MADAR: Efficient Continual Learning for Malware Analysis with Distribution-Aware Replay

Mohammad Saidur Rahman §†, University of Texas at El Paso, msrahman3@utep.edu
Scott Coull §, Google, scottcoull@google.com
Qi Yu †, Rochester Institute of Technology, qyuvks@rit.edu
Matthew Wright †, Rochester Institute of Technology, matthew.wright@rit.edu

***Abstract*—Millions of new pieces of malicious software (i.e., malware) are introduced each year. This poses significant challenges for antivirus vendors, who use machine learning to detect and analyze malware, and must keep up with changes in the distribution while retaining knowledge of older variants. Continual learning (CL) holds the potential to address this challenge by relaxing the requirements of the incremental storage and computational costs of regularly retraining over all the collected data. Prior work, however, shows that CL techniques, which are designed primarily for computer vision tasks, fare poorly when applied to malware classification. To address these issues, we begin with an exploratory analysis of a typical malware dataset, which reveals that malware families are heterogeneous and difficult to characterize, requiring a wide variety of samples to learn a robust representation. Based on these findings, we propose *M*alware *A*nalysis with *D*istribution-*A*ware *R*eplay (MADAR), a CL framework that accounts for the unique properties and challenges of the malware data distribution. Through extensive evaluation on large-scale Windows and Android malware datasets, we show that MADAR significantly outperforms prior work. This highlights the importance of understanding domain characteristics when designing CL techniques and demonstrates a path forward for the malware analysis domain.**

***Index Terms*—Malware Analysis; Windows Malware; Android Malware; Catastrophic Forgetting; Continual Learning;**

Resources. The code and datasets of this paper are available at: https://github.com/IQSeC-Lab/MADAR.

I. Introduction

Advances in machine learning have significantly enhanced the detection and classification of malicious software, achieving notable success across various domains such as Windows executables [1]–[3], PDFs [4], and Android applications [5]–[7]. Traditional models, trained on static datasets, are typically expected to perform well on new data under the assumption of a constant data distribution. However, in reality, both malicious (*malware*) and benign (*goodware*) software evolve continuously, necessitating regular model updates to adapt to changes in data distribution and maintain effectiveness. For example, the AV-TEST Institute reports approximately 450,000 new malware samples daily [8], while VirusTotal processes over one million unique submissions each day [9]. This scale creates immense challenges in training and even storing all the samples.

Training a malware classification model solely on new data can lead to *catastrophic forgetting* (CF) [10], [11], where previously learned information is forgotten, resulting in increased misclassification and allowing attackers to evade detection with older malware strains, known as *Retrograde Malware Attacks (RMA)* (see Section II). As such, anti-virus vendors must deal with difficult trade-offs: (i) removing older samples from the training set, risking exposure to revived older malware; (ii) ignoring newer samples, risking failure to detect emerging trends; (iii) reducing the frequency of retraining, compromising accuracy during intervals between updates; or (iv) incurring significant effort and cost to frequently retrain on the combined new and older samples. These challenges highlight the need for agile and adaptive malware classification techniques capable of learning incrementally and responding to the dynamic malware landscape.

Continual learning (CL) provides a promising solution to this problem by enabling models to adapt to new data without requiring the retention of large datasets or frequent retraining [12]–[17]. By addressing catastrophic forgetting, CL techniques ensure that models remain effective and efficient in the face of evolving malware distributions. While designs for CL have been extensively studied in the context of computer vision [15], [18], [19], there are very few such studies in the malware classification domain [12], [20]. Rahman et al. [20] observed that CL techniques originally developed for computer vision problems fail to deliver acceptable performance in malware classification, due in part to the strong semantics of malware features and the high level of heterogeneity found in the malware ecosystem.

In this study, we first delve into the complexities of malware data distributions using the EMBER dataset [21] [1]. Our analysis highlights the heterogeneity in malware, both between and even within *families*, or groups of related malware. Leveraging this insight, we devise MADAR – *M*alware *A*nalysis with *D*istribution-*A*ware *R*eplay, a replay-based continual learning strategy that accounts for heterogeneity and achieves improved malware classification performance. In particular, MADAR replays a mix of representative samples and novel samples (i.e., outliers) to enhance the model's ability to retain knowledge and identify new malware variants despite memory constraints. Our techniques employ Isolation Forests (IF) [23] to identify critical novel samples, either directly through the raw feature vectors (MADAR), or through the use of the hidden representations of the model for a more compact representation (MADAR$^{\theta}$). For both of these approaches, we consider two mechanisms to

[1]The recently released EMBER 2024 dataset [22] is outside the scope of this study, as it was not available at the time the experiments were conducted.

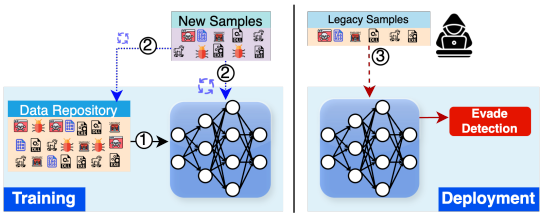


Fig. 1: Retrograde Malware Attack (RMA).

control how the number of replay samples is chosen, which we refer to as *Ratio* and *Uniform* strategies.

We evaluate these techniques with comprehensive experiments on two large-scale datasets across three CL scenarios representative of real-world malware analysis tasks, covering domain incremental learning (Domain-IL), class incremental learning (Class-IL), and task incremental learning (Task-IL). These datasets include the well-known EMBER dataset [21], containing one million examples of Windows executables, and a new benchmark dataset of Android malware from the AndroZoo repository [24] specifically created to explore CL scenarios. Our experimental results across these datasets demonstrate that MADAR is effective and outperforms prior state-of-the-art continual learning methods when confronted with realistic distribution shifts in malware data.

In summary, the contributions of this study are:

- We provide an exploratory analysis of the heterogeneity of malware distributions and show how it creates unique challenges for continuous learning.
- We develop a large-scale, realistic Android malware benchmark dataset covering all three CL scenarios – Domain-IL, Class-IL, and Task-IL.
- In Domain-IL scenarios, we show that MADAR performs much better than prior CL techniques. On the AndroZoo dataset, for example, MADAR comes within 0.4% average accuracy of the retraining baseline using just 50K training samples versus 680K for full retraining.
- MADAR is also effective in Class-IL scenarios, where it consistently outperforms all prior methods over a wide range of budgets. With a budget of 20K training samples on EMBER, MADAR gets an average accuracy of 85.8% versus 66.8% for the best method from prior work.
- For Task-IL, MADAR outperforms all prior methods across all memory budget configurations for both the EMBER and AndroZoo datasets. For example, in the AndroZoo dataset, the MADAR-U variant of MADAR achieves an average accuracy of 98.7% (within 0.1% of full retrain) with a budget of only 20K replay samples (versus approximately 250K for full retrain).

Through these contributions, this study stands as a significant advancement in continual learning for malware classification, highlighting the importance of understanding the domain distribution to effectively combat catastrophic forgetting.

II. Threat Model

The *Retrograde Malware Attack (RMA)* describes an attack scenario where adversaries can exploit catastrophic forgetting in machine learning-based malware detection systems. RMA occurs when a system, updated incrementally with only *new samples*, loses the ability to recognize older malware, allowing attackers to reintroduce legacy or slightly modified variants that can evade detection.

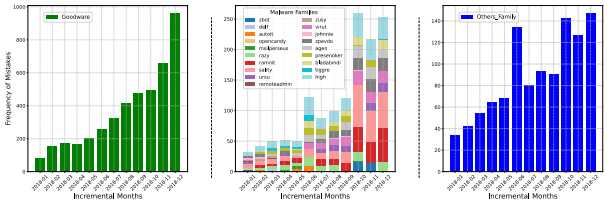


Fig. 2: Frequencies of forgotten goodware, top malware families, and malware with unlabeled families (i.e., `others_family`) across incremental monthly learning episodes for EMBER dataset.

As shown in Figure 1, the attack can be realized in the deployment phase of the detection system:

- **Initial Training and Deployment (①):** The system is trained on an initial dataset of malware and benign software and deployed for classification.
- **Incremental Updates and Forgetting (②):** As new samples emerge, the model is retrained with only the latest data. This process leads to catastrophic forgetting, where older malware signatures are no longer retained, reducing detection accuracy for previously known threats. In addition, false positives increase as the model struggles to differentiate between benign and malicious samples, leading to the misclassification of legitimate software. Figure 2 illustrates the frequencies of forgotten goodware, top malware families, and malware with unlabeled families (i.e., `others_family`) over incremental monthly learning episodes for EMBER dataset [21], demonstrating the extent of catastrophic forgetting in the model.
- **Retrograde Malware Attack (RMA) (③):** Attackers exploit this limitation by *reintroducing forgotten malware or slightly modified versions*, bypassing detection as the model has lost prior knowledge. The success of this attack increases with each incremental update if no mechanisms are in place to retain past information.

This attack presents a challenge for CL-based malware detection as maintaining detection accuracy for both new and previously known threats is critical. We address this by proposing MADAR a replay-based CL framework that mitigates catastrophic forgetting and improves robustness against RMA.

III. Related Work

A. Replay in Continual Learning

The fundamental challenge in developing a CL system is addressing catastrophic forgetting, and one of the widely studied methods to overcome forgetting is *replay*. In general, replay techniques complement the training data for each new task with older data that are representative of the tasks observed by the model so far. These techniques can further be classified into one of three subcategories – exact replay, generative replay, and compressed replay.

Exact Replay. Selecting and utilizing replay samples in exact replay involves determining a memory budget, denoted as $\mathcal{B}$.

Finding the optimal way to choose $\mathcal{B}$ remains an open research question [17], [25]. Exact-replay techniques are designed to choose replay samples from previously learned data to be combined with new samples for retraining. The goal of these techniques is to maximize the performance with minimal replay samples [16], [26]–[28].

Generative/Pseudo Replay. Generative or pseudo-replay strategies are designed to replicate the original data [12], [15], [18], [29]. These techniques either generate a representative of the original data using a separate generative model or generate pseudo-data by using an earlier model's predictions as soft labels for training subsequent models.

B. CL in Malware and Related Domains.

Despite extensive work in CL, very few studies have ventured into applying CL in the realm of malware. To the best of our knowledge, Rahman et al. [20] were the first to explore CL for malware classification. They concluded that existing CL methods fall short in tackling forgetting in malware classification systems due to differences in the underlying nature of the data distribution shifts that occur in practice versus those explored in the computer vision domain. Malware representations leverage tabular features with strong semantic constraints that limit the space of feasible samples, and within that space, samples exhibit a high level of heterogeneity. Replay-based techniques are found to perform better compared to other approaches in this setting. Another recent work, MalCL [12], introduces a generative replay-based continual learning approach for malware classification using a generative adversarial network (GAN). It achieves state-of-the-art performance compared to prior generative replay methods. However, its evaluation is restricted to the Class-IL scenario.

Furthermore, Chen et al. proposed to combine contrastive learning with active learning to continuously train Android malware classifiers [3]. They focus on the detection of *concept drift*, rather than overcoming forgetting.In addition, some studies have used *online learning* for malware classification, which deals with adding new samples as they are observed but does not directly address catastrophic forgetting [30]. Another CL domain in cybersecurity is network intrusion detection (NID), looking for malicious activity based on network packets. Amalapuram et al. explored a replay-based CL technique that incorporates class-balancing reservoir sampling and perturbation assistance for parameter approximation NID [**?**]. Another recent work explored semi-supervised CL for NID in a class incremental setting [31]. We note that NID is a different domain with different data characteristics than malware. Further, these works do not focus on reducing CF.

IV. Preliminaries

A. CL Scenarios for Malware Classification

Continual Learning (CL) is categorized into three scenarios: Domain Incremental Learning (Domain-IL), Class Incremental Learning (Class-IL), and Task Incremental Learning (Task-IL) [32]. In this work, we demonstrate how the three CL scenarios naturally fit into a typical malware analysis pipeline

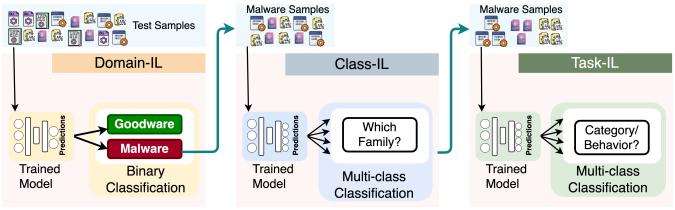


Fig. 3: CL scenarios in a typical malware analysis pipeline.

(see Figure 3). The first step in the pipeline is to determine whether a test sample is malware or goodware. Next, the pipeline identifies the specific family of the detected malware, which is formulated as a multi-class classification problem. Finally, the pipeline classifies the broader category or behavior of the malware, such as adware or ransomware, which is also a multi-class classification problem.

1) Domain-IL: The primary challenge in malware classification lies in distinguishing between goodware and malware. Each day, VirusTotal receives one million never-before-seen samples [9], highlighting the persistent and evolving nature of software, known as *concept drift* [3]. This evolution underscores the importance of rapidly integrating these new samples into operational systems to maintain effective protections against new threats. In addition, with the continual emergence of new benign software programs and the massive class imbalance in practice (i.e., significantly more goodware than malware), it is of utmost importance to not increase the false positive rate of the classifiers.

In this adversarial context, attackers might deploy older malware to evade detection by systems that have *forgotten* their signatures, necessitating a balance between adapting to new threats and preventing catastrophic forgetting. To address this, we segment our Domain-IL datasets into monthly tasks for EMBER and yearly tasks for AZ to mirror natural temporal shifts in the threat distribution.

2) Class-IL: Another significant task in malware analysis involves classifying malware into families, which are groups of programs with substantial code overlap and similar functionalities, as recognized by experts [33]. For instance, the `zeus` banking trojan has evolved into 556 variants across 35 families, including `citadel` and `gameover`. Labeling new samples often relies on consensus from multiple anti-virus engines and occurs when a significant set of similar samples forms a new family [33], [34]. In our incremental multi-class model, we start with known malware families and add new ones as they emerge, continuously adjusting and assessing the model across all known classes.

3) Task-IL: In malware analysis, leveraging insights from additional methods can prove beneficial. This may involve identifying the broader malware category (e.g., adware, ransomware, etc.), malware behaviors [35], or the infection vector (e.g., phishing, downloader, etc.). Task-IL encapsulates this concept of constrained tasks, where the introduction of a new task may symbolize a new category or behavior. This event occurs less frequently than adding a new family, as seen in Class-IL, yet it poses a genuine issue. Unlike Class-IL, the task identity is provided to the model at test time, significantly simplifying the problem. In malware, this could mean learning

the task identity from a separate model, manual analysis, or field reports of the malware's behavior. However, as our datasets do not possess naturally defined tasks, we partition our dataset into tasks comprising an equal number of independent and non-overlapping classes to act as a proxy to new behaviors, following common practice in the CL literature [15], [32]. In other words, a given task would be to perform family classification among one subset of families, and the subset that each sample belongs to is known to the classifier. The model is expected to be able to handle multiple tasks at once, and new tasks are being added during each experiment.

B. Dataset

In this study, we use large-scale Windows and Android malware datasets: EMBER [21], a Windows malware dataset from prior work, recognized as a benchmark for malware classification, and two new Android malware datasets derived from AndroZoo [24], specifically assembled for this research.

1) Windows PE Files: For our experiments, we leverage the EMBER 2018 dataset, containing features from one million Windows Portable Executable (PE) files, predominantly scanned in 2018 [2]. The dataset comprises 400K goodware and 350K malware, with the rest labeled as unknown. EMBER provides a diverse array of 2,381 hand-crafted features, covering general file information, header data, import/export functions, and section details. Notably, these features capture strong semantic concepts that have a limited space of feasible settings, outside of which the executable does not actually run.

In our Class-IL experiments, we focused on 2018 malware samples from 2,900 families. After filtering out families with fewer than 400 samples, we narrowed the remaining samples down to the top 100 families, leaving 337,035 samples for analysis. For Domain-IL, we included both goodware and malware from the entire year of 2018 for binary classification, excluding unknown samples.

2) Android APK Files: Additionally, we collected two datasets from AndroZoo [24] (AZ) for our experiments: AZ-Domain for Domain-IL and AZ-Class for Class-IL and Task-IL. These datasets contain Android APK files, and both use a 9:1 ratio of goodware to malware to reflect real-world class imbalance. Following the practice of prior work [30], the malware samples are selected with a VirusTotal detection count of $>= 4$. The AZ-Domain dataset includes 80,690 malware and 677,756 goodware samples from 2008 to 2016. We divided the AZ-Domain dataset into non-overlapping yearly training and testing sets. The AZ-Class dataset consists of 285,582 samples from 100 Android malware families, each with at least 200 samples.

We extracted Drebin features [5] from the apps for both datasets. These features cover various aspects of app behavior, including hardware access, permissions, app component names, filtered intents, restricted API calls, used permissions, suspicious API calls, and network addresses. Again, we note that these capture strong semantic concepts from the operation of the application. The training sets of AZ-Domain and AZ-Class have 3,858,791 and 1,067,550 features, respectively. We processed the test datasets to match the training feature sets and reduced dimensionality by filtering features with low variance (< 0.001) using `scikit-learn`'s `VarianceThreshold`. This resulted in final feature dimensions of 1,789 for AZ-Domain and 2,439 for AZ-Class, respectively.

[2] https://github.com/elastic/ember

C. Model Selection and Implementation

We use a multi-layer perceptron (MLP) model for malware classification, similar to the model used by Rahman et al. [20], for experiments with the EMBER dataset. For the AZ dataset, we developed a new MLP model with five fully-connected layers, quite similar to the MLP used for EMBER. This model uses the Adam optimizer with a learning rate of 0.001, and batch normalization and dropout for regularization.

The implementation of the output layer varies among Domain-IL, Class-IL, and Task-IL scenarios. Domain-IL operates as a series of binary classification tasks over 12 months for EMBER, and over 9 years for the AZ dataset, with two output units in each case: malicious and benign. In Class-IL, the output layer comprises units – one for each malware family. Output units are active only if they correspond to family that have been seen by that point in the experiment. Class-IL begins with an initial set of 50 families in the first task and progressively adds five more families in each of the remaining 10 tasks for both EMBER and AZ datasets. In Task-IL, only the output units of the families in the current task are active. Both the EMBER and AZ-Class datasets divide the families equally into 20 tasks, with each task containing five families.

D. Baselines and Metric

We adopt two baselines for comparison: *None* and *Joint*. *None* sequentially trains the model on each new task without any CL techniques, serving as an informal minimum baseline. By contrast, *Joint* uses all new and prior data for training at each step, acting as an informal maximum baseline. Despite its resource demands, *Joint* ensures strong performance throughout the dataset. We also introduce an additional baseline – Global Reservoir Sampling – which provides an unbiased sampling of the underlying class distributions and serves as a strong point of comparison for our distribution-aware approach.

Global Reservoir Sampling (GRS): GRS simply selects samples at random from a global stored data pool [36], [37]. Given a memory budget β, GRS randomly picks β samples from a data pool $\mathcal{P}$, with each incremental learning task contributing to the pool. If $\beta \geq \mathcal{P}$, GRS selects all the available samples in $\mathcal{P}$. Rahman et al. [20] investigated GRS – which they refer to as Partial Joint Replay – only for Domain-IL scenario of EMBER dataset. In this work, we present a deeper investigation of GRS in both Domain-IL and Class-IL scenarios with both EMBER and AZ datasets.

Global average accuracy ($\overline{AP} \in [0, 100]\%$): To maintain consistency with prior work, we present results using *global average accuracy* as the primary metric for our evaluations [15], [16], [20]. Note that we conducted a subset of evaluations using other metrics, such as F1 score, precision, and recall, which are not included in this paper. The conclusions remain unchanged for all of these metrics.

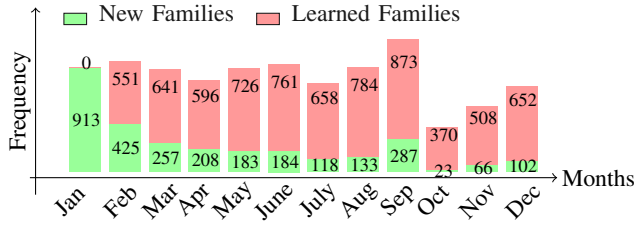


Fig. 4: New and already learned families in each task.

Let $P_{i,j}$ be the accuracy of the model on the test set of task T_j, $j \leq i$, after continually training the model on tasks 1 to i. Then the average accuracy AP at task T_i is defined as $AP_i = \frac{1}{i}\sum_{j=1}^{i} P_{i,j}$. For N total tasks, the global average accuracy $\overline{AP}$ over all tasks is computed as $\overline{AP} = \frac{1}{N}\sum_{i=1}^{N} AP_i * 100$.

E. Training and Evaluation Protocol

A continual learning (CL) model is sequentially trained to learn tasks from $t_1, t_2, ..., t_T$, each with its distinct data distribution $p(x, y|t_i)$. The goal is to adapt to new tasks without forgetting the old ones. CL training involves three sets of parameters: shared parameters (θ_s) across all tasks, old task-specific parameters (θ_0), and new task parameters (θ_n) [29]. The *Joint* training benchmark trains the model with all the available training samples up to the current task and optimizes all these parameters simultaneously; however, it incurs incremental storage and training costs. In contrast, CL training strives to optimize and update θ_s and θ_n, while maintaining θ_0 in a relatively fixed state for each new task t_n. However, updating any of the shared weights θ_s risks confusing the classifier when faced with older data, as those classification decisions depend not only on θ_0 but also on θ_s. CL training typically boasts significantly faster speeds and far less storage requirements than *Joint* training, thus permitting more frequent model retraining to adapt to evolving data distributions or other requirements.

In our evaluations, we use a non-overlapping hold-out set corresponding to each task. For example, the AZ-Domain dataset contains 8 years of training samples from 2008 to 2016, resulting in 9 hold-out sets, one for each year. A CL model is evaluated on all the hold-out sets up to the current task; formally, the model is evaluated on tasks t_i to t_T, for $1 \leq i \leq T$, after it been trained on the current task t_T. In this work, each set of experiments is performed around 10-15 times with different random parameter initializations. We use PyTorch on a CentOS-7 machine with an Intel Xeon processor, 40 CPU cores, 128GB RAM, and four GeForce RTX 2080Ti GPUs, each with 12GB memory.

V. Exploratory Analysis of EMBER

In this section, we provide an analysis of the EMBER dataset, which sheds light on the distribution across various families and tasks, aiding in selecting representative samples for replay. We identified 2,899 unique malware families within a subset of EMBER, and an additional 11,433 samples lacking clear family labels were assigned the label *Other*.

We investigate the prevalence of malware families over time, distinguishing between recurring and newly identified families

TABLE I: Number of samples and families per task in EMBER

Task	#of Goodware	#of Malware	#of Families
January	29423	32491	913
February	22915	31222	976
March	21373	20152	898
April	25190	26892	804
May	23719	22193	909
June	23285	25116	945
July	24799	26622	776
August	23634	21791	917
September	26707	37062	1160
October	29955	56459	393
November	50000	50000	574
December	50000	50000	754

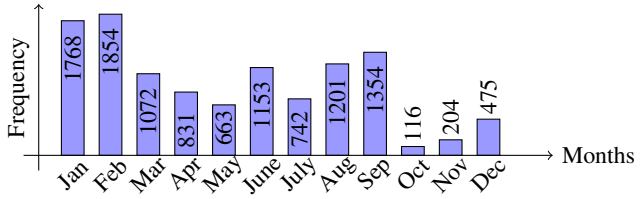


Fig. 5: EMBER Malware samples without AV-Class labels.

each month in Figure 4. Unlike many datasets used in CL research, we see significant churn in the representation of families over time. Of the 913 families seen in January, for example, only 551 are seen in February, while 425 new families emerge. This churn indicates a potential issue in training data continuity, which may aggravate catastrophic forgetting and underscores the need for different CL strategies in the malware domain. Table I shows the number of goodware and malware samples in each month, as well as the number of families. Generally, each family has its own distinct distribution pattern, and together these patterns make up the total distribution of malware for a particular month.

Worse, many malware samples do not have family labels at all (see Figure 5). Correctly labeling samples is a challenging endeavor and often takes time and expert knowledge [34], so this lack of labels matches real-world conditions. Family labels for malware are based on the *av-class* labels provided by the av-test engine [8]. Analysis of the *Other*-labeled samples do not align with the labeled families, which shows that many of them indeed come from unknown families.

Furthermore, our analysis shows that the prominent malware families change with the evolution of tasks. We identified the 10 most common malware families based on the frequency of samples from each family and found that the top 10 families vary across tasks. The most prevalent families at the beginning of the experiment (i.e., January) do not remain prevalent in later tasks (i.e., from February on). For example, the `emotet` malware family was the most consistent, appearing in 11 out of 12 tasks. The next most consistent families were `fareit` and `zusy`, appearing in eight and seven tasks, respectively. This indicates considerable concept drift in malware data, highlighting the need to regularly update classifiers.

In addition, many malware families display complex distributional patterns in feature space, making for additional heterogeneity within classes. Figure 6, for example, shows a t-SNE projection of the EMBER features for all malware samples from January 2018. Each class (represented by color) is not clustered into a single well-defined region. Rather, the larger classes are split up into subsets that spread out in feature

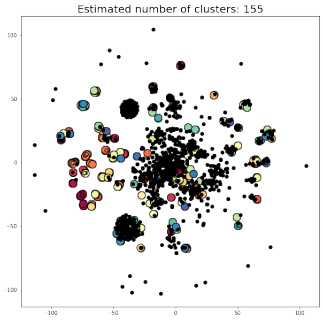


Fig. 6: t-SNE projection of EMBER malware from Jan. 2018.

space. To accurately represent the malware distribution, it is thus important to select samples not only from each class, but also from multiple areas within each class.

Despite some degree of overlap among different families, it is possible to discern distinct clusters indicative of the heterogeneity across and even within malware families. It is vital to capture samples from each of these smaller clusters, including those belonging to minority families, to accurately represent the malware landscape for a specific task month. To further complicate the situation, it is often not possible to provide definitive family labels for a sample due to the inherent subjectivity involved in malware analysis, which results in a large, diverse set of additional unlabeled samples which must be considered [34]. These attributes of the malware landscape make it difficult to characterize classes as contiguous regions of the feature space with relatively simple class boundaries. Rather, each class might only be understood as a collection of smaller pockets of the feature space that might be closer to pockets from other classes than the same class. This may explain why prior CL techniques designed for computer vision datasets are less effective when applied to the malware domain [20].

In light of these analysis, we propose that selecting samples based on families and variations within those families could more effectively capture the heterogeneity within the replay data distribution, potentially mitigating catastrophic forgetting.

VI. Distribution-Aware Replay

Here, we introduce the our proposed continual learning (CL) framework for malware classification with two distribution-aware replay variants: MADAR and MADAR$^{\theta}$. MADAR operates on the basis of the raw-feature space and MADAR$^{\theta}$ operates on the basis of the model's weights-space of raw-features.

A. MADAR

Building on the exploratory analysis in Section V, we postulate that stratified sampling, where replay samples are chosen based on their representation in malware families, may better preserve the model's stability versus random sampling as used in GRS. Moreover, we also seek to capture the heterogeneity *within* each family's data distribution. We do this by selecting a balance between *representative samples* that are close to each other and *anomalous samples* that are separated. While any single anomalous sample is not as important to learn and remember as a single representative sample, a collection of anomalous samples helps to track the heterogeneity within a class.

Algorithm 1: MADAR in Domain-IL.

Input : c – Current Task number, X_c, Y_c – Samples and labels of c, $\mathcal{P}$ – Data pool, β – Memory budget, γ – Split of β for malware and goodware, Ω – Split of anomalous/similar samples, ξ – Ratio budgeting, Ψ – Uniform budgeting

init $\mathcal{P}$; **init** $\mathcal{D} \leftarrow \{M_f : M_c\}$;
if $c = 0$ **then**
  $\mathcal{P} \leftarrow X_c, Y_c$;
  $X_{good}, X_{mal} \leftarrow \mathcal{P}$;
  $\nabla\mathcal{D} \leftarrow X_{mal}$
  $X_{train},\ Y_{train} \leftarrow X_c, Y_c$
else
  $X_{good}, X_{mal} \leftarrow \mathcal{P}$;
  $\beta_M, \beta_G \leftarrow \beta\ .\ \gamma$;
  $\beta_A, \beta_S \leftarrow \beta_M\ .\ \Omega$; **if** Ψ **then**
    $\mathcal{NF} \leftarrow |\mathcal{D}|$; $\mathcal{B_F} \leftarrow \beta_M / \mathcal{NF}$;
  $R_{mal} \leftarrow [\ ]$;
  for $X_f \subseteq X_{mal}$ **do**
    $\mathcal{F_{MC}} \leftarrow X_f$;
    if ξ **then**
      $\mathcal{MC} \leftarrow |\mathcal{D}|$; $\mathcal{B_F} \leftarrow (\mathcal{F_{MC}}/\mathcal{MC}) * \beta_M$;
    if $\mathcal{F_{MC}} <= \mathcal{B_F}$ **then**
      R_{mal}.append(X_f);
    else
      $(A_f, S_f) \leftarrow$ IF(X_f, β_A, β_S);
      R_{mal}.append(A_f, S_f)
  $R_{good} \leftarrow$ sample$(X_{good}, len(R_{mal}))$;
  $X_{replay} \leftarrow (R_{good}, R_{mal})$;
  $Y_{replay} \leftarrow ([0] *$ len$(R_{good}), [1] *$ len$(R_{mal}))$;
  $X_{train} \leftarrow$ concat$((X_c, X_{replay}))$;
  $Y_{train} \leftarrow$ (concat$((Y_c, Y_{replay}))$;
  $\mathcal{P}$.append(X_c, Y_c); $\nabla\mathcal{D} \leftarrow X_{mal}$;
return $(X_{train},\ Y_{train})$

Isolation Forest (IF) [23] is a technique for identifying outliers in high-dimensional data. IF uses decision trees to isolate anomalous data points based on the intuition that outliers are distinct and easy to separate from the rest of the data. An important parameter in IF is the contamination rate C_r, which represents the expected fraction of outliers in the data. We found that $C_r = 0.1$ works best and used it in all our experiments. The algorithm for MADAR in the Domain-IL setting is provided in Algorithm 1. The algorithm for MADAR in the Class-IL and Task-IL settings is provided in Algorithm 2.

Procedure: We now describe MADAR using the framework of Domain-IL; the process is similar for Class-IL and Task-IL. The procedure begins by initializing a global data pool $\mathcal{P}$, containing both goodware and malware samples, and a dictionary $\mathcal{D}$ that tracks malware families and their frequencies in the data up to the current task.

For each new task c, MADAR divides the data into goodware (X_{good}) and malware (X_{mal}) subsets from $\mathcal{P}$, allocating memory budgets β_M for malware and β_G for goodware from the total memory budget β, based on a split ratio γ:

$$\beta_G = \gamma \cdot \beta, \quad \beta_M = (1 - \gamma) \cdot \beta. \tag{1}$$

For balanced datasets like EMBER, $\gamma = 0.5$ ensures an equal split between malware and goodware. For an imbalanced dataset, it is better to tune γ. Our Android malware (AZ) datasets, for example, have a 9:1 ratio of goodware to malware, so we use $\gamma = 0.9$.

Before training for a new task, MADAR incrementally trains the classifier using a combination of new samples from the current task and replay samples from previous tasks. The replay samples include both goodware ($R_{good} \subset X_{good}$) and malware ($R_{mal} \subset X_{mal}$), with R_{mal} sampled from specific malware families rather than randomly from all of X_{mal}.

For each family f, we set its *family budget* $\mathcal{B}_f$— the number of samples to select from f—using two sub-sampling variants: *Ratio budgeting* and *Uniform budgeting*.

- **Ratio Budgeting:** Select the number of samples from a family f proportional to that family's representation in the dataset. The family budget $\mathcal{B}_f$ is $\mathcal{B}_f = \frac{|X_f|}{|X_{mal}|} \cdot \beta_M$, where $|X_f|$ is the number of samples in family f, and $|X_{mal}|$ is the total number of malware samples. This strategy may be more suitable in binary classification, as it provides proportional representation of the malware families for training on the malicious class.
- **Uniform Budgeting:** In this method, the memory budget β_M is uniformly distributed across all malware families: $\mathcal{B}_f = \frac{\beta_M}{|\mathcal{F}|}$, where $|\mathcal{F}|$ is the total number of malware families. Compared with Ratio budgeting, Uniform budgeting may work well for multi-class classification to determine which family a sample belongs to, as it ensures better class balance.

Within each family f, we further split the family budget $\mathcal{B}_f$ into two parts: representative samples S_f and anomalous samples A_f, using IF, controlled by a split parameter α:

$$|S_f| = \alpha \cdot \mathcal{B}_f, \quad |A_f| = (1 - \alpha) \cdot \mathcal{B}_f \tag{2}$$

We found empirically that a balanced split ($\alpha = 0.5$) between representative and anomalous samples provides optimal performance. In this setup, the model learns equally the core class characteristics from representative samples and less common variations from anomalous samples.

The malware replay set R_{mal} is then constructed by combining the representative and anomalous samples from all malware families:

$$R_{mal} = \bigcup_{f \in F} \{S_f \cup A_f\}. \tag{3}$$

The total replay set consists of both goodware and malware replay samples, which are then concatenated with the new task samples to form the training set for the current task c. After training, the data pool $\mathcal{P}$ is updated with the new task samples, $\mathcal{P} \leftarrow \mathcal{P} \cup \{X_c, Y_c\}$, and the malware family dictionary $\mathcal{D}$ is updated to reflect the new frequencies of malware families in X_{mal}: $\mathcal{D} \leftarrow \mathcal{D} + freq(X_{mal})$.

Algorithm 2: MADAR in Class-IL and Task-IL

Input : c – Task number, X_c, Y_c – Malware samples and their family labels, $\mathcal{P}$ – Malware data pool, β – Memory budget, Ω – Split of anomalous/similar samples, ξ – Ratio budgeting, Ψ – Uniform budgeting

init $\mathcal{P}$; **init** $\mathcal{D} \leftarrow \{M_f : M_c\}$;
if $c = 0$ **then**
  $\mathcal{P} \leftarrow X_c, Y_c$; $X_{mal} \leftarrow \mathcal{P}$; $\nabla\mathcal{D} \leftarrow X_{mal}$;
  $X_{train}, Y_{train} \leftarrow X_c, Y_c$;
else
  $X_{mal} \leftarrow \mathcal{P}$; $\beta_A, \beta_S \leftarrow \beta \cdot \Omega$;
  if Ψ **then**
    $\mathcal{NF} \leftarrow \mathcal{D}$; $\mathcal{B_F} \leftarrow \beta / \mathcal{NF}$;
  $R_{mal} \leftarrow [\,]$; **for** $X_f \subseteq X_{mal}$ **do**
    $\mathcal{F_{MC}} \leftarrow X_f$; **if** ξ **then**
      $\mathcal{MC} \leftarrow \mathcal{D}$; $\mathcal{B_F} \leftarrow (\mathcal{F_{MC}} / \mathcal{MC}) \cdot \beta$;
    if $\mathcal{F_{MC}} \leq \mathcal{B_F}$ **then**
      R_{mal}.append(X_f);
    else
      $(A_f, S_f) \leftarrow \texttt{IF}(X_f, \beta_A, \beta_S)$;
      R_{mal}.append(A_f, S_f);
  $X_{replay} \leftarrow R_{mal}$; $Y_{replay} \leftarrow ([1] \times \texttt{len}(R_{mal}))$;
  $X_{train} \leftarrow \texttt{concat}(X_c, X_{replay})$;
  $Y_{train} \leftarrow \texttt{concat}(Y_c, Y_{replay})$;
  $\mathcal{P}$.append(X_c, Y_c); $\nabla\mathcal{D} \leftarrow X_{mal}$;
return (X_{train}, Y_{train})

B. MADAR$^\theta$

While Isolation Forest (IF) outperforms other distance-based anomaly detection techniques on high-dimensional data, it struggles with correlated features [38]. Let $\mathbf{X} \in \mathbb{R}^D$ represent the input data, where D is the feature dimension. For example, the EMBER dataset has $D = 2381$, while the AZ datasets have $D = 1789$ for AZ-Domain and $D = 2439$ for AZ-Class, respectively. Instead of applying dimensionality reduction techniques such as PCA, we propose leveraging a neural network $\mathcal{M}$ to generate compact feature representations $\mathbf{Z} \in \mathbb{R}^d$, where $d \ll D$. This approach is particularly advantageous in continual learning contexts, as it complements ongoing model development and adapts to evolving tasks [39], [40].

To address these, we introduce MADAR$^\theta$, a variant of MADAR that leverages learned representations from a trained model $\mathcal{M}$ to identify both representative and diverse samples effectively. For an input sample $\mathbf{x}$, the model $\mathcal{M}$ computes activations $\mathcal{W}(\mathbf{x})$, representing its internal state. Specifically, for a set of inputs $\mathbf{X}_f$ belonging to a family f, MADAR$^\theta$ extracts activations:

$$\Theta^{\mathcal{L}}(\mathbf{X}_f) = \{\mathcal{W}^{\mathcal{L}}(\mathbf{x}) : \mathbf{x} \in \mathbf{X}_f\}, \quad \text{where} \quad \Theta^{\mathcal{L}}(\mathbf{X}_f) \in \mathbb{R}^d, \tag{4}$$

from a chosen layer $\mathcal{L}$ of the model.

These activations are analyzed using the Isolation Forest (IF) algorithm to identify anomalous activations $\mathcal{A}_w \subseteq \Theta^{\mathcal{L}}(\mathbf{X}_f)$. The corresponding anomalous samples in the original input space are denoted as:

$$A_f = \{\mathbf{x} \in \mathbf{X}_f : \mathcal{W}^{\mathcal{L}}(\mathbf{x}) \in \mathcal{A}_w\}. \tag{5}$$

Non-anomalous samples are similarly sampled to form the set S_f, ensuring a balanced and representative replay set:

$$S_f = \{\mathbf{x} \in \mathbf{X}_f : \mathcal{W}^{\mathcal{L}}(\mathbf{x}) \notin \mathcal{A}_w\}. \tag{6}$$

TABLE II: Summary of Results for EMBER Domain-IL Experiments.

Group	Method	Budget						
		1K	10K	50K	100K	200K	300K	400K
Baselines	Joint				96.4±0.3			
	None				93.1±0.1			
	GRS	93.6±0.3	94.1±1.3	95.3±0.2	95.3±0.7	95.9±0.1	95.8±0.6	96.0±0.3
Prior Work	ER [26]	80.6±0.1	73.5±0.5	70.5±0.3	69.9±0.1	70.0±0.1	70.0±0.1	70.0±0.1
	AGEM [27]	80.5±0.1	73.6±0.2	70.4±0.3	70.0±0.1	70.0±0.2	70.0±0.1	70.0±0.1
	GR [18]				93.1±0.2			
	RtF [41]				93.2±0.2			
	BI-R [15]				93.4±0.1			
MADAR	MADAR-R	**93.7±0.1**	**94.7±0.1**	**95.4±0.1**	**95.3±0.6**	**96.0±0.1**	**96.1±0.1**	**96.1±0.1**
	MADAR-U	**93.6±0.2**	94.0±0.2	95.1±0.1	**95.3±0.1**	95.5±0.1	95.7±0.1	95.8±0.1
	MADAR$^\theta$-R	**93.6±0.1**	**94.4±0.3**	**95.3±0.2**	**95.8±0.1**	**96.1±0.1**	**96.1±0.1**	**96.1±0.1**
	MADAR$^\theta$-U	93.5±0.2	94.1±0.2	94.9±0.1	95.2±0.2	95.6±0.1	95.7±0.1	95.7±0.1

TABLE III: Summary of Results for AZ Domain-IL Experiments.

Group	Method	Budget						
		1K	10K	50K	100K	200K	300K	400K
Baselines	Joint				97.3±0.1			
	None				94.4±0.1			
	GRS	95.3±0.1	96.4±0.1	96.9±0.1	97.1±0.1	97.1±0.1	97.2±0.1	97.2±0.1
Prior Work	ER [26]	40.4±0.1	40.1±0.1	41.1±0.2	42.6±0.1	44.0±0.1	45.9±0.1	48.6±1.1
	AGEM [27]	45.4±0.1	47.4±0.2	49.2±0.2	53.7±0.6	54.2±0.3	54.8±0.4	56.7±0.3
	GR [18]				93.3±0.4			
	RtF [41]				93.4±0.2			
	BI-R [15]				93.5±0.1			
MADAR	MADAR-R	**95.8±0.1**	**96.6±0.1**	**96.9±0.1**	**97.0±0.1**	**97.0±0.1**	**97.0±0.1**	**97.0±0.1**
	MADAR-U	**95.7±0.1**	95.5±0.1	95.2±0.2	95.2±0.1	95.4±0.1	95.8±0.2	96.3±0.2
	MADAR$^\theta$-R	**95.8±0.2**	**96.6±0.1**	**96.9±0.1**	**96.9±0.1**	**97.1±0.1**	**97.1±0.1**	**97.2±0.1**
	MADAR$^\theta$-U	95.6±0.1	96.1±0.1	96.6±0.1	96.8±0.1	**97.0±0.1**	**97.1±0.1**	**97.1±0.1**

The replay samples $A_f \cup S_f$ are then used in subsequent training episodes to maintain knowledge retention and mitigate catastrophic forgetting.

Selection of the Layer $\mathcal{L}$. The choice of $\mathcal{L}$ is critical in MADAR$^\theta$ to ensure that feature representations are captured without interference from the model's final classification stage. Empirical testing revealed that the penultimate layers, denoted as `act4` for the EMBER dataset and `act5` for the AZ datasets, provide optimal results such that $\mathcal{L}_{\text{EMBER}} = \texttt{act4}, \quad \mathcal{L}_{\text{AZ}} = \texttt{act5}$.

Adaptation of Batch Normalization. During the forward pass, batch normalization is omitted in MADAR$^\theta$ to preserve the heterogeneity of the activation distributions. While batch normalization typically stabilizes learning and improves generalization, it can homogenize activations, reducing the heterogeneity essential for identifying anomalies. This adaptation enhances the performance of MADAR$^\theta$, as confirmed by empirical results.

Efficiency of MADAR$^\theta$. MADAR$^\theta$ is computationally efficient compared to MADAR applied directly to the input space. The hidden layers `act4` and `act5` have $d = 128$, significantly reducing the dimensionality from D. As a result, MADAR$^\theta$ offers reduced computational complexity.

VII. Evaluation

We present the results of our MADAR framework and MADAR$^\theta$ in the Domain-IL, Class-IL, and Task-IL scenarios using the EMBER and AZ datasets discussed in Section IV-B. To denote our techniques, we use the following abbreviations: **MADAR-R** for MADAR-Ratio, **MADAR-U** for MADAR-Uniform, **MADAR$^\theta$-R** for MADAR$^\theta$-Ratio, and **MADAR$^\theta$-U** for MADAR$^\theta$-Uniform.

For all three scenarios, we compare MADAR against widely studied replay-based continual learning (CL) techniques, including experience replay (ER) [26], average gradient episodic memory (AGEM) [27], deep generative replay (GR) [18], Replay-through-Feedback (RtF) [41], and Brain-inspired Replay (BI-R) [15]. Additionally, we evaluate MADAR against iCaRL [16], a replay-based method specifically designed for Class-IL. For the Class-IL and Task-IL scenarios, we additionally compare MADAR with Task-specific Attention Modules in Lifelong Learning (TAMiL) [14]. Furthermore, we benchmark MADAR against MalCL [12], a method specifically designed for Class-IL. Notably, most recent work focuses primarily on Class-IL and Task-IL scenarios, limiting direct comparisons in the Domain-IL scenario. In our results tables, the best-performing methods and those within the error margin of the top results are highlighted.

A. Domain-IL

In EMBER, we have 12 tasks, each representing the monthly data distribution spanning January–December 2018. Our results, detailed in Table II, provide a comprehensive view of each method's performance, reported as the average accuracy over all tasks $\overline{\mathbf{AP}}$.

TABLE IV: Summary of Results for EMBER Class-IL Experiments.

Group	Method	Budget						
		100	500	1K	5K	10K	15K	20K
Baselines	Joint				86.5±0.4			
	None				26.5±0.2			
	GRS	51.9±0.4	70.3±0.5	75.4±0.7	82.0±0.2	83.5±0.1	84.3±0.3	84.6±0.2
Prior Work	TAMiL [14]	32.2±0.3	33.1±0.2	35.3±0.2	36.7±0.1	38.2±0.3	37.2±0.2	38.8±0.2
	iCaRL [16]	53.9±0.7	58.7±0.7	60.0±1.0	63.9±1.2	64.6±0.8	65.5±1.0	66.8±1.1
	ER [26]	27.5±0.1	27.8±0.1	28.0±0.1	27.9±0.1	28.0±0.1	28.0±0.1	28.2±0.1
	AGEM [27]	27.3±0.1	27.4±0.1	27.7±0.1	28.5±0.1	28.2±0.1	28.3±0.1	28.2±0.1
	GR [18]				26.8±0.2			
	RtF [41]				26.5±0.1			
	BI-R [15]				26.9±0.1			
	MalCL [12]				54.5±0.3			
MADAR	MADAR-R	**68.0±0.4**	73.6±0.2	76.0±0.3	81.5±0.2	83.2±0.2	83.8±0.2	84.0±0.2
	MADAR-U	66.4±0.4	**76.5±0.2**	**79.4±0.4**	**83.8±0.2**	**84.8±0.1**	**85.5±0.1**	**85.8±0.3**
	MADAR$^\theta$-R	**67.9±0.3**	72.7±0.5	72.7±0.5	81.7±0.2	83.2±0.1	83.9±0.1	84.5±0.2
	MADAR$^\theta$-U	67.5±0.3	**76.4±0.4**	**78.5±0.4**	**84.1±0.1**	**85.3±0.1**	**85.8±0.2**	**86.2±0.2**

The informal lower and upper performance bounds for this configuration are approximated by the *None* and *Joint* methods, achieving $\overline{\mathbf{AP}}$ scores of 93.1% and 96.4%, respectively. Meanwhile, *GRS* serves as a strong baseline, providing unbiased sampling without incorporating sample heterogeneity awareness.

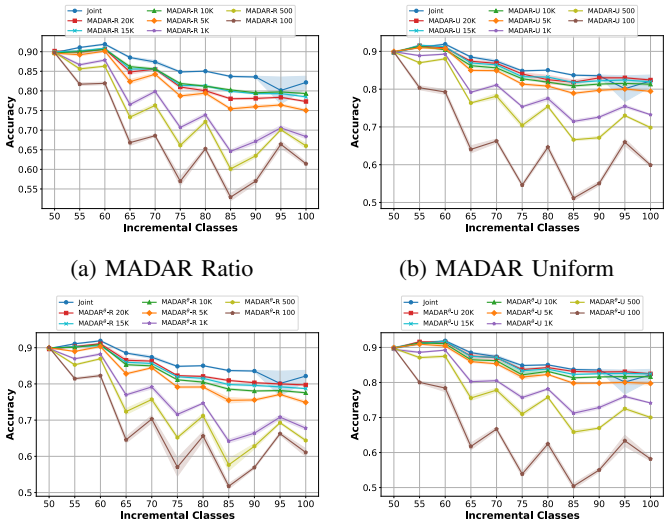
At a lower budget of 1K, MADAR-R, MADAR-U, and MADAR$^\theta$-R exhibit competitive performance, all achieving $\overline{\mathbf{AP}}$ of over 93.6%, significantly outperforming prior approaches. This highlights their ability to effectively utilize limited resources. In particular, MADAR-R achieves the highest accuracy at this budget, with $\overline{\mathbf{AP}}$ of 93.7%. As the memory budget increases, the performance of all MADAR and MADAR$^\theta$ variants improves steadily. At a budget of 200K, MADAR-R and MADAR$^\theta$-R achieve an impressive $\overline{\mathbf{AP}}$ of 96.0% and 96.1%, respectively, closely approaching the 96.4% achieved by the *Joint* baseline, which utilizes over 670K samples. Uniform strategies, including MADAR-U and MADAR$^\theta$-U, are only slightly behind, with $\overline{\mathbf{AP}}$ values of 95.5% and 95.6%, respectively.

For the experiments with AZ-Domain, we consider 9 tasks, each representing a yearly data distribution from 2008 to 2016. The performance of each method is presented in Table III as $\overline{\mathbf{AP}}$ and compared to two baselines: *None*, which achieves 94.4%, and *Joint*, which reaches 97.3%.

Similar to the results observed with EMBER, our MADAR techniques consistently outperform prior methods such as ER, AGEM, GR, RtF, and BI-R across all budget levels. For lower budgets, such as 1K, MADAR-R achieves $\overline{\mathbf{AP}}$ of 95.8% and coming within 1.5% of the *Joint* baseline.

At higher budgets, ranging from 100K to 400K, MADAR-R continues to demonstrate high $\overline{\mathbf{AP}}$ scores of up to 97.0%, closely matching GRS and only marginally below the *Joint* baseline, which requires significantly more training samples (680K). Notably, MADAR$^\theta$-R exhibits comparable performance, reaching a peak $\overline{\mathbf{AP}}$ of 97.2% at the highest budget level, further underscoring the efficacy of our distribution-aware approach.

In summary, these results empirically demonstrate the effectiveness of MADAR's distribution-aware sample selection in enhancing the efficiency and accuracy of malware classification in Domain-IL scenarios. MADAR-R and MADAR$^\theta$-R, in particular, consistently either yield on-par or outperform GRS while delivering significant improvements over prior methods.



(a) MADAR Ratio (b) MADAR Uniform

(c) MADAR$^\theta$ Ratio (d) MADAR$^\theta$ Uniform

Fig. 7: EMBER Class-IL: Comparison of the MADAR-R, MADAR-U, MADAR$^\theta$-R, and MADAR$^\theta$-U with Joint baseline.

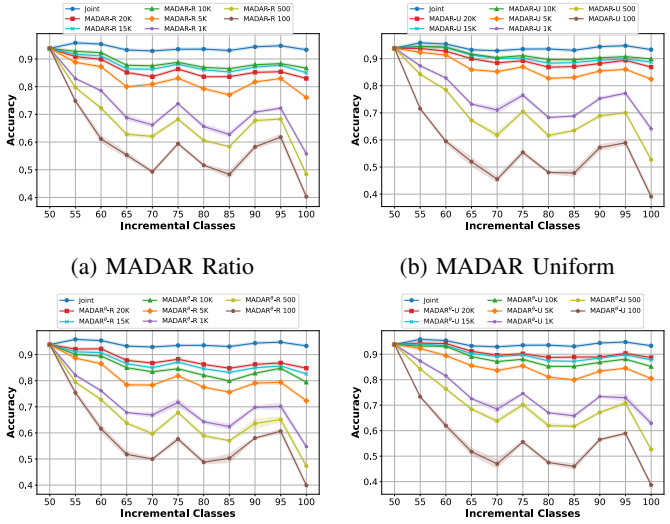
B. Class-IL

In this set of experiments with EMBER, we consider 11 tasks, starting with 50 classes (representing distinct malware families) in the initial task, and incrementally adding five new classes in each subsequent task. Table IV presents the performance of each method, measured by average accuracy $\overline{\mathbf{AP}}$. The *None* and *Joint* baselines achieve $\overline{\mathbf{AP}}$ values of 26.5% and 86.5%, respectively, providing informal lower and upper bounds. Figure 7 illustrates the progression of average accuracy across tasks, showing how the MADAR and MADAR$^\theta$ methods compare to the *Joint* baseline.

At a very low budget of just 100 samples, MADAR-R achieves a notable $\overline{\mathbf{AP}}$ of 68.0%, outperforming GRS and

TABLE V: Summary of Results for AZ Class-IL Experiments.

Group	Method	Budget						
		100	500	1K	5K	10K	15K	20K
Baselines	Joint				94.2±0.1			
	None				26.4±0.2			
	GRS	43.8±0.7	62.9±0.8	70.2±0.4	83.0±0.3	86.4±0.2	88.2±0.2	89.1±0.2
Prior Work	TAMiL [14]	53.4±0.3	55.2±0.3	57.6±0.3	60.8±0.2	63.5±0.1	65.3±0.5	67.7±0.3
	iCaRL [16]	43.6±1.2	54.9±1.0	61.7±0.7	77.2±0.4	81.5±0.6	83.4±0.5	84.6±0.5
	ER [26]	50.8±0.7	58.3±0.6	58.9±0.2	59.2±0.8	62.9±0.7	63.1±0.5	64.2±0.4
	AGEM [27]	27.3±0.7	28.0±1.4	27.1±0.3	28.0±0.6	28.2±1.0	29.8±2.6	28.0±0.8
	GR [18]				22.7±0.3			
	RtF [41]				22.9±0.3			
	BI-R [15]				23.4±0.2			
	MalCL [12]				59.8±0.4			
MADAR	MADAR-R	**59.4±0.6**	67.8±0.9	71.9±0.5	82.9±0.2	86.3±0.1	88.2±0.2	89.1±0.1
	MADAR-U	57.3±0.5	**70.4±0.4**	**76.2±0.2**	**86.8±0.1**	**89.8±0.1**	**91.0±0.1**	**91.5±0.1**
	MADAR$^{\theta}$-R	**58.8±0.3**	66.2±0.7	71.0±0.7	81.2±0.3	85.1±0.2	86.9±0.2	88.1±0.1
	MADAR$^{\theta}$-U	58.5±0.7	**70.1±0.2**	**74.7±0.2**	**85.5±0.1**	**88.7±0.1**	**90.3±0.2**	**90.7±0.1**



(a) MADAR Ratio (b) MADAR Uniform

(c) MADAR$^{\theta}$ Ratio (d) MADAR$^{\theta}$ Uniform

Fig. 8: AZ Class-IL: Comparison of the MADAR-R, MADAR-U, MADAR$^{\theta}$-R, and MADAR$^{\theta}$-U with Joint baseline.

prior methods by a significant margin. As the budget increases, MADAR-U emerges as the top performer, achieving $\overline{\mathbf{AP}}$ values of 76.5% and 79.4% at 1K and 10K budgets, respectively, surpassing all other methods, including GRS.

At higher budgets, MADAR-U and MADAR$^{\theta}$-U continue to excel, with MADAR$^{\theta}$-U achieving the best results overall. At a 20K budget, MADAR$^{\theta}$-U reaches an $\overline{\mathbf{AP}}$ of 86.2%, nearly equaling the *Joint* baseline, which uses over **150 times** more training samples. These results clearly demonstrate the effectiveness of MADAR's distribution-aware sample selection and the effectiveness of MADAR-U and MADAR$^{\theta}$-U in leveraging limited resources.

In contrast, prior methods such as ER, AGEM, GR, RtF, and BI-R fail to exceed 30% $\overline{\mathbf{AP}}$, while more advanced techniques like TAMiL and MalCL achieve only 38.2% and 54.8%, respectively. Even iCaRL, designed specifically for Class-IL, achieves only 64.6%, further highlighting the significant advantage of our approaches in the malware domain.

In the Class-IL setting of AZ-Class, we consider 11 tasks. The summary results of all experiments are provided in Table V, with comparisons against the *None* and *Joint* baselines, which achieve $\overline{\mathbf{AP}}$ scores of 26.4% and 94.2%, respectively. Figure 8 illustrates the progression of average accuracy across tasks, showing how each method performs relative to the *Joint* baseline.

As shown in Table V, among the prior methods, iCaRL performs best across most budget configurations, outperforming techniques such as MalCL, TAMiL, ER, AGEM, GR, RtF, and BI-R. Therefore, we focus on comparing MADAR's performance with iCaRL. At a low budget of 100 samples, iCaRL and GRS achieve less than 44% $\overline{\mathbf{AP}}$, while all MADAR methods surpass 57%. In particular, MADAR-R and MADAR$^{\theta}$-R achieve $\overline{\mathbf{AP}}$ scores of 59.4% and 58.8%, respectively, highlighting their efficiency at low-resource levels.

As the budget increases, all methods improve, but MADAR-U consistently delivers the best results. At a budget of 1K, MADAR-U achieves the highest $\overline{\mathbf{AP}}$ at 70.4%, followed closely by MADAR$^{\theta}$-U at 70.1%. This trend continues as budgets increase, with MADAR-U outperforming all other methods, achieving $\overline{\mathbf{AP}}$ scores of 89.8% at 10K and 91.5% at 20K. Compared to GRS, which achieves 90.1% at 20K, and iCaRL, which trails at 84.6%, MADAR-U demonstrates clear superiority. MADAR$^{\theta}$-U also performs GRS reaching 90.7% at 20K.

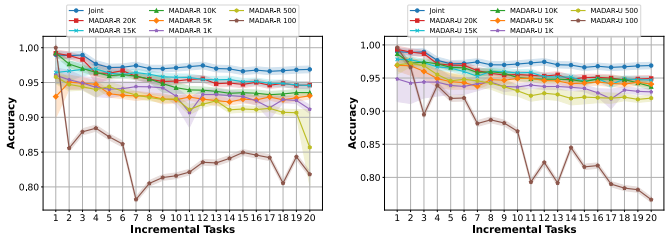
In summary, our experiments demonstrate the effectiveness of MADAR's distribution-aware replay techniques in the Class-IL setting for both EMBER and AZ datasets. While GRS shows significant improvement with larger budgets, MADAR's uniform variants consistently outperform it across all budget levels. These results underscore MADAR's ability to mitigate catastrophic forgetting and enhance malware classification performance, even in resource-constrained environments.

C. Task-IL

In this set of experiments with EMBER, we consider 20 tasks, with 5 new classes added in each task. The summarized results are shown in Table VI, where performance is reported as the average accuracy over all tasks ($\overline{\mathbf{AP}}$). It is worth

TABLE VI: Summary of Results for EMBER Task-IL Experiments.

Group	Method	Budget						
		100	500	1K	5K	10K	15K	20K
Baselines	Joint				97.0±0.3			
	None				74.6±0.7			
	GRS	86.9±0.3	87.4±0.3	93.6±0.3	94.4±0.2	94.7±0.3	94.9±0.1	95.0±0.1
Prior Work	TAMiL [14]	72.8±0.1	81.5±0.3	86.9±0.2	88.1±0.3	90.3±0.1	93.2±0.3	94.2±0.7
	ER [26]	67.4±0.3	84.9±0.2	89.5±0.5	93.9±0.2	94.8±0.2	95.2±0.1	95.4±0.1
	AGEM [27]	79.6±0.2	81.7±0.2	83.8±0.4	84.9±0.2	86.1±0.2	88.9±0.2	89.3±0.1
	GR [18]				79.8±0.3			
	RtF [41]				77.8±0.2			
	BI-R [15]				87.2±0.3			
MADAR	MADAR-R	92.1±0.2	92.3±0.9	93.8±0.2	94.2±0.1	94.8±0.2	**95.7±0.2**	**95.6±0.1**
	MADAR-U	**93.4±0.2**	**93.7±0.3**	**93.9±0.3**	**94.8±0.2**	**95.6±0.1**	**95.7±0.1**	**95.8±0.2**
	MADAR$^{\theta}$-R	**93.1±0.2**	**93.3±0.1**	**93.6±0.1**	94.3±0.1	94.6±0.2	94.8±0.2	94.7±0.3
	MADAR$^{\theta}$-U	**93.2±0.1**	93.1±0.2	**93.8±0.2**	**94.4±0.1**	**94.8±0.1**	**95.3±0.2**	**95.5±0.3**



(a) MADAR Ratio (b) MADAR Uniform

(c) MADAR$^{\theta}$ Ratio (d) MADAR$^{\theta}$ Uniform

Fig. 9: EMBER Task-IL: Comparison of the MADAR-R, MADAR-U, MADAR$^{\theta}$-R, and MADAR$^{\theta}$-U with Joint baseline.

noting that Task-IL is considered the easiest scenario in continual learning [15], [32]. The *None* and *Joint* methods serve as informal lower and upper bounds, achieving $\overline{\mathbf{AP}}$ scores of 74.6% and 97%, respectively. Figure 9 visualizes the progression of average accuracy across tasks, highlighting comparisons with the *Joint* baseline.

As shown in Table VI, ER consistently outperforms TAMiL, A-GEM, GR, RtF, and BI-R across all budget configurations and even surpasses GRS in some cases. However, MADAR variants significantly outperform all prior methods, particularly under lower budget constraints (100–1K). For example, MADAR-U achieves the highest $\overline{\mathbf{AP}}$ of 93.4% and 93.7% at budgets of 100 and 1K, respectively, outperforming GRS and all other approaches. Similarly, MADAR$^{\theta}$-U performs competitively, with $\overline{\mathbf{AP}}$ of 93.2% at a 100 budget and 93.8% at 1K.

As the budget increases, the performance gap among MADAR, ER, and GRS narrows; however, MADAR variants continue to either outperform or match other techniques. Notably, the MADAR-U variant of MADAR achieves the best overall performance at a budget of 20K, attaining a $\overline{\mathbf{AP}}$ of 95.8%, which closely approaches the *Joint* baseline. Similarly, MADAR-R yields $\overline{\mathbf{AP}}$ of 95.6% at 20K.

Task-IL for AZ consists of 20 tasks, each with 5 non-overlapping classes. The results are summarized in Table VII and benchmarked against the *None* and *Joint* baselines, which achieve $\overline{\mathbf{AP}}$ values of 74.5% and 98.8%, respectively. Figure 10 illustrates the progression of average accuracy across tasks, showing how each method performs relative to the *Joint* baseline.

As seen in Table VII, ER consistently outperforms TAMiL, AGEM, GR, RtF, BI-R, and GRS across most budget configurations, making it a strong baseline for comparison. At a low budget of 100 samples, MADAR-U achieves $\overline{\mathbf{AP}}$ of 88.1%, which is 4.5% higher than ER's performance. Similarly, MADAR$^{\theta}$-U demonstrates competitive performance, achieving 87.9% at the same budget.

As the budget increases, MADAR-U continues to deliver the best performance, reaching $\overline{\mathbf{AP}}$ scores of 94.5% at a 1K budget and 98.1% at a 10K budget, outperforming all other methods, including ER and GRS. At the highest budget of 20K, MADAR-U achieves an $\overline{\mathbf{AP}}$ of 98.7%, surpassing ER by 1.2% and nearly matching the *Joint* baseline. Notably, MADAR$^{\theta}$-U also performs well, achieving 98.1%. In contrast, MADAR-R and MADAR$^{\theta}$-R perform slightly lower but remain competitive, with $\overline{\mathbf{AP}}$ values of 97.9% and 96.9% at a 20K budget, respectively. These results indicate that ratio-based methods, while effective, are slightly less robust than uniform sampling in this scenario.

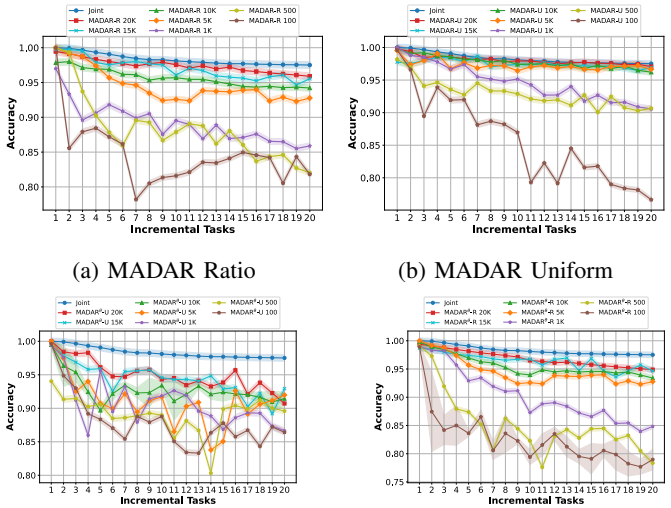
In summary, MADAR-U and MADAR$^{\theta}$-U consistently demonstrate better performance across most of the budget levels, particularly excelling at low-resource settings and achieving near-optimal results at higher budgets. These findings underscore the effectiveness of MADAR framework in Task-IL scenarios and their ability to approach joint-level performance with significantly fewer resources.

D. Analysis and Discussion

Our results demonstrate that MADAR yields markedly better performances compared to previous methods for both the EMBER and AZ datasets across all CL settings. This clearly indicates that distribution-aware replay is effective in preserving

TABLE VII: Summary of Results for AZ Task-IL Experiments.

Group	Method	Budget						
		100	500	1K	5K	10K	15K	20K
Baselines	Joint				98.8±0.2			
	None				74.5±0.2			
	GRS	85.2±0.1	89.2±0.2	90.8±0.1	91.6±0.2	93.5±0.1	93.9±0.1	95.2±0.1
Prior Work	TAMiL	80.5±0.4	85.3±0.6	91.5±0.2	92.1±0.1	93.5±0.1	94.0±0.2	94.8±0.2
	ER	83.6±0.2	90.2±0.1	92.3±0.3	95.6±0.1	96.2±0.1	96.8±0.2	97.5±0.2
	AGEM	76.7±0.5	82.8±0.2	85.3±0.1	85.6±0.2	86.7±0.2	88.9±0.2	91.3±0.3
	GR				75.6±0.2			
	RtF				74.2±0.3			
	BI-R				85.4±0.2			
MADAR	MADAR-R	86.0±0.3	90.3±0.2	92.4±0.1	95.8±0.2	96.7±0.1	97.1±0.1	97.9±0.2
	MADAR-U	**88.1±0.3**	**92.9±0.2**	**94.5±0.3**	**97.2±0.2**	**98.1±0.1**	**98.5±0.1**	**98.7±0.1**
	MADAR$^\theta$-R	87.3±0.3	**90.6±0.2**	93.2±0.2	95.7±0.2	95.9±0.1	96.6±0.1	96.9±0.1
	MADAR$^\theta$-U	**87.9±0.2**	**90.8±0.2**	**93.6±0.1**	**96.2±0.3**	**97.2±0.2**	**97.5±0.2**	**98.1±0.1**



(a) MADAR Ratio (b) MADAR Uniform

(c) MADAR$^\theta$ Ratio (d) MADAR$^\theta$ Uniform

Fig. 10: AZ Task-IL: Comparison of the MADAR-R, MADAR-U, MADAR$^\theta$-R, and MADAR$^\theta$-U with Joint baseline.

the stability of a CL-based system for malware classification, while prior CL techniques largely fail to achieve acceptable performance.

MADAR in low-budget settings. In Domain-IL, MADAR achieves competitive performance even with a 1K budget, surpassing prior work by over 3 percentage points in EMBER and AZ. At higher budgets, ratio-based selection (MADAR-R and MADAR$^\theta$-R) achieves near Joint baseline performance (96.4% in EMBER and 97.3% in AZ) while using significantly fewer resources. This demonstrates MADAR's efficiency in leveraging limited samples to achieve robust classification.

MADAR is both effective and scalable. Traditional CL methods, including ER and AGEM, experience significant performance degradation as tasks increase. In contrast, MADAR maintains high accuracy across 20 Task-IL tasks, with MADAR-U achieving 95.8% in EMBER and 98.7% in AZ at a 20K budget, nearly matching the *Joint* baseline.

Ratio vs. Uniform Budgeting. A consistent trend across our experiments is that ratio-based selection performs best in Domain-IL, whereas uniform-based selection is superior in Class-IL and Task-IL. MADAR$^\theta$-U reaches 91.5% in AZ at 20K, significantly outperforming iCaRL and TAMiL. Furthermore, in EMBER, MADAR-U achieves near *Joint* baseline performance at just a 5K budget, underscoring the effectiveness of uniform selection in class-incremental settings. Intuitively, this makes sense because ratio budgeting for binary classification in the Domain-IL setting naturally captures the contributions of each family to the overall malware distribution. Additionally, since there are many small families in the Domain-IL datasets, uniformly sampling from them consumes budget while offering little improvement in malware coverage. In contrast, our Class-IL and Task-IL experiments perform classification across families, which is better supported by Uniform budgeting to maintain class balance and ensure coverage over all families. Moreover, in most settings we can leverage efficient representations using MADAR$^\theta$ to scale the approach regardless of feature dimension without significant loss of performance.

GRS remains a strong baseline at high budgets. While MADAR consistently outperforms GRS in low-resource settings, GRS performs comparably at higher budgets, particularly in Domain-IL. This suggests that distribution-aware replay is most impactful when the number of available samples per class is limited, whereas uniform selection provides sufficient representation at larger budgets.

VIII. Conclusion

In this paper, we propose MADAR, a framework for distribution-aware replay in continual learning specially designed for the challenging setting of malware classification. Our comprehensive evaluation across Domain-IL, Class-IL, and Task-IL scenarios against Windows executable (EMBER) and Android application (AZ) datasets demonstrates that distribution-aware sampling is helpful for effective CL in malware classification. As malware and goodware continue to evolve, these insights steer continual learning towards strategic, resource-efficient methods, ensuring model effectiveness amid the constantly shifting landscape of cybersecurity threats.

Acknowledgements

This research was funded in part by the National Science Foundation under Grant no. 2422241. The authors acknowl-

edge Research Computing [42] at the Rochester Institute of Technology for providing computational resources and support.

REFERENCES

[1] E. Kovacs, "FireEye MalwareGuard uses machine learning to detect malware," https://www.securityweek.com/fireeye-malwareguard-uses-machine-learning-detect-malware/, 2018.

[2] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending Transcend: Revisiting malware classification in the presence of concept drift," in *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[3] Y. Chen, Z. Ding, and D. Wagner, "Continuous learning for Android malware detection," in *USENIX Security*, 2023.

[4] D. Maiorca, G. Giacinto, and I. Corona, "A pattern recognition system for malicious PDF files detection," in *MLDM Workshop*, 2012.

[5] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, 2014.

[6] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," in *USENIX Security*, 2021.

[7] A. Abusnaina, A. Anwar, S. Alshamrani, A. Alabduljabbar, R. Jang, D. Nyang, and D. Mohaisen, "Systematically evaluating the robustness of ml-based iot malware detection systems," in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2022.

[8] AV-TEST, "Malware statistics and trends report," https://www.av-test.org/en/statistics/malware/, 2025.

[9] VirusTotal, "VirusTotal – Stats," https://www.virustotal.com/gui/stats, 2025.

[10] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, 1999.

[11] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, 1995.

[12] J. Park, A. Ji, M. Park, M. S. Rahman, and S. E. Oh, "MalCL: Leveraging gan-based generative replay to combat catastrophic forgetting in malware classification," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

[13] Z. Wang, Y. Li, L. Shen, and H. Huang, "A unified and general framework for continual learning," in *International Conference on Learning Representations (ICLR)*, 2024.

[14] P. Bhat, B. Zonooz, and E. Arani, "Task-aware information routing from common representation space in lifelong learning," in *International Conference on Learning Representations (ICLR)*, 2023.

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

[16] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *CVPR*, 2017.

[17] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Gradient based sample selection for online continual learning," *NeurIPS*, 2019.

[18] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," *NeurIPS*, 2017.

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

[20] M. S. Rahman, S. E. Coull, and M. Wright, "On the limitations of continual learning for malware classification," in *First Conference on Lifelong Learning Agents (CoLLAs)*, 2022.

[21] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," *arXiv:1804.04637*, 2018.

[22] R. J. Joyce, G. Miller, P. Roth, R. Zak, E. Zaresky-Williams, H. Anderson, E. Raff, and J. Holt, "EMBER2024–a benchmark dataset for holistic evaluation of malware classifiers," *arXiv preprint arXiv:2506.05074*, 2025.

[23] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *International Conference on Data Mining (ICDM)*, 2008.

[24] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *MSR*, 2016.

[25] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato, "On tiny episodic memories in continual learning," in *ICML*, 2019.

[26] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *NeurIPS*, 2019.

[27] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with A-GEM," in *ICML*, 2019.

[28] J. S. Smith, L. Valkov, S. Halbe, V. Gutta, R. Feris, Z. Kira, and L. Karlinsky, "Adaptive memory replay for continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 3605–3615.

[29] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 40, no. 12, pp. 2935–2947, 2017.

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

[31] S. K. Amalapuram, B. R. Tamma, and S. S. Channappayya, "Spider: A semi-supervised continual learning-based network intrusion detection system," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 571–580.

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

[33] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online anti-malware engines," in *USENIX Security*, 2020.

[34] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar, "Better malware ground truth: Techniques for weighting anti-virus vendor labels," in *ACM AISec*, 2015.

[35] K. Berlin, D. Slater, and J. Saxe, "Malicious behavior detection using Windows audit logs," in *ACM AISec*, 2015.

[36] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, 1985.

[37] S. Zhang and R. S. Sutton, "A deeper look at experience replay," arXiv:1712.01275, 2017.

[38] L. Puggini and S. McLoone, "An enhanced variable selection and isolation forest based methodology for anomaly detection with OES data," *Engineering Applications of Artificial Intelligence*, vol. 67, 2018.

[39] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan, "Remind your neural network to prevent catastrophic forgetting," in *ECCV*, 2020.

[40] O. Ostapenko, T. Lesort, P. Rodríguez, M. R. Arefin, A. Douillard, I. Rish, and L. Charlin, "Foundational models for continual learning: An empirical study of latent replay," *arXiv:2205.00329*, 2022.

[41] G. M. van de Ven and A. S. Tolias, "Generative replay with feedback connections as a general strategy for continual learning," *arXiv preprint arXiv:1809.10635*, 2018.

[42] Rochester Institute of Technology, "Research computing services," 2024. [Online]. Available: https://www.rit.edu/researchcomputing/