Windows PE malware detection when labelled positives are scarce, testing if performance (classification) gains persist under distribution shifts. We evaluated various augmentation methods for the BODMAS, EMBER, and SOREL datasets. Our key finding is that feature-space evolution (EVO) is the most reliable method at extreme data scarcity. EVO is comparably stable as the real-only baseline once the fraction leaves the extreme-scarcity regime and is the most stable method under temporal shift. This supports the hypothesis that EVO yields the highest task performance (on average), and is more predictable across experimental regimes. Thus, when positives are extremely scarce, especially under temporal drift or unseen-family evaluation, EVO is the best performing method. However, as the real-data fraction increases toward one percent, augmentation benefits diminish and real-only training becomes sufficient across regimes.

References

[1] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, pp. 1–22, 2020.

[2] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2008.

[3] H. Menéndez, D. Clark, and E. Barr, "Getting ahead of the arms race: Hothousing the coevolution of virustotal with a packer," *Entropy*, vol. 23, no. 395, pp. 1–42, 2021.

[4] T. Abdullahi, G. Nitschke, and N. Sweijd, "Predicting Diarrhoea Outbreaks with Climate Change," *PLoS ONE*, vol. 17, no. 4, p. e0262008, 2022.

[5] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of android malware detection with deep neural models," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 1–36, Dec. 2020.

[6] H. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," *arXiv:1804.04637*, pp. 1–8, 2018.

[7] R. Harang and E. Rudd, "SOREL-20M: A large scale benchmark dataset for malicious PE detection," *arXiv:2012.07634*, pp. 1–7, 2020.

[8] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.

[9] R. Jordaney, K. Sharad, S. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *26th USENIX Security Symposium*. Vancouver, BC: USENIX, 2017, pp. 625–642.

[10] J. Fraley and M. Figueroa, "Polymorphic malware detection using topological feature extraction with data mining," in *SoutheastCon 2016*. New York, NY, USA: IEEE, 2016, pp. 1–7.

[11] D. Bruschi, L. Martignoni, and M. Monga, "Code normalization for self-mutating malware," *IEEE Security&Privacy*, vol. 5, pp. 46–54, 2007.

[12] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *28th USENIX Security Symposium*. USENIX, 2019, pp. 729–746.

[13] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 60, pp. 1–48, 2019.

[14] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: Evaluating android anti-malware against transformation attacks," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013, pp. 329–334.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[16] M. Alotaibi, "A multifaceted deep generative adversarial networks model for mobile malware detection," *Applied Sciences*, vol. 12, no. 19, pp. 1–12, 2022.

[17] M. Amin, B. Shah, A. Sharif, T. Ali, K.-I. Kim, and S. Anwar, "Android malware detection through generative adversarial networks," *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 2, pp. 1–29, 2022.

[18] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Information Sciences*, vol. 460-461, pp. 83–102, 2018.

[19] R. Burks, K. Islam, Y. Lu, and J. Li, "Data augmentation with generative models for improved malware detection: A comparative study," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference*. IEEE, 2019, pp. 660–665.

[20] K. Babaagba, Z. Tan, and E. Hart, "Improving classification of metamorphic malware by augmenting training data with a diverse set of evolved mutant samples," in *IEEE Congress on Evolutionary Computation*, 2020.

[21] Y. Kossale, M. Airaj, and A. Darouichi, "Mode collapse in generative adversarial networks: An overview," in *2022 8th International Conference on Optimization and Applications (ICOA)*, 2022, pp. 1–6.

[22] C. Little, M. Elliot, R. Allmendinger, and S. Samani, "Generative adversarial networks for synthetic data generation: a comparative study," *arXiv:2112.01925*, 2021.

[23] S. Sen, E. Aydogan, and A. Aysan, "Coevolution of mobile malware and anti-malware," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2563–2574, 2018.

[24] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[25] N. Chawla, K. Bowyer, L. Hall, and P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[26] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, vol. 70, 2017, pp. 214–223.

[27] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "BODMAS: An open dataset for learning based temporal analysis of pe malware," in *Deep Learning and Security Workshop*, 2021, pp. 78–84.

[28] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of Wasserstein GANs," in *Advances in Neural Information Processing Systems*, 2017.

[29] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2015.