# MalCL: Leveraging GAN-Based Generative Replay to Combat Catastrophic Forgetting in Malware Classification

**Jimin Park[1], AHyun Ji[1], Minji Park[1], Mohammad Saidur Rahman[2], Se Eun Oh[1]***

[1]Ewha Womans University
[2]University of Texas at El Paso
242aig12@ewha.ac.kr, nolli77@ewha.ac.kr, 02mjpark@ewha.ac.kr, msrahman3@utep.edu, seoh@ewha.ac.kr

## Abstract

Continual Learning (CL) for malware classification tackles the rapidly evolving nature of malware threats and the frequent emergence of new types. Generative Replay (GR)-based CL systems utilize a generative model to produce synthetic versions of past data, which are then combined with new data to retrain the primary model. Traditional machine learning techniques in this domain often struggle with catastrophic forgetting, where a model's performance on old data degrades over time.

In this paper, we introduce a GR-based CL system that employs Generative Adversarial Networks (GANs) with feature matching loss to generate high-quality malware samples. Additionally, we implement innovative selection schemes for replay samples based on the model's hidden representations.

Our comprehensive evaluation across Windows and Android malware datasets in a class-incremental learning scenario – where new classes are introduced continuously over multiple tasks – demonstrates substantial performance improvements over previous methods. For example, our system achieves an average accuracy of 55% on Windows malware samples, significantly outperforming other GR-based models by 28%. This study provides practical insights for advancing GR-based malware classification systems.

**Source Code —**
https://github.com/MalwareReplayGAN/MalCL

## Introduction

Machine learning (ML) has become essential for protecting computers and networked systems, with successful applications in malware detection and classification across domains like Windows and Android (Arp et al. 2014; Dahl et al. 2013; Raff et al. 2021; Kovacs 2018; Maiorca, Giacinto, and Corona 2012). Researchers have developed advanced models that distinguish malware from benign software, classify malware families, and categorize malware types by analyzing features from both benign and malicious source codes.

Malware evolves rapidly, requiring frequent retraining on large datasets to stay effective. Daily, the AV-TEST Institute logs 450,000 new malware and "Potentially Unwanted Applications (PUA)" (AV-TEST 2024), and VirusTotal processes over one million software submissions (VirusTotal 2024), posing significant challenges for antivirus companies. In response, vendors face tough choices: remove old samples from training sets, risking the resurgence of older malware; overlook new samples, missing emerging malware trends; reduce training frequency, sacrificing accuracy; or invest heavily in continuous retraining.

To address these challenges, the approach suggested by Raff et al. (Raff et al. 2021) provides a viable method to enhance memory efficiency in training malware classification models. However, this method does not tackle the inherent tendency of ML models to forget previously learned malware feature distributions when they are continuously trained with new data – a necessity driven by the dynamic and incremental nature of the malware domain.

Continual Learning (CL) offers an effective solution to this challenge, commonly referred to as Catastrophic Forgetting (CF). CL continually adapts to a data stream, refining knowledge over time through techniques such as *replay* (i.e., storing and reusing data) and *regularization* (i.e., regularizing learned representation). These strategies help reduce the storage and computational burdens associated with frequent retraining (Rahman, Coull, and Wright 2022; Channappayya, Tamma et al. 2024).

In particular, among replay techniques (Cao et al. 2024; van de Ven, Siegelmann, and Tolias 2020; Rebuffi et al. 2017), Generative Replay (GR) (Shin et al. 2017), which utilizes generative deep neural networks to produce synthetic samples that represent past data distributions, is advantageous since GR approach eliminates the need to store past raw data, offering a significant benefit in scenarios where data storage is constrained due to privacy regulations.

While CL techniques are well-established in computer vision (CV) (Hsu et al. 2018; van de Ven, Siegelmann, and Tolias 2020), their application in malware classification remains underdeveloped. Rahman, Coull, and Wright (2022) demonstrated that CL methods tailored for CV often perform poorly in malware detection due to the complex and diverse nature of malware features (Rahman, Coull, and Wright 2022). They found that directly applying CV-specific CL systems to malware classification without considering the complexities of malware data distribution leads to poor detection of known malware classes, performing no bet-

*corresponding author

ter than the "*None*" baseline, which involves retraining the model with only new data.

In this paper, we introduce MalCL, a novel malware family classification system that utilizes a GR-based CL approach, incorporating a Generative Adversarial Network (GAN) architecture for its replay methodology (Goodfellow et al. 2020) and a one-dimensional Convolutional Neural Network (CNN) for classifying malware families. We have implemented feature matching in the design of the loss function for the GAN's generator to enhance the quality of generated samples. Additionally, we developed several replay sample selection techniques to select the most effective samples for each malware class.

MalCL is designed to accurately classify malware families using feature vectors in a class-incremental learning scenario, as outlined in prior research (van de Ven, Tuytelaars, and Tolias 2022), addressing multiple continual tasks with each introducing new malware classes (or families) and effectively classifying both past and newly encountered malware families. The primary goal of this system is to mitigate CF of previously observed malware classes while efficiently adapting to new malware families.

We summarize our key contributions as follows:

- **Malware Domain-Specific CL Model:** We introduce MalCL, a novel CL model tailored specifically for malware family classification. It achieves an average accuracy of 55% in correctly identifying 100 malware families across 11 CL tasks, each introducing new malware families. This performance surpasses existing approaches such as Generative Replay (GR) (Shin et al. 2017) and Brain-Inspired Replay (BI-R) (van de Ven, Siegelmann, and Tolias 2020) by 28%. Our model incorporates robust architectures designed for precise malware classification.
- **Improved Replay with Feature Matching:** MalCL effectively replays past malware samples using a GAN's generator. We have enhanced this capability by incorporating Feature Matching Loss (FML), which reduces the discrepancy between the richer features of original malware samples – extracted from the hidden layers of the GAN's discriminator – and the synthetic malware samples created by the GAN's generator.
- **Replay Sample Selection Schemes:** We have developed innovative replay sample selection schemes for MalCL, particularly focusing on features derived from the intermediate layer of the classifier. This approach improves the alignment between generated malware samples and original data, thereby improving classification accuracy.
- **Strategic Task Set Construction:** We have explored various task set construction strategies, especially assigning larger classes to initial tasks, which has optimized MalCL's performance and showcased a strategic approach to effectively mitigate CF in malware classification domain.

## Threat Model

A schematic representation of the threat model, where an attacker reuses legacy malwares to exploit vulnerabilities in machine learning (ML) systems, is shown in Figure 1. ① in Figure 1 represents the initial phase, where the system is trained on available data from the repository and deployed. When new samples are observed, the system is retrained with these samples, as depicted by ②, and the new samples are added to the data repository. However, due to catastrophic forgetting (CF), the system's ability to detect earlier threats diminishes over time.

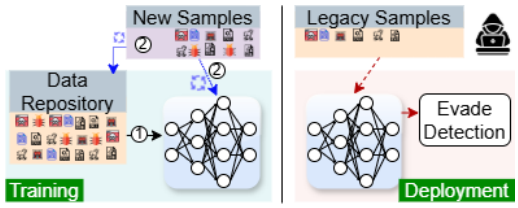



Figure 1: An attacker reuses legacy malware to evade systems updated with *only* new malwares.

This limitation allows attackers to reuse older malware and malicious code snippets, which can bypass the updated system. The likelihood of successful evasion increases as the temporal gap between the original training and the attack widens. Therefore, addressing this challenge is critical to developing defense systems that maintain consistent detection capabilities for both recent and legacy malware.

## Related Work

The fundamental challenge in developing a continual learning (CL) system is addressing *catastrophic forgetting* (CF). CF is a phenomena in which the performance of a neural network degrades significantly on older tasks after being trained on a new task (French 1999; McCloskey and Cohen 1989). Over the years, several mechanisms have been proposed to overcome CF (Zenke, Poole, and Ganguli 2017; van de Ven, Siegelmann, and Tolias 2020; Rebuffi et al. 2017), which fall into two major categories (Parisi et al. 2019): replay methods and regularization methods. Replay techniques supplement the training data for each new task with representative data from previously learned tasks.

**Replay.** Replay techniques can be further classified into major two subcategories: exact replay and generative replay. Exact replay methods, such as Experience Replay (ER) (Rolnick et al. 2019) and Incremental Classifier and Representation Learning (iCaRL) (Rebuffi et al. 2017), allocate a memory budget $\mathcal{M}$ to replay samples from previous tasks, combining them with new data during retraining to optimize performance. Generative replay approaches, such as Generative Replay (GR) (Shin et al. 2017), Brain-Inspired Replay (BI-R) (van de Ven, Siegelmann, and Tolias 2020), diffusion-based generative replay (DDGR) (Gao and Liu 2023) generate representation of the stored samples with a secondary generative models to generate previous data distributions, making them useful when access to historical data is limited.

**Regularization.** Regularization-based methods, such as Elastic Weight Consolidation (EWC)(Kirkpatrick et al. 2017) and Synaptic Intelligence (SI)(Zenke, Poole, and Gan-

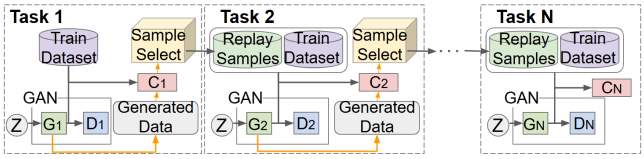



Figure 2: MalCL continual learning pipeline for malware classification using N tasks.

guli 2017), work by limiting changes to weights that are crucial for previously learned tasks. These methods introduce an additional loss function, known as regularization loss, which is added to the training loss. The total loss therefore reflects both the new learning and the retention of knowledge from previous tasks. This approach helps strike a balance between acquiring new information and preserving important past learnings, thereby enhancing the system's adaptability.

### CL and Related Approaches for Malware Classification

Few studies have explored the application of CL in malware detection. Rahman, Coull, and Wright (2022) are among the first to investigate CL for malware classification, finding that existing methods fall short in effectively addressing CF. More recently, Chen, Ding, and Wagner (2023) combined contrastive learning with active learning to train Android malware classifiers, focusing on detecting concept drift rather than overcoming CF. Channappayya, Tamma et al. (2024) explored a replay-based CL technique in network intrusion detection, which involves class-balancing reservoir sampling and perturbation assistance, though this domain differs from the challenges faced in malware classification.

Other approaches, such as *online learning*, have been used for malware classification to incorporate new samples as they appear, but they do not directly tackle CF (Xu et al. 2019). Transfer learning has also been studied for malware detection due to its adaptability to evolving threats (Neyshabur, Sedghi, and Zhang 2020), but it often neglects the retention of knowledge from previous tasks, which is crucial to avoid the resurfacing of vulnerabilities (Chai et al. 2022). Considering these limitations, our paper focuses on CL, which is specifically designed to address CF in malware detection.

## MalCL Overview

In this section, we detail our proposed MalCL, which aims to accurately detect malware families based on features extracted from Windows Portable Executable (PE) and Android APK files.

### MalCL Pipeline

A schematic representation of the MalCL pipeline is depicted in Figure 2. MalCL which consists of two primary components: a GAN and a classifier. The GAN is responsible for generating malware samples of the training data from previous tasks to be replayed, while the classifier is tasked with malware family classification, identifying the family to which each feature vector belongs. It is important to note that the feature vectors utilized in this study were constructed based on previous literature (Rahman, Coull, and Wright 2022).

As depicted in Figure 2, throughout each of the $n$ tasks, the GAN is trained with a generator ($G_i$) designed to produce high-quality synthetic malware samples that closely resemble the training data of the $i$-th task. Concurrently, a discriminator ($D_i$) works to differentiate these generated samples from the actual training data. The classifier model ($C_i$) is trained using the training data to accurately identify malware families. Once $G_i$ produces sufficiently realistic malware samples, these are fed into $C_i$, which then outputs `logits` from the middle layer of the classifier model.

Subsequently, a replayed sample set is created by selecting the most effective synthetic malware samples using a selection algorithm. We refer to these samples as "replay samples" throughout the paper. Once the replayed set is established, it is carried forward to the next task, where it is used alongside the new training data to train both the GAN and the classifier models.

### Overview of the Models

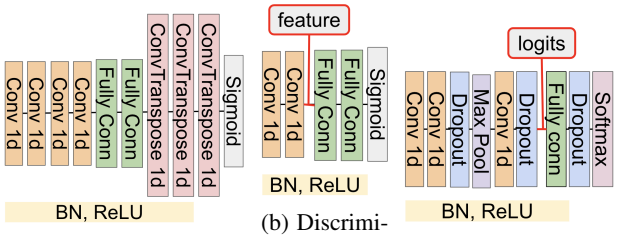
In this section, we detail the architectures of the GAN and classifier models chosen for our study.

**GANs.** As shown in Figure 3a, the generator comprises four 1D CNN layers, two Fully-Connected (FC) layers, and three deconvolution layers, with ReLU activation and batch normalization applied to all but the final deconvolution layer. The discriminator (Figure 3b) includes two convolutional layers followed by two FC layers, with ReLU activation and batch normalization in all but the final FC layer. Flattened logits from the second convolutional layer are used for feature matching to analyze differences between fake and original malware samples. A sigmoid layer serves as the output for both the generator and discriminator. Optimization is performed using Adam with a batch size of 256.

**Classifier.** The classifier model in Figure 3c consists of three convolutional layers followed by a FC layer. Max pooling and dropout follow the first two convolutional layers, while dropout is applied after the third convolutional layer and the FC layer. The final output is a softmax layer. Flattened logits from the third convolutional layer are used for relay sample selection. The model is optimized using Stochastic Gradient Descent (SGD) with a learning rate of $e^{-3}$, momentum of 0.9, and weight decay of $e^{-6}$, with a batch size of 256.

### Replay Sample Selection Schemes

To maximize the effectiveness of continual learning in MalCL, selecting high-quality synthetic malware samples is crucial. A robust selection metric must pinpoint samples that closely resemble those in the original malware training set. Additionally, the generator's tendency to produce imbalanced samples per class can lead to model collapse, with only a few classes represented by synthetic samples. To address these challenges, we propose several replay sample selection methods in this section.




(a) Generator (b) Discriminator (c) Classifier

Figure 3: GAN architecture: (a) Generator and (b) Discriminator, adapted for generating replay malware samples, and (c) Classifier, optimized for malware family classification in continual learning tasks.

Let $F(s)$ represent a transformation applied to a synthetic sample $s$, where $s \in S$ and $S \subseteq \mathbb{R}^l$, with $l$ denoting the dimension of the generator's output layer. $H(x)$ represents a similar transformation applied to an original sample $(x, y)$, where $x \in X$, $y \in Y$, $X \subseteq \mathbb{R}^m$, and $Y \subseteq \{0,1\}^n$, representing the training set $M$. The feature vector length is $m$, and $n$ is the total number of malware classes. The distance $\delta$ between these transformations is calculated as:

$$\delta(F(s), H(x)) \tag{1}$$

To implement the $k$-Nearest Neighbor ($k$-NN) approach, we select the $k$ synthetic samples in $S$ with the lowest distances to the original samples $x$ from each class $c$, represented as:

$$S_{c,k} = \{s \mid \arg\min_{s} \delta(F(s), H(x)), \forall s \in S, \forall x \in X_c\} \tag{2}$$

where $X_c$ includes the original samples from malware class $c$, and $\delta$ denotes a distance metric, such as L1 or L2.

In addition to the $k$-NN approach, we also propose a global selection across $n$ classes, aimed at selecting samples that *generally* resemble the overall malware data distribution. This approach is formulated as:

$$S_{km} = \{s \mid \arg\min_{s} \delta(F(s), H(x)), \forall s \in S, \forall x \in X\} \tag{3}$$

where $m$ is the number of batches.

Note that $S_{km}$ does not explicitly enforce a balanced selection per class as $S_{c,k}$ does, which may result in zero samples for certain classes. Thus, this methodology has the limitation that the selection scheme may not fully overcome the mode collapse problem of the generator. However, we explore this scheme to demonstrate the impact of the generator's ability to tune toward producing samples that look good overall, rather than targeting each class specifically, even though it produces synthetic samples for only a subset of $n$ classes.

These strategies construct an effective replay set and ensures class balance, addressing the generator's tendency to produce imbalanced samples. This section outlines selection algorithm adjustments to enhance replay sample effectiveness, compensating for training data loss in previous tasks. We modify transformations $F$, $H$, and metric $\delta$ to achieve this.

**Selection with L2 Distance to One-Hot Labels.** In this setup, we use L2 distance ($\delta$) with one-hot-encoded labels. When a synthetic sample $s$ is input into the classifier for the $i$-th task, the outputs from the final softmax layer are:

$$F(s) = C_i(s), \quad H(x) = y \tag{4}$$

where $C_i(s)$ is the classifier's softmax output, and $y$ is the true label vector.

The selection process computes the Euclidean distance between $C_i(s)$ and $y$, selecting the top $k$ synthetic samples for each class $c$ based on:

$$S_{c,k} = \{s \mid \arg\min_{s} \sqrt{(C_i(s) - y_c)^2}, \forall s \in S, y_c \in Y\} \tag{5}$$

where $y_c$ denotes the label vector for class $c$. These selected samples are then used to train the next classifier model, $C_{i+1}$.

**Selection with L1 Distance to Logits.** In this setup, logits $\mathcal{L}_i(x)$ and $\mathcal{L}_i(s)$ are extracted from the middle layer of the $i$-th classifier $C_i$ when $x$ (from the $i$-th task training set $X_i$) and $s$ are input. The selection process computes the L1 distance ($\delta$) between these logits and selects either the top $k$ samples with the shortest distances for each class $c$ or the top $km$ samples across $n$ classes using $m$ batches. These samples are then used to train the subsequent classifier, $C_{i+1}$.

We explored two logit representations for $H$:

1. **Per-Batch Mean Logit Vector**: The mean logit vector $\bar{\mathcal{L}}_{i,b}(x)$ for batch $b$ with size $n_b$ is given by:

$$\bar{\mathcal{L}}_{i,b}(x) = \frac{1}{n_b} \sum_{x \in X_b} \mathcal{L}_i(x) \tag{6}$$

Then, $F(s)$ and $H(x)$ are defined as:

$$F(s) = \mathcal{L}_i(s), \quad H(x) = \bar{\mathcal{L}}_{i,b}(x) \tag{7}$$

The top $km$ samples across classes are selected by minimizing:

$$S_{km} = \{s \mid \arg\min_{s} |\mathcal{L}_i(s) - \bar{\mathcal{L}}_{i,b}(x_j)|, \forall s \in S, \forall x_j \in X_b\} \tag{8}$$

where $m$ is the number of batches, and $X_b$ is the batch, with $X_b \in X_i$. Finally, each $s$ is labeled with the class label outputted by $C_i(s)$. This scheme selects the synthetic samples closer to the global mean, that is the centroids of each batch, and adopts the natural decision of the classifier for labeling.

2. **Per-Class Mean Logit Vector**: The mean logit vector $\bar{\mathcal{L}}_{i,c}(x)$ for class $c$ with size $n_c$ is:

$$\bar{\mathcal{L}}_{i,c}(x) = \frac{1}{n_c} \sum_{x \in X_c} \mathcal{L}_i(x) \tag{9}$$

Then, $F(s)$ and $H(x)$ are:

$$F(s) = \mathcal{L}_i(s), \quad H(x) = \bar{\mathcal{L}}_{i,c}(x) \tag{10}$$

Algorithm 1: Training a GAN in MalCL

Initialize $G$ and $D$ networks with random weights
Define batch size $n$, learning rate $\eta$, and number of epochs $E$
**for** $epoch = 1$ to $E$ **do**
  **for** each batch $B$ in the training dataset **do**
    Sample $m$ noise samples $\{z_1, ..., z_m\}$ from noise prior $p_z(z)$.
    Generate synthetic malware samples $\{x_1, ..., x_m\}$ where $x_i = G(z_i)$.
    Sample $m$ real malware instances $\{x'_1, ..., x'_m\}$ from malware data distribution $p_{data}(x)$.
    Update $D$ by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x'_i) + \log(1 - D(G(z_i)))]$$

    Sample $\{z_1, ..., z_m\}$ from $p_z(z)$.
    Update $G$ by descending its stochastic gradient:

$$\nabla_{\theta_g} L_G$$

  **end for**
**end for**

The top $k$ samples for each class $c$ are selected by minimizing:

$$S_{c,k} = \{s \mid \arg\min_{s} |\mathcal{L}_i(s) - \bar{\mathcal{L}}_{i,c}(x_j)|, \quad \forall s \in S, \forall x \in X_c\} \tag{11}$$

Compared to the former, this approach selects replay samples closer to the centroid of each class and labels them according to the closest centroid. With this approach, we achieve $k$ samples for each class, leading to a balanced dataset.

These selection methodologies are aimed at effectively training the classifiers by refining the selection of synthetic samples through sophisticated distance calculations.

## Loss Functions For Generator

To ensure the quality of replay malware samples, selecting an appropriate loss function for the generator is essential. In this section, we discuss two loss functions for the generator. We detail the training process of GAN in Algorithm 1.

**Binary Cross Entropy Loss.** Binary Cross Entropy (BCE) is one of the most widely used loss functions for training the generator in GANs because it effectively quantifies the difference between two data distributions. BCE assesses how successfully the generator has deceived the discriminator into classifying the generated outputs as real rather than synthetic. When employing BCE loss for the generator, the labels for its outputs are flipped to 1, indicating "real." Thus, the generator aims to minimize the following loss:

$$L_G = -\frac{1}{m} \sum_{i=1}^{m} \log(D(G(z_i))) \tag{12}$$

where $m$ denotes the batch size.

**Feature Matching Loss.** During the discriminator's forward pass with real and synthetic malware samples, intermediate features from a middle hidden layer are extracted. A distance metric then calculates the difference between the average features of real and synthetic malware samples. The generator's loss function is defined as:

$$L_G = \frac{1}{m} \sum_{i=1}^{m} \|\mathbb{E}_{x \sim p_{\text{data}}}[D^{(f)}(x)] - \mathbb{E}_{z \sim p_z}[D^{(f)}(G(z))]\| \tag{13}$$

where $D^{(f)}(\cdot)$ denotes the output of a middle layer of the discriminator, $x$ represents real malware samples, $G(z)$ represents synthetic malware samples, and $m$ denotes the batch size.

This loss function, adopted by previous research (Salimans et al. 2016), provides a compelling alternative to BCE loss by refocusing training objectives from the final discriminator output to the richer intermediate feature layers, fostering a more nuanced learning process for the generator.

# Experimental Details

In this section, we detail the datasets and features used in the evaluations, introduce baseline models and those from prior literature (Shin et al. 2017; van de Ven, Siegelmann, and Tolias 2020) for comparison with MalCL, and discuss the experimental settings to demonstrate the effectiveness of MalCL as a malware family detection methodology.

## Datasets

We use two large-scale malware datasets for our experiments: EMBER (Anderson and Roth 2018), which includes Windows PE malware samples, and AZ-Class, a dataset we collected from the AndroZoo repository (Allix et al. 2016) containing Android malware samples. Consistent with prior work (Xu et al. 2019), we selected samples with a VirusTotal detection count $>=$ four.

**EMBER.** We use the 2018 EMBER dataset, known for its challenging classification tasks, focusing on a subset of 337,035 malicious Windows PE files labeled by the top 100 malware families, each with over 400 samples. Features include file size, PE and COFF header details, DLL characteristics, imported and exported functions, and properties like size and entropy, all computed using the feature hashing trick.

**AndroZoo.** The AZ-Class dataset contains 285,582 samples from 100 Android malware families, each with at least 200 samples. We extracted Drebin features (Arp et al. 2014) from the apps, covering eight categories like hardware access, permissions, API calls, and network addresses. Initially, the training set had 1,067,550 features, which were aligned in the test set. We used `scikit-learn's VarianceThreshold` to reduce low-variance features, resulting in a final feature dimension of 2,439.

| Approach | Method | | EMBER | |
|---|---|---|---|---|
| | | | ***Mean*** | ***Min*** |
| Baselines | None | | 27.5 | 0.6 |
| | Joint | | 88.7 | 74.5 |
| Prior Works | GR | | 26.8 | 09.5 |
| | BI-R | | 27.0 | 09.2 |
| MalCL | L2 to Labels | FML | 50.2 | 17.8 |
| | | BCE | 47.8 | 19.4 |
| | L1 to CMean | FML | **54.5** | 21.8 |
| | Logits | BCE | 52.1 | 26.2 |
| | L1 to BMean | FML | 50.3 | 32.0 |
| | Logits | BCE | 47.6 | 24.2 |

Table 1: Comparisons to Baseline and Prior Replay Models Using Ember Dataset. We report the mean accuracy scores (Mean) and minimum (Min) computed from every 11 tasks.

## Baseline and Prior Work

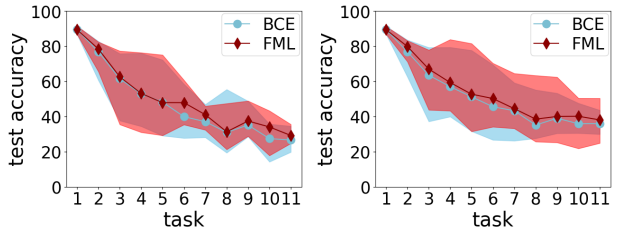
This work uses two baselines: *None* and *Joint*. In the *None* baseline, the classifier is trained only on the new data for each task, simulating catastrophic forgetting (CF) and serving as the informal lower bound. The *Joint* baseline combines data from all previous tasks into a single dataset for training, representing an ideal scenario that preserves all prior information but is impractical for large-scale malware datasets, serving as the informal upper bound.

The baseline results are empirically grounded. A standard scaler is applied in all scenarios, with variations in its usage: it is refitted for each new task in the None and Joint baselines but incrementally updated in CL scenarios. This highlights a key distinction: the None baseline trains exclusively on new data, the Joint baseline integrates all data, and CL scenarios adaptively learn evolving data distributions as new classes are introduced.

We also compare the performance of MalCL with two prior GR-based CL techniques: GR (Shin et al. 2017) and BI-R (van de Ven, Siegelmann, and Tolias 2020), both well-studied in CV domains. GR uses a GAN to GR samples, while BI-R, which employs a Variational Autoencoder (VAE) (Kingma and Welling 2013), enhances GR by adding Conditional Replay (CR), Gating based on Internal Context (Gating), and Internal Replay (IR).

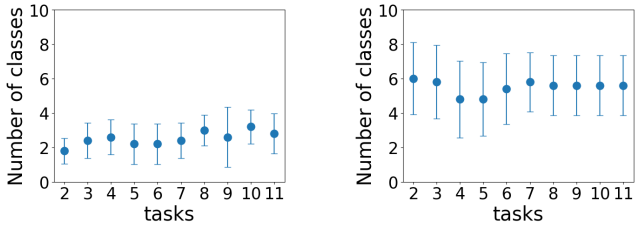
## Task Set Construction and Metrics

We constructed the training set for the first task by randomly selecting 50 classes, adding 5 randomly chosen classes for each subsequent task, resulting in 11 task datasets. Each dataset simulates incremental additions of malware classes to the training set. For the testing set, we used all available classes observed up to each task – for instance, 50 classes for the first task, 55 for the second, and so forth. To evaluate the impact of random selection, we conducted five experiments, reporting the minimum and mean accuracies achieved. Figures 5 and 6 illustrate these accuracies; each line represents mean accuracies, while the shaded areas reflect the range between the maximum and minimum accuracies at each step.




(a) L2 to Labels: 48% vs. 50%. (b) L1 to CMean: 52% vs. 55%.

Figure 4: Impact of three replay sample selections on MalCL performance when using the EMBER Dataset. We further report the mean accuracies for each task when comparing BCE with FML.




(a) BCE (b) FML

Figure 5: The number of replay sample classes per task using L1 to BMean Logits. We omit the first task, as no replay sample is utilized to train the classifier.

# Results

In this section, we discuss the performance of MalCL by comparing it to baseline models and those from prior literature (Shin et al. 2017; van de Ven, Siegelmann, and Tolias 2020), exploring various generator loss functions, replay sample selection methods, and task set generation scenarios.

## Comparisons to Baseline and Prior Work

MalCL demonstrated superior performance compared to the None baseline, GR, and BI-R. In the optimal setting, where FML with L1 distance to Per-Class Mean Logit Vectors (L1 to CMean Logits) was employed, MalCL improved upon the None baseline by 27%. This substantial performance gain indicates that MalCL is significantly more robust in mitigating CF, a critical issue observed in the None baseline. Furthermore, MalCL outperformed existing CL approaches with replay, such as GR and BI-R, by 28%, highlighting that while these methods are effective in image classification tasks, they are less effective in classifying malware families when new malware samples are continually introduced. Our observations also revealed that CL techniques optimized for image classification, such as GR and BI-R, performed similarly to the None baseline, indicating their limited effectiveness in addressing CF in malware classification. This underscores the necessity of designing and optimizing CL methodologies specifically targeted at malware family classification.

However, we also note that the selection scheme based on L1 distance to Per-Batch Mean Logit Vectors (L1 to BMean Logits) demonstrated poor performance, producing

synthetic samples for only a few classes and thus restricting the replay ability of MalCL (Refer to Figure 5). This scheme does not explicitly enforce sample selection per class, leading to the natural inability of the generator to diversify its production to cover all malware families. We plan to work on improving the global selection scheme in the future.

In addition, although there remains a substantial gap between the performance of MalCL and the Joint baseline, there is potential for further improvement, particularly in more accurately emulating original samples and generating even more robust synthetic malware samples. Enhancing these aspects could bring performance closer to that of the Joint baseline, which we identify as a promising direction for future research.

### Generator Loss and Replay Sample Selection

We compare the performance of two selection schemes: L2 to Labels and L1 to CMean Logits, using both BCE and FML. As depicted in Figure 4b, FML slightly outperforms BCE across all tasks.

For both loss functions, MalCL demonstrates increased effectiveness with the L1 to CMean Logits selection scheme. This improvement indicates that synthetic malware samples, which closely resemble the mean vectors derived from the richer features of the classifier's hidden layer, effectively facilitate the replay of training data from prior tasks. This sampling scheme aligns well with the generator model's objective of accurately mimicking original malware samples, as captured by mean logit vectors. Consequently, these replay samples are likely to exhibit lower classification errors when identified as the target class. Overall, FML with L1 to CMean Logits achieves the highest accuracy across all tasks in classifying malware families.

### Task Set Generation Methods

The method of constructing task sets is crucial for the CL capabilities of MalCL. While we have employed random selections in all previously described experiments, we also explored a scenario where larger classes containing more malware samples were assigned to the initial tasks, and smaller classes with fewer samples to subsequent tasks. In this configuration, we fixed the first task to include 50 "giant" classes, each having an average sample count of 5,397, and applied random selection to the remaining classes, which have a mean sample count of 670, to construct the sets for later tasks. Surprisingly, MalCL, with the EMBER dataset, replay sample selection scheme of L1 to CMean Logits and BCE loss, achieved a mean accuracy of 74%, significantly surpassing the previous best result of 55% and closely approaching the performance of the Joint baseline (refer to Table 1). This outcome underscores that assigning large classes with a greater number of samples to the first task can effectively mitigate the impact of CF on earlier task data in the CL process.

### MalCL on AZ-Class

We further evaluated MalCL using the AndroZoo dataset (AZ-Class) and observed, as illustrated in Figure 6, that

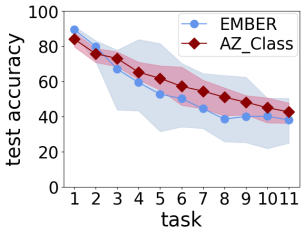



Figure 6: MalCL Performance on the EMBER and AZ-Class datasets using FML and L1-norm to Mean Logits.

MalCL performed better compared to when using the EMBER dataset.

Additionally, the gap between the minimum and maximum accuracy ranges is narrower. This improvement can be attributed to the EMBER dataset's more imbalanced set, characterized by a highly variable sample count per class. Such imbalance weakened MalCL's ability to retain learning from previous tasks and effectively learn from new observations, which added confusion to the malware family classification.

We summarize key findings through our experiments as follows.

- MalCL improved upon the None baseline by 27% and surpassed GR and BI-R by 28%, underscoring its effectiveness in malware classification compared to models optimized for other domains like image classification.
- The effectiveness of MalCL's performance is closely linked to the alignment of replay samples with the original data's feature space as defined by the classification model, especially when employing the L1 distance to Per-Class Mean Logits selection scheme.
- Assigning larger classes with more samples to earlier tasks significantly improved MalCL's accuracy (up to 74%), highlighting the importance of strategic task set construction in CL scenarios.
- The evaluation on the EMBER and AZ-Class datasets revealed that dataset imbalance can weaken MalCL's learning retention and accuracy, suggesting the need for strategies to handle class imbalances effectively in CL models.

## Conclusion

MalCL achieves state-of-the-art performance in mitigating CF in malware classification across Windows and Android platforms, underscoring the efficacy of our GR-based CL techniques. Our research robustly confirms that these methods can be effectively adapted to the malware domain. Looking ahead, improving the quality of synthetic malware generation remains critical. We aim to develop more advanced generative models and investigate hybrid training approaches that merge the advantages of GR and joint training. Future research will expand to address a wider variety of malware types and integrate more sophisticated features that reflect the dynamic nature of malware threats. Additionally, we will explore adaptive mechanisms to proactively anticipate and counter shifts in malware tactics, further bolstering MalCL's robustness.

## Acknowledgments

We would like to thank our anonymous reviewers for helpful comments. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)), and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00222385).

## References

Allix, K.; Bissyandé, T. F.; Klein, J.; and Le Traon, Y. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *MSR*.

Anderson, H. S.; and Roth, P. 2018. EMBER: an open dataset for training static pe malware machine learning models. *arXiv:1804.04637*.

Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; and Rieck, K. 2014. Drebin: Effective and explainable detection of android malware in your pocket. In *Network and Distributed System Security Symposium (NDSS)*.

AV-TEST. 2024. Malware Statistics and Trends Report. https://www.av-test.org/en/statistics/malware/.

Cao, X.; Lu, H.; Huang, L.; Liu, X.; and Cheng, M.-M. 2024. Generative Multi-modal Models are Good Class Incremental Learners. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 28706–28717.

Chai, Y.; Du, L.; Qiu, J.; Yin, L.; and Tian, Z. 2022. Dynamic prototype network based on sample adaptation for few-shot malware detection. *IEEE Transactions on Knowledge and Data Engineering*.

Channappayya, S.; Tamma, B. R.; et al. 2024. Augmented Memory Replay-based Continual Learning Approaches for Network Intrusion Detection. *Advances in Neural Information Processing Systems (NeurIPS)*.

Chen, Y.; Ding, Z.; and Wagner, D. 2023. Continuous Learning for Android Malware Detection. In *USENIX Security*.

Dahl, G. E.; Stokes, J. W.; Deng, L.; and Yu, D. 2013. Large-scale malware classification using random projections and neural networks. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

French, R. M. 1999. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*.

Gao, R.; and Liu, W. 2023. DDGR: Continual learning with deep diffusion-based generative replay. In *International Conference on Machine Learning (ICML)*, 10744–10763. PMLR.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144.

Hsu, Y.-C.; Liu, Y.-C.; Ramasamy, A.; and Kira, Z. 2018. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv:1810.12488*.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*.

Kovacs, E. 2018. FireEye MalwareGuard Uses Machine Learning to Detect Malware. https://www.securityweek.com/fireeye-malwareguard-uses-machine-learning-detect-malware/.

Maiorca, D.; Giacinto, G.; and Corona, I. 2012. A pattern recognition system for malicious PDF files detection. In *MLDM Workshop*.

McCloskey, M.; and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*. Elsevier.

Neyshabur, B.; Sedghi, H.; and Zhang, C. 2020. What is being transferred in transfer learning? *Advances in Neural Information Processing Systems (NeurIPS)*.

Parisi, G. I.; Kemker, R.; Part, J. L.; Kanan, C.; and Wermter, S. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks*.

Raff, E.; Fleshman, W.; Zak, R.; Anderson, H. S.; Filar, B.; and McLean, M. 2021. Classifying sequences of extreme length with constant memory applied to malware detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 9386–9394.

Rahman, M. S.; Coull, S. E.; and Wright, M. 2022. On the Limitations of Continual Learning for Malware Classification. In *First Conference on Lifelong Learning Agents (CoLLAs)*.

Rebuffi, S.-A.; Kolesnikov, A.; Sperl, G.; and Lampert, C. H. 2017. iCaRL: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Rolnick, D.; Ahuja, A.; Schwarz, J.; Lillicrap, T.; and Wayne, G. 2019. Experience replay for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; and Chen, X. 2016. Improved techniques for training gans. *Advances in neural information processing systems*, 29.

Shin, H.; Lee, J. K.; Kim, J.; and Kim, J. 2017. Continual learning with deep generative replay. *Advances in Neural Information Processing Systems (NeurIPS)*.

van de Ven, G. M.; Siegelmann, H. T.; and Tolias, A. S. 2020. Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*.

van de Ven, G. M.; Tuytelaars, T.; and Tolias, A. S. 2022. Three types of incremental learning. *Nature Machine Intelligence*.

VirusTotal. 2024. VirusTotal – Stats. https://www.virustotal.com/gui/stats.

Xu, K.; Li, Y.; Deng, R.; Chen, K.; and Xu, J. 2019. Droidevolver: Self-evolving android malware detection system. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual learning through synaptic intelligence. *Journal of Machine Learning Research (JMLR)*.