5-2024

Continual Learning for an Ever Evolving and Intelligent Malware Classification System

Mohammad Saidur Rahman
mr6564@rit.edu

Continual Learning for an Ever Evolving and Intelligent Malware Classification System

by

Mohammad Saidur Rahman

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences

Rochester Institute of Technology
Rochester, New York
May 2024

Continual Learning for an Ever Evolving and Intelligent Malware Classification System

by

Mohammad Saidur Rahman

Committee Approval:

We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computing and Information Sciences.

Matthew Wright, Ph.D. Date
Dissertation Advisor

Scott E. Coull, Ph.D. Date
Dissertation Committee Member

Qi Yu, Ph.D. Date
Dissertation Committee Member

Rui Li, Ph.D. Date
Dissertation Committee Member

Sean Hansen, Ph.D. Date
Dissertation Defense Chairperson

Certified by:

Pengcheng Shi, Ph.D. Date
Ph.D. Program Director, Computing and Information Sciences

Continual Learning for an Ever Evolving and Intelligent Malware Classification System

by

Mohammad Saidur Rahman

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences Ph.D. Program in Computing and Information Sciences
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

Malware classification poses unique challenges for continual learning (CL) systems, driven by the daily influx of new samples and the evolving nature of malware threats that exploit new vulnerabilities. Antivirus vendors encounter hundreds of thousands of unique software pieces daily, encompassing both malicious and benign files. Over its operational life, a malware classifier can accumulate more than a billion samples. Training malware classification system with only new samples and classes leads to catastrophic forgetting (CF), where the system forgets previously learned data distribution. While retraining with all old and new samples effectively combats CF, it is computationally expensive and necessitates storing vast amounts of older software and malware samples. Employing sequential training with CL strategies offers a potential solution to mitigate these challenges by reducing both training and storage demands. However, the adoption of CL for malware classification has not been extensively explored. This work represents the first in-depth examination of CL not just in the realm of malware classification, but also more broadly within the cybersecurity domain.

In this thesis, first we systematize the malware classification pipeline through the lens of three continual learning scenarios: Domain Incremental Learning (Domain-IL), Class Incremental Learning (Class-IL), and Task Incremental Learning (Task-IL), detailed in Chapter 3. Our objective is to bridge the research gap between existing CL literature and the specific needs of malware classification. We undertake a thorough examination of state-of-the-art CL methods within the frameworks of these three CL scenarios. Chapter 4 presents an in-depth

exploration of the catastrophic forgetting phenomenon in the context of malware classification. We analyze the applicability and performance of 11 leading CL techniques across three categories – regularization, replay, and replay with exemplars, initially developed for computer vision tasks, to uncover if they can also mitigate catastrophic forgetting in malware domain. Contrary to expectations, our findings indicate that none of the CL approaches tested successfully mitigate catastrophic forgetting in malware classification systems, pointing to a significant research opportunity in this domain.

The unexpected results presented in Chapter 4 prompted a detailed exploratory analysis of the EMBER dataset [10], which comprises Windows malware and benign software samples, in Chapter 5. This analysis revealed significant diversity within malware data distributions, both across and within malware *families*. Drawing on these insights, we developed MADAR– *M*alware *A*nalysis with *D*iversity-*A*ware *R*eplay. This innovative strategy for malware classification adopts a diversity-aware, replay-based approach, integrating a mix of representative and novel samples into the training regimen to enhance the stability of the model to retain learned information and identify emerging malware threats, even with limited memory budget. Moreover, we have created two new benchmarks using Android malware from the AndroZoo repository [8] for testing in both Domain-IL (*AZ-Domain*) and Class-IL (*AZ-Class*) scenarios. The results from these benchmarks underscore the effectiveness of the MADAR framework, establishing it as the new state-of-the-art and demonstrating its enhanced performance over existing leading CL methods in adapting to realistic shifts in malware data distribution.

In Chapter 6, we conclude with promising future research direction to advance continual learning research for an ever evolving and intelligent malware classification systems, focusing on adaptability to evolving threats and tackling challenges relevant to both industry and academia.

Acknowledgments

The completion of this dissertation and the research behind it would not have been possible without the guidance, support, and encouragement of many individuals. I would like to take this opportunity to express my heartfelt gratitude to them.

Firstly, I am profoundly grateful to my advisor, Professor Dr. Matthew Wright, who believed in me, gave me the opportunity to learn from him, and guided me toward becoming an independent researcher. If he had not given me this chance, I would not be where I am today. His continuous inspiration, encouragement, feedback, and invaluable advice have shaped me as a researcher over the years. He is an exemplary professor and a remarkable human being. He went above and beyond to help and support me in difficult situations.

Afterwards, I am immensely grateful to Dr. Scott Coull for introducing me to malware analysis research during my Data Science Internship in 2020. His guidance, feedback, and industry perspectives made our work even more practical and timely. The guidance from him and my mentors at the Mandiant Data Science team, Dr. Phil Tully and Dr. Ethan Rudd, inspired me to focus my dissertation on this area, fostering a collaboration that continues to this day.

I would like to express my gratitude to my PhD committee members, Dr. Qi Yu and Dr. Rui Li, for their intellectual support and feedback. As prominent machine learning experts, their insights have greatly improved our work. I would also like to thank our PhD directors, Dr. Pengcheng Shi and Dr. Yin Pan, for their support and advice, which have helped me become a better PhD student. In addition, I would like to express my gratitude to Dr. Ziming Zhao for his mentorship and support over the years.

In addition, I would like to express my heartfelt gratitude to those who taught, inspired, encouraged, and supported me in pursuing my higher education. The complete list would be too long, but I want to acknowledge a few who are directly related to this endeavor from the MIS department of the University of Dhaka: Md. Ariful Islam, Dr. Hasibur Rashid, Dr. K. M. Salah Uddin, and Dr. Md. Rakibul Hoque. I would like to express my gratitude to all the teachers who have taught, inspired, and supported me throughout my life. I am and always will be grateful to them. I am committed to dedicating my lifetime of service to making my parents and teachers proud.

The last but not the least, I would like to thank my excellent mentors, collaborators, friends, and lab siblings throughout the graduate school for their support and help – Mohsen Imani, Payap Sirinam, Saniat Sohrawardi, Nate Mathews, Se Eun Oh, Kelly Wu, Shaikh Akib Shahriyar, Luke Kurlandski, Sovantharith Seng, Kantha Girish Gangadhara, and many more.

As I write this acknowledgment, I miss the beloved person who would have been happiest to see me graduate with a PhD — the person who was full of love and care for me, who gave up her own happiness for the sake of mine and my well-being. She never lost hope for me, never said no to my endeavors, and did everything within her power to ensure I had everything I needed. She believed in my dream of pursuing higher education in the US — my loving and caring mother, who bravely fought COVID and left us with countless cherished memories. I cannot adequately express my gratitude for all she did for me. All I can say is, "I exist because of you." I had planned for us to celebrate this graduation together. I would like express my heartfelt gratitude to my beloved father who abandoned all his happiness to provide for us, who did not even buy an expensive shirt to save money for my education, and worked so hard to make sure I have everything. Only being thankful to my parents is not enough who taught me being a better human being with value sense.

Finally, I would like to express my tremendous gratitude to my beloved wife, Mt. Tahmina Akter, who stood by me through the ups and downs of my Ph.D. journey, including the most challenging time of my life — the loss of my mother. I consider myself incredibly blessed and fortunate to have her in my life. Her unconditional love and care for the past 11 years, and her hard work to make our living space a home, are beyond my capability to match. I am profoundly grateful for her, and I cherish every moment we share together. I also thank my two sisters for their unconditional love and support at every step of my life. Everything in my life would be meaningless without my family members.

To my Beloved ***Parents****, my Loving* ***Wife****, and All the Great* ***Educators*** *who Made me Who I am Today!*

Contents

List of Figures

List of Tables

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."

— Alan Turing (1912 - 1954)

1

Introduction

Over the past few years, the fields of machine learning (ML) and deep learning (DL) have revolutionized how we approach complex challenges across various disciplines. These technologies have been instrumental in advancing computer vision [54,71,74], enhancing natural language understanding [37,144], improving speech recognition and audio processing [1,55,107], and bolstering cybersecurity defenses [61,110]. As a result, ML and DL have become foundational components in these fields, driving the development of innovative and practical intelligent systems. Their integration has not only solved existing problems but also opened new avenues for research and application, setting the stage for the next generation of technological advancements.

In the domain of cybersecurity, the integration of ML and DL has marked a significant evolution in protecting computer and network systems [15,36,61,77,93,110]. In particular, DL advancements have led to notable enhancements in identifying malicious websites [133], classifying malicious phone calls [77], detecting malware [33,77,132], and recognizing network intrusions [46, 93]. Research specific to malware detection and classification has applied ML and DL techniques, achieving tremendous success in a wide range of problem domains including Windows malware [35,135], PDF malware [72,84], malicious URLs [75,133], and Android malware [11,49,98]. The cross-domain applications of ML and DL showcase their versatility and power in solving complex problems, making them indispensable tools in the modern technological landscape.

DL models are fundamentally inspired by the structure of biological neurons, a concept highlighted in seminal works such as those by Rosenblatt [121] and Fukushima [45]. However, the manner in which these artificial systems learn diverges significantly from the learning processes observed in humans. Human brains exhibit a remarkable capability for continuous, sequential learning, seamlessly integrating new information and skills without substantially compromising previously established knowledge. Contrastingly, DL models, once trained on a predetermined dataset, essentially become static. Their ability to adapt or learn from new data at this point is limited; they undergo a phase of evaluation where their performance is gauged against previously unseen data, under the assumption that this new data mirrors the statistical properties of the training set. This approach presumes a fixed distribution for both training and testing datasets, a premise often at odds with real-world scenarios where data can evolve or change over time. To maintain accuracy and relevance in the face of such shifts, DL models necessitate periodic retraining, a process that involves revisiting the

training phase with new or updated data to accommodate these changes.

In the context of malware classification and detection, the continuous emergence of varied and novel malicious software (i.e., malware) represents a significant challenge in cybersecurity [12]. Developing systems capable of detecting and classifying these new types of malware with minimal human intervention remains a critical area of research [30,75,98,135]. ML and DL-based systems are extensively investigated in both academic studies and practical applications to detect and categorize malware, covering various types such as Windows malware [35, 63, 135], PDF malware [72, 84], malicious URLs [75, 133], and Android malware [11,98]. These approaches typically involve training models on known malware samples and deploying them with the expectation that they will be effective against new, unseen threats. However, the ever-evolving nature of malware, coupled with the continuous changes in benign software (referred to as goodware), presents a dynamic and challenging problem.

To accommodate shifts in the data distribution over time, the model needs to be retrained regularly to maintain its effectiveness. Unfortunately, the speed with which new malware and goodware are produced results in large datasets that can be both costly to maintain and difficult to train on. For example, the AV-TEST institute registers more than 450,000 unique pieces of malware and "potentially unwanted applications (PUA)" each day [12], while VirusTotal, a crowdsourced antivirus scanning service, regularly receives more than 1 million unique pieces of software each day [145]. Over the lifetime of a malware classification model, these daily feeds can result in datasets containing upwards of a billion unique training samples spanning multiple years. Given the realities of training these models, antivirus companies must decide whether to:

- remove some older samples from the training set, at the risk of allowing attackers to revive older malware instead of writing new ones;
- train less frequently, at the cost of not adjusting to changes in the distribution; or
- expend tremendous effort to frequently retrain over all the data which will incur incremental computational cost.

The challenges mentioned above can be addressed by an evolving and intelligent malware classification system, capable of dynamically adjusting to changes in data distribution with-

out significant storage and computational demands. Fortunately, continual learning (CL) presents a promising solution by allowing the incremental addition of new information and adaptation to shifts in data distribution, all while avoiding the need for large datasets and excessive training overhead [79,114,139]. Integrating CL techniques into the training pipeline of a malware classification system could offer significant advantages, such as:

- reducing or eliminating the need to retain past data;
- lessening the necessity for access to previous datasets;
- maintaining the privacy of training data while enabling model updates; and
- cutting down on computational costs.

However, implementing continual learning (CL) faces a significant challenge: catastrophic forgetting (CF), where a ML model forgets previously learned information [44,87,113]. This issue is closely related to the *stability-plasticity* dilemma in neural networks [2,83,91], which represents the balance between retaining existing knowledge (i.e., stability) and learning new information (i.e., plasticity). As a model learns to identify new threats, it needs to remain adaptable or *plastic*. However, this adaptability can compromise the model's *stability*, leading to the loss of previously learned information, or CF. Traditional neural networks often struggle with this balance. Updating these networks with new data can unintentionally alter the weights that encode older knowledge, effectively *overwriting* vital information. This issue is particularly problematic in the field of malware detection and classification, where recognizing both new and old threats is crucial. Therefore, addressing CF is essential to maintain the efficacy of CL models amidst the constantly evolving cybersecurity landscape.

1.1 Research Contribution

The challenge of catastrophic forgetting has been extensively addressed in the field of computer vision, with significant research and applications exploring solutions using datasets such as MNIST [73], CIFAR10, and CIFAR100 [70], as well as ImageNet [124]. However, the relevance and effectiveness of these approaches in the context of malware classification remain

unexplored. This thesis aims to bridge this gap by developing a continual learning system specifically designed for malware classification, addressing the dynamic and ever-evolving nature of malware and benign software. Our work positions us among the first to explore this critical research intersection between continual learning and the malware domain.

In this thesis, we propose a systematization of the malware classification pipeline through the lens of three continual learning scenarios [141, 142]: Domain Incremental Learning (Domain-IL), Class Incremental Learning (Class-IL), and Task Incremental Learning (Task-IL). This systematization is presented in Chapter 3. Domain-IL addresses changes in data distribution, typically over time, and in our context, it involves adapting to variations in both benign software and malware while maintaining the accuracy of classifying older samples. For Domain-IL, our focus is on the binary classification of software as either malicious or benign, with an emphasis on handling data distribution shifts over time. Class-IL poses a significant challenge in continual learning by requiring the model to accommodate an expanding array of classes. This scenario demands the ability to identify newly introduced categories (such as cats, dogs, apples, bananas) without the benefit of distinct sub-task (i.e., pets and fruits) indications. On the other hand, Task-IL is considered the more straightforward approach, where the focus is on mastering new distinct tasks (i.e., apples vs. bananas) while maintaining proficiency in previously learned tasks (cats vs. dogs). Here, each task is well-defined and segregated (for instance, categorizing subjects as either fruits or pets).

For both Class-IL and Task-IL, we explore malware *family* classification, where each malware sample is assigned to a specific family based on characteristics like its codebase, capabilities, and structure. In Class-IL, we progressively introduce new malware families, reflecting the continuous discovery of new malware types in the real world. Task-IL also adds new families over time but does so within clearly defined classification tasks such as adware, ransomware, trojans, spyware, and so on. These three scenarios encapsulate key challenges faced by the anti-malware industry.

In Chapter 4 of this thesis, we delve into a comprehensive study of the catastrophic forgetting phenomenon within malware classification. We assess the effectiveness and relevance of 11 state-of-the-art continual learning (CL) techniques across three categories – regularization, replay, and replay-with-exemplars. These techniques, initially devised for computer vision tasks, are tested for their ability to mitigate catastrophic forgetting in the malware domain.

Our investigation utilizes two large-scale real world malware datasets, Drebin [11] and EMBER [10], to explore both binary and multi-class malware classification challenges. Surprisingly, our study finds that none of the CL methods examined effectively prevent catastrophic forgetting in malware classification systems. This marks the first comprehensive study of CL not only in malware classification but also more broadly within the cybersecurity field. The research also provides evidence that incorporating some form of replay is essential for reducing catastrophic forgetting, echoing findings from previous studies [139, 141].

Additionally, we identify the *partial joint replay* (PJR) approach – later defined as *global reservoir sampling* (GRS) in Chapter 5 – as a promising method. By replaying a percentage of past data alongside new data, PJR significantly mitigates catastrophic forgetting while also reducing training costs. We hypothesize that the unique challenges of malware data distribution, such as stringent feature semantics and the evolving nature of malware, may explain the discrepancies between our findings and existing CL literature. These characteristics potentially hinder the effectiveness of generative, distillation, and regularization-based approaches, as they may not fully capture the domain's complexity. Nevertheless, these observations also suggest new directions for future research in this area and these insights highlight the need for further exploration of CL in dynamic and complex settings.

The surprising findings of the Chapter 4 motivated us to conduct a comprehensive exploratory analysis of the EMBER dataset [10], which includes samples of Windows malware and benign software (goodware), to uncover unique characteristics and complexities within malware data distributions, presented in Chapter 5. Our analysis reveals a notable churn in the representation of malware families over time. For instance, of the 913 families identified in January, only 551 continue into February, with 425 new families emerging. This churn highlights potential challenges in maintaining training data continuity, which could exacerbate catastrophic forgetting, thus emphasizing the need for adaptive continual learning strategies in the malware domain. Moreover, many malware families exhibit complex distributional patterns in the feature space, adding further diversity within classes. 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 [65]. Our findings underscore the diversity in malware, both between and even within *families*, or groups of related malware.

Leveraging these insights, we introduce MADAR– *M*alware *A*nalysis with *D*iversity-*A*ware *R*eplay, a strategy specifically designed for malware classification that utilizes a diversity-aware, replay-based approach. This method incorporates a strategic mix of representative and novel samples (i.e., outliers) into the training process, enhancing the model's ability to maintain its knowledge base and recognize new malware variants, even under memory constraints. To identify these crucial novel samples, our technique employs Isolation Forests (IF) [81]. MADARfeatures two variations: Isolation Forest-based Sampling (IFS), which relies on the model's input features, and Anomalous-Weights-based Sampling (AWS), which opts for a more compact representation using model weights. We rigorously evaluate these methods on the EMBER dataset across two CL scenarios in malware classification: Domain-IL and Class-IL, aligning with previous studies [109]. Furthermore, we have developed two new benchmarks of Android malware from the AndroZoo repository [8] to conduct experiments in both Domain-IL (*AZ-Domain*) and Class-IL (*AZ-Class*) settings. Our findings across these datasets demonstrate MADAR's effectiveness, outperforming existing state-of-the-art CL approaches in the face of realistic shifts in the malware data distribution. 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.

In Chapter 6, we conclude by outlining several promising directions for future research. Our goal is to advance the development of continual learning systems that are adaptable to the rapidly evolving landscape of malware threats and capable of addressing complex challenges that span both industry and academia. Specifically, we see potential in exploring novel algorithms that enhance model adaptability without compromising on efficiency, developing techniques to better manage the balance between retaining old knowledge and acquiring new information, and investigating the integration of domain-specific knowledge into learning processes. Additionally, the creation of more realistic and challenging benchmarks could significantly contribute to pushing the boundaries of current methodologies. By focusing on these areas, we aim to draw a path forward for continual learning in the malware classification domain, ultimately contributing to more resilient and intelligent cybersecurity defenses.

"Look deep into nature, and then you will understand everything better."

- Albert Einstein

2

Background & Preliminaries

2.1 Continual Learning

Continual learning (CL), a branch of Machine Learning (ML), addresses the challenge of learning from a sequence of evolving data distributions and generalizing across all encountered distributions [51]. The current static ML training paradigm assumes that the training data are independent and identically distributed (i.i.d.), which contrasts with the practical learning setting where both training and testing data distributions are generally dynamic. CL attempts to overcome the limitations of the current static ML training paradigm, in which an ML model is trained and deployed under the assumption that the training and test data belong to similar distributions, with the expectation that the model will generalize to unseen distributions. Broadly, research on CL addresses two significant challenges inherent in this traditional static ML training paradigm.

Firstly, the standard training paradigm for ML models, especially stochastic gradient descent [117], encounters difficulties in adapting to changing and evolving data distributions. This challenge leads to a phenomenon termed as *catastrophic forgetting* or *catastrophic interference* [44, 87, 113]. Fundamentally, the neural network becomes too adapted to the most recent data it has seen, neglecting what it learned from earlier data. This results in significantly reduced effectiveness, particularly when evaluated on test sets that include a variety of samples belonging to earlier learned data distributions, as the neural network fails to maintain a balance between new and old information.

Secondly, the constant changes in real-world data distribution indicates that addressing non-stationarity is essential for developing evolving, practical, and intelligent systems. Depending on the problem space, various factors can contribute to the constantly shifting nature of data, including climate change, urban development, and the creation of digital content. In security applications such as malware analysis, the reason behind the constant shift in data distribution is the evolution of malware and the release of benign software. These changes are usually gradual and interconnected. The experimental setting becomes even more challenging when we consider that the intelligent system is deployed in a resource-constrained environment such as various internet-of-things (IoT) devices and and can only process a small fraction of the available data.

The research gap between the static i.i.d. assumption used in many machine learning meth-

ods and the ever evolving nature of the real-world data distributions warrants for novel and more sophisticated solutions [94]. Current strategies, such as storing and randomly resampling data, are limited and not suitable for all situations. Therefore, the challenge of non-stationarity highlights the crucial importance of continual learning, marking it as a key area for exploration and application in both theoretical and practical fields. In the CL training framework, a ML model deals with a series of tasks, denoted as $t_1, t_2, ..., t_T$, each linked to a unique data distribution represented as $p(x, y|t_i)$. For each task, the model has a set of parameters, indicated as $\theta_{t_1}, \theta_{t_2}, ..., \theta_{t_T}$. Initially, the model is trained on task t_1, which adjusts its parameters to θ_{t_1}. Upon encountering a new task, say θ_{t_2}, the model undergoes retraining, leading to an update in its parameters to θ_{t_2}. However, this retraining process may result in *catastrophic forgetting* of the data distribution $p(x, y|t_{i-1})$ related to the previous task, as the model may lose the knowledge it gained earlier. As such, the fundamental goal of CL is to develop an intelligent CL system capable of adapting to new tasks while preserving the knowledge from earlier ones.

2.2 Continual Learning Training Protocols

In continual learning, models sequentially learn tasks $t_1, t_2, ..., t_T$, each with its distinct data distribution $p(x, y|t_i)$. The goal is to adapt to new tasks while remembering old ones. CL uses three types of parameters: shared parameters (θ_s) across all tasks, old task-specific parameters (θ_0), and new task parameters (θ_n). The *Joint* training method, considered the performance benchmark, optimizes all these parameters simultaneously but requires extensive resources as it needs access to all past and present data during training [78].

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 n. The updating, however, of any of the shared weights θ_s risks confusing the classifier when faced with older data, as those classification decisions depend on not only θ_0, but also θ_s. CL training typically boasts significantly faster speed and far less storage requirements than *Joint* training, thus permitting more frequent model retraining to adapt to evolving data distributions or other requirements.

Training in Continual learning (CL) involves considering three sets of parameters [78] –

- θ_s: parameters that are shared across all the tasks
- θ_0: parameters specific to the previous tasks, and
- θ_n: parameters specific to the new task

Here, θ_0 and θ_n contain the corresponding weights and the output layer of the older tasks and the new task, respectively. The weights of all the layers except the classification layer belong to θ_s.

Traditional machine learning (ML) training involves optimizing all three sets of parameters together, and is referred to as *Joint* training. This training method requires the availability – and hence the storage – of all the older data from previous tasks. The performance of this training mechanism is considered to be an upper bound, as data from all the previous tasks as well as the new tasks are used to train the model. Training this way, however, is slow and costly.

In CL training, for each new task n, θ_s and θ_n are optimized and updated while trying to keep θ_0 fixed. CL training does not require the older data to be available, which makes this a difficult task. Significant effort has been spent to boost the performance of models on the tasks in cases where there is no previous data available. Training is typically significantly faster than in *Joint* training, which may enable more frequent retraining of the model to keep up with changing data distributions or other needs without risking earlier concepts.

2.3 Continual Learning Scenarios

Depending on the type of the data distribution shift observed in the real-world, continual learning (CL) can be formalized into three scenarios [76, 141, 142] – Task Incremental Learning (Task-IL), Domain Incremental Learning (Domain-IL), and Class Incremental Learning (Class-IL).

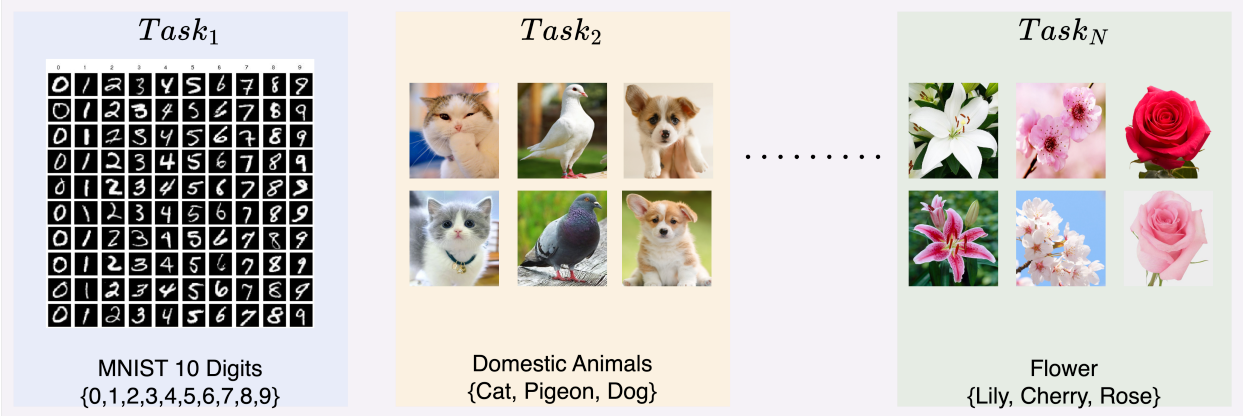


Figure 2.1: Schematic Representation of Task Incremental Learning (Task-IL) Scenario.

2.3.1 Task Incremental Learning (Task-IL)

In this scenario, the capability of the model increases with respect to different objectives of the model, and the task essentially defines the objective. Task-IL is considered as the easiest continual learning (CL) scenario, where the task identity is always known. This means that the inference objective of the model is to make a decision based on the task. A schematic representation of the Task-IL scenario is depicted in Figure 2.1. In this figure, we can see that the first task is to learn and classify the MNIST-10 digits [73]. The second objective of the model is to learn and classify three domestic animals: cat, pigeon, and dog. The final objective of the model is to learn and classify three flowers: lily, cherry, and rose. In this continual learning setting, given the objectives of the model (i.e., tasks) — MNIST-10 digit classification, domestic animal classification, and flower classification — the model needs to infer which object is the test sample within the given objective.

2.3.2 Domain Incremental Learning (Domain-IL)

In a Domain-IL scenario, the number of classes in the learning process remains fixed, but the distribution of the data for each class changes due to covariate shift. A schematic representation of the Domain-IL scenario is depicted in Figure 2.2. We can see that the learning objective of the model remains fixed, which is to learn and classify three domestic animals: cat, pigeon, and dog. However, the underlying data distribution changes in each

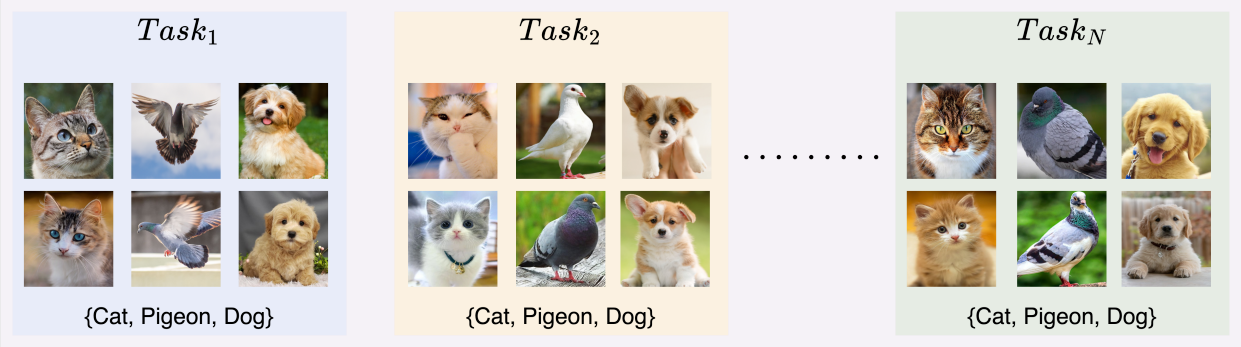


Figure 2.2: Schematic Representation of Domain Incremental Learning (Domain-IL) Scenario.

subsequent task. This means that the model observes different variants of the same object in each learning episode. As shown in the figure, the model encounters variants of the same objects in each learning episode, from $Task_1$ to $Task_N$.

2.3.3 Class Incremental Learning (Class-IL).

In a Class-IL scenario, the class distribution changes as new classes emerge in each learning episode. A schematic representation of the Class-IL scenario is depicted in Figure 2.3. We can see that the first objective of the model is to learn and classify three objects: cat, pigeon, and dog. In the second task, the model encounters two more classes: lily and cherry. Now, the model's objective is to learn these two new classes; however, the inference objective changes from classifying three objects to classifying five objects: cat, pigeon, dog, lily, and cherry. In the final task, the model encounters two additional classes: apple and orange. The model's objective is to learn these two new classes, and the inference objective is to classify all the classes encountered so far: cat, pigeon, dog, lily, cherry, apple, and orange.

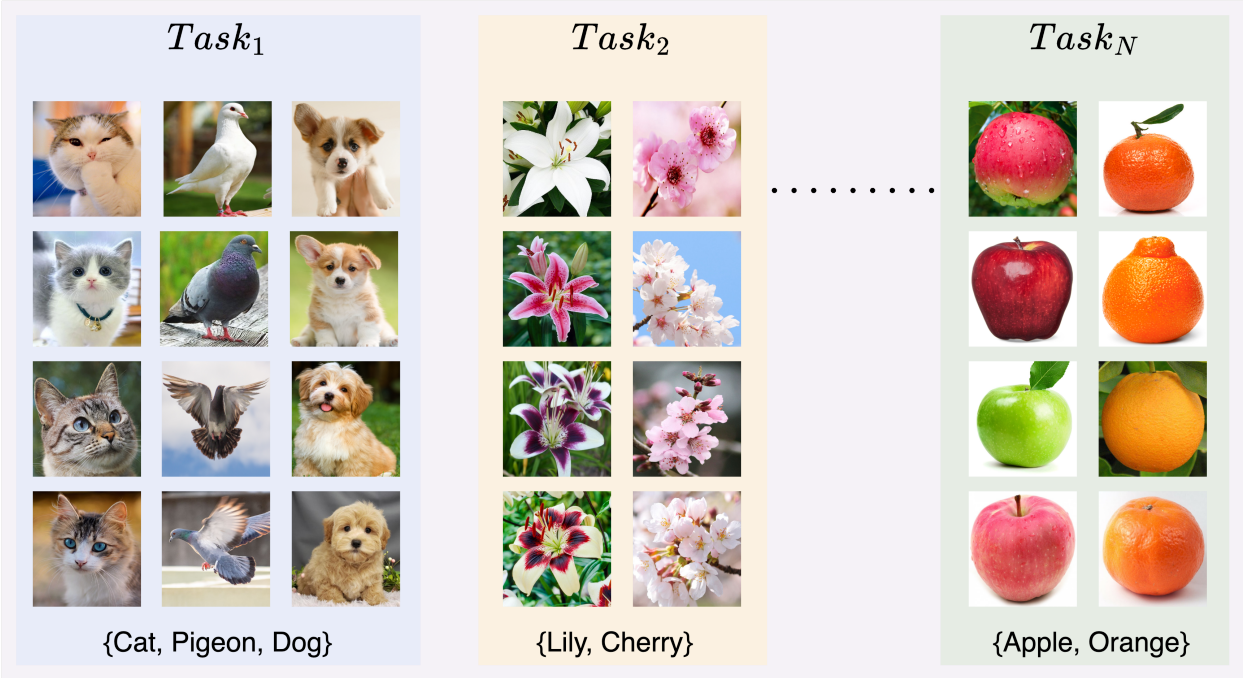


Figure 2.3: Schematic Representation of Class Incremental Learning (Class-IL) Scenario.

2.4 Continual Learning vs. Related Learning Paradigm

2.4.1 Online Learning

Online learning (OL) is a branch of machine learning (ML) in which training data is observed sequentially [29, 39]. The base model is updated as soon as a new sample is observed, and the model becomes effective moving forward. In contrast to the static ML training paradigm, where retraining is done using all current and historical data, OL retrains with only the new data, making it efficient and fast. Another important benefit of OL is its capacity to accommodate the model to *concept drift*. Concept drift occurs when new samples of existing classes are observed. In essence, OL is characterized by incremental updates to the model with each new data point, effectively setting the batch size to one. This method is particularly well-suited for scenarios requiring immediate processing of information, either to quickly adapt the model or due to limitations in data storage.

Although the learning procedures of OL and CL may seem similar, they serve different

objectives. OL is fundamentally designed to efficiently learn new samples incrementally and be robust to concept drift. CL, on the other hand, is designed not only to learn new samples incrementally and be robust to concept drift, but also to ensure that the model remains robust against catastrophic forgetting. A CL model needs to retain the knowledge of prior data distributions while learning new data distributions and experiencing new learning objectives. In some cases, these prior and current learning experiences help in learning future experiences more easily.

Recently, a few studies have aimed to bridge OL and CL, focusing on models that learn from a continuous, potentially infinite stream of data, processing one instance or a small batch at a time. This approach assumes each new sample is encountered only once, necessitating rapid learning within constrained resources [6, 7, 19, 20, 31, 50, 51, 52, 53, 80, 126].

2.4.2 Transfer Learning

Transfer learning (TL) enables an ML model to leverage its prior learning experience from one task to learn the next task [43, 105, 153]. Fundamentally, learning from one type of objective (i.e., prior task) helps improve the learning of the next objective (i.e., next task). The prior task in this context is also referred to as the *source domain*, while the next task is referred to as the *target domain*. TL is also referred to as a *domain adaptation technique* in natural language processing (NLP) research [34, 101].

TL and CL differ fundamentally in their objectives. TL does not focus on preparing the model to be robust against catastrophic forgetting, whereas CL is designed to achieve this. Additionally, CL focuses on knowledge aggregation and continuous learning, while TL is not designed for this purpose.

2.5 Catastrophic Forgetting

Catastrophic forgetting (CF) is a phenomenon where a neural network's performance on previously learned tasks significantly worsens after it is trained on a new task. This occurs under the assumption that the network learns tasks sequentially. A "task" refers to a spe-

cific data distribution, such as a new class of data, a different type of task (for example, transitioning from classifying MNIST digits to CIFAR10 objects), or a variation in the data distribution of existing classes due to factors like time, weather conditions, or seasons. CF happens because the neural network's weights, adjusted for new tasks, tend to overwrite those that were adapted for older tasks, leading to a decline in performance on those older tasks.

This phenomenon is also known in the literature as the *stability-plasticity dilemma* [2, 3, 118]. The ideal scenario involves the neural network being sufficiently *plastic* to adapt to new environments and learn novel tasks, while also being *stable* enough to retain critical information over time. Plasticity is crucial for the ongoing learning process, allowing neural networks to incorporate new patterns and data. This capability is akin to the learning process in biological systems, where synaptic plasticity underlies learning and memory formation [85]. On the other hand, stability ensures that the acquisition of new information does not lead to the unlearning of previous knowledge, a process that is safeguarded in biological neurons through various mechanisms, such as synaptic consolidation [88].

The tension between these two requirements presents a significant challenge: excessive plasticity may lead to catastrophic forgetting, where new learning disrupts old knowledge, while excessive stability can impede the incorporation of new information, rendering the network less responsive to changes in its environment. Several approaches have been proposed to mitigate this dilemma, such as Elastic Weight Consolidation (EWC) [69], which introduces a regularization term to protect important weights from significant changes, thereby balancing between the need for plasticity in learning new tasks and the need for stability to retain previous knowledge. Understanding and addressing the stability-plasticity dilemma is crucial for the development of more robust and adaptive neural networks, capable of continuous learning without forgetting previous knowledge.

The human brain's capacity for learning and memory storage is underpinned by the sophisticated interplay between various regions, notably the prefrontal cortex and the hippocampus. The prefrontal cortex, particularly its medial regions, is instrumental in the executive functions associated with learning, such as attention, decision-making, and the regulation of emotional responses [92]. It processes complex information and is involved in the abstraction and generalization of knowledge, thereby facilitating the application of learned

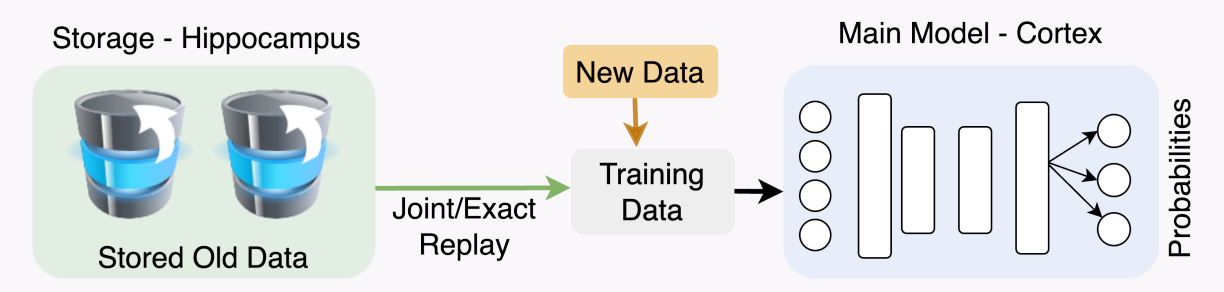


Figure 2.4: Schematic Overview of Joint Replay Mechanism in Continual Learning (CL) Processes.

information to new, similar contexts [13]. Once information has been initially processed and encoded through these mechanisms, it is then transferred to the hippocampus, a structure deeply embedded within the medial temporal lobe, which is critical for the consolidation of this information into long-term memory [13]. The hippocampus is particularly adept at encoding the contextual details of experiences, forming the rich, associative memories that are essential for spatial navigation and episodic memory [40]. This division of labor between the prefrontal cortex's role in the initial learning and processing of information and the hippocampus's role in memory consolidation and storage reflects the intricate neural architecture supporting human cognition.

Continual learning (CL) research aims to emulate the human learning process by training a neural network to learn a series of tasks sequentially, while simultaneously addressing the challenge of catastrophic forgetting (CF). One primary method to mitigate CF involves retaining previous data and revisiting it during the learning of new tasks (see Figure 2.4). This approach is known as *Joint/Exact replay* [119, 140] in contemporary literature and as *rehearsal* [113, 118] in earlier works. It involves periodically reintegrating previously learned data into the training process of new tasks to mitigate catastrophic forgetting. In this conceptual framework, the primary model is likened to the visual cortex, responsible for perceiving new information. Analogously, the storage mechanism is equated to the brain's hippocampus, functioning as a memory buffer. Previous research indicates that the process known as *replay*, crucial for consolidating and reinforcing memories, originates within the hippocampus, manifesting as episodic memory recall [32, 122, 138]. This analogy underscores the pivotal role of the hippocampus in memory formation and retrieval, drawing a parallel between neural network operations and cognitive processes in the human brain.

However, this approach is often criticized for its inefficiency, primarily due to the necessity of

continually retraining the system on all previously learned tasks, which can lead to unmanageable data storage demands over time [102,128]. Moreover, the comprehensive storage of all data may contradict with various data privacy regulations, including the EU's General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA). Contrary to this, the human brain does not retain all raw data for memory recall. Instead, it relies on the crucial process of replaying past experiences to solidify new memories [62, 108]. This reactivation of neurons associated with past experiences, originating from the hippocampus and manifesting in the cortex, occurs during both sleep and waking states [21].

2.6 Overcoming Catastrophic Forgetting

Over the years, numerous strategies have been developed to address the challenge of catastrophic forgetting, which can broadly be categorized into three major families [102]: i) *regularization* methods, which constrain the update of certain parameters to retain previous knowledge; ii) *replay* methods, which involve revisiting previous tasks or data to reinforce old memories; and iii) *adaptive expansion* methods, which dynamically adjust the network architecture to accommodate new tasks without overwriting existing information.

2.6.1 Regularization Methods

Regularization-based techniques attempt to penalize changes to weights that are determined to be important to the previous tasks. This is done by introducing a new loss function known as *regularization loss*. The regularization loss is added to the training loss to compute the total loss. This category includes Elastic Weight Consolidation (EWC) [69], EWC Online [128], Synaptic Intelligence (SI) [151], Memory Aware Synapses [5], Riemannian Walk (RWALK) [24], Online Laplace Approximator [116], Hard Attention to the Task [129], and Learning without Memorizing [38]. In this section, we explore a selection of key techniques relevant to our research.

Elastic Weight Consolidation (EWC) and EWC Online. [69] proposed EWC to overcome catastrophic forgetting in a neural network. EWC is inspired by human brain, in which the plasticity of the synapses of the previously learned tasks are reduced to facilitate continual learning. As mentioned earlier, excessive plasticity of the weights of the previous tasks is the major cause of the catastrophic forgetting. If the plasticity of the weights is loosely connected to the previous task, then the network becomes less prone to catastrophic forgetting. Leveraging this idea, EWC quantifies the importance of the weights in terms of their impact on the previous tasks' performance, and selectively reduces the plasticity of the most important such weights.

A Bayesian approach is used to measure the importance of the parameters of a task in EWC. Given two tasks T_1 and T_2, EWC tries to converge to a point where both of these tasks have low error using the following loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_{T_2}(\theta) + \sum_i \frac{\lambda}{2} F_i \left(\theta_i - \theta^*_{T_1,i}\right)^2 \tag{2.1}$$

Here, $\mathcal{L}_{T_2}(\theta)$ is the loss of task T_2 only, $F_i(\theta_i - \theta^*_{T_1,i})^2$ approximates a Gaussian distribution with mean given by the parameters $\theta^*_{T_1}$ which is with respect to task T_1 and a diagonal of the Fisher information matrix F. i is an index into the weight vector. λ controls this distribution in such as way that the weights do not move too far away from the low error region of task T_1. Similarly, when a new task T_3 is observed, the loss function is updated in such a way that forces the parameters θ to be close to $\theta^*_{T_1,T_2}$, where $\theta^*_{T_1,T_2}$ is the parameters learned for the previous tasks T_1 and T_2.

The major drawback of the proposed EWC is scalability. The quadratic term of the regularization loss grows linearly with the increase of task which hinders to enable EWC be truly applicable in a practical continual learning scenario where the tasks keep growing. EWC Online attempts solve this problem by strict Bayesian treatment which results in a single quadratic penalty term considering the important parameters of the current task and a running sum of the Fisher Information matrices of the previous task's parameters [128]. EWC Online modifies the second part of Equation 2.1 where $\tilde{F}_i$ is the running sum of the previous task. The modified loss function becomes as follows:

$$\mathcal{L}(\theta) = \mathcal{L}_{T_2}(\theta) + \sum_i \lambda \tilde{F}_i \left(\theta_i - \theta^*_{T_1,i}\right)^2 \tag{2.2}$$

Synaptic Intelligence (SI). SI is a variant of EWC, in which changes to weights that are important to the previous tasks are penalized when training on new tasks [151]. Fundamentally, regularizer loss is added to the loss of the current task to get the total loss, $\mathcal{L}_{total}$. To compute this regularization loss, we first compute the importance of the weights I_t^w after every task t, with respect to the change in total loss $\mathcal{L}_{total}$. To get the estimated importance of the weights I_t^{N-1} for $N-1$ tasks, all the I_t^w are summed up together after a normalization step. Finally, the regularization loss $\mathcal{L}_r(\theta)$ for tasks $N > 1$ is computed in Equation 2.3. Refer to the paper [151] for more details.

$$\mathcal{L}_r(\theta) = \sum_{t=1}^{K} I_t^{N-1} \left(\theta_t - \hat{\theta}_t^{(N-1)}\right)^2 \tag{2.3}$$

2.6.2 Replay Methods

Replay methods are designed to complement the training data for each new task with older data that are representative of the tasks seen so far [26, 119, 134]. Distillation loss, for instance, is often used in replay-based techniques [22, 78, 114]. Learning without Forgetting (LwF) [78] replays *pseudo-data*, which is generated by first running the older model on new data to generate the softmax outputs, and then using those outputs as soft labels for training the model. This technique is similar to model distillation.

In an alternative form of replay called *generative replay (GR)* [130, 139, 140], a second model is trained to capture the distribution of data from the previous tasks and generate new samples from the distribution to be replayed. In the current task T, both the main model $\mathcal{M}_T$ and a generative model $\mathcal{G}_T$ is trained using a mix of task T data and data generated by the prior generator model $\mathcal{G}_{T-1}$. GR is a powerful technique when access to older data is not available or restricted. Replay through Feedback (RtF) [140] reduces the training cost for dual-memory-based GR by coupling the generative model into the main model. RtF enhances the capability of the main model to generate samples by adding feedback connections and

a layer of latent variables z, which are responsible for reconstructing inputs. Brain-Inspired Replay (BI-R) [139] is an improvement over RtF and proposes three new components: i) replacement of the standard normal prior with Gaussian mixture, ii) internal context [86], and iii) internal replay.

Another variant of replay, which intelligently picks the samples (i.e., called exemplars) to complement the training data of the current task, includes Experience Replay [119], Gradient Episodic Memory (GEM) [82], Averaged Gradient Episodic Memory (A-GEM) [25], and Incremental Classifier and Representation Learning (iCaRL) [114]. ER leverages a reinforcement learning based learner to learn new and old experiences. In particular, ER balances stability and plasticity using on-policy for plasticity and off-policy for stability. iCaRL sets a memory budget beforehand, and the memory budget is equally divided into the previously learned classes. It picks and stores only exemplars that are close to the feature mean of each class. Loss of the current classes is minimized along with the distillation loss between targets obtained from the predictions of the previous model and the predictions of the current model on the previously learned classes. A-GEM also follows iCaRL to replay selective samples during training from the learned tasks to be replayed in the current task. When the original data is available, however, prior work has recommended *exact replay* instead [66, 97, 114]. We briefly discuss the specific techniques studied for this proposal below.

Experience Replay (ER). ER technique jointly trains the network utilizing both the examples (i.e., data) from the current task and examples stored in the very small episodic memory. ER is based on a distributed actor-critic training in which the single learner is fed both new and replayed experiences (i.e., tasks) [119]. To adjust the off-policy distribution shifts during training, ER adapts V-Trace off-policy learning algorithm [41]. Three losses – $L_{policy-gradient}$, L_{value}, and $L_{entropy}$ are common to both new and replayed experiences and another two losses – $L_{policy-cloning}$ and $L_{value-cloning}$ are unique to only the replayed experiences. Refer to the paper [119] for more details.

Learning without Forgetting (LwF). LwF tries to reduce the catastrophic forgetting of an older task t_{n-1} by learning the parameters of new task t_n, θ_n, with the help of the shared parameters θ_s and the parameters of the older tasks θ_0. The objective is to optimize θ_n and θ_s such that the prediction of t_n using θ_s and θ_0 does not drift significantly [78]. The

objective function of LwF algorithm is given below:

$$\theta_s^*, \theta_0^*, \theta_n^* \longleftarrow \text{argmin}_{\hat{\theta}_s, \hat{\theta}_0, \hat{\theta}_n}(\mathcal{L}_{new}(\hat{y}_n, y_n) + \lambda_0 \mathcal{L}_{old}(\hat{y}_0, y_0) + \mathcal{R}(\theta_s, \theta_0, \theta_n)) \tag{2.4}$$

$\hat{y}_n$ is the prediction of the test samples of the new task t_n using current shared parameters $\hat{\theta}_s$ and current task's parameters $\hat{\theta}_n$. A multinomial logistic loss function is used to compute $\mathcal{L}_{new}$. Thus, $\mathcal{L}_{new}(\hat{y}_n, y_n)$ minimizes the difference between predicted $\hat{y}_n$ and actual y_n. y_0 is the prediction of the test sample of the new task t_n using the shared parameters θ_s and previous task's parameters θ_0 (i.e., the model before being trained with the samples of the new task). $\hat{y}_0$, on the other hand, is the prediction of the test samples of the new task t_n using current shared parameters $\hat{\theta}_s$ and previous task's parameters $\hat{\theta}_0$ (i.e., the model as trained with the samples of the new task). Distillation loss [56] is used to compute the $\mathcal{L}_{new}$ so that the output of one network can be approximated using the outputs of another. Thus $\mathcal{L}_{old}(\hat{y}_0, y_0)$ minimizes the difference between $\hat{y}_0$ and y_0. λ_0 works as a balancing factor between the new task t_n and the previous task t_{n-1}. In the algorithm, $\mathcal{R}(\theta_s, \theta_0, \theta_n)$ works as a regularizer to avoid overfitting in the model.

Generative Replay (GR) and GR with Distillation. Earlier studies indicate that the cerebral cortex, serving as the primary model, is more effective when coupled with a generative model rather than a replay buffer [28, 67, 112]. Shin et al. [130] introduced Deep Generative Replay (DGR) in 2017, utilizing a Generative Adversarial Network (GAN) [48] to produce synthetic examples of past tasks, thereby obviating the need to store real data from these tasks. Figure 2.5 depicts a schematic of Generative Replay (GR), illustrating how the generator, analogous to the hippocampus, synthesizes representative samples of historical data for integration during new task training. This process allows the GAN generator to effectively recreate the model's understanding of prior tasks.

Given a series of tasks $t_0, t_1, t_2, ..., t_n$, an expert model $\mathcal{S}$ – which contains a generator model $\mathcal{G}$ and a solver model $\mathcal{M}$ (called the main model in some works [139, 140]) – holds the knowledge of the previous tasks and thus prevents the system from catastrophic forgetting. In the Figure 2.5, at the outset of the learning process in Task 0 (t_0), the main model

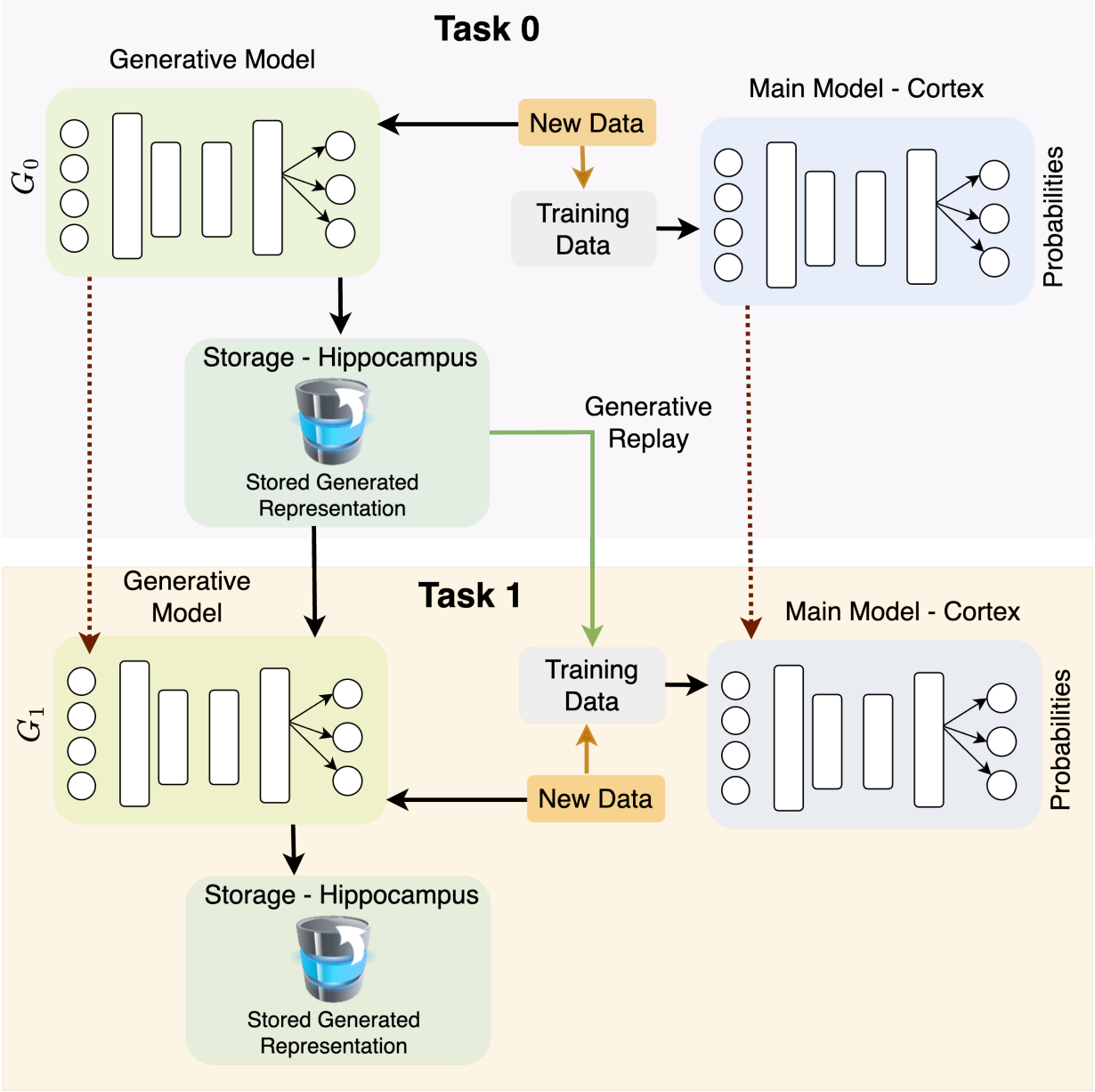


Figure 2.5: Conceptual Diagram of Generative Replay in Neural Networks: Integrating Past and Present Learning.

undergoes training with the current new data. Concurrently, the generative model G_0 creates representative data of this new input and stores it. Moving on to Task 1 (t_1), the data generated by G_0 is replayed to the main model for retraining, alongside the new data for Task 1. Meanwhile, G_0 evolves into G_1, which then generates representative samples of both the current task's data and the previously stored representative generated samples. This means G_1 produces representative samples for both the current and previous tasks. This

cycle of generation and storage continues with subsequent tasks and learning episodes.

In DGR, $\mathcal{S}$ is learned and maintained in a continual learning fashion. $\mathcal{S}$ for the series of previous n tasks can be represented as $\mathcal{S}_n = (\mathcal{G}_n, \mathcal{M}_n)$. Given a new task t_{n+1} and the new training data D_{n+1}, the objective of $\mathcal{S}$ is to learn $\mathcal{S}_{n+1} = (\mathcal{G}_{n+1}, \mathcal{M}_{n+1})$. There are two steps involved in the learning process of $\mathcal{S}_{n+1}$ considering $D_{n+1} = (x, y)$ where (x, y) represents *(data, label)*:

1. At first, the *scholar* model $\mathcal{S}_{n+1}$ is updated with the input x of new task t_{n+1} and replayed with the generated data, $\hat{x}$, from previous scholar model $\mathcal{G}_n$. Real data x and replayed data $\hat{x}$ are mixed together at a ratio based on the importance of the new task t_{n+1} compared to the older task t_n. This is referred to as intrinsic replay or pseudo-rehearsal [118].

2. Then the main model $\mathcal{M}_{n+1}$) is trained with the real and replayed data with the following loss function:

$$\begin{aligned} \mathcal{L}_{train}(\theta_{n+1}) = r\,\mathbb{E}(x,y) \sim \mathcal{D}_{n+1}[\mathcal{L}(\mathcal{M}(x;\theta_{n+1}),y)] + \\ (1-r)\,\mathbb{E}(x,y) \sim \mathcal{D}_n[\mathcal{L}(\mathcal{M}(x;\theta_{n+1}),y)] \end{aligned} \tag{2.5}$$

   Here, θ_n represents the parameters of the main model $\mathcal{M}_n$) and r represents the ratio of the mixture of the real and replayed data.

The DGR framework is designed in such a way that choice of the generative model is not limited to a GAN and can instead be a variational autoencoder (VAE) [68] or any other such type.

For the experiments of GR, we need two models – the main model $\mathcal{M}$ and a generative model $\mathcal{G}$. $\mathcal{G}$ is responsible for generating representative samples of the previous tasks to be replayed in the current task. We use the base model architecture for both $\mathcal{M}$ and $\mathcal{G}$. The loss function of $\mathcal{M}$ consists of two parts – one for the data of the current task and another for the replayed samples. The cumulative loss of these two parts are weighted in terms of the number of tasks the model has observed so far.

For $\mathcal{G}$ in our experiments, we use a symmetric VAE [68], where there is an encoder that maps the input data distribution to a latent distribution and a decoder that reconstructs the

inputs from the latent distribution. For both the encoder and the decoder, the base model architecture is used. In all of our experiments, we used a stochastic latent variable layer with 100 Gaussian units parameterized by the mean and the standard deviation of the output of the encoder given input x.

The data to be replayed are sampled from the generative model, and then the selected samples are fed to the main model and then labeled based on the predicted class of the model. The samples to be replayed during task T are generated by the version of the main model and the generator after training on task $T-1$. Hence, we need to store a copy of both $\mathcal{M}$ and $\mathcal{G}$ after each task.

GR with distillation is a variant of GR where the generated samples are replayed with the output probabilities (i.e., soft targets) instead of the actual labels. Previous work show that GR with distillation often works better than GR [139, 140, 141].

Replay through Feedback (RtF). GR has two models – a main model and a generative model. RtF proposes to merge the generator model into the main model [140]. This is inspired by the fact that replay in the brain is originated in the hippocampus and then it propagates to the cortex, and in our brain's processing hierarchy, the hippocampus sits on top of the cortex. The merged model will work as a brain, where the first n layers will work as the visual cortex and the last $m-n$ layers will work as the hippocampus. Technically, an additional `softmax` classification layer is added on top of the encoder of our generator VAE model. This technique requires only a single model to be trained, and the loss of the current tasks has two terms – cross entropy loss and generative loss.

Brain-Inspired Replay (BI-R). Recently proposed by [139], BI-R seeks to improve upon RtF with another three add-on components – Conditional Replay (CR), Gating based on Internal Context (Gating), and Internal Replay (IR). For CR, BI-R proposes to replace the standard normal prior over the VAE's latent variables by a Gaussian mixture with a separate mode for each class so that class-specific samples can be generated. This is due to the fact that a vanilla VAE is limited to generate class-specific samples, but humans do have control over which memories to recall. Conditional replay (CR) is intended to provide the network a human-like capacity to generate samples of the class the network needs most.

Context-dependent gating was originally proposed by [86]. The idea is to reduce interference between different tasks by gating different and randomly selected network nodes for each task. However, this technique requires the task identity to be known for all the tasks, which is not realistic in the Class-IL scenario. BI-R proposes to use this gating technique in the decoder of the VAE with a conditional of the internal context. The task or class to be generated and reconstructed is the conditioned internal context. To note, not all of the nodes of the decoder network are gated. Mental images are not propagated all the way to the retina, and thus the brain does not replay memories to the input level [16]. This insight is corroborated by evidence from neuroscience that our brain's early visual cortex does not change significantly from childhood to adulthood [131]. To accommodate these observations into continual learning, BI-R proposes to replay internally or at a hidden layer, instead of to the input level. From the machine learning perspective, the first n layers will need a limited amount of change, since there is no replay in them.

Incremental Classifier and Representation Learning (iCaRL). iCaRL [114] is one of the earliest replay-based methods specifically designed for Class-IL scenario. Given a fixed buffer size (i.e., allocated memory), iCaRL stores samples of the earlier learned classes which are closest to the feature mean of those classes obtained from the feature maps of the network. iCaRL minimizes two loss functions – i) one is the categorical cross entropy loss of new classes, and ii) distillation loss obtained from the predictions of the current model's and the previous model's targets.

Averaged Gradient Episodic Memory (A-GEM). A-GEM [25] is an improved version of GEM [82]. GEM attempts to reduce catastrophic forgetting by constraining the updates of the new task not to interfere with the previous tasks. GEM utilizes the first order Taylor series approximation to estimate the direction of the gradient on the possible areas laid out by the gradients of the previously learned tasks. A-GEM relaxes the constraint to project the gradient into only one direction estimated from the randomly selected samples stored in a replay buffer. The replay buffer contains samples of the previously learned tasks.

2.6.3 Adaptive expansion methods

Adaptive expansion methods grow the capacity of the neural network as new tasks are observed. Several techniques have been proposed, including Progressive Neural Networks [125], Dynamically Expandable Networks [150], Adaptation by Distillation [57], and Dynamic Generative Memory [100]. These methods require incremental increases in memory as the new tasks are observed, and thus face scalability problems. Given the scale of malware classification problems, we leave the investigation of these techniques to future work, and instead focus on methods that do not require us to increase model capacity with dataset or problem size.

3

Continual Learning in Malware Domain

In this Chapter, we begin by addressing the current challenges in the malware domain, particularly in light of the continuously expanding landscape of both malware and benign software. We then propose a formalization of continual learning scenarios, discussed in Chapter 2, tailored for malware classification. Next, we delve into the specifics of the malware datasets analyzed in this dissertation. While two of these datasets, EMBER [10] and Drebin [11], were sourced from prior studies, the remaining two were collected specifically for this research from AndroZoo repository [8]. Following this, we provide insights into the model utilized in our study, along with the training mechanism employed. Additionally, we discuss the implementation details, baselines utilized, and the metrics employed for evaluation.

3.1 Malware Domain

The explosive growth in both malicious and benign software presents significant challenges for machine learning models in cybersecurity. According to AV-TEST Institute, over 450,000 new malware instances and potentially unwanted applications (PUAs) are detected daily [12], highlighting the relentless advance of cyber threats. Furthermore, VirusTotal registers more than 1.5 million new benign software applications each day [145]. This rapid proliferation complicates the task of distinguishing between malware and benign programs, as the sheer volume and variety of new software strains traditional detection methods.

Continual Learning (CL) offers a promising solution to these challenges by enabling models to learn incrementally from new data without forgetting previous knowledge. This approach is particularly suited to the dynamic nature of cybersecurity, where the ability to adapt to evolving threats in real-time is crucial. By incorporating CL, malware classification systems can better manage the influx of new software and malware, maintaining their efficacy in identifying and neutralizing threats amidst the ever-growing digital ecosystem.

However, we note that no prior work has explored continual learning or machine unlearning to the malware domain. However, [9] has proposed a continual learning-based network intrusion detection system (IDS) to mitigate catastrophic forgetting in a class-incremental scenario leveraging partial replay-based approaches. While they primarily focus on the role of class imbalance, some of the results of their study mirror our own findings, namely that the number and selection of samples used during replay are key to the success of continual

learning in the cybersecurity space. A recent work on continual learning for Android malware classification proposes to combine contrastive learning with active learning to continuously train Android malware classifiers [27]. However, their focus is on the *concept drift* detection rather than overcoming CF.

Additionally, there is some prior work on *online learning* (OL) applied to malware classification [60, 95, 96, 148]. Online learning considers the problem of incorporating new samples into the model as they are observed, and notably does not directly address the problem of catastrophic forgetting. Furthermore, previous work has shown that producing high-quality labels for malware can be a difficult task [65], and often requires weeks of time for vendors to come to consensus on newly-discovered variants [154], making immediate incorporation of observed samples risky. Recently, researchers have explored malware detection systems leveraging transfer learning in an attempt to address the challenge of adapting to ever-evolving malware samples [4, 23, 47, 58, 120, 136]. Transfer learning, however, is not focused on retaining knowledge of the prior tasks when being applied to the new domain. In malware, this distinction is significant, since an inability to detect previous malware variants would reintroduce vulnerabilities without the user's knowledge. We thus focus in this paper only on CL techniques, as they explicitly address the issue of catastrophic forgetting.

3.2 Continual Learning Scenarios for Malware Classification

We setup our continual learning problem in such a way that a ML model, M, learns a series of sequential tasks, $t_0, t_1, ..., t_n$. During the training of t_i, only data from t_i is available. Below, we describe three continual learning scenarios for malware classification [141]: Domain Incremental Learning (Domain-IL), Class Incremental Learning (Class-IL), and Task Incremental Learning (Task-IL). We show a schematic representation of the utility of these three scenarios in a malware classification pipeline in Figure 3.1.

Domain-IL. The most important problem in malware classification is binary classification of a test sample as either benign software (*goodware*) or malware. Many new malware

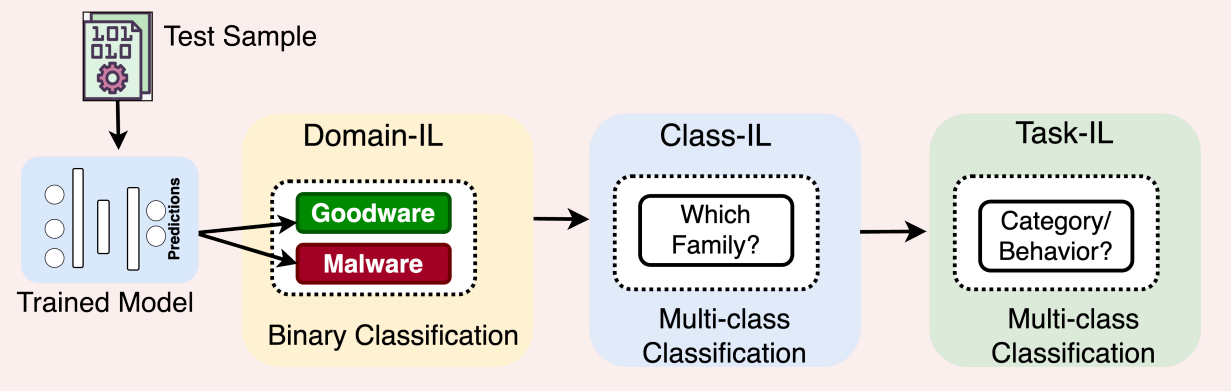


Figure 3.1: CL Scenarios in Malware Classification Pipeline.

samples, unidentified applications, and goodware are created and released every day. New malware and goodware often behave differently from prior ones, resulting in concept drift over time. This makes it valuable to incorporate new samples into production systems sooner than later. Previously proposed malware classification techniques generally do not consider an evolving model trained continuously to incorporate this distribution shift without complete retraining [64, 149]. Further, in this adversarial setting, an attacker may leverage older malware specifically to evade classifiers that have forgotten about it. Thus, the distribution shift must be captured while avoiding catastrophic forgetting.

In this binary malware classification setting, we divide our dataset into monthly tasks, where each month captures the natural concept drift of both malware and goodware due to evolving malware capabilities and benign software releases. We seek to incorporate this new knowledge in each monthly incremental learning iteration while maintaining previous discriminative knowledge about earlier malware and goodware.

We note that, unlike most studies of Domain-IL that rely on artificial manipulations of image datasets, the malware dataset we use shows real-world shifts in the data distributions over time. For example, we find that some malware families become more or less popular during the span of the dataset.

Class-IL. The second type of malware task is multi-class family classification. A *family* is a set of malware programs that have significant overlap in their code, such that they are considered by experts to be a group with common functionality. The famous Zeus banking

trojan, for example, has evolved since 2006 to include 556 versions of software spread out among 35 different families with names like Citadel and Gameover [127].

It takes some time to label new malware samples and unknown applications with the help of experts and, in many cases, a consensus of scores from multiple anti-virus engines. Classes can be added when enough samples share enough similarity for the experts to decide that they are a new family [65, 154]. This occurs fairly infrequently in practice, but it requires the model to be adaptable to incorporate this new knowledge.

In this multi-class malware classification setting, we divide our dataset so that the model would learn new malware classes incrementally, extending its capabilities. We assume that the base model would start with a non-trivial number of classes, and then we would increment with new classes afterwards. Each new task is defined as adding new classes, but test time performance is measured on all classes that the model has been trained on so far.

In practice, an analysis pipeline is used to detect and triage malicious programs. The first step is binary classification of benign/malicious files. Once a file is determined to be malicious, it is useful to provide additional context to the security analyst about the malware family or its capabilities, which is encapsulated in Class IL. Since benign files are classified at the first stage in the pipeline, only malware is used in these settings.

Task-IL. To facilitate classification of malware into families, it can help to constrain the task based on information gained from other analysis methods, such as the broader category of the malware (e.g. adware, ransomware, etc.), behaviors of the malware [17], or the infection vector that the malware uses (e.g. phishing, downloader, etc.). Task-IL captures this notion of constrained tasks, where adding a new task may represent a new category or new set of behaviors. This is likely to be less frequent than simply adding a new family, as we have in Class-IL, but it represents a real problem in malware classification. Unlike in Class-IL, the task identity is given to the model at test time, making it a much easier problem. In malware, this could mean learning the task identity from a separate model, manual analysis, or field reports of the malware's behavior. As we do not have naturally defined tasks in our datasets, we divide our dataset into tasks that contain an equal number of independent and non-overlapping classes, as is common in the continual learning literature [69, 128, 151].

Table 3.1: Drebin. 4525 malware samples from top 18 malware families.

Label	Family	#of Samples	Label	Family	#of Samples
0	FakeInstaller	925	9	Geinimi	92
1	DroidKungFu	667	10	Adrd	91
2	Plankton	625	11	DroidDream	81
3	Opfake	613	12	MobileTx	69
4	GingerMaster	339	13	FakeRun	61
5	BaseBridge	330	14	SendPay	59
6	Iconosys	152	15	Gappusin	58
7	Kmin	147	16	Imlog	43
8	FakeDoc	132	17	SMSreg	41
Total 4,525					

3.3 Dataset

For our experiments, we have utilized two popular, large-scale datasets from the malware research community: Drebin [11] for Android malware and EMBER [10] for Windows Portable Executable (PE) malware, specifically the EMBER 2018 version due to its updated complexity and challenge for classifiers. Additionally, we have compiled two datasets from AndroZoo [8] (AZ), named AZ-Domain and AZ-Class, tailored for Domain Incremental Learning (Domain-IL) and Class Incremental Learning (Class-IL) respectively. These datasets comprise Android APK files.

Drebin. The Drebin dataset consists of 5,560 Android malware samples.[1] For our experiments, we use malware samples from the top 18 malware families totaling 4,525 malware samples. Unfortunately, we could not find the samples belonging to *LinuxLotoor* and *GoldDream* in the dataset. We can see the details of Drebin dataset used in this study from Table 3.1.

Drebin features are organized as sets of strings, such as permissions, API calls, and network addresses, and they are embedded in a joint vector space as Boolean expressions representing

[1] https://www.sec.cs.tu-bs.de/~danarp/drebin/download.html

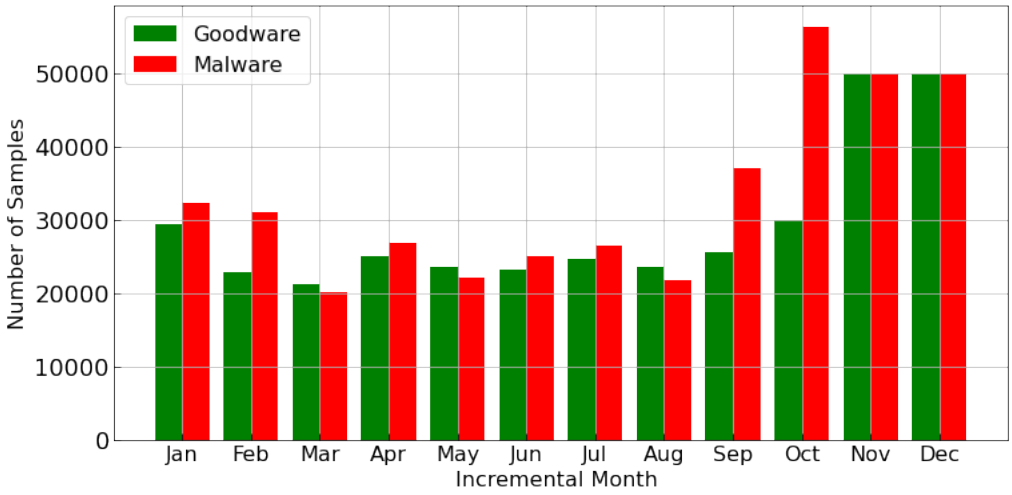


Figure 3.2: EMBER Data: Goodware and malware data statics of 12 months of 2018.

the presence or absence of the attribute. Drebin features are highly sparse, making it a relatively easy dataset. The total number of original features in these samples is 8,803. With binary features, we do not need to use standardization, though we do pre-process the features using `scikit-learn's VarianceThreshold` [104] to omit features with very low variance (< 0.001). This leaves a final set of 2,492 features.

We can only perform task-IL and class-IL experiments with this dataset. Even though the samples span over five years, the families in this dataset do not consistently contain enough samples for each time period.

EMBER. The EMBER 2018 dataset contains features from one million PE files scanned mostly in 2018, with 50K samples from before 2018.[2] There are 400K goodware samples, 400K malware samples, and 200K unknown samples. Figure 3.2 shows the number of goodware and malware samples of each month of 2018.

EMBER contain a variety of features, including general file information, header information, imported and exported functions, and section information. EMBER features also contain three groups of format-agnostic features: byte histogram, byte-entropy histogram, and string information. Some of these features with high cardinality (e.g., identified strings) are then processed using the *hashing trick* [147] with different bin sizes. Each of the groups of EMBER

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

features have unique distribution characteristics, which makes this dataset complex. It is also worth noting that the EMBER feature space encodes rich semantics for the underlying executable data that naturally constrain the feasible regions of that space. For more detail, refer to [10].

There are 2,381 features in total, and we use `scikit-learn's StandardScaler` [104] to standardize the feature space. StandardScaler provides a `partial_fit` method, which can be updated incrementally to standardize the dataset using each month representing a continuous flow of data.

For the Task-IL and Class-IL experiments, we only use the malware samples from 2018, which belong to 2,900 families. As we found that the majority of the families contain only a few samples, we filtered out the families containing fewer than 400 samples. This left us with 106 families, from which we select the top 100 malware families containing 337,035 samples for our experiments. For the study of Domain-IL, we take both goodware and malware samples from 2018, removing the unknown samples. This subset of the data spans 12 months, January to December. We focus on binary classification for this set of experiments.

AndroZoo – AZ Datasets. We have collected two datasets of Android APK files from AndroZoo [8] for our experiments: AZ-Domain for Domain Incremental Learning (Domain-IL) and AZ-Class for Class Incremental Learning (Class-IL). The AZ-Domain dataset comprises 80,690 malware and 677,756 goodware samples collected from 2008 to 2016, with each year's samples spanning from January to December, totaling 758,446 samples. Following established practices [148], we selected malware samples with a VirusTotal detection count of at least 4. We aimed for a 9:1 goodware-to-malware ratio, although this was not always achievable for each year. The AZ-Domain dataset was divided into training and testing sets for each year at a 9:1 ratio, resulting in 682,598 training samples and 75,848 testing samples. The AZ-Class dataset includes 285,582 samples across 100 Android malware families, each with a minimum of 200 samples and a VirusTotal detection count of at least 4. This dataset was split into 223,608 training samples and 61,974 testing samples, maintaining the same 9:1 ratio.

For both datasets, we extracted Drebin features [11], which encompass eight feature categories to analyze app behavior thoroughly. These categories include hardware access, re-

quested permissions, component names, filtered intents, restricted API calls, used permissions, suspicious API calls, and network addresses. The AZ-Domain training set yielded 3,858,791 features, with the test set transformed accordingly. The AZ-Class training set had 1,067,550 features, with the test set also transformed accordingly. To enhance computational efficiency, we applied `scikit-learn's VarianceThreshold` to both datasets, eliminating features with a variance below 0.001. This process resulted in reduced feature dimensions of 1,789 for AZ-Domain and 2,439 for AZ-Class.

3.4 Model Selection and Training

For the purposes of our experiments, we standardize our model as a multi-layer perceptron (MLP). Our MLP model has four fully-connected (FC) layers with $[1024, 512, 256, 128]$ hidden units, using dropout ($rate = 0.5$) and batch normalization in each layer. We use SGD as the optimizer, with learning rate = 0.01, momentum = 0.9, and weight decay = 0.000001.

In our approach, we refined the model's performance through selective hyperparameter tuning, which included adjustments to the number of layers, hidden units, activation functions, optimization algorithms, and learning rates. This process led to our model achieving an AUC score of 0.99512. For context, the EMBER 2018 dataset, as referenced by [10], utilized a LightGBM model for binary classification, achieving a slightly higher AUC score of 0.99642. Interestingly, substituting `StandardScaler` with `QuantileTransformer` in our model improved the ROCAUC score to 0.99655. However, due to the lack of a `partial fit` method in `QuantileTransformer`, this approach was not feasible for our final model. Despite this, the ROCAUC score obtained with `StandardScaler` closely approaches the benchmark set by the LightGBM model in [10], leading us to adopt it for subsequent experiments on both EMBER and Drebin datasets.

There are, however, a few data- and experiment-specific changes to the optimizer and learning rate that we finalized after conducting multiple sets of experiments in each setting. For experiments with the Drebin dataset, we observed that an Adam optimizer with learning rate 0.001 provided us slightly better performance than SGD, so we switch to it for Task-IL and Class-IL experiments on Drebin. For Task-IL and Class-IL with the top 100 classes of EMBER, we observe that SGD with learning rate 0.001 gave better accuracy than learning

rate 0.01. Meanwhile, for the Domain-IL experiments with EMBER, SGD with learning rate 0.01 yielded the best performance.

For all scenarios and training protocols, the model is trained in a sequential manner, such that each of the N tasks $t_1, t_2, t_3, ..., t_N$ are independent, and their corresponding data distribution is also independent. The model only has access to the data of the current task. We use the standard multi-class cross-entropy loss for all the experiments except in Domain-IL. As Domain-IL is binary, we use binary cross entropy loss. We use mini-batch sizes of 32 and 256 for Drebin and EMBER, respectively.

3.5 Implementation Details

In Task-IL, we equally divide the number of classes into nine tasks for Drebin and and 20 tasks for EMBER data, where each task has two classes and five classes, respectively. In Class-IL with Drebin data, the model starts with 10 classes, and then we add two classes in each task, making five tasks in total. For EMBER, the model starts with 50 classes and then five classes are added in each task, making 11 tasks in total. In Class-IL, a task means an episode of learning new class(es). In Domain-IL, there are 12 tasks, one for each month's data of malware and goodware.

The output layer of each of the scenarios is implemented differently. The output layers of Task-IL and Class-IL have a distinct output unit for each class to be learned. The major difference of these two scenarios, however, is in the *active* output units. In Task-IL, only the output units of the classes in the current task are active. In Class-IL, however, all the output units of all the classes seen so far are active. In Domain-IL, all the output units – both of them in our binary classification task – are active, as only the data distribution is changing. The `softmax` function only considers these *active* units while assigning probabilities.

We use `PyTorch` [103] and run our models on a `CentOS-7` machine with an Intel Xeon processor with 40 CPU cores, 128GB RAM, and four GeForce RTX 2080Ti GPU cards, each with 12GB GPU memory.

3.6 Baselines

We have two baselines: i) *None* and ii) *Joint*. In the *None* baseline, the model is trained sequentially without any CL techniques. The performance of this method can be interpreted as an informal lower bound on our results, though it is not a true lower bound in theory or practice. In the *Joint* baseline, the accumulated data of all the tasks observed so far are used to train the model. The performance of this mechanism can be interpreted as an informal upper bound, though it too is not a theoretical upper bound. *Joint* replay requires storage and training effort proportional to all the data of the tasks observed so far. It is expensive, but it ensures high performance across the entire dataset up to the current iteration. The effectiveness of a particular technique to overcome catastrophic forgetting can then be seen as its ability to move the accuracy from being near to the lower bound to near the upper bound.

3.7 Metrics

We measure the performance of our experiments throughout this work using two metrics – i) global average accuracy over all tasks, and ii) minimum of the average accuracies of tasks.

Global average accuracy over all tasks ($\overline{AT} \in [0, 1]$) : Let $A_{P,T}$ be the accuracy of the model on the test set of task Q after the model is trained on task T. Then the average accuracy $\overline{AT}$ at task T can be defined as follows:

$$AT = \frac{1}{T} \sum_{Q=1}^{T} A_{Q,T} \tag{3.1}$$

Assuming the number of task is N, then the average accuracies AT over all tasks T_N can be defined as:

$$\overline{AT} = \frac{1}{T_N} \sum_{T=1}^{T_N} AT \tag{3.2}$$

Minimum of the average accuracies of tasks ($\widehat{AT} \in [0, 1]$) : Assuming the number of task is N and Z is the set of the minimum average accuracy, $Z = \{AT_1, AT_2, ..., AT_N\}$, of all tasks T_N, the minimum of the Z is $\widehat{AT}$ that can be defined as:

$$\widehat{AT} = \arg\min Z \tag{3.3}$$

4

Catastrophic Forgetting for Malware Classification

The content of this chapter has been adapted from a paper that is accepted and published at the Conference on Lifelong Learning Agents (CoLLAs) 2022. This paper can be found in this URL.

Current machine learning systems for malware classification often fail to account for the dynamic nature of training and test distributions, the emergence of new malware families, and the daily discovery of thousands of new samples. Training solely on new samples and classes leads to *catastrophic forgetting* (CF), a phenomenon where the system forgets previously learned tasks. Although retraining with both old and new data is effective against CF, it is costly and necessitates storing vast amounts of older software and malware samples. Continual Learning (CL) seeks to mitigate this issue through sequential training, reducing the risk of CF.

In this Chapter, we present a comprehensive study of state-of-the-art (SOTA) continual learning techniques to overcome catastrophic forgetting. Towards that end, we assess the effectiveness of widely adopted state-of-the-art CL techniques for malware classification. We evaluate 11 CL techniques – elastic weight consolidation (EWC), Online EWC, synaptic intelligence (SI), learning without forgetting (LwF), generative replay (GR), and GR with distillation, across three CL scenarios: task incremental learning (Task-IL), class incremental learning (Class-IL), and domain incremental learning (Domain-IL). We utilize two real-world malware datasets, Drebin and EMBER, noting that Drebin presents a less complex feature space than EMBER. Our Domain-IL experiments reflect realistic shifts in data distribution over time. Our comprehensive evaluation reveals that: i) in Task-IL, only LwF scales effectively with increasing task numbers and dataset complexity, and ii) all CL techniques struggle to deliver satisfactory performance in both Class-IL and Domain-IL. Interestingly, we find that replaying just 20% of stored data can save approximately 50% of training time while achieving performance near the *Joint* level. Our conclusion discusses the findings and poses open questions, encouraging further research into adaptive and intelligent malware classification/detection systems.

4.1 Introduction

Machine learning (ML) and deep learning (DL) models have become important tools in the defense of computers and networked systems. In particular, for addressing malicious software (*malware*), ML-based approaches have been used extensively in both academic literature and real-world systems for malware detection and classification, including for Win-

dows malware [35, 63, 135], PDF malware [72, 84], malicious URLs [75, 133], and Android malware [11, 49, 98]. Although DL models are fundamentally inspired by biological neurons [45, 121], their learning paradigm differs significantly from human learning processes. Biological neural networks are capable of continual, sequential learning, allowing our brains to assimilate new data and tasks on the go without significantly forgetting previously acquired information. In contrast, DL models, once trained on a dataset, become fixed and are then applied to new data in production without further adaptation. This approach typically assumes that the distributions of training and testing data remain constant. However, the adversarial nature of malware and continual evolution of benign software (*goodware*) makes for an inherently non-stationary problem.

To accommodate shifts in the data distribution over time, the model needs to be retrained regularly to maintain its effectiveness. Unfortunately, the speed with which new malware and goodware are produced results in large datasets that can be both costly to maintain and difficult to train on. For example, the AV-TEST institute registers more than 450,000 unique pieces of malware and "potentially unwanted applications (PUA)" each day [12], while VirusTotal, a crowdsourced antivirus scanning service, regularly receives more than 1 million unique pieces of software each day [145]. Over the lifetime of a malware classification model, these daily feeds can result in datasets containing upwards of a billion unique training samples spanning multiple years. Given the realities of training these models, antivirus companies must decide whether to: (i) remove some older samples from the training set, at the risk of allowing attackers to revive older malware instead of writing new ones; (ii) train less frequently, at the cost of not adjusting to changes in the distribution; or (iii) expend tremendous effort to frequently retrain over all the data.

Continual learning (CL) offers an appealing alternative to these options by enabling incremental incorporation of new information and adaptation to data distribution shifts without maintaining large datasets or incurring significant training overhead. In this work, we investigate the extent to which malware classification models suffer from catastrophic forgetting [44, 87, 113], and whether we can address this using approaches from continual learning research. While combating the problem of catastrophic forgetting has been extensively studied, explored, and applied in the context of computer vision [42, 59, 62, 130, 139, 140] using the MNIST, CIFAR10 and CIFAR100, and ImageNet datasets, its role in and applicability to malware classification tasks remains unknown.

Using two large-scale malware datasets – Drebin [11] and EMBER [10] – we examine the problems of binary malware classification and multi-class classification in the context of three CL scenarios: *domain incremental learning (Domain-IL)*, *class incremental learning (Class-IL)*, and *task incremental learning (Task-IL)*. For Domain-IL, we focus on the binary classification task of labeling software as malicious or benign, and consider the problem of incorporating shifts in the data distribution over time without losing the ability to classify older samples. For Class-IL and Task-IL, we examine the task of malware *family* classification, where a piece of malware is categorized into a well-defined family based on its code base, capabilities, and overall structure. In Class-IL, we incrementally add newly-discovered families at each iteration to mirror the ever-expanding universe of malware families found in the wild. The Task-IL formulation also incrementally adds new families to be classified, but constrains the classification task. All three scenarios represent problems faced in the anti-malware industry.

For each of these settings, we study 11 proposed CL approaches in our experiments, representing three major categories: regularization, replay, and replay with exemplars. We investigate their ability to reduce catastrophic forgetting in our two malware datasets compared with the baselines of (i) using no CL techniques and (ii) full *Joint replay*, which retrains on the full available dataset at each iteration. Finally, we investigate how much stored data may be enough to be replayed while still achieving high accuracy with lower storage and retraining costs. Our contributions are as follows:

- We are among the first to investigate continual learning in security problems, specifically in malware classification using deep learning models.
- We present a study of Domain-IL scenario using a real-world dataset with data distribution shift over the span of twelve months.
- We applied 11 distinct CL techniques in three CL scenarios using two large-scale malware datasets. We find that several CL techniques work reasonably well on Task-IL.
- We empirically show that none of the CL techniques are effective in the Domain-IL setting.
- For Class-IL, 10 of the 11 methods performed poorly, with only iCaRL [114] performing marginally better against the *Joint* replay baseline.

- Instead of storing all prior data, we find that replaying 20-50% data is enough to significantly outperform all CL techniques in Domain-IL while also reducing the training cost by 35-50% compared with 100% *Joint* replay.
- We discuss the probable causes of the poor performance offered by CL techniques to help spur future research in this problem setting.

4.2 Continual Learning Techniques Studied

In this work, we apply 11 widely studied CL techniques belonging to three major categories: regularization, replay, and replay with exemplars. We provide a brief overview of each of the studied techniques in this section.

Regularization. Elastic Weight Consolidation (EWC) [69] quantifies the importance of the weights in terms of their impact on the previous tasks' performance. As proposed, however, EWC is not scalable to a large number of tasks, as the regularization term grows with the number of tasks. EWC Online [128] is a modified version of EWC proposed to overcome this limitation. Synaptic Intelligence (SI) [151] is similar to EWC Online, but it uses a different method to measure the importance of weights and a different regularization loss.

Replay. For each task, LwF trains with both the hard labels, to adapt to the new data distribution, and the soft labels, to retain aspects of the old model. Generative Replay (GR) [130], on the other hand, replays representative data of the previous tasks using a generative model. For the generative model, $\mathcal{G}$, in our experiments, we use a symmetric VAE [68], in which the base model architecture is used for both the encoder and the decoder [139, 140]. We use a variant of GR with distillation loss, as well. We also evaluate Replay through Feedback (RtF) [140], which attempts to reduce the extensive cost of training the dual-memory-based GR technique by integrating the generative model into the main model. In addition, we study Brain-Inspired Replay (BI-R) [139] which improves upon RtF.

Replay + Exemplars. Experience replay (ER) [119] trains the model to learn new experiences (i.e., new tasks) coupled with replayed experiences (i.e., old tasks). iCaRL [114] proposes to store $\mathcal{X}$ number of samples of the previously learned classes based on a memory budget. A-GEM [25] contains an episodic memory which stores a subset of the observed examples from previously learned task and replayed along with the new sample during training.

LwF labels the current data using the model of the old task. This forms an input, output pair. The outputs in this case is not the actual label rather the predicted softmax probabilities of the current data using the old model. This input, output pairs are replayed while training the old model with the current data. Basically two things are happening here. i) we get the input, output pair of the current data using the old model, ii) we update the old model with the current data with actual input, output pairs and the input, output pairs we got from (i).

4.3 Evaluation

In this section, we describe the results from our main set of experiments, investigating the three continual learning (CL) scenarios – Domain-IL, Class-IL, and Task-IL – using 11 continual learning techniques described in Section 4.2. We compare the results with two baselines – *None* and *Joint* – and perform experiments using two malware datasets – Drebin and EMBER. We summarize our findings in Table 4.1. The results of each of the experiments are represented in both **Mean** and **Min** metrics. Mean represents the mean accuracy of all the tasks in a single experiment, such as across 10, 12, 14, 16, and 18 classes tested for Class-IL with a given CL method on Drebin. Similarly, Min represents the minimum accuracy among all the tasks, which we highlight because the weakest performance shows the degree to which a technique may not be suitable for use.

Domain-IL. In our experiments, we have 12 tasks representing the monthly data distribution shift of malware and goodware in EMBER from January to December 2018. Figure 4.1 shows the results, which are summarized in the rightmost column of Table 4.1. The mean accuracies of *None* and *Joint* baselines over the 12 tasks are 93.1% and 95.9%, respectively. We can see that Joint performance trends upward with each incremental task. This may be

Table 4.1: **Summary of the Experiments.** The average accuracy (**Mean** ($\overline{AT}$)) and minimum accuracy (**Min** ($\widehat{AT}$)) from all the tasks in each experiment. Results in **Bold** indicate accuracy values closer to *Joint* performance than *None*. **EWC-O**: EWC Online, **GR-D**: GR + Distill. Error range is omitted for the results with less than 1.0 standard deviation .

Approach	**Method**	**Drebin**				**EMBER**					
		Task-IL		**Class-IL**		**Task-IL**		**Class-IL**		**Domain-IL**	
		$\overline{AT}$	$\widehat{AT}$	$\overline{AT}$	$\widehat{AT}$	$\overline{AT}$	$\widehat{AT}$	$\overline{AT}$	$\widehat{AT}$	$\overline{AT}$	$\widehat{AT}$
Baselines	None	85.3	63.7±6	45.3	19.7	75.7	60±3.5	26.6	09.2	93.1	91.3
	Joint	**99.7**	**99.3**	**99.0**	**97.5**	**97.1**	**95±3**	**87.7**	**85±2.5**	**95.9**	**93.2**
Regul.	EWC	85.1	62±9	46.3	20±2	85.9	72±16	8.4	00.1	92.8	90.0
	EWC-O	83.9	64±7	47.1	20±2	78.8	57±31	9.0	00.2	93.1	91.5
	SI	90.7	78±7	45.3	20.0	73.6	58±4	27.3	09.5	93.0	91.1
Replay	LwF	**94.9**	88±4	27.8	5±1.5	**93.9**	**91±9**	11.9	00.7	93.2	91.7
	GR	**99.1**	**98.1**	55.1	26.2	80.8	70±6	26.9	09.3	93.2	91.6
	GR-D	**99.3**	**98.5**	55.1	26.1	82.9	73±3	27.0	09.0	93.2	91.7
	RtF	**99.4**	**98.9**	55.0	25.7	77.7	68±8.5	26.6	09.1	93.1	91.2
	BI-R	**95.9**	88±6	58.7	30±2.5	86.9	81±4	26.7	9.0	93.4	91.6
Replay + Exemplars	ER	**99.6**	**99.1**	55.2	26.7	**94.0**	**91±1**	28.0	09.4	75.9	65±4.5
	A-GEM	92.6	79±5.5	47.8	20±2	**90.4**	82±3	28.0	09.9	77.5	67.4
	iCaRL	-	-	**96.2**	**95±1**	-	-	**62.8**	**46±2.5**	-	-

expected, despite changes in the data distribution, as there is more training data in each additional task. We also see that none of the CL techniques are effective, with all of them performing closer to *None* than to *Joint*. Note that the data distribution does not change dramatically during the year – even *None* reaches at least 91.3% or better in all cases.

Class-IL. Figure 4.4 and Figure 4.2 show the results of our experiments in this scenario for Drebin and EMBER, respectively. Since Class-IL is the most difficult CL scenario, it is not surprising that the mean accuracies for *None* and *Joint* are so far apart at 45.3% and 99.0%, respectively, on Drebin. All the regularization-based techniques and LwF perform poorly and very close to *None*. Among replay techniques, BI-R performs the best with 58.7% mean accuracy. iCaRL outperforms all the other CL techniques with 96.2% mean accuracy.

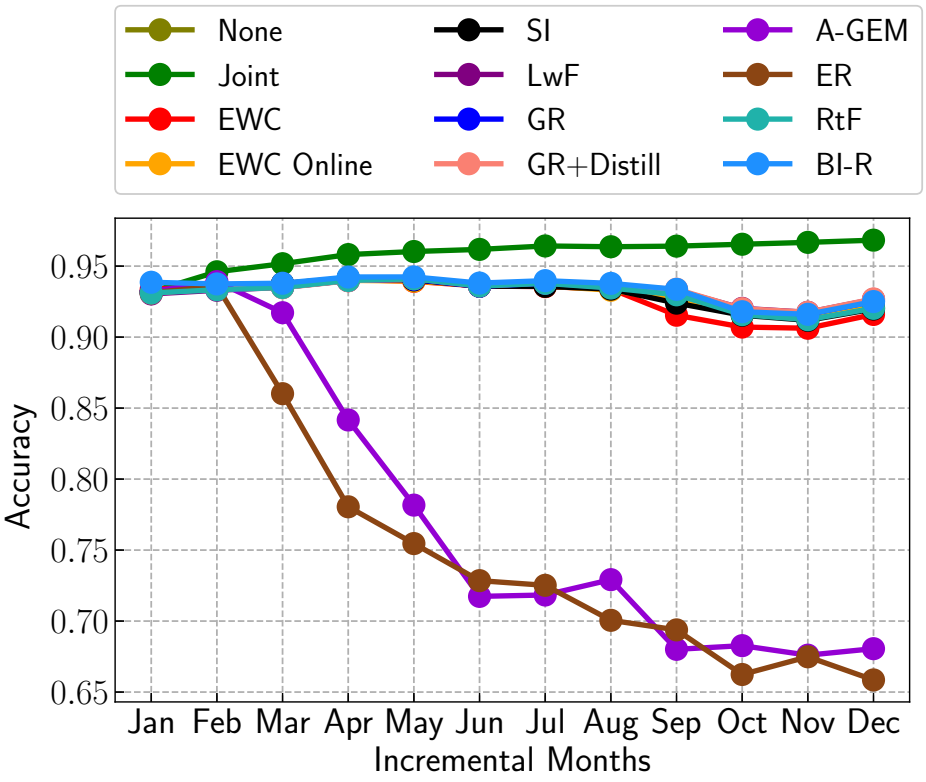


Figure 4.1: Domain-IL on EMBER: Accuracy over time.

On EMBER, the performance of all methods is about the same or even worse than *None* except iCaRL, which yields 62.8% mean accuracy. Even though iCaRL outperforms other techniques by a substantial margin, it still is far from the *Joint* baseline, which is notable given the volume of executables scanned by malware detection models each day – even a small increase in false positive or false negative rate can cause significant operational problems.

Task-IL. Figures 4.5 and 4.3 show the results of the Task-IL experiments on Drebin and EMBER, respectively. The average accuracies on the Drebin dataset of *None* and *Joint* training of the nine tasks, where each task contains two classes, are 85.3% and 99.7%, respectively. These form the effective lower and upper bounds in this setting. Among the regularization techniques, only SI yields closer to *Joint* level performance with 90.7% accuracy. The replay and replay-with-exemplars techniques are mostly effective, with GR, GR+Distill, RtF, and ER reaching over 99.0% average accuracy.

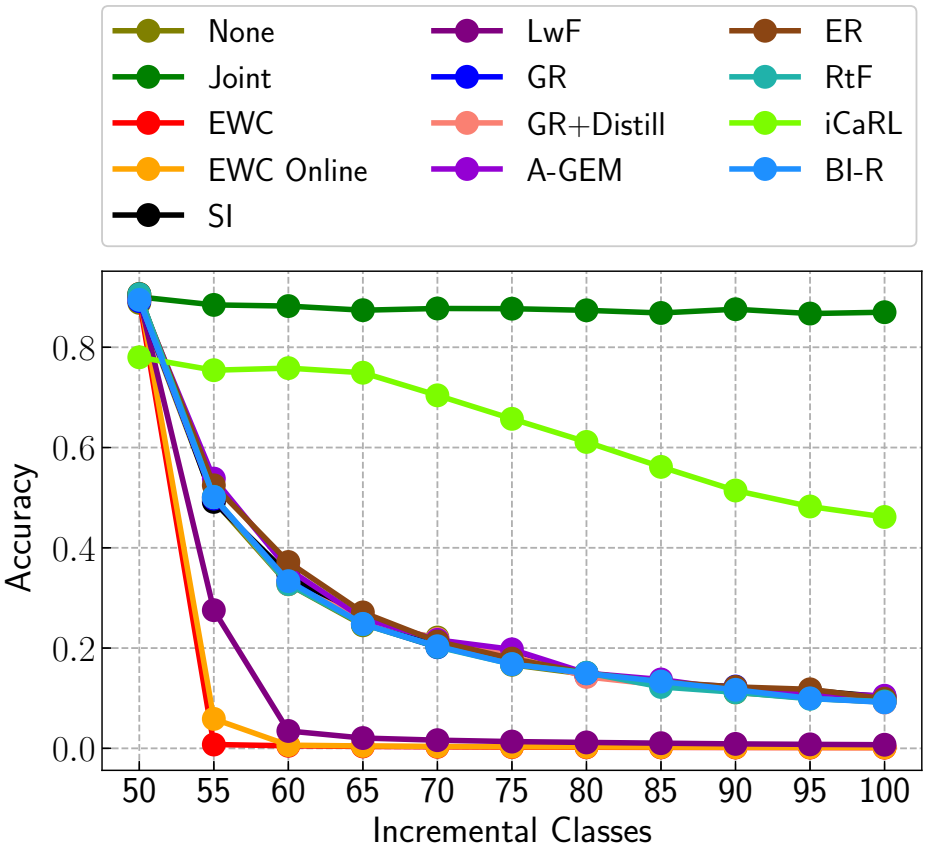


Figure 4.2: Class-IL on EMBER: Accuracy as the number of classes grows.

On EMBER, the average accuracy of *None* and *Joint* on the 20 tasks, where each task contains five classes, are 75.7% and 97.1%, respectively. None of the regularization-based techniques yield closer to *Joint* baseline performance. Among replay-based techniques, only LwF performs close to the *Joint* baseline with 93.9% mean accuracy. Replay-with-exemplars-based techniques – ER and A-GEM – outperform other techniques with 94.0% and 90.4% average accuracy, respectively. Considering both datasets, we can see that only ER performs reasonably well on both. GR, GR+Distill, RtF, BI-R, and SI all have difficulty with the larger and more complex EMBER dataset. In contrast, LwF does well on EMBER, but underwhelms on Drebin.

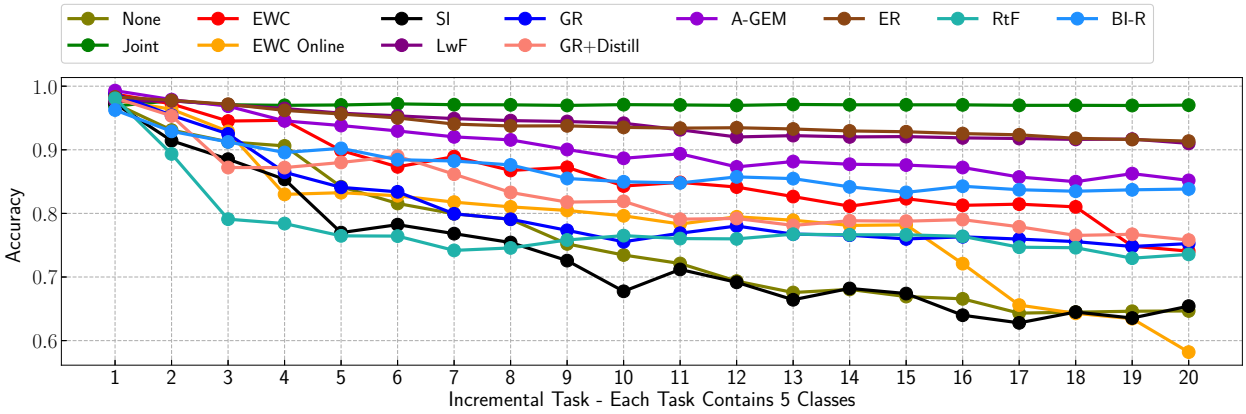


Figure 4.3: Task-IL on EMBER: Accuracy as the number of tasks grows.

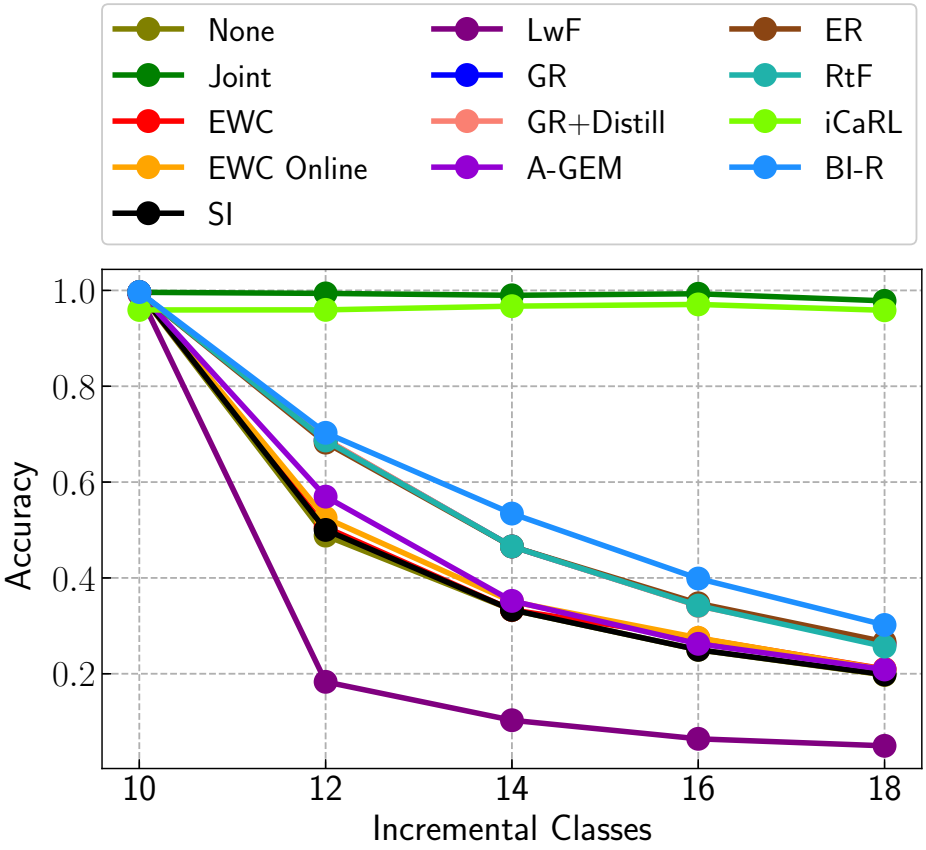


Figure 4.4: Class-IL on Drebin: Accuracy as the number of classes grows.

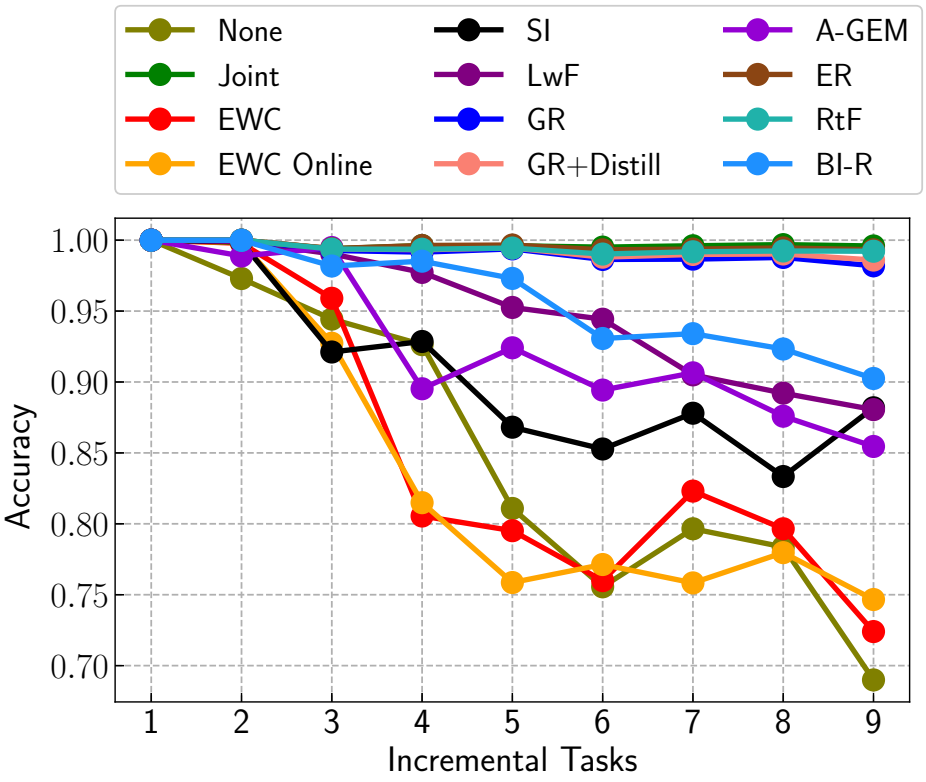


Figure 4.5: Task-IL on Drebin: Accuracy as number of tasks grows.

4.4 Partial Replay with Stored Data

In this section, we perform a second set of experiments using *partial joint replay (PJR)*. This assumes that we have ample storage capacity and can maintain huge amounts of both historical and current data to be accessed at any time. Indeed, this may be the case for some companies doing malware detection. In this setting, CL techniques are not required, as the real data is available. Nevertheless, the huge volume of historical malware data makes it very expensive to train over all of it using full *Joint* replay. Thus, our question is how much of the historical data is needed to achieve high levels of accuracy using a strategy of sampled partial joint replay. In these experiments, we use early stopping with patience = 5, as replaying stored data causes the model to converge faster.

We focus on Domain-IL, which is the CL setting most applicable to malware classification. We performed seven sets of experiments with varying fractions – *None* (0%), 1%, 5%, 10%, 20%, 50%, and *Joint* (100%) – of the stored data to be replayed in the subsequent tasks.

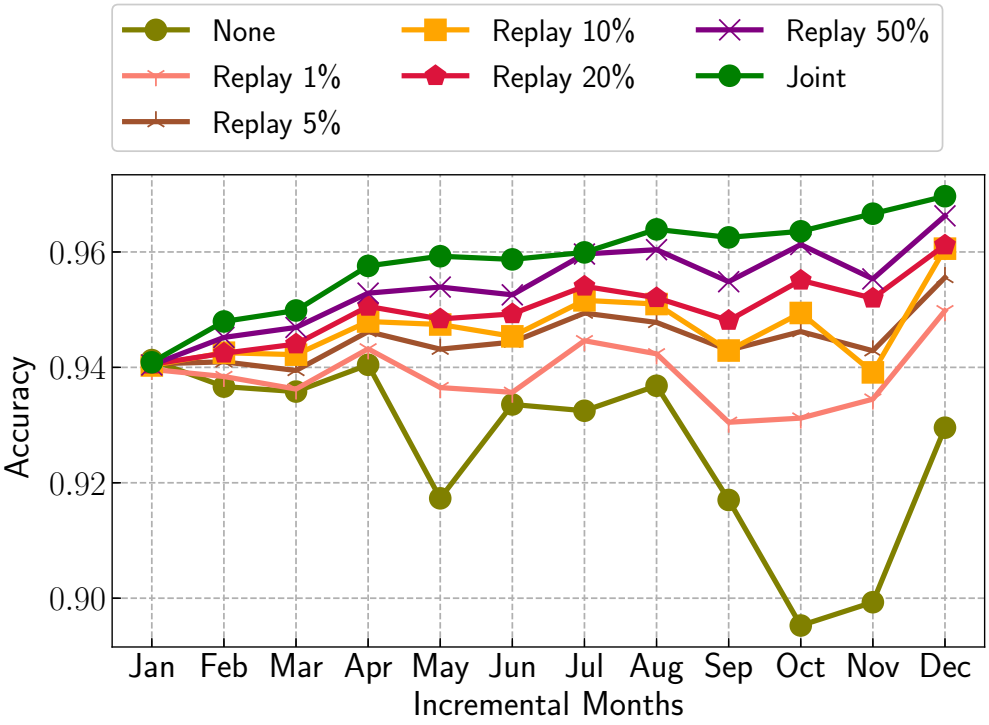


Figure 4.6: Partial Joint Replay in EMBER: Accuracy over time.

Figure 4.6 shows the results, while Figure 4.7 shows the corresponding training times. The mean accuracy of the *Joint* and *None* baselines is 96.4% and 93.0%, respectively.

The average accuracy over all the tasks with 1% of replayed data is 93.9% – nearly 1% higher than *None* and easily outperforming all CL techniques evaluated above. It is perhaps surprising to see that only 1% replayed data results in such a significant improvement in the average accuracy, but note that the total set of old training samples grows larger with each task, making it *multiple times larger than the new data* for most tasks in our experiment. Even a small sample of 1% thus becomes a significant fraction of all training data. When we increase the replayed data fraction to 5%, 10%, 20%, and 50%, average accuracy grows to 94.5%, 94.7%, 95.0%, and 95.4%, respectively. With 20% replayed data, the accuracy is only 1.4% lower than *Joint* replay. Reducing the replay data improves training efficiency significantly, as we can see in Figure 4.7. With our dataset, the expected amount of training effort using 20% of the replay data shrinks by 50% compared to full *Joint* replay. Even with 50% of the replay data, the expected training effort shrinks by 35%.

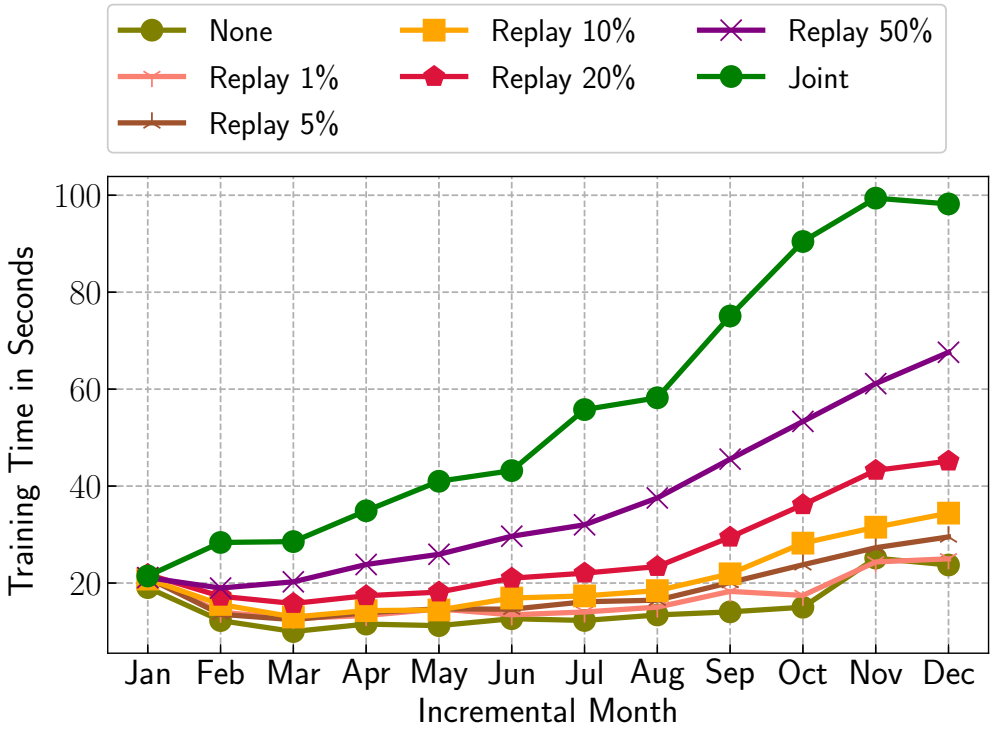
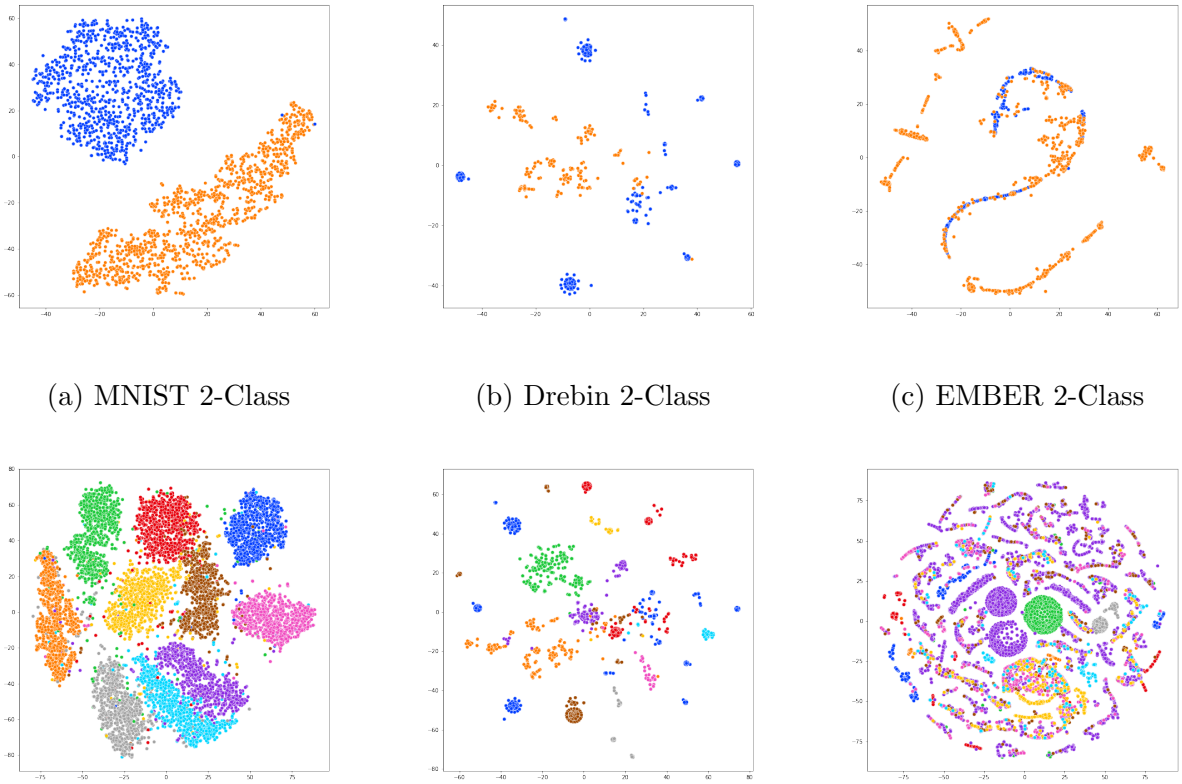


Figure 4.7: Partial Joint Replay in EMBER: Change in training time over time.

4.5 Discussion

Continual learning (CL) for malware classification can provide a number of benefits including: i) alleviating the need to store some or all of the previous data, ii) relaxing the requirement for access to previous data, iii) keeping training data private while sharing a model that can be updated, and iv) reducing computational cost. From our experiments, however, we can see that none of the CL techniques could contribute significantly in the Domain-IL setting, and only iCaRL provided reasonable performance in Class-IL setting, though even in that setting the gap between *Joint* and iCaRL was substantial – upwards of 24% on EMBER. For Task-IL, we observed reasonable performance by several methods on Drebin and on EMBER. Task-IL for malware, however, is of least significance, as adding new tasks is likely to occur less frequently than adding new classes or observing domain shift. Given these limitations, we now examine probable causes to help spur future research in CL considering these problem settings and datasets.



(a) MNIST 2-Class (b) Drebin 2-Class (c) EMBER 2-Class

(d) MNIST 10-Class (e) Drebin 10-Class (f) EMBER 10-Class

Figure 4.8: Feature space visualization using t-SNE in Class-IL scenario.

Dataset Complexity. Most papers on CL techniques use the MNIST dataset for their experiments, but MNIST is relatively simple in its data distribution and feature space. Figure 4.8 shows a t-SNE projection [143] of the feature space to demonstrate the complexity of MNIST, Drebin, and EMBER dataset with both two and 10 classes for each. In MNIST, we can see a clear separation for the two classes and relatively clear separations for 10 classes. In Drebin, we see a somewhat sparser, yet complex space, particularly for 10 classes where some classes become difficult to separate. Meanwhile, the EMBER feature space is extremely complex with significant overlap between classes. Additionally, we see class imbalance in the EMBER dataset, which reflects the realistic sampling of real-world data. MNIST has equal numbers of samples in both Class-IL and Task-IL.

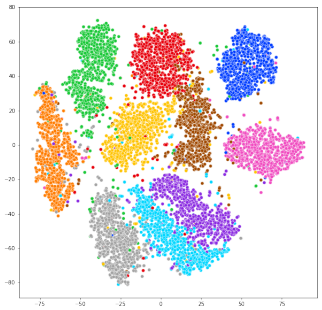
Another interesting distinction is the semantically-rich feature space of the malware classifi-

cation datasets. Unlike the image domain, malware classifiers typically use tabular features derived from parsing and analysis of the binary program (e.g., headers, byte sequences, APIs). There are strong semantic constraints on the values that these features can take on and their relationships, which affects the shape of the feature space that feasible samples occupy. It is possible that these constraints and the inherent complexity of the classification task render generative, distillation, and regularization-based CL methods ineffective. Taken together, our results show that it is important to evaluate CL techniques on more complex, realistic datasets to understand how their effectiveness may generalize and apply in practical settings.

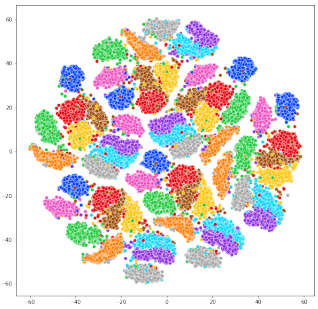
Real Domain Shifts. Prior work has studied Domain-IL only in a simulated experimental setting to mimic data distribution shift using the *permuted MNIST* protocol, in which pixels of the MNIST images are permuted around the image in a consistent way across the dataset to create the next task. This protocol clearly does not represent a realistic data distribution shift, and certainly not the strongly constrained feature space found in malware classification tasks. As such, the performance offered by CL techniques in this simulated setting is not a realistic measurement that would be likely to generalize to other data. In contrast, our experimental setting in Domain-IL represents a sample of real-world data that shifts over time.

In Figure 4.9, we show a t-SNE visualization of MNIST and EMBER data distribution shift in Domain-IL. In summary, even though permuted MNIST is challenging for people to visualize, it appears to actually create reasonably solvable classification tasks for DL models. The natural distribution shift found in EMBER, on the other hand, creates very difficult tasks for the classifier.

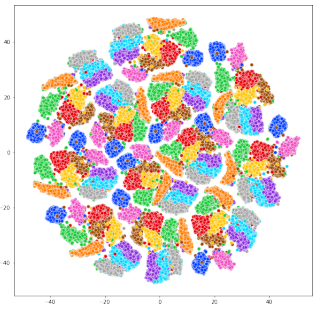
Partial Joint Replay (PJR). Simple partial joint replay offers significant improvements over the *None* benchmark while also reducing computational effort. It would be interesting for future work to evaluate the effectiveness of combining partial joint replay and CL techniques, and our results here offer a pragmatic benchmark for future efforts. In some ways, the widely studied iCaRL CL technique is analogous to PJR in that iCaRL replays the samples stored in the memory buffer in the training phase, coupling those old samples with the new ones. As originally designed, however, iCaRL has a fixed memory budget that can limit its



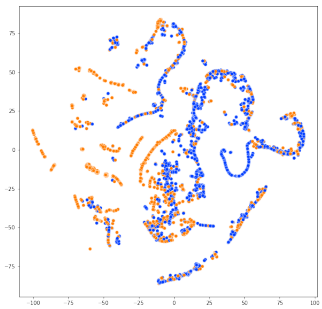
(a) **Task 1**: Original MNIST (no permutation).



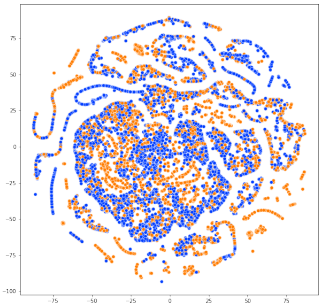
(b) **Task 6**: Cumulative MNIST data from Task 1 to Task 6 using *permuted* MNIST protocol.



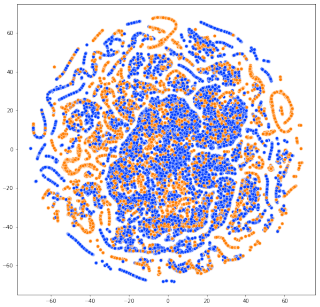
(c) **Task 12**: Cumulative MNIST data from Task 1 to Task 12 using *permuted* MNIST protocol.



(d) **Task 1**: EMBER data of January



(e) **Task 6**: Accumulated EMBER data from January to June.



(f) **Task 12**: Accumulated EMBER data from January to December.

Figure 4.9: MNIST and EMBER data distribution shift in Domain-IL scenario using t-SNE plot.

effectiveness with increasing classes. We can validate this by observing the performance of iCaRL for Drebin and EMBER datasets shown in Table 4.1 and Figures 4.4 and 4.2. Drebin has 18 classes and the performance gap between iCaRL and *Joint* is around 3%. On the other hand, EMBER has 100 classes and iCaRL's performance gap grows to almost 25%.

4.6 Conclusion

In this work, we investigate 11 continual learning techniques in three scenarios using two large, real-world malware datasets. Unfortunately, our findings demonstrate that in almost all cases those CL techniques are ineffective at preventing catastrophic forgetting and maintaining classification performance. Of the techniques evaluated, only iCaRL [114] performed near *Joint* replay baselines. Meanwhile, we also found that a simple partial joint replay strategy of training on a small fraction of the historical data is enough to achieve reasonable performance with lower cost.

Taken as a whole, our results underscore the need for additional study of CL in complex, non-stationary problem settings. We hypothesize that the strong semantic constraints of the features used for malware classification tasks, along with the complex data distributions induced by natural drift in this space, lie at the heart of the performance differences between existing CL literature and the results in our paper. In particular, these unique aspects of the malware classification domain may severely limit the applicability of generative, distillation, and regularization-based methods due to their inability to sufficiently capture the inherent complexity. At the same time, these results also hint at possible avenues of future work. For example, the relative success of partial joint replay and iCaRL demonstrate the importance of sample selection during replay and the possibility of developing more effective strategies that ensure optimal coverage over feasible regions of the feature space. Most importantly, however, we hope this work spurs further exploration of CL in the cybersecurity domain and a broader conversation about real-world application of CL techniques.

5

Malware Analysis with Diversity-Aware Replay

The content of this chapter has been adapted from a paper that is submitted to a conference and under review.

The findings of Chapter 4 empirically show us that the conventional CL techniques, often developed for image processing tasks, struggle to maintain performance on older malware samples while adapting to new data. This limitation is particularly critical in malware detection, as it might enable attackers to recycle old malware. The results of Chapter 4 motivated us to investigate even further to uncover the unique nature of malware which may be inherent to malware data distribution. And based on the outcome of the analysis, we endeavor to explore a continual learning system that is tailored for malware classification tasks taking into account of the unique nature of malware data distribution.

Towards that end, in this Chapter, we first present a comprehensive exploratory analysis, revealing unique challenges and characteristics of malware datasets. We find that malware families are diverse and difficult to characterize easily, requiring a variety of samples to learn a robust representation. Based on these findings, we propose *M*alware *A*nalysis with *D*iversity-*A*ware *R*eplay (MADAR), a CL framework that accounts for the unique properties and challenges of the malware data distribution. MADAR consists of two novel CL techniques – Isolation Forest-Based Sampling (IFS) and Anomaly Weights-Based Sampling (AWS). We extensively evaluate these techniques using both Windows and Android malware, and 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 classification domain.

5.1 Introduction

Advancements in machine learning have significantly improved malware detection and classification, with notable successes across various types of software, such as Windows executables [35, 63], PDFs [84], and Android applications [11]. Traditional models, trained on static datasets, are expected to perform well on new data under the assumption of a constant data distribution. In reality, though, both malware and benign software (i.e., *goodware*) are ever-evolving and require regular model updates to keep up with these changes in data distribution to maintain effectiveness. For example, the AV-TEST Institute logs about 450,000 new malware samples daily [12], and VirusTotal processes about one million new submissions each day [145].

Training a malware classification model solely on new data can lead to *catastrophic forgetting (CF)* [44], which may result in both misclassifying known benign software and allowing attackers to bypass detection with older malware strains. To address this, antivirus vendors can keep older samples and retrain over all of them during model updates, but the enormous volume of such samples makes the storage and computational costs of this approach excessive. Continual learning (CL) offers a solution to this problem by enabling models to adapt to new data without the need for maintaining large datasets or extensive retraining [7, 139].

While combating catastrophic forgetting has been extensively studied, explored, and applied in the context of computer vision [59, 130, 139], there are very few studies of CL in the context of malware classification. Previous work reveals that none of the CL techniques originally designed for computer vision problems offer acceptable performance in malware classification, due in part to the strong semantics of malware features and the high level of diversity found in the malware ecosystem [109].

In this study, we first delve into the unique complexities of malware data distributions using the EMBER dataset [10] of Windows malware and goodware. Our analysis highlights the diversity 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*iversity-*A*ware *R*eplay, a diversity-aware, replay-based continual learning strategy tailored for malware classification. This approach 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) [81] to pinpoint these critical novel samples. MADAR consists of two variants: Isolation Forest-based Sampling (IFS), which utilizes the model's input features, and Anomalous-Weights-based Sampling (AWS), which uses model weights for a more compact representation.

We then evaluate these techniques with comprehensive experiments on the EMBER dataset in two key CL scenarios that mirror common malware classification tasks: domain incremental learning (Domain-IL) and class incremental learning (Class-IL), as highlighted in prior work [109]. Additionally, we have curated two new benchmarks of Android malware from the AndroZoo repository [8] to experiment in both Domain-IL (*AZ-Domain*) and Class-IL (*AZ-Class*) scenarios. Our results on these datasets confirm that MADAR is indeed effective and much better than the prior state-of-the-art CL methods in the face of realistic data

distribution shifts.

In summary, the contributions of this study are:

- We provide an exploratory analysis of malware's diversity and show how it creates unique challenges in the continuous learning setting.
- We develop two large-scale, realistic Android malware benchmarks covering both Domain-IL and Class-IL scenarios.
- In Domain-IL scenarios, we show that MADAR performs much better than prior CL techniques. On the AZ dataset, for example, MADAR comes within 0.4% average accuracy of the joint training baseline using just 50K training samples versus 680K.
- 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.

5.2 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. The malware samples in the dataset are labeled with the *avclass* labels that have been identified as belonging to a particular family. After conducting a thorough examination of the month-by-month malware samples in the EMBER data, we discovered that there are 2899 distinct malware families present in the accumulated malware samples. Additionally, we found 11433 samples without any *avclass* labels but were still identified as malware. These additional 11,433 samples lacking clear family labels are assigned the label *other*.

Table 5.1 provides a summary of the number of goodware and malware samples in each task month, along with the unique number of malware families represented. For example, a closer look at January's data shows that 32,491 malware samples are spread across 913 families.

Table 5.1: EMBER task based frequency of Goodware and Malware Samples, and frequencies of families in malware samples in each task.

Task	**#of Goodware**	**#of Malware**	**#of Malware 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

On average, each task month, ranging from January to December, includes samples from over 800 distinct malware families.

Furthermore, the distributional patterns of many malware families within the feature space add another layer of diversity. As depicted in Figure 5.1, the t-SNE projection of EMBER's January 2018 malware features illustrates that classes, denoted by different colors, are not confined to single, well-defined regions. Instead, larger classes fragment into subsets scattered across the feature space. Therefore, to faithfully represent the malware distribution, it's crucial to include samples from various regions within each class, capturing the diversity both across and within families. While there is some overlap between families, distinct clusters can still be identified, emphasizing the necessity of including samples from each, particularly those from less represented families, to accurately portray the malware ecosystem for any given period.

Assigning specific family labels to malware samples is challenging due to the subjective nature of malware classification. This leads to many samples remaining unlabeled, complicating the dataset and highlighting the importance of careful sample selection for analysis. In

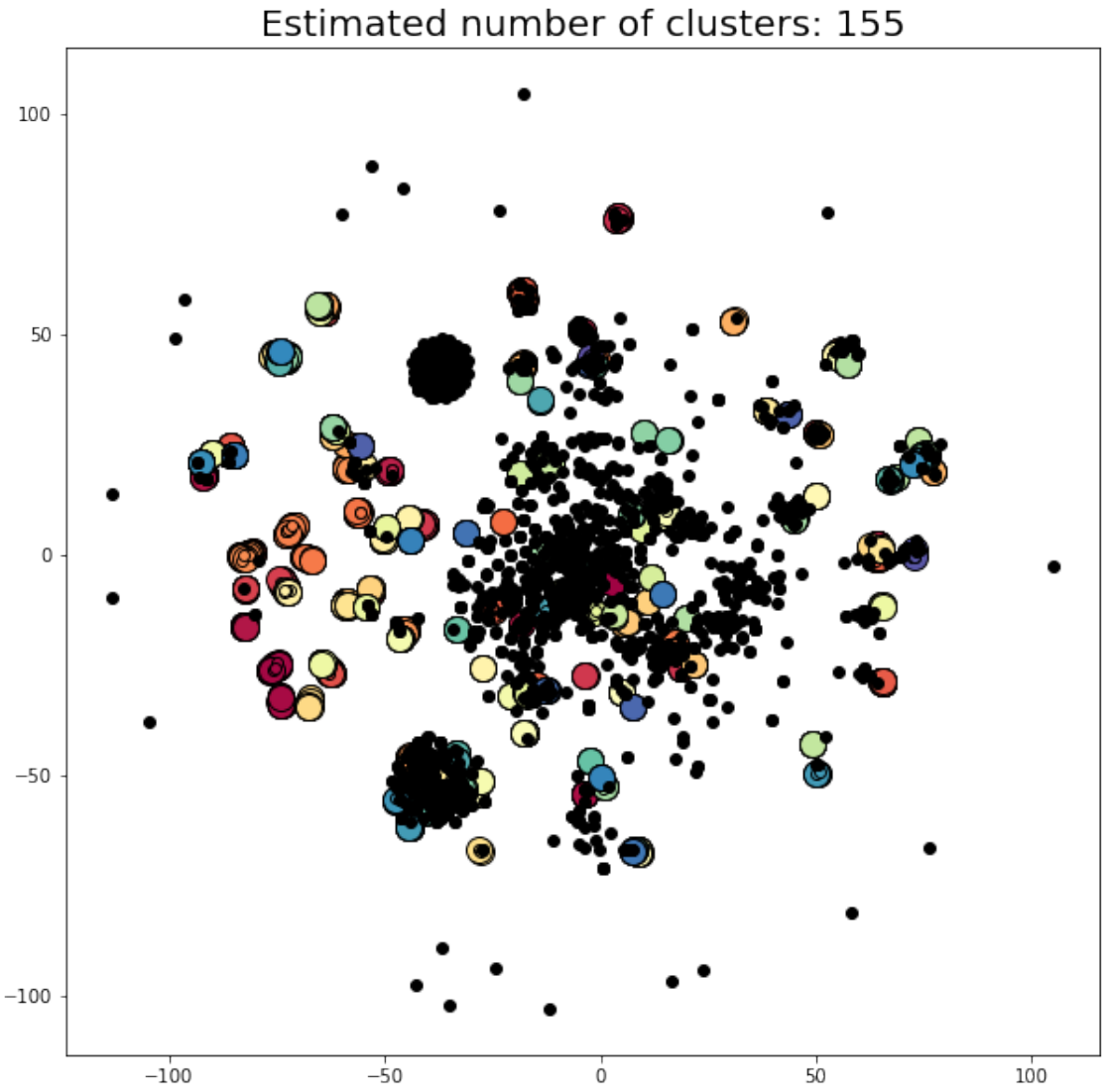


Figure 5.1: t-SNE projection of EMBER malware from January 2018.

our examination of the EMBER dataset, we found over 10,000 malware samples without *avclass* labels, shown in Figure 5.2. This situation makes malware classification difficult, as accurately identifying new malware requires significant expertise and time. Additionally, these unlabeled samples often don't group together in simplified visualizations, suggesting they might stand out as outliers compared to samples from known families.

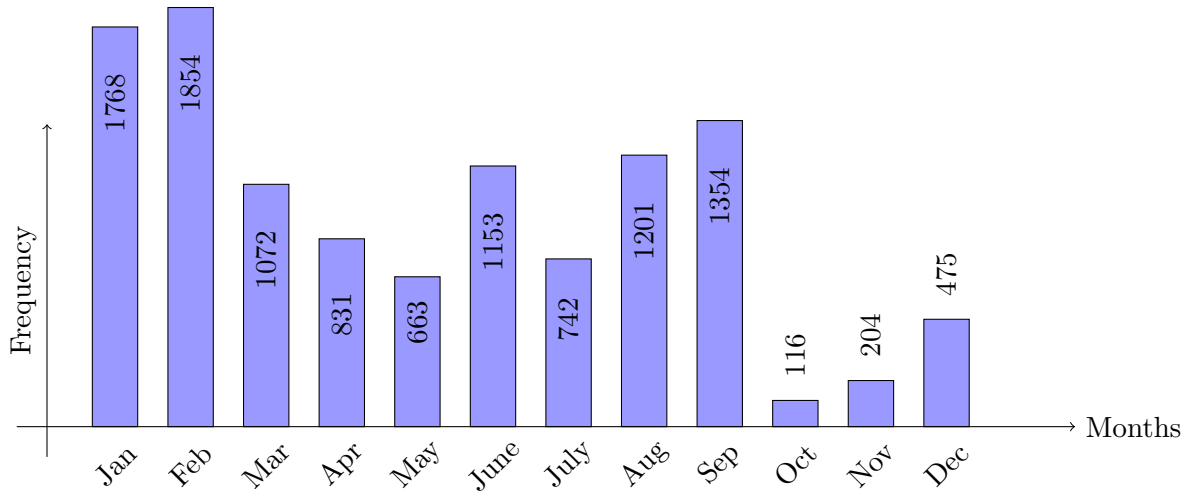


Figure 5.2: EMBER Malware samples without AV-Class labels.

In addition, we analyze the prevalence of malware families over time, identifying both recurring and newly detected families each month, as illustrated in Figure 5.3. Unlike the static datasets commonly used in continual learning research, our dataset exhibits significant variation in family representation over time. For instance, of the 913 families identified in January, only 551 reappear in February, with 425 new families emerging. This dynamic is visualized by the green bars, which represent families the model is encountering for the first time, and the red bars, which denote families previously observed. Notably, several families learned in earlier months do not recur in subsequent datasets. This discontinuity suggests that the model's performance on these non-recurring families may decline, potentially exacerbating catastrophic forgetting. Such churn underscores the challenge of maintaining training data continuity and highlights the necessity of continually updating and refining machine learning models to maintain efficacy in malware detection. This situation also points to the need for adaptive continual learning strategies tailored to the malware domain.

We conducted a supplementary analysis to examine how the most common malware families vary across different tasks in the EMBER dataset. We identified the top 15 families based on their sample frequency within each task and their appearance across task months, as illustrated in Figure 5.4. The figure does not display the frequency of samples for each malware family. Instead, it illustrates the frequency of appearance in each task month. For instance, the *emotet* family is observed in 11 out of 12 task months. However, we do not provide the specific sample counts for *emotet* across different task months, such as January,

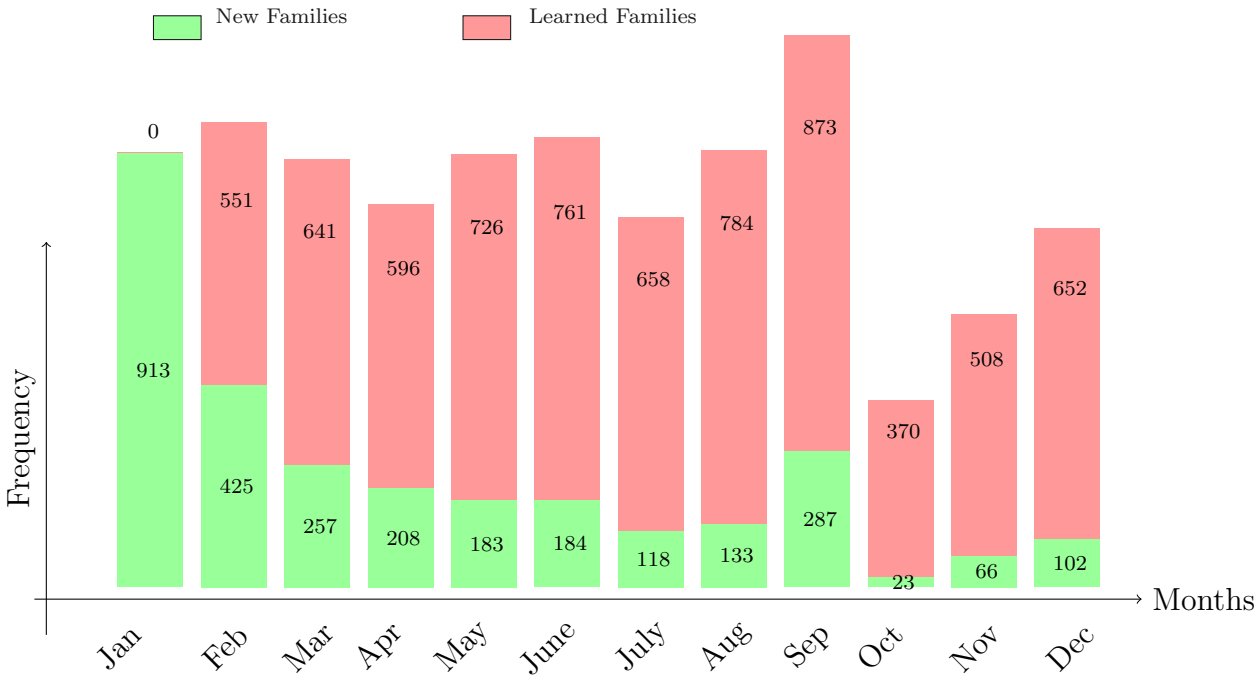


Figure 5.3: New and already learned families in each task.

February, and so on.

Our findings indicate variability in these top families, suggesting that families prevalent in earlier tasks might not maintain their prominence in later ones. Notably, the `emotet` family is the most consistent, appearing in 11 of the tasks, followed by `fareit` and `zusy`, each appearing in 8 and 7 tasks, respectively. This variability demonstrates significant concept drift within the malware dataset, posing a challenge for research and highlighting the necessity for continual learning techniques that can adapt to such drift [14, 64, 149]. Effective continual learning approaches should, therefore, be designed to accommodate the concept drift observed in malware datasets.

In summary, 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 [109].

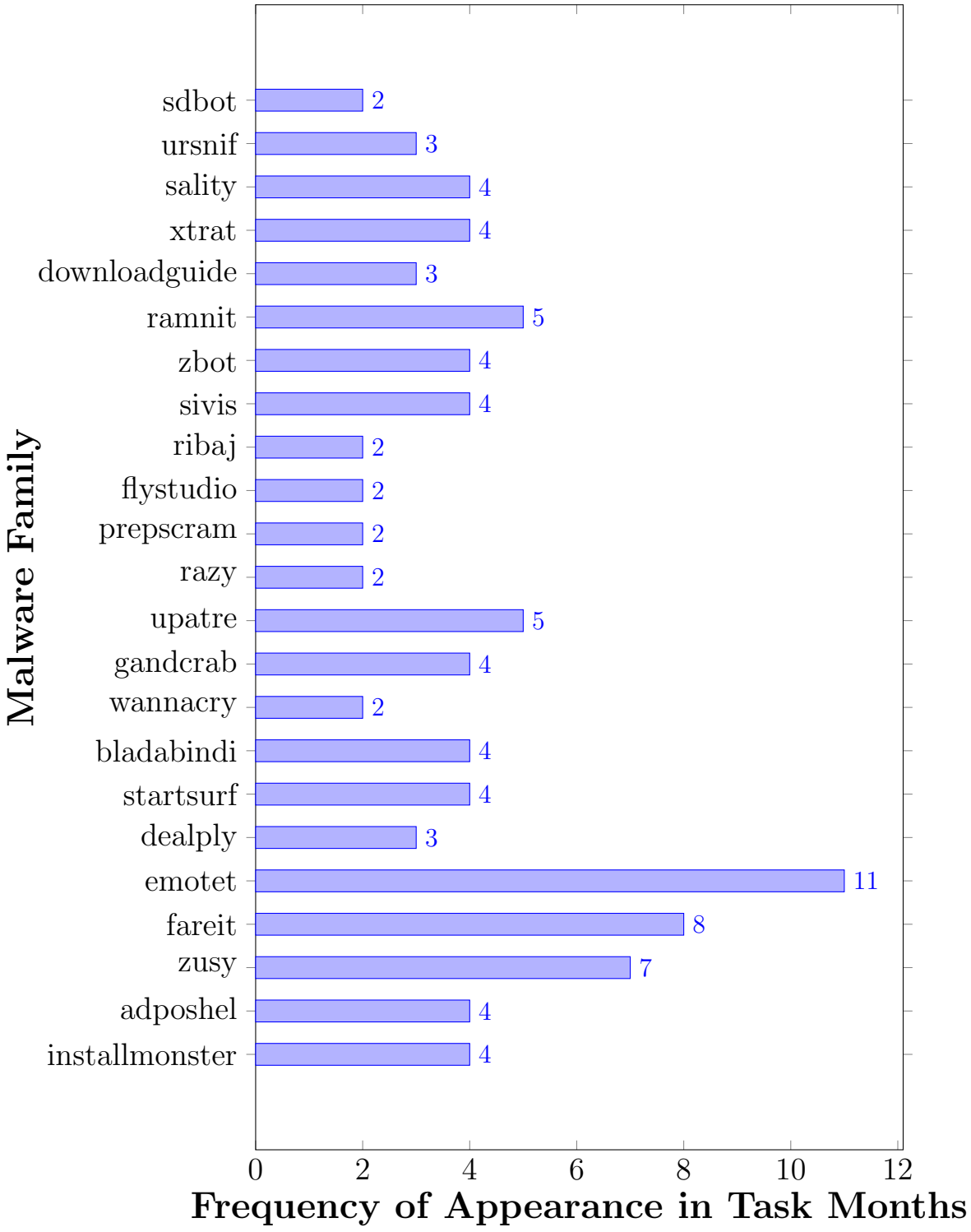


Figure 5.4: Frequency of Top 15 Malware Families based on the Appearance in Task Months.

5.3 Additional Baseline – Global Reservoir Sampling (GRS)

Algorithm 1: Global Reservoir Sampling (GRS).

Input : $\mathcal{T}_c$ – Current task
X_{T_c}, Y_{T_c} – Samples and associated Labels of current task
$\mathcal{G}_{dp}$ – Global Data Pool
$\mathcal{R}_p$ – Memory Buffer Constraint

Output: X_{replay}, Y_{replay} – Replay Samples and associated Labels

if $T_c == 0$ **then**
    $\nabla\mathcal{G}_{dp} \leftarrow X_{T_c}, Y_{T_c}$ `// Create global data pool with the samples from current task`
else
    $X_{stored}, Y_{stored} \leftarrow \mathcal{G}_{dp}$ `// Get stored samples and associated labels from the global data pool`
    $I_{all} =$ `random.shuffle(range(`Y_{stored}`))` `// Randomly shuffled index numbers of the stored samples`
    $I_{replay} = I_{all}[: \mathcal{R}_p]$ `// Index numbers of the replay samples to be picked`
    $X_{replay}, Y_{replay} = X_{stored}[: I_{replay}], Y_{stored}[: I_{replay}]$ `// Get the replay samples and associated labels`
    $\nabla\mathcal{G}_{dp} \leftarrow X_{T_c}, Y_{T_c}$ `// Update global data pool with the samples from current task`

Partial joint replay (PJR) randomly selects $X\%$-of the stored data till the previous task T_{N-1} to be used along with the samples of the current task T_N. PJR is a term used in our previous work where we investigate various continual learning techniques in the malware classification problem space [109]. In this work, we attempt to formalize PJR in light of the literature of the continual learning space. One of the widely term of PJR technique used in the previous work is called *Reservoir Sampling* (RS) [26, 115, 146, 152]. Given a memory β, RS randomly selects M-samples without replacing from the global data pool $\mathcal{G}_{dp}$ of $\mathcal{Z}$-samples. For each of the continual task, samples in the β is updated adhering some constraint $\mathcal{C}$. For our work, $\mathcal{C}$ is the number of replay samples to choose from the $\mathcal{G}_{dp}$. The samples are chosen from $\mathcal{G}_{dp}$ with equal probability $\beta/\mathcal{Z}$ by randomly shuffling the indexes of $\mathcal{Z}$ samples. Unless otherwise specified, from this point we will refer random sample selection as RS instead of PJR with some configuration specific modification. In the continual learning literature, each of the current training batch is updated with samples from the current data $X_{current}$ and β for mini-batch stochastic training. For our work, however, we mix $X_{current}$ and β in the form of `concatenation`, apply `random shuffling`, and then prepare training mini-batch.

As we choose samples for β from the global data pool $\mathcal{G}_{dp}$ and global data pool stores all

the previous data up to the previous task T_{N-1}, we refer this sampling technique `Global Reservoir Sampling` (GRS). Note that, Goodware and Malware samples for this configuration form a mixed data pool for Domain-IL. A detailed algorithmic representation of GRS is depicted in Algorithm 1. Given current task $\mathcal{T}_c$, samples of the current task (X_{T_c}, Y_{T_c}), global data pool $\mathcal{G}_{dp}$ which is empty at the beginning of the learning process, and a constraint $\mathcal{C}$ for the memory β, this algorithm returns the replay samples for the current task. For the beginning task, there is no available replay samples, hence, $\mathcal{G}_{dp}$ is updated only. For any other task than the initial task, the algorithm selects replay samples X_{replay} for T_c and after the selection, it updates $\mathcal{G}_{dp}$ with (X_{T_c}, Y_{T_c}) to be used in the next task T_{c+1}. Rahman et al. [109] investigated GRS 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.

5.4 Diversity Aware Replay

Here, we introduce the MADAR framework for CL in malware classification with two diversity-aware replay variants: Isolation Forest-based Sampling (IFS) and Anomalous Weights-based Sampling (AWS).

5.4.1 Isolation Forest-based Sampling (IFS)

Building on the exploratory analysis in Section 5.2, 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 diversity *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 diversity within a class.

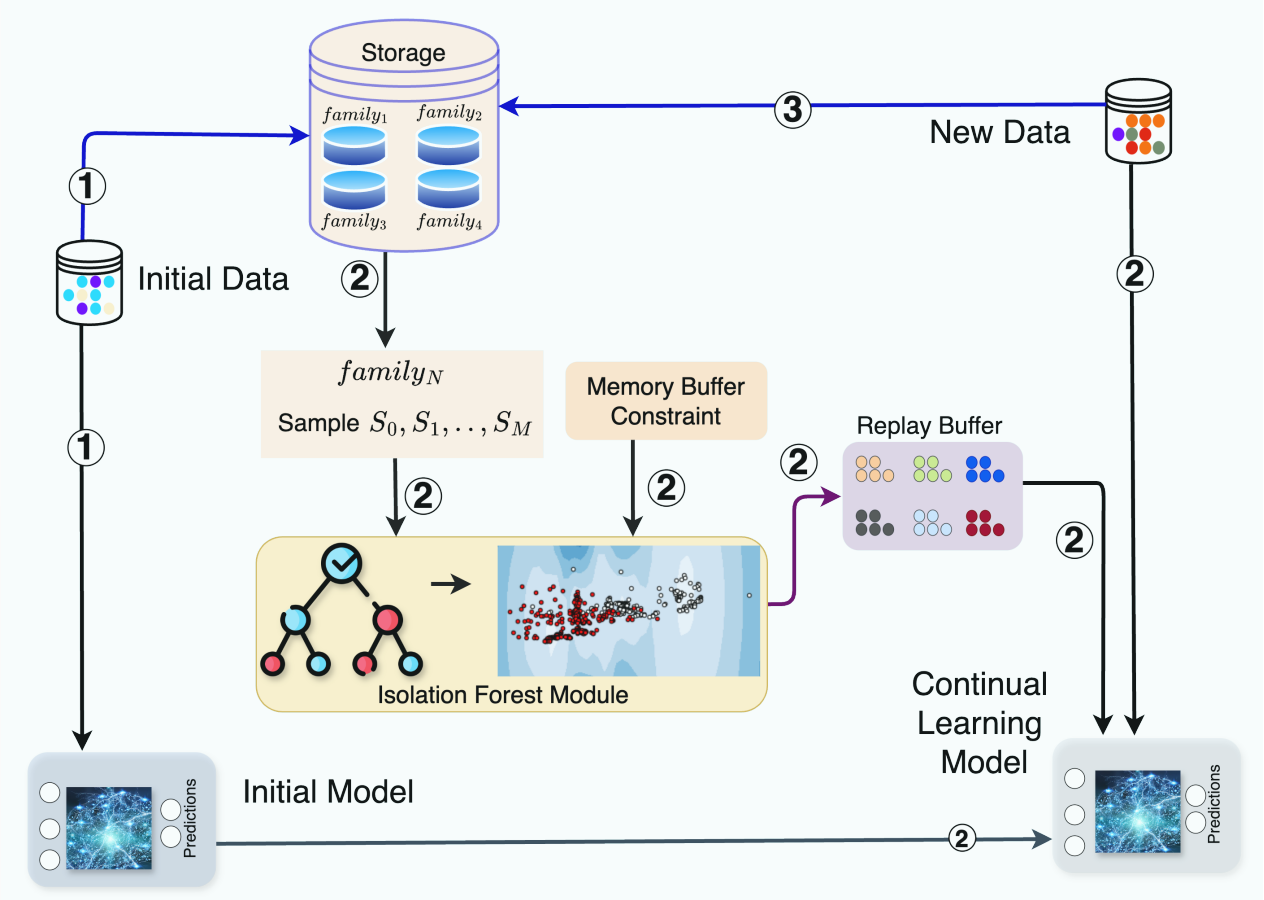


Figure 5.5: Depiction of MADAR framework with Isolation Forest based Sampling (IFS).

Isolation Forest (IF) [81] 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 the Algorithm 2.

A high-level representation of the Isolation Forest-based Sampling (IFS) is shown in Figure 5.5 within the MADAR framework, which operates in three distinct phases. (1) in the figure marks the initial phase, where the model undergoes its first training using a static training approach with the initial dataset. It is important to note that weight initialization occurs solely during this stage. This phase involves a thorough exploration of the model's hyperparameters such as learning rate, batch size, number of layers, and optimizer—to en-

hance model performance. Upon completing this initial training, the utilized data is stored, ensuring it is organized based on malware family categories.

The model then progresses to the continual learning (CL) phase, indicated by (2) in the figure. During this phase, the model is exposed only to new data and selected data from the replay buffer, assuming a replay-based technique is employed to ensure model stability. The replay buffer is filled with samples chosen by the IFS, adhering to a predefined memory constraint. The isolation forest module then analyzes samples from each family, identifying both anomalous and similar samples based on IFS's hyperparameters (explained in the next section), and selects replay samples for each family within the memory buffer's limits.

Finally, as depicted by (3), newly encountered data is stored, continuing the organization by malware family. This structured approach facilitates the model's ongoing learning and adaptation to new data while maintaining the integrity of previously acquired knowledge.

5.4.2 Procedure

We now describe IFS using Domain-IL, noting that Class-IL is similar. IFS processes goodware and malware separately, with malware further categorized by family. It begins by creating a global data pool $\mathcal{P}$ for all goodware and family-based malware samples, and a dictionary $\mathcal{D}$ to track malware families and their frequencies up to the current task. For each new task, IFS divides goodware (X_{good}) and malware (X_{mal}) 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. If the dataset, like EMBER, is relatively balanced between goodware and malware, we use $\gamma = 0.5$. For a significantly imbalanced datasets like AZ, it is better to tune γ. The ratio of goodware and malware samples in the AZ datasets is 9:1, so we use $\gamma = 0.9$ to get the β_G and the rest as β_M.

Before each new task, MADAR trains the already trained classifier with a mix of new samples from the most recent task and replay samples from prior tasks. The replay samples include benign replay samples $R_{good} \subset X_{good}$ and malicious replay samples $R_{mal} \subset X_{mal}$. R_{mal} is sampled from malware families rather than randomly from all of X_{mal}.

To determine how many samples to select from each family f, its *family budget* $\mathcal{B}_f$, we use

Algorithm 2: Isolation Forest based Sampling (IFS)

Input : c – Current task number
X_c, Y_c – Samples and labels of c
$\mathcal{P}$ – Data pool
β – Memory budget
γ – Split of goodware and malware
Ω – Split of Typical and anomalous samples
Output: X_{train}, Y_{train} – Training samples and labels

init $\mathcal{P}$; // Global data pool
init $\mathcal{D} \leftarrow \{f : s_f\}$; // Dictionary of each malware family f and its number of samples s_f
if $c = 0$ **then**
  $\mathcal{P} \leftarrow X_c, Y_c$;
  $X_{good}, X_{mal} \leftarrow \mathcal{P}$;
  $\nabla\mathcal{D} \leftarrow X_{mal}$; // Update dictionary
  $X_{train},\ Y_{train} \leftarrow X_c, Y_c$;
else
  $X_{good}, X_{mal} \leftarrow \mathcal{P}$; // Separate out goodware and malware samples from the global family data pool
  // Set budget ratios
  $\beta_G \leftarrow \beta \cdot \gamma$; // Goodware budget
  $\beta_M \leftarrow (1-\gamma) \cdot \beta$; // Malware budget
  $\beta_T \leftarrow \beta_M \cdot \Omega$; // Typical budget
  $\beta_A \leftarrow \beta_M \cdot (1-\Omega)$; // Anomalous budget
  if *Uniform budgeting* **then**
    $f_{tot} \leftarrow \mathcal{D}$; // # of malware families in $\mathcal{D}$
    $\mathcal{B}_f \leftarrow \beta_M / f_{tot}$; // Family budgets are Uniform
  $R_{mal} \leftarrow [\,]$; // Malware replay samples
  for $X_f \subseteq X_{mal}$ // Each family f
  do
    $m_f \leftarrow |X_f|$; // $of malware in X_f
    if *Ratio Budgeting* **then**
      $m_{tot} \leftarrow |\mathcal{P}|$; // Total #of malware
      $B_f \leftarrow (m_f / m_{tot}) * \beta_M$; // Family budget
    if $m_f <= B_f$ **then**
      R_{mal}.append(X_f);
    else
      $(T_f, A_f) \leftarrow$ IF$(X_f, \beta_T, \beta_A, m_f)$; // Typical and Anomalous samples selected w/ IF
      R_{mal}.append(T_f, A_f)
  $R_{good} \leftarrow$ sample$(X_{good}$, len$(R_{mal}))$; // Randomly select as much goodware as malware
  $X_{replay} \leftarrow (R_{good}, R_{mal})$;
  $Y_{replay} \leftarrow ([0] *$ 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); // Update $\mathcal{P}$ with samples of current task
  $\nabla\mathcal{D} \leftarrow X_{mal}$ // Update global dictionary
return $(X_{train},\ Y_{train})$

two sub-sampling variants – Ratio budgeting and Uniform budgeting. In Ratio budgeting, IFS selects the number of samples from a family in direct proportion to that family's representation in the dataset. This may be more suitable in binary classification of samples as being either benign or malicious, as it provides proportional representation of the malware families for training on the malicious class. In Uniform budgeting, IFS evenly distributes the available malware budget β_M among all 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, Isolation Forest provides two sets of samples: representative samples S_f and anomalous samples A_f. We found that an even split of representative and anomalous samples provides the best performance.

To get X_{train} for the current task c, we concatenate the samples X_c of the current task c with X_{replay}. Finally, after training, data pool $\mathcal{P}$ is updated with X_c and dictionary $\mathcal{D}$ is updated with X_{mal} for use in the next task.

5.4.3 Anomalous Weights-based Sampling (AWS)

While IF works better than other distance-based anomaly detection technique on high-dimensional data, it can struggle with correlated features [106]. The EMBER dataset has 2,381 features for both Domain-IL and Class-IL scenarios, and while the AZ datasets have 1,789 and 2,439 features for Domain-IL and Class-IL, respectively. Instead of applying dimensionality reduction, employing a neural network to generate a more compact feature representation could be a beneficial approach, especially in continual-learning contexts where it complements ongoing model development efforts [51, 99].

As such, we introduce Anomalous Weights-based Sampling (AWS), an advancement of IFS that leverages learned representations from a trained model, $\mathcal{M}$, to more effectively identify both representative and diverse samples. By inputting a sample $\mathcal{X}$ into $\mathcal{M}$, we obtain weights $\mathcal{W}$ reflecting the sample's internal state. For inputs Xf from family f, AWS extracts weights $\Theta^L X_f \in \mathbb{R}^D$ from a chosen layer $\mathcal{L}$. These weights are then analyzed with the Isolation Forest (IF) algorithm to pinpoint anomalous activations, $\mathcal{A}_w$. AWS then maps these anomalies

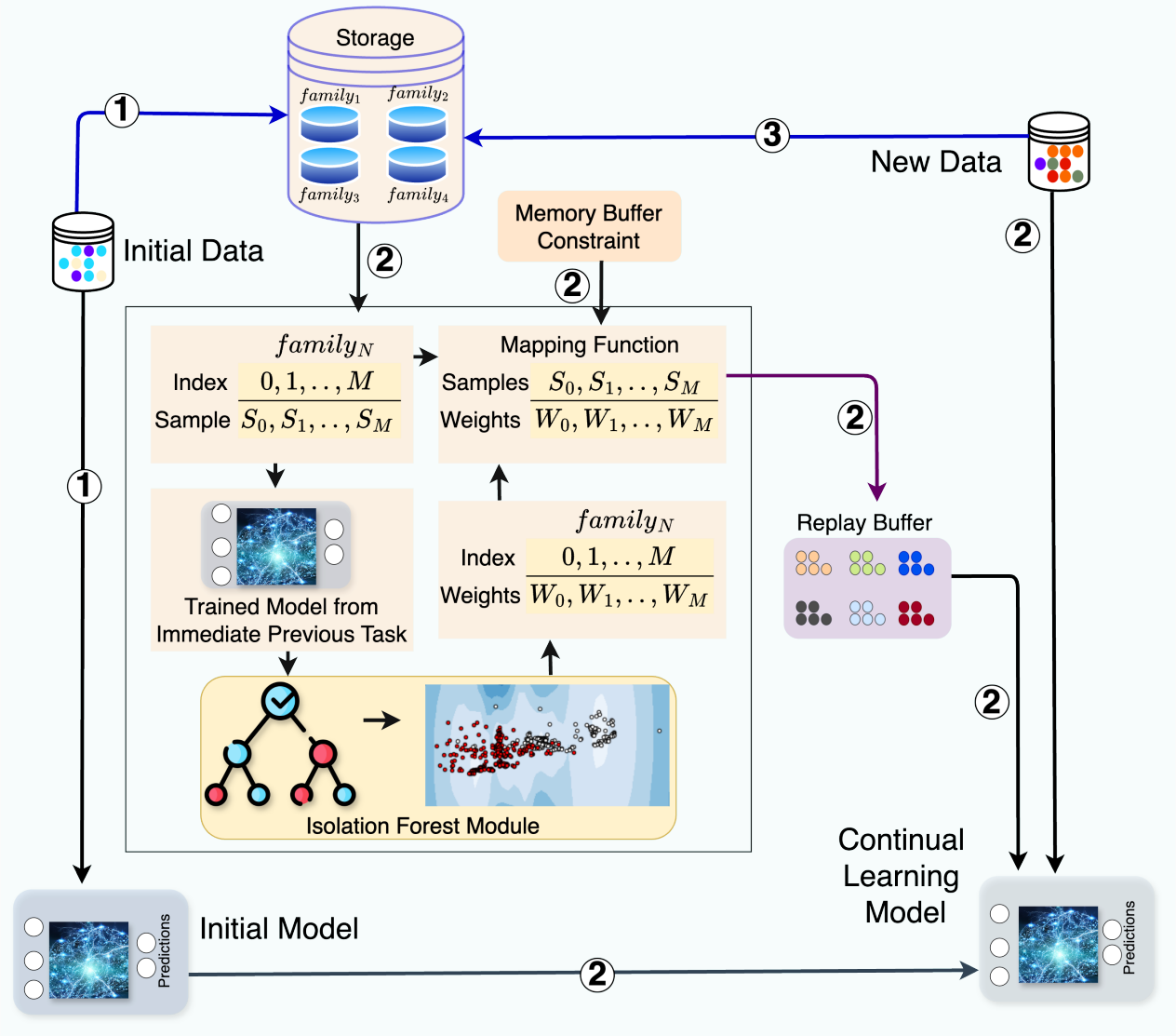


Figure 5.6: Depiction of MADAR framework with Anomalous Weights-based Sampling (AWS).

back to the original input space of Xf to select the anomalous samples A_f. Non-anomalous samples are similarly sampled to form S_f. Aside from these steps, AWS proceeds similarly to IFS.

In selecting the layer $\mathcal{L}$ for AWS, we aim to capture feature representations while avoiding the model's final classification stage. Testing various layers revealed that the final activation layers, `act4` for EMBER and `act5` for AZ, yield the best results. Thus, we utilized these layers for the remainder of our study. We also adapted the AWS algorithm by omitting batch

normalization during the forward pass. Although batch normalization generally stabilizes learning and improves model generalization by normalizing input layers, in the AWS context, it can diminish the weight distribution diversity we seek to maintain. Our tests confirmed that excluding batch normalization indeed enhances AWS's performance.

Besides enhancing replay sample selection, AWS is also more efficient than IFS. The `act4` and `act5` hidden layers have just 128 dimensions each, considerably less than the original feature dimensions. The IF algorithm thus operates more quickly in this reduced dimension space.

A high-level representation of the Anomalous Weights-based Sampling (AWS) within the MADAR framework is depicted in Figure 5.6, illustrating the three distinct operational phases. The initial phase, marked by (1) in the figure, involves the model's first training session using a static approach with the initial dataset. It is crucial to note that weight initialization is exclusive to this stage. This phase includes an in-depth analysis of the model's hyperparameters such as learning rate, batch size, number of layers, and optimizer to optimize performance. Following this initial training, the data used is stored and categorized according to malware family.

The model then transitions to the continual learning (CL) phase, as indicated by (2). In this phase, the model is introduced solely to new data and selected data from the replay buffer, under the assumption that a replay-based strategy is in place to preserve model stability. The replay buffer incorporates samples chosen by AWS, subject to a specific memory limit. Similar to IFS, AWS selects replay samples by analyzing each family individually. The model processes samples from a given family, using the trained model obtained from the previous learning episode, to extract specific layer weights. These weights are then analyzed using the isolation forest module to identify anomalous and similar patterns. Subsequently, a mapping function correlates these weights with the original samples. Based on the memory constraints, a selection of replay samples is finalized and allocated to the replay buffer.

As shown by (3), the introduction of new data continues the systematic categorization by malware family. This methodical strategy supports the model's continuous learning and adaptation to novel data, safeguarding the knowledge previously acquired.

Table 5.2: **Summary of the EMBER Domain-IL Experiments. None = 93.1±0.1** and **Joint = 96.4±0.3**.

Budget	Method							
	ER	AGEM	GR	GRS	IFS-R	IFS-U	AWS-R	AWS-U
1K	80.6±0.1	80.5±0.1		**93.6±0.3**	**93.7±0.1**	**93.6±0.2**	**93.6±0.1**	93.5±0.2
10K	73.5±0.5	73.6±0.2		94.1±1.3	**94.7±0.1**	94.0±0.2	94.4±0.3	94.1±0.2
50K	70.5±0.3	70.4±0.3		**95.3±0.2**	**95.4±0.1**	95.1±0.1	**95.3±0.2**	94.9±0.1
100K	69.9±0.1	70.0±0.1	93.1±0.2	95.3±0.7	95.3±0.6	95.3±0.1	**95.8±0.1**	95.2±0.2
200K	70.0±0.1	70.0±0.2		**95.9±0.1**	**96.0±0.1**	95.5±0.1	**96.1±0.1**	95.6±0.1
300K	70.0±0.1	70.0±0.1		95.8±0.6	**96.1±0.1**	95.7±0.1	**96.1±0.1**	95.7±0.1
400K	70.0±0.1	70.0±0.1		**96.0±0.3**	**96.1±0.1**	95.8±0.1	**96.1±0.1**	95.7±0.1

Table 5.3: **Summary of the EMBER Class-IL Experiments. None = 26.5±0.2** and **Joint = 86.5±0.4**.

Budget	Method								
	iCaRL	ER	AGEM	GR	GRS	IFS-R	IFS-U	AWS-R	AWS-U
100	53.9±0.7	27.5±0.1	27.3±0.1		51.9±0.4	**68.0±0.4**	66.4±0.4	67.9±0.3	66.5±0.3
500	58.7±0.7	27.8±0.1	27.4±0.1		70.3±0.5	73.6±0.2	**76.5±0.2**	72.7±0.5	**76.4±0.4**
1K	60.0±1.0	28.0±0.1	27.7±0.1		75.4±0.7	76.0±0.3	**79.4±0.4**	76.0±0.2	**79.7±0.3**
5K	63.9±1.2	27.9±0.1	28.5±0.1	26.8±0.2	82.0±0.2	81.5±0.2	**83.8±0.2**	81.7±0.2	**84.1±0.1**
10K	64.6±0.8	28.0±0.1	28.2±0.1		83.5±0.1	83.2±0.2	**84.8±0.1**	83.2±0.1	**85.3±0.1**
15K	65.5±1.0	28.0±0.1	28.3±0.1		84.3±0.3	83.8±0.2	**85.5±0.1**	83.9±0.1	**85.8±0.2**
20K	66.8±1.1	28.2±0.1	28.2±0.1		84.6±0.2	84.0±0.2	**85.8±0.3**	84.5±0.2	**86.2±0.2**

5.5 Evaluation

We now present the results of our MADAR framework in the Domain-IL and Class-IL scenarios for EMBER and AZ datasets. We use the following four abbreviations to denote our techniques – **IFS-R**: IFS-Ratio, **IFS-U**: IFS-Uniform, **AWS-R**: AWS-Ratio, and **AWS-U**: AWS-Uniform.

5.5.1 EMBER

Domain-IL. In our experiments, we have 12 tasks, each representing the monthly data distribution shift of malware and goodware in EMBER spanning from January to December 2018. Our results, detailed in Table 5.2, present a nuanced view of each method's performance, reported as the average accuracy over all tasks $\overline{\mathbf{AP}}$. The informal lower and upper performance bounds for this configuration can be approximated by the *None* and *Joint* methods, which get $\overline{\mathbf{AP}}$ of 93.1% and 96.4%, respectively. We compare the performances our our proposed techniques with three of the most widely used state-of-the-art replay-based CL

techniques – experience replay (ER) [119], average gradient episodic memory (AGEM) [25], and deep generative replay (GR) [130].

As shown in Table 5.2, at lower budgets (1K and 10K), GRS, IFS-R, and IFS-U exhibited competitive performance, all significantly better than prior works with $\overline{\mathbf{AP}}$ above 93.6%, indicating their effective utilization of limited resources. ER and AGEM performed far below even the *None* baseline, while GR could only match it. As the budget increased, IFS-R and AWS-R consistently demonstrated better performance, suggesting their greater effectiveness in adapting to incremental learning scenarios. For instance, at a 200K budget, both IFS-R and AWS-R achieved $\overline{\mathbf{AP}}$ of 96.0%, close to the 96.4% reached by the Joint baseline that used over 670K training samples. On the other hand, the uniform strategies (IFS-U and AWS-U) showed slightly lower yet consistent performance across all budget levels, indicating a stable but less dynamic approach to incremental learning. All three prior works remained at or below the *None* baseline, showing low effectiveness in the malware domain.

In summary, our results empirically depict the effectiveness of MADAR's diversity-aware sample selection in maximizing the efficiency and effectiveness of a malware classifier in Domain-IL. IFS-R and AWS-R are either better or on par with GRS and significantly better than prior work.

Class-IL. In this set of experiments, we have 11 tasks, where the initial task starts with 50 classes, one for each of 50 malware families, and five classes are added in each subsequent task. The performance of these methods, detailed in Table 5.3, is measured by average accuracy $\overline{\mathbf{AP}}$ with *None* and *Joint* training baselines at an $\overline{\mathbf{AP}}$ of 26.5 ± 0.2 and 86.5 ± 0.4, respectively. For a very low budget of 100 samples, MADAR methods all greatly outperform GRS, with IFS-R getting 16% higher $\overline{\mathbf{AP}}$. For more reasonable budgets, however, the uniform variants IFS-U and AWS-U offer the best performance. For example, with a 5K budget, both methods reach at least 83.8% $\overline{\mathbf{AP}}$, which is better than GRS at 82% $\overline{\mathbf{AP}}$. They also fare far better than all prior works, with ER, AGEM, and GR below 30% and iCaRL at only 63.9%. These poor results for the prior methods are in line with other findings in the malware domain [109]. For a budget of 20K, AWS-U reaches 86.2 ± 0.2, nearly as good as the Joint baseline using a maximum budget over 150 times larger.

In summary, our experiments clearly demonstrate the effectiveness of MADAR's diversity-

Table 5.4: **Summary of the AZ Domain-IL Experiments. None = 94.4±0.1** and **Joint = 97.3±0.1**

Budget	Method							
	ER	AGEM	GR	GRS	IFS-R	IFS-U	AWS-R	AWS-U
1K	40.4±0.1	45.4±0.1		95.3±0.1	**95.8±0.1**	**95.7±0.1**	**95.8±0.2**	95.6±0.1
10K	40.1±0.1	47.4±0.2		96.4±0.1	**96.6±0.1**	95.5±0.1	**96.6±0.1**	96.1±0.1
50K	41.1±0.2	49.2±0.2		**96.9±0.1**	**96.9±0.1**	95.2±0.2	**96.9±0.1**	96.6±0.1
100K	42.6±0.1	53.7±0.6	93.3±0.4	**97.1±0.1**	**97.0±0.1**	95.2±0.1	96.9±0.1	96.8±0.1
200K	44.0±0.1	54.2±0.3		**97.1±0.1**	**97.0±0.1**	95.4±0.1	**97.1±0.1**	**97.0±0.1**
300K	45.9±0.1	54.8±0.4		**97.2±0.1**	97.0±0.1	95.8±0.2	**97.1±0.1**	**97.1±0.1**
400K	48.6±1.1	56.7±0.3		**97.2±0.1**	97.0±0.1	96.3±0.2	**97.2±0.1**	**97.1±0.1**

Table 5.5: **Summary of the AZ Class-IL Experiments.** None = **26.4±0.2** and **Joint = 94.2±0.1**.

Budget	Method					
	iCaRL	GRS	IFS-R	IFS-U	AWS-R	AWS-U
100	43.6±1.2	43.8±0.7	**59.4±0.6**	57.3±0.5	**58.8±0.3**	57.5±0.7
500	54.9±1.0	62.9±0.8	67.8±0.9	**70.4±0.4**	66.2±0.7	**70.1±0.2**
1K	61.7±0.7	70.2±0.4	71.9±0.5	**76.2±0.2**	71.0±0.7	**74.7±0.2**
5K	77.2±0.4	83.0±0.3	82.9±0.2	**86.8±0.1**	81.2±0.3	**85.5±0.1**
10K	81.5±0.6	86.4±0.2	86.3±0.1	**89.8±0.1**	85.1±0.2	**88.7±0.1**
15K	83.4±0.5	88.2±0.2	88.2±0.2	**91.0±0.1**	86.9±0.2	**90.3±0.2**
20K	84.6±0.5	89.1±0.2	89.1±0.1	**91.5±0.1**	88.1±0.1	**90.7±0.1**

aware replay techniques in Class-IL. These methods significantly improve performance in malware classification by mitigating catastrophic forgetting, and they do so using fewer resources.

5.5.2 Android APK File - AZ

Domain-IL. In this set of experiments, we have nine tasks, each representing a year from 2008 to 2016. The performance of each method is shown in Table 5.4 as $\overline{\mathbf{AP}}$ and compared with two baselines: *None* at 94.4 ± 0.1 and *Joint* at 97.3 ± 0.1, representing the informal lower and upper performance bounds, respectively.

As with EMBER, we find that our MADAR techniques greatly surpass previous methods like ER, AGEM, and GR for every budget level. For lower budgets (1K-10K), IFS-R and

AWS-R slightly outperform GRS and are within 1.5% of *Joint*. For higher budgets (50K-400K), IFS-R, AWS-R, and AWS-U all perform well – in line with GRS and just slightly below *Joint*, which requires 680K training samples.

Class-IL. We have 11 tasks for the Class-IL scenario of AZ. The summary results of all the experiments are shown in Table 5.5 and is benchmarked against *None* and *Joint* with $\overline{\mathbf{AP}}$ of 26.4 ± 0.2 and 94.2 ± 0.1, respectively. In addition, we compare our proposed techniques with iCaRL [114] as it is the performant technique among the compared prior work for the EMBER dataset as shown in § 5.5.1.

For a low budget of 100, iCaRL and GRS get less than 44%, while all MADAR methods achieve over 57%. As budgets increase, all methods improve, with IFS-U offering the best results at every budget from 500 to 20K. At 20K, it reaches 91.5 ± 0.1%, which is 1.4% higher than GRS and 6.9% higher than iCaRL.

Overall, mirroring the success seen with the EMBER dataset, our proposed techniques also surpass previous work in Class-IL in the context of the AZ-Class dataset. Additionally, while GRS shows significant improvement with an increased budget, the uniform variants of MADAR are more effective at every budget level.

5.6 Discussion

Our results demonstrate that in both Domain-IL and Class-IL scenarios, our MADAR techniques yield markedly better performances compared to previous methods for both the EMBER and AZ datasets. This clearly indicates that diversity-aware replay is effective in preserving the stability of a continual learning system for malware classification.

Analyzing the results of Domain-IL scenario, we can see that at lower budgets, all MADAR methods show better performance than both prior work and GRS in both datasets. As budgets increase, the *Ratio* variants – IFS-R and AWS-R – perform well and approach the *Joint* baselines of 96.4% in EMBER and 97.3% in AZ. It is noteworthy that our techniques require a substantially smaller number of replay samples compared with *Joint* to attain a

similar average accuracy.

In the Class-IL scenario, our proposed techniques significantly outperform previous work in both the EMBER and AZ datasets. In EMBER, AWS-U nearly matches the Joint baseline at just a 5K budget, outperforming the iCaRL method with fewer resources. In the AZ dataset, both IFS-U and AWS-U perform well, with IFS-U reaching an impressive 91.5% $\overline{\mathbf{AP}}$ at a 20K budget. In both datasets, the Uniform variants – IFS-U and AWS-U – consistently surpass other methods, proving their efficiency in resource management and adaptability to new classes.

The Ratio variants worked better for Domain-IL experiments, while Uniform variants worked well in Class-IL. 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 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 AWS to scale the approach regardless of feature dimension without loss of performance.

5.7 Conclusion

In this paper, we propose MADAR, a framework for diversity-aware replay in continual learning specially designed for the challenging setting of malware classification. Our comprehensive evaluation across Domain-IL and Class-IL scenarios against Windows executable (EMBER) and Android application (AZ) datasets demonstrates that diversity-aware sampling is helpful for effective CL in malware classification. In Domain-IL scenarios, IFS-R and AWS-R, and in Class-IL scenarios, IFS-U and AWS-U, demonstrate superior adaptability and resource efficiency. 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.

6

Conclusion and Future Work

6.1 Conclusion

Malware classification poses significant challenges for continual learning (CL) techniques due to the daily influx of new malware samples and the rapid evolution of malware exploiting new vulnerabilities. Antivirus vendors encounter hundreds of thousands of unique software pieces daily as well. This enormous daily feed of both malware and benign software can lead to a potential accumulation of over a billion samples over the lifetime of a malware classifier. Relying solely on new samples for training can lead to catastrophic forgetting, where the system loses its ability to recall previously learned tasks. Although retraining with both old and new samples is effective, it is expensive and necessitates storing an enormous volume of past software and malware samples. Hence, applying CL techniques in a sequential training framework could significantly reduce both training and storage demands.

This thesis conducts a thorough investigation into developing a CL system capable of adapting to the continuous growth and evolution of malware and benign software, accounting for shifts in data distribution. We begin by identifying a research gap between CL and the malware domain, evaluating state-of-the-art CL methods across three CL scenarios: task, class, and domain incremental learning. Utilizing two extensive, real-world malware datasets – EMBER [10] and Drebin [11]—our empirical findings indicate that these CL techniques generally fail to prevent catastrophic forgetting and maintain classification accuracy for malware classification system. Furthermore, an in-depth analysis of the EMBER dataset reveals the unique properties and challenges of malware data distributions, highlighting the diversity within and across malware *families*. In response, we introduce MADAR – a diversity-aware, replay-based CL strategy specifically designed for malware classification. Additionally, we have developed two new benchmarks using Android malware from the AndroZoo repository [8] for testing in both Domain-IL (*AZ-Domain*) and Class-IL (*AZ-Class*) scenarios. Our extensive evaluations demonstrate the effectiveness of MADAR framework over existing state-of-the-art CL methods, effectively handling realistic shifts in data distribution.

6.2 Future Work

As continual learning for malware domain is an emergent field of research, we believe that extensive investigations are necessary to answer various research questions unique to malware domain before we can realize a practical and effective continual learning system for malware classification and detection. Moreover, the principles and methodologies developed in this context hold promising implications for broader cybersecurity areas. These include, but are not limited to, intrusion detection, phishing prevention, and securing emerging technologies such as IoT devices. By extending continual learning approaches to these areas, we can significantly enhance the resilience and responsiveness of cybersecurity defenses, making them more adept at handling the complexities of modern cybersecurity threats.

In this section, we outline promising future research directions that could provide significant insights and advance the field of continual learning in the malware domain.

6.2.1 Analysis of Complex Image Data vs. Malware Data

We have conducted an investigation into the feature space complexity of standard image datasets such as MNIST [73] and standard malware datasets such as EMBER [10]. This analysis is presented in Section 4.5. Currently, the MNIST dataset is widely regarded as one of the easiest datasets to solve, characterized by a simpler feature space. In contrast, the EMBER dataset, a benchmark in malware research, is considered highly complex. Thus, comparing these datasets may initially seem unfair, but it offers an opportunity to explore feature space complexity in the context of more challenging datasets such as CIFAR-100 [70] and ImageNet [124]. A comprehensive comparative study of the feature space of these complex image datasets and the EMBER dataset could yield valuable insights. These insights may serve as a foundation for the development of more advanced continual learning systems for malware classification and detection.

6.2.2 Continual Learning for Dynamic Malware Analysis

This study examines malware classification systems in the context of static malware analysis. Among various classification and detection techniques, static malware analysis is widely utilized in industry, where expert-designed features are employed to develop the system. The rationale behind using a hand-engineered feature-based system lies primarily in its computational efficiency and scalability. Additionally, leveraging existing knowledge of common malware and goodware attributes allows for the improvement of such systems even further. Notably, these systems have demonstrated impressive performance, with models like Light GBM achieving a ROC AUC of 0.996 on benchmark malware dataset EMBER [10]. Consequently, in this work, both the exploratory analysis and the proposed continual learning system are designed to operate within the hand-engineered feature space.

At its current stage, this work does not delve into studying a continual learning system in the context of dynamic malware analysis. In dynamic malware analysis, a malware sample is executed within a controlled virtual environment to observe its behavior and its impact on the system. This sophisticated analysis provides insights into the true nature of the malware and its ability to evade detection systems. While dynamic analysis is considered more effective compared to static analysis, it requires significantly more resources and time. Despite these challenges, an exploration of continual learning systems utilizing dynamic malware features could yield even greater effectiveness. The feature space in dynamic analysis is more fine-grained and unique to specific types of malware. We hypothesize that this approach could further reduce the reliance on accessing older malware samples and subsequently decrease storage and computation costs.

6.2.3 Harnessing the Advancement of Generative AI

The Generative Replay (GR) technique has emerged as a promising solution to counteract catastrophic forgetting, particularly within the computer vision domain [130, 139, 140]. However, our findings present an anomaly. Despite its theoretical potential, GR has not demonstrated the anticipated level of effectiveness in our studies on malware classification. This gap between its theoretical promise and the observed results in our research opens up a series of compelling questions. Could there be characteristics intrinsic to malware data that

diminish the effectiveness of GR? Or might external factors be influencing the outcomes?

Investigating the underlying reasons for this discrepancy is not just academically intriguing; it could fundamentally transform our approach to malware classification. Successfully harnessing the GR technique could lead to significant advancements: eliminating the necessity to store raw data would greatly reduce storage requirements, streamline the training process, and enhance data security by reducing exposure. Moreover, perfecting the use of GR in malware classification could set a precedent for its application across various domains. A promising avenue for future research is to unravel this mystery and fully exploit the potential of generative artificial intelligence (AI) techniques, ensuring our models are both efficient and robust in real-world applications.

6.2.4 Continual Large Language Model for Malware Analysis

Given the challenges associated with managing long and complex sequences in binary executables, current methods often favor feature-based systems for classifying malware. In our previous research, we explored the potential of Transformer models in an innovative end-to-end approach, evaluating various architectural designs, training techniques, and experimental setups to identify key factors for effective detection models [123]. Building on these insights, further research can be advanced and conducted by integrating more sophisticated large language models [18, 111, 137] with continual learning strategies. This area of research would be suitable to tackle the challenges of processing extensive sequence lengths within an end-to-end framework for continual malware detection.

6.2.5 Generalization of the System

Our current research is based on malware datasets specific to Windows and Android platforms. This focus narrows the scope of our findings, potentially limiting the broader applicability of our proposed systems to the diverse and evolving landscape of malware threats. Recognizing this limitation, we advocate for a more expansive approach in future studies. Incorporating a wider array of malware datasets from a variety of platforms—including, but not limited to, iOS, PDF documents, web-based threats, and vulnerabilities within office

suite applications—would not only enrich the research but also strengthen the effectiveness and relevance of the developed systems. Furthermore, extending the temporal range of these datasets to span multiple years can offer invaluable insights into the longitudinal trends and transformations within the malware domain, thereby enhancing the predictive and adaptive capacities of continual learning frameworks in cybersecurity.

6.3 Publications

6.3.1 Continual Learning & Malware Analysis

1. **Under Review**: **Rahman, Mohammad Saidur**, Scott Coull, Qi Yu, and Matthew Wright. "MADAR: Continual Learning for Malware Analysis with Diversity-Aware Replay." 2024.
2. **CoLLAs-2022**: **Rahman, Mohammad Saidur**, Scott Coull, and Matthew Wright. "On the Limitations of Continual Learning for Malware Classification." In Conference on Lifelong Learning Agents, pp. 564-582. PMLR, 2022.
3. **WoRMA-2022**: Rudd, Ethan M., **Mohammad Saidur Rahman**, and Philip Tully. "Transformers for End-to-End InfoSec Tasks: A Feasibility Study." In Proceedings of the 1st Workshop on Robust Malware Analysis, pp. 21-31. 2022.

6.3.2 Traffic Analysis & Website Fingerprinting

1. **IEEE S&P-2023**: Mathews, Nate, James K. Holland, Se Eun Oh, **Mohammad Saidur Rahman**, Nicholas Hopper, and Matthew Wright. "SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses." In 2023 IEEE Symposium on Security and Privacy (SP), pp. 344-361. IEEE Computer Society, 2022.
2. **IEEE S&P-2022**: Oh, Se Eun, Taiji Yang, Nate Mathews, James K. Holland, **Mohammad Saidur Rahman**, Nicholas Hopper, and Matthew Wright. "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification." In

2022 IEEE Symposium on Security and Privacy (SP), pp. 1915-1932. IEEE Computer Society, 2022.

3. **PoPETS-2021**: Oh, Se Eun, Nate Mathews, **Mohammad Saidur Rahman**, Matthew Wright, and Nicholas Hopper. "GANDaLF: GAN for data-limited fingerprinting." Proceedings on Privacy Enhancing Technologies 2021, no. 2 (2021).

4. **IEEE TIFS-2020**: **Rahman, Mohammad Saidur**, Mohsen Imani, Nate Mathews, and Matthew Wright. "Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces." IEEE Transactions on Information Forensics and Security 16 (2020): 1594-1609.

5. **PoPETS-2020**: **Rahman, Mohammad Saidur**, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. "Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks." Proceedings on Privacy Enhancing Technologies 3 (2020): 5-24.

6. **ACM CCS-2019**: Sirinam, Payap, Nate Mathews, **Mohammad Saidur Rahman**, and Matthew Wright. "Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning." In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1131-1148. 2019.

7. **ACM CCS-2019**: **Rahman, Mohammad Saidur**, Nate Matthews, and Matthew Wright. "Poster: video fingerprinting in tor." In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2629-2631. 2019.

8. **ACM CCS-2019**: Mathews, Nate, **Mohammad Saidur Rahman**, and Matthew Wright. "Poster: Evaluating Security Metrics for Website Fingerprinting." In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2625-2627. 2019.

9. **ACM CCS-2018**: Imani, Mohsen, **Mohammad Saidur Rahman**, and Matthew Wright. "Adversarial traces for website fingerprinting defense." In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 2225-2227. 2018.

6.3.3 Quantum Secure Network

1. **Under Review**: **Rahman, Mohammad Saidur**, Stephen DiAdamo, Miralem Mehic, Charles Fleming. "Quantum Secure Anonymous Communication Network." 2024.

References

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[2] Wickliffe C Abraham and Mark F Bear. Metaplasticity: the plasticity of synaptic plasticity. *Trends in Neurosciences*, 19(4):126–130, 1996.

[3] Wickliffe C Abraham and Anthony Robins. Memory retention and weight plasticity in ann simulations. *Trends in Neurosciences*, 2(28):73–78, 2005.

[4] Laha Ale, Longzhuang Li, Dulal Kar, Ning Zhang, and Aayasha Palikhe. Few-shot learning to classify android malwares. In *2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP)*, pages 1001–1007. IEEE, 2020.

[5] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[6] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11254–11263, 2019.

[7] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

[8] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

[9] Suresh Kumar Amalapuram, Thushara Tippi Reddy, Sumohana S Channappayya, and Bheemarjuna Reddy Tamma. On handling class imbalance in continual learning based network intrusion detection systems. In *The First International Conference on AI-ML-Systems*, pages 1–7, 2021.

[10] Hyrum S Anderson and Phil Roth. EMBER: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[11] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Network and Distributed System Security Symposium (NDSS)*, volume 14, pages 23–26, 2014.

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

[13] David Badre and Anthony D Wagner. Left ventrolateral prefrontal cortex and the cognitive control of memory. *Neuropsychologia*, 45(13):2883–2901, 2007.

[14] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.

[15] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *USENIX Security Symposium*, pages 807–822, 2016.

[16] Daniel Bendor and Matthew A Wilson. Biasing the content of hippocampal replay during sleep. *Nature Neuroscience*, 15(10):1439–1444, 2012.

[17] Konstantin Berlin, David Slater, and Joshua Saxe. Malicious behavior detection using windows audit logs. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 35–44, 2015.

[18] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[19] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Page-Caccia, Issam Hadj Laradji, Irina Rish, Alexandre Lacoste, David Vázquez, et al. Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. *Advances in Neural Information Processing Systems*, 33:16532–16545, 2020.

[20] Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8281–8290, 2021.

[21] Margaret F Carr, Shantanu P Jadhav, and Loren M Frank. Hippocampal replay in the awake state: a potential substrate for memory consolidation and retrieval. *Nature Neuroscience*, 14(2):147, 2011.

[22] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018.

[23] Yuhan Chai, Lei Du, Jing Qiu, Lihua Yin, and Zhihong Tian. Dynamic prototype network based on sample adaptation for few-shot malware detection. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[24] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[25] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *International Conference on Machine Learning (ICML)*, 2019.

[26] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. In *International Conference on Machine Learning (ICML)*, 2019.

[27] Yizheng Chen, Zhoujie Ding, and David Wagner. Continuous learning for android malware detection. In *USENIX Security Symposium*, 2023.

[28] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

[29] Min Chee Choy, Dipti Srinivasan, and Ruey Long Cheu. Neural networks for continuous online learning and control. *IEEE Transactions on Neural Networks*, 17(6):1511–1531, 2006.

[30] Mihai Christodorescu, Somesh Jha, Sanjit A Seshia, Dawn Song, and Randal E Bryant. Semantics-Aware Malware Detection. In *IEEE Symposium on Security and Privacy (S&P)*, pages 32–46. IEEE, 2005.

[31] Aristotelis Chrysakis and Marie-Francine Moens. Online continual learning from imbalanced data. In *International Conference on Machine Learning*, pages 1952–1961. PMLR, 2020.

[32] Martin A Conway. Episodic memories. *Neuropsychologia*, 47(11):2305–2313, 2009.

[33] Scott E Coull and Christopher Gardner. Activation analysis of a byte-based deep neural network for malware classification. In *IEEE Security and Privacy Workshops (SPW)*, pages 21–27. IEEE, 2019.

[34] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017.

[35] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *International Conference on Acoustics, Speech, & Signal Processing (ICASSP)*, pages 3422–3426. IEEE, 2013.

[36] Vacha Dave, Saikat Guha, and Yin Zhang. Viceroi: Catching click-spam in search ad networks. In *ACM Conference on Computer and Communications Security (CCS)*, pages 765–776, 2013.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[38] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5138–5146, 2019.

[39] Mark Dredze and Koby Crammer. Online methods for multi-domain learning and adaptation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 689–697, 2008.

[40] Howard Eichenbaum. Hippocampus: cognitive processes and neural representations that underlie declarative memory. *Neuron*, 44(1):109–120, 2004.

[41] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. In *International conference on machine learning (ICML)*, pages 1407–1416. PMLR, 2018.

[42] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

[43] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[44] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[45] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*, pages 267–285. Springer, 1982.

[46] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R Kulkarni, and Prateek Mittal. SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In *USENIX Security Symposium*, pages 639–656, 2018.

[47] David Escudero García, Noemí DeCastro-García, and Angel Luis Muñoz Castañeda. An effectiveness analysis of transfer learning for the concept drift problem in malware detection. *Expert Systems with Applications*, 212:118724, 2023.

[48] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2014.

[49] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security (ESORICS)*, pages 62–79. Springer, 2017.

[50] Yiduo Guo, Bing Liu, and Dongyan Zhao. Online continual learning through mutual information maximization. In *International Conference on Machine Learning*, pages 8109–8126. PMLR, 2022.

[51] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 466–483. Springer, 2020.

[52] Tyler L Hayes and Christopher Kanan. Online continual learning for embedded devices. *arXiv preprint arXiv:2203.10681*, 2022.

[53] Jiangpeng He and Fengqing Zhu. Online continual learning via candidates voting. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3154–3163, 2022.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[55] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. In *IEEE Signal Processing Magazine*, 2012.

[56] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[57] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Lifelong learning via progressive distillation and retrospection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 437–452, 2018.

[58] Shou-Ching Hsiao, Da-Yu Kao, Zi-Yuan Liu, and Raylin Tso. Malware image classification using one-shot learning with siamese networks. *Procedia Computer Science*, 159:1863–1871, 2019.

[59] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

[60] Ngoc Anh Huynh, Wee Keong Ng, and Kanishka Ariyapala. A new adaptive learning algorithm and its application to online malware detection. In *International Conference on Discovery Science (ICDS)*, pages 18–32. Springer, 2017.

[61] Mohsen Imani, Mohammad Saidur Rahman, and Matthew Wright. Adversarial traces for website fingerprinting defense. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[62] Daoyun Ji and Matthew A Wilson. Coordinated memory replay in the visual cortex and hippocampus during sleep. *Nature Neuroscience*, 10(1):100–107, 2007.

[63] Jeff Johns. MalwareGuard: FireEye's Machine Learning Model to Detect and Prevent Malware. https://www.fireeye.com/blog/products-and-services/2018/07/malwareguard-fireeye-machine-learning-model-to-detect-and-prevent-malware.html, 2018.

[64] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *USENIX Security Symposium*, pages 625–642, 2017.

[65] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D Joseph, and J Doug Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 45–56, 2015.

[66] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[67] William DS Killgore, Ellen T Kahn-Greene, Erica L Lipizzi, Rachel A Newman, Gary H Kamimori, and Thomas J Balkin. Sleep deprivation reduces perceived emotional intelligence and constructive thinking skills. *Sleep Medicine*, 9(5):517–526, 2008.

[68] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[69] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[70] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[71] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[72] Pavel Laskov and Nedim Šrndić. Static detection of malicious javascript-bearing pdf documents. In *Annual Computer Security Applications Conference (ACSAC)*, pages 373–382, 2011.

[73] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[74] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, pages 253–256. IEEE, 2010.

[75] S. Lee and J. Kim. WarningBird: Detecting Suspicious URLs in Twitter Stream. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[76] Timothée Lesort, Massimo Caccia, and Irina Rish. Understanding continual learning settings with data distribution drift analysis. In *International Conference of Machine*

Learning 2021 (ICML) Workshop on Theory and Foundation of Continual Learning, 2021.

[77] Huichen Li, Xiaojun Xu, Chang Liu, Teng Ren, Kun Wu, Xuezhi Cao, Weinan Zhang, Yong Yu, and Dawn Song. A machine learning approach to prevent malicious calls over telephony networks. In *IEEE Symposium on Security and Privacy (S&P)*, pages 53–69. IEEE, 2018.

[78] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(12):2935–2947, 2017.

[79] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[80] Bing Liu. Learning on the job: Online lifelong and continual learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13544–13549, 2020.

[81] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[82] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems (NeurIPS)*, 30, 2017.

[83] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, pages 23190–23211. PMLR, 2023.

[84] Davide Maiorca, Giorgio Giacinto, and Igino Corona. A pattern recognition system for malicious pdf files detection. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition (MLDM)*, pages 510–524. Springer, 2012.

[85] Stephen J Martin, Paul D Grimwood, and Richard GM Morris. Synaptic plasticity and memory: an evaluation of the hypothesis. *Annual review of neuroscience*, 23(1):649–711, 2000.

[86] Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115(44):E10467–E10475, 2018.

[87] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989.

[88] James L McGaugh. Memory–a century of consolidation. *Science*, 287(5451):248–251, 2000.

[89] Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pages 33–42. IEEE, 2017.

[90] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11):205, 2017.

[91] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:54654, 2013.

[92] Earl K Miller and Jonathan D Cohen. An integrative theory of prefrontal cortex function. *Annual review of neuroscience*, 24(1):167–202, 2001.

[93] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.

[94] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[95] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):157–175, 2017.

[96] Annamalai Narayanan, Liu Yang, Lihui Chen, and Liu Jinliang. Adaptive and scalable android malware detection through online learning. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2484–2491. IEEE, 2016.

[97] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[98] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–34, 2019.

[99] Oleksiy Ostapenko, Timothee Lesort, Pau Rodríguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin. Foundational models for continual learning: An empirical study of latent replay. *arXiv preprint arXiv:2205.00329*, 2022.

[100] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11321–11329, 2019.

[101] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2):199–210, 2010.

[102] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[103] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[104] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research (JMLR)*, 12:2825–2830, 2011.

[105] Lorien Y Pratt. Discriminability-based transfer between neural networks. *Advances in neural information processing systems*, 5, 1992.

[106] Luca Puggini and Seán McLoone. An enhanced variable selection and isolation forest based methodology for anomaly detection with oes data. *Engineering Applications of Artificial Intelligence*, 67:126–135, 2018.

[107] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[108] Yu-Lin Qin, Bruce L McNaughton, William E Skaggs, and Carol A Barnes. Memory reprocessing in corticocortical and hippocampocortical neuronal ensembles. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 352(1360):1525–1533, 1997.

[109] Mohammad Saidur Rahman, Scott E. Coull, and Matthew Wright. On the Limitations of Continual Learning for Malware Classification. In *First Conference on Lifelong Learning Agents (CoLLAs)*, 2022.

[110] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *IEEE Transactions on Information Forensics and Security*, 2020.

[111] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[112] Steve Ramirez, Xu Liu, Pei-Ann Lin, Junghyup Suh, Michele Pignatelli, Roger L Redondo, Tomás J Ryan, and Susumu Tonegawa. Creating a false memory in the hippocampus. *Science*, 341(6144):387–391, 2013.

[113] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285, 1990.

[114] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2001–2010, 2017.

[115] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations (ICLR)*, 2019.

[116] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3738–3748, 2018.

[117] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[118] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[119] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 350–360, 2019.

[120] Candong Rong, Gaopeng Gou, Chengshang Hou, Zhen Li, Gang Xiong, and Li Guo. UMVD-FSL: Unseen Malware Variants Detection Using Few-Shot Learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[121] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[122] Gideon Rothschild, Elad Eban, and Loren M Frank. A cortical–hippocampal–cortical loop of information processing during memory consolidation. *Nature Neuroscience*, 20(2):251–259, 2017.

[123] Ethan M Rudd, Mohammad Saidur Rahman, and Philip Tully. Transformers for end-to-end infosec tasks: A feasibility study. In *Proceedings of the 1st Workshop on Robust Malware Analysis*, pages 21–31, 2022.

[124] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[125] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[126] Mattia Sangermano, Antonio Carta, Andrea Cossu, and Davide Bacciu. Sample condensation in online continual learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 01–08. IEEE, 2022.

[127] Dennis Schwarz. zeusmuseum. https://zeusmuseum.com/, 2022.

[128] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning (ICML)*, pages 4528–4537, 2018.

[129] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. 2018.

[130] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in Neural Information Processing Systems (NeurIPS)*, 30:2990–2999, 2017.

[131] Stelios M Smirnakis, Alyssa A Brewer, Michael C Schmid, Andreas S Tolias, Almut Schüz, Mark Augath, Werner Inhoffen, Brian A Wandell, and Nikos K Logothetis. Lack of long-term cortical reorganization after macaque retinal lesions. *Nature*, 435(7040):300–307, 2005.

[132] Nedim Šrndic and Pavel Laskov. Detection of malicious pdf files based on hierarchical document structure. In *Network and Distributed System Security Symposium (NDSS)*, pages 1–16. Citeseer, 2013.

[133] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *ACM Conference on Computer and Communications Security (CCS)*, pages 133–144, 2013.

[134] Timothy Tadros, Giri P Krishnan, Ramyaa Ramyaa, and Maxim Bazhenov. Sleep-like unsupervised replay reduces catastrophic forgetting in artificial neural networks. *Nature Communications*, 13(1):1–12, 2022.

[135] Gil Tahan, Lior Rokach, and Yuval Shahar. Mal-id: Automatic malware detection using common segment analysis and meta-features. *Journal of Machine Learning Research (JMLR)*, 13(1):949–979, 2012.

[136] Zhijie Tang, Peng Wang, and Junfeng Wang. Convprotonet: Deep prototype induction towards better class representation for few-shot malware classification. *Applied Sciences*, 10(8):2847, 2020.

[137] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[138] Endel Tulving et al. Episodic memory: From mind to brain. *Annual Review of Psychology*, 53(1):1–25, 2002.

[139] Gido M van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11(1):1–14, 2020.

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

[141] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[142] Gido M van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, 2022.

[143] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9(11), 2008.

[144] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[145] VirusTotal. VirusTotal - Stats. https://www.virustotal.com/gui/stats.

[146] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[147] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *International Conference on Machine Learning (ICML)*, pages 1113–1120, 2009.

[148] Ke Xu, Yingjiu Li, Robert Deng, Kai Chen, and Jiayun Xu. Droidevolver: Self-evolving android malware detection system. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 47–62. IEEE, 2019.

[149] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. CADE: Detecting and explaining concept drift samples for security applications. In *USENIX Security Symposium*, pages 2327–2344, 2021.

[150] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[151] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *Journal of Machine learning research (JMLR)*, 70:3987, 2017.

[152] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

[153] Chenyang Zhao, Timothy Hospedales, Freek Stulp, and Olivier Sigaud. Tensor based knowledge transfer across skill categories for robot control. In *International Joint Conference in Artificial Intelligence (IJCAI)*, pages 1–7, 2017.

[154] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *USENIX Security Symposium*, 2020.

Appendices

A

Additional Replay Buffer Techniques

Before outlining our proposed diversity-aware replay-based continual learning system for malware classification in Chapter 5, we conducted an extensive preliminary study. In Appendix A, we present a subset of this preliminary investigation, focusing on aspects that can help understand and evaluate various alternative replay buffer proposals. Note that all the analysis and investigations in this appendix are conducted using the EMBER [10] dataset in the Domain-IL setting.

A.1 Comparison of the Required Number of Replay Samples

Table A.1: Comparison of the number of replay samples required in global random sample selection and family based random sample selection.

Replay Config.	Task Months											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Joint	-	55722	93094	139967	181287	224847	271125	312007	368499	446271	536271	**626271**
GRS-20%	-	11144	20888	28362	37736	46000	54712	63967	72143	83441	98995	116995
GRS-50%	-	27861	52222	70908	94344	115004	136784	159923	180364	208610	247496	**292496**
FRS-50/f	-	14744	23580	29276	32998	38532	43690	46682	50716	56230	56870	58204
FRS-100/f	-	19694	31460	39402	45102	53238	61452	66842	73136	81894	82980	84936
FRS-200/f	-	25548	41418	51462	59374	71902	84188	92592	101614	115216	117122	**120196**
FRS-500/f	-	33252	58044	72202	83922	102174	123014	137072	150758	173768	178916	**184410**
FRS-1K/f	-	39832	70862	89110	106426	129036	156724	179624	198554	230716	241464	**249662**
FRS-5K/f	-	55722	99613	131888	166044	196648	234104	274652	307200	365770	416912	447866
FRS-10K/f	-	55722	104445	138825	185409	225772	268458	313680	353367	409127	481803	547802

One crucial question that arises in sample selection is: "*How many samples should be selected for each family?*" The answer to this question may vary depending on the capacity or memory budget (i.e., replay buffer) of specific antivirus vendors. However, it is important to understand the number of samples required in different configurations of sample selection to facilitate better comparison of the performance of each explored configuration. Our objective is to identify an effective sample selection configuration that can achieve higher or comparable performance with an equivalent or reduced number of replay samples compared to partial joint replay (PJR).

To this end, we conducted an exploratory analysis to compare the number of replay samples required in different configurations of family-based replay sample selection with partial joint replay (PJR) based sample selection. The number of replay samples for each task month is shown in Table A.1. In this context, PJR follows the experimental setting outlined in Section 4.4, where $X-\%$ of stored samples are selected for replay alongside the new data of the current learning episode. In our continual learning framework, the Joint configuration accumulates samples until the previous task, T_{n-1}, and combines them with the new data of the current task, T_n. The resulting dataset is then used to train the model for the

current task. We track the total number of accumulated samples for different tasks in this configuration. However, no replay samples are available for the January task month, as it marks the beginning of the learning task and there are no previous tasks to accumulate samples from.

The notation $GRS - X\%$ is used to denote different configurations of the PJR technique in continual learning. This technique involves randomly selecting samples from the global repository of the previous data pool. Please note that the random sample selection technique discussed in Section 5.3 differs from the one used here. In this context, we select $X\%$ of the stored data, whereas in Section 5.3, a fixed number of replay samples are chosen.

In our comparison, we examine two configurations: one with 20% of the replay samples and the other with 50%. Previous research has indicated that using less than 20% of the replay samples leads to suboptimal performance, while utilizing more than 50% of the stored data for replay is not desirable. Although any value for X can be chosen, higher values are expected to yield better performance. The primary objective is to achieve maximum performance using as few replay samples as possible.

It is worth noting that the performance achievable with the Joint configuration will be the highest, as this configuration utilizes all available samples from the previous task and trains the model using a static training mechanism. Therefore, if we can achieve performance equivalent to the Joint configuration with fewer replay samples, such as those provided by $GRS - 50\%$ or $GRS - 20\%$, or even with fewer samples, we can consider the configuration to be effective.

The notation $FRS - N/f$, where N ranges from 50 to 10K, represents various configurations of replay sample selection based on family. Here, N signifies the number of samples to select from each family. In this configuration, FRS denotes random selection from the data pool of a specific family. The algorithm for this configuration is detailed in Section A.2.1.

To ensure that the number of replay samples falls within our desired limits, we can refer to the Joint configuration, which provides an upper limit for the number of replay samples for the last task. Our objective is to maintain the number of replay samples at or below $GRS - 50\%$ but above $GRS - 20\%$. Among the family-based configurations, only $FRS - 200/f$, $FRS - 500/f$, and $FRS - 1K/f$ meet this criterion.

The green cell in the table denotes the number of replay samples for the final task (December), representing the highest number of replay samples for the Joint, $GRS - 50\%$, $FRS - 200/f$, $FRS - 500/f$, and $FRS - 1K/f$ configurations, respectively. This allows us to make an informed decision regarding the number of replay samples for the replay buffer while ensuring high performance. We plan to investigate each of these three configurations for their ability to reduce catastrophic forgetting.

A.2 Replay Buffer Techniques

A.2.1 Family based Reservoir Sampling (FRS)

Algorithm 3: Family based Reservoir Sampling (FRS).

Input : $\mathcal{T}_c$ – Current task
X_{T_c}, Y_{T_c} – Samples and associated Labels of current task
$\mathcal{G}_{fd}$ – Global Family Data Pool
β – Threshold number of samples per family

Output: X_{replay}, Y_{replay} – Replay Samples and associated Labels

if $T_c == 0$ **then**
  $\nabla\mathcal{G}_{fd} \leftarrow X_{T_c}, Y_{T_c}$ // Create global family data pool with samples of the current task
else
  $X_{goodware}, X_{malware} \leftarrow \mathcal{G}_{fd}$ // Separate out goodware and malware samples from global family data pool
  $R_{malware} = []$ // Initialize empty list for malware replay samples
  for *family*, X_{family} *in* $X_{malware}.items()$ **do**
    if *len*$(X_{family}) <= \beta$ **then**
      $R_{malware}$.append(X_{family})
    else
      $NX_{family} =$ random.sample(X_{family}, β)
      $R_{malware}$.append(NX_{family})
  $R_{goodware} =$ random.sample$(X_{goodware},$ len$(R_{malware}))$ // Randomly select goodware samples equal to the number of malware samples
  $X_{replay} =$ Concatenate$((R_{goodware}, R_{malware}))$
  $Y_{replay} =$ Concatenate$(([0] *$ len$(R_{goodware}), [1] *$ len$(R_{malware})))$
  $\nabla\mathcal{G}_{fd} \leftarrow X_{T_c}, Y_{T_c}$ // Update global family data pool with samples of the current task

Family based reservoir sampling (FRS) treats goodware and malware samples independently. Instead of a global data pool $\mathcal{G}_{dp}$, there is a global family data pool $\mathcal{G}_{fd}$ for this configuration which stores malware samples based on their associated families. In particular, $\mathcal{G}_{fd}$ is considered to be a global family data pool `dictionary` in which the family labels are the `keys` and samples of the family are `values`. A detailed algorithmic representation of FRS is depicted in Algorithm 3. β for FRS is treated as the threshold of the number of samples per family. For the initial task, the FRS creates the $\mathcal{G}_{fd}$. For the goodware samples, an additional key `goodware` is added to the $\mathcal{G}_{fd}$ to avoid using a separate data pool for the goodware.

For the task other than the initial task, we separate out goodware $X_{goodware}$ and malware $X_{malware}$ data pool. Malware replay samples $R_{malware}$ are selected at first from $X_{malware}$ pool. The families which contain samples less than β, all of the samples are included the $R_{malware}$. We only need to select β samples from the families which have more than β samples. We utilize `reservoir sampling` to choose samples from a particular family. After populating $R_{malware}$ from all the families, we use `reservoir sampling` to select goodware replay samples $R_{goodware}$ equal to the number of $R_{malware}$ samples from the $X_{goodware}$ pool. After the replay samples selection for both goodware and malware, we mix $R_{goodware}$ and $R_{malware}$ in the form of `concatenation`. Finally, FRS updates the $\mathcal{G}_{fd}$ with (X_{T_c}, Y_{T_c}) to be used in the next task T_{c+1}.

A.2.2 Family based Recency Sampling (FReS)

In this configuration of sample selection, we still select family based samples, however, instead of reservoir sampling, we choose samples based on the recentness of the malware samples in the $\mathcal{G}_{fd}$. The only change in the Algorithm 3 is in the `line 10` where recency based sampling chooses either the first or the last β samples from X_{family}. This set of investigation will reveal whether the latest β samples or the oldest β samples contribute most to reduce catastrophic forgetting.

Algorithm 4: HDBSCAN based Sampling (HS).

Input : $\mathcal{T}_c$ – Current task
X_{T_c}, Y_{T_c} – Samples and associated Labels of current task
$\mathcal{G}_{fd}$ – Global Family Data Pool
β – Threshold number of samples per family

Output: X_{train}, Y_{train} – Training Samples and associated Labels

if $T_c == 0$ **then**
    $\nabla\mathcal{G}_{fd} \leftarrow X_{T_c}, Y_{T_c}$ `// Create global family data pool with samples of the current task`
else
    $\nabla\mathcal{G}_{fd} \leftarrow X_{T_c}, Y_{T_c}$ `// Update global family data pool with samples of the current task`
    $X_{goodware}, X_{malware} \leftarrow \mathcal{G}_{fd}$ `// Separate out goodware and malware samples from global family data pool`
    $R_{malware} = []$ `// Initialize empty list for malware replay samples`
    for *family*, X_{family} *in* $X_{malware}.items()$ **do**
        if *len*$(X_{family}) <= \beta$ **then**
            $R_{malware}$.append(X_{family})
        else
            $NX_{family} =$ HDBSCAN(X_{family}, β)
            $R_{malware}$.append(NX_{family})
    $R_{goodware} =$ random.sample$(X_{goodware},$ len$(R_{malware}))$ `// Randomly select goodware samples equal to the number of malware samples`
    $X_{train} =$ Concatenate$((R_{goodware}, R_{malware}))$
    $Y_{train} =$ Concatenate$(([0] *$ len$(R_{goodware}), [1] *$ len$(R_{malware})))$

A.2.3 HDBSCAN based Sampling (HS)

One of the reasons we endeavor to investigate to select samples based on malware family is due to the variety of malware families. Selecting samples based on family itself has its merits to offer diversity in the replay samples. We hypothesize that we can do even better than family based reservoir sampling provided that we can sort out within family diversity. In particular, if we use a clustering algorithm to cluster the samples of a particular malware family in a low dimensional space, we can probably visually distinguish some patterns within that family. With the reference of those clusters, we can choose to select samples which represent each of those small clusters in addition to the outliers within that family. Intuitively, this process might better represent diversity in the selected replay samples.

Among the various clustering algorithms, hierarchical density-based spatial clustering of applications with noise (HDBSCAN) [89] is the most robust clustering algorithm that can handle noisy data points and can also offer to detect outliers. HDBSCAN is built upon DBSCAN. HDBSCAN overcomes the limitations of DBSCAN by offering to identify clusters of varying density and faster computation. HDBSCAN algorithm starts with finding out the *core distances* between data points. Points in denser regions would have smaller core distances while points in sparser regions would have larger core distances. HDBSCAN will help us to identify the outliers with the family based data distribution. Among the various parameters of HDBSCAN, minimum cluster size `min-cluster-size` impacts the most to the quality of the resulting clusters. Minimum cluster size parameter puts a threshold on the least number of samples to have to consider a cluster. Reducing the `min-cluster-size` will produce more clusters and increasing the `min-cluster-size` will produce less number of clusters and produce more outliers. There is another important parameter called minimum samples `min-samples` which is by default set equal to the `min-cluster-size` in the implementation [90]. While tuning `min-cluster-size`, it is also crucial to tune `min-samples`, otherwise HDBSCAN might produce either significant high or low number of outliers.

A detailed algorithmic representation of the HDBSCAN based sampling is provided in Algorithm 4. In this configuration of sample selection, we impose another specification on top of the constraint $\mathcal{C}$ of the number of replay samples per family β. As HDBSCAN will be able to select both outliers (discriminative samples) and non-outliers (representative samples), we hypothesize that we might not need to include all of the samples of the current task T_N. Instead we can effectively down-sample further and select both discriminative and representative samples after combining both the stored samples till task T_{N-1} and samples from the current task T_N. In the algorithm, we can see that we do not down-sample for the first task as the initial model needs to be good enough with enough number of samples and it is usually not recommended to down-sample in the initial task. From the second task, we update the global family data pool $\mathcal{G}_{fd}$ with data from current task (X_{T_c}, Y_{T_c}) and then start sub-sampling and down-sampling processing with HDBSCAN (see `line` $5-13$). For this configuration, the selected goodware and malware samples $(R_{goodware}, R_{malware})$ form the training samples (X_{train}, Y_{train}) of the current task T_N.

A.3 Evaluation

A.3.1 FRS

Based on our exploratory analysis, we choose and study three configurations of family based reservoir sampling (FRS) such that $FRS : \{200, 500, 1000\}/f$ and compare the performance with the baselines explained above. The performance configuration is used for further comparison in next set of investigations. For each task other than the initial task, in each of these FRS configurations, the replay samples are mixed with the task specific samples and randomly shuffled. Standard scaler is used to standardize the samples before starting the training. Training procedure is shown in Algorithm 5. Training is performed in batch-by-batch with stochastic gradient descent (SGD) optimization.

We can see the performances of each of the studied configurations in Figure A.1. The figure shows the average accuracies of each task as the task grows. As expected all the three FRS configurations outperform GRS-20%. Compared with GRS-50%, FRS-1000/f performs better with a slight margin, FRS-500/f yields relatively similar performance, and FRS-200/f performance slightly low. The average accuracies over all tasks ($\overline{AT}$) and minimum accuracy among all task ($\widehat{AT}$) can provide even better comparison among all these configurations. For all the three configurations of FRS: $\{FRS - 200/f, FRS - 500/f, FRS - 1000/f\}$, $\overline{AT} = \{94.9\%, 95.1\%, 95.3\%\}$ and $\widehat{AT} = \{93.5, 94.0, 93.9\}$, respectively. Compared to these performances, GRS-20% and GRS-50% yield $\overline{AT} = 94.4, 95.0$ and $\widehat{AT} = 93.8, 94.2$, respectively. Carefully analyzing these results, we can see that $RS - 500/f$ is actually performing very closely to $GRS - 50\%$. Even though $FRS - 1000/f$ is performing slightly better compared to $GRS - 500\%$, the number of replay samples required for $FRS - 1000/f$ is higher than $FRS - 500/f$. Our goal is to get similar or better performance with as few replay samples as possible. In this case, $GRS - 500\%$ seems to be a viable candidate to explore to improve this even further.

Finally, the joint configuration yields $\overline{AT} = 95.9$ and $\widehat{AT} = 95.2$ indicating that we still need to improve the performance significantly. Even though, the gap among the performances might seem to look small, this is significant for a security applications. In addition, the dataset just spans over one year. Having a dataset that would span over multiple year could

magnify the impact of these replay configurations in a more realistic way.

Algorithm 5: Training Procedure.

Input : $\mathcal{T}$ – Set of Tasks
$\mathcal{M}$ – Model
β – Threshold number of samples per family

`init` $\mathcal{S}$ – Initialize standard scaler
`init` $\mathcal{G}_{dp}$ – Initialize global family data pool
`init` ξ – Initialize SGD optimizer

for T_c *in* T **do**
  $X_{T_c}, Y_{T_c} \leftarrow T_c$
  if $T_c == 0$ **then**
    $\nabla\mathcal{G}_{dp} \leftarrow X_{T_c}, Y_{T_c}$ `// Create global data pool with the samples from current task`
    $\mathcal{S}_{scaler} = \mathcal{S}$.`partialFit`$(X_{T_c})$
    $X_{T_c} = \mathcal{S}_{scaler}$.`transform`$(X_{T_c})$
  else
    $X_{replay}, Y_{replay} \leftarrow$ `SampleSelection`$(\mathcal{G}_{dp}, \beta)$ `// Get the replay samples and associated labels`
    $X_{T_c}, Y_{T_c} \leftarrow$ `Concatenate`$((X_{T_c}, X_{replay}))$, `Concatenate`$((Y_{T_c}, Y_{replay}))$
    $\nabla\mathcal{G}_{dp} \leftarrow X_{T_c}, Y_{T_c}$ `// Update global data pool with the samples from current task`
  $Training_{Loss} =$ `train`$(\mathcal{M}, X_{T_c}, Y_{T_c}, \xi)$

A.3.2 FReS

For this set of investigations, we attempt to understand whether the latest samples or the oldest samples in the queue of family based samples contribute most to yield a better performance compared to FRS. A simplification of the oldest sample selection and latest sample selection can be expressed as first in first out (FIFO) and last in first out (LIFO). Intuitively, a family data queue will add new family samples at the end of the queue every time we encounter new samples. As such the sample update in the memory buffer will only happen in LIFO method as the last set of N samples where $N = \{500, 1000\}$ in the queue will change as the task grows. On the other hand, the first set of N samples where $N = \{500, 1000\}$ in the queue will remain the same from the first task.

We perform our investigation with only two configurations of number of sample selection $\{500, 1000\}$ which we choose based on the results of our investigations of FRS. $FRS-500/f$

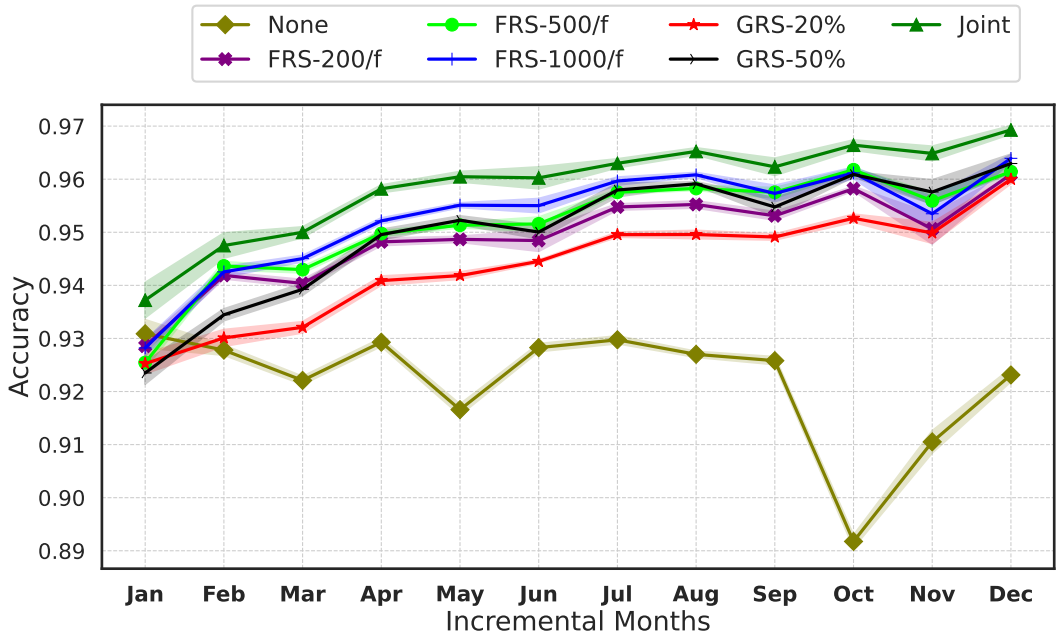


Figure A.1: FRS: Accuracy over time with different configurations of FRS in Domain-IL.

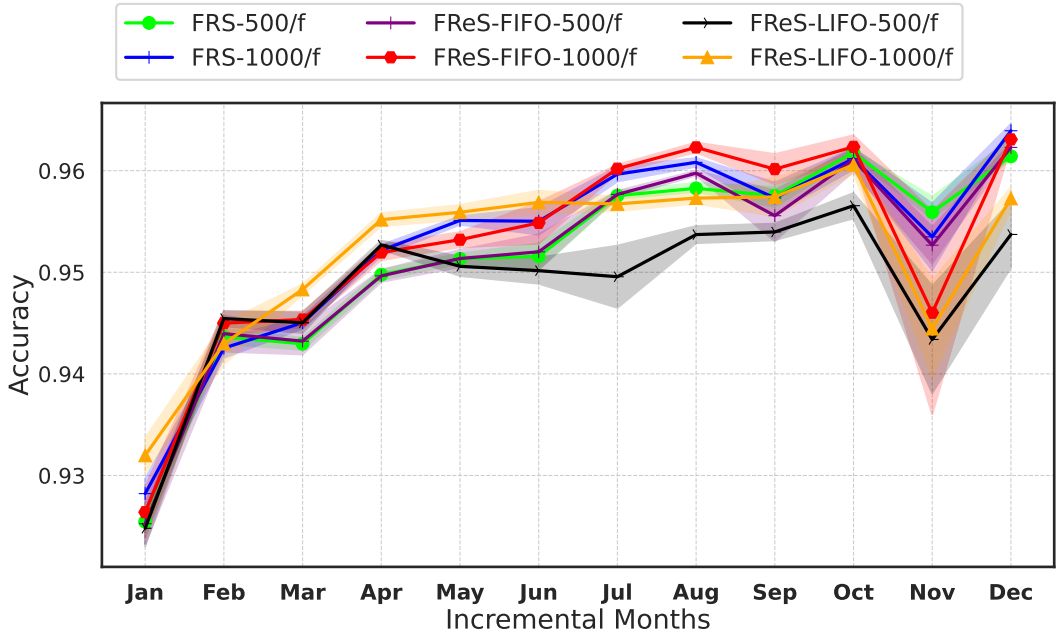


Figure A.2: FReS: Accuracy over time with different configurations of FReS in Domain-IL.

and $FRS - 1000/f$ performed comparably with $FRS - 50\%$. Hence, we only consider only these two configurations for two FReS sample selection criteria – LIFO and FIFO.

Figure A.2 depicts the performances of each of the studied configurations. In the figure, we can observe that $\{500, 1000\} \in FReS - LIFO$ are clearly struggling to compete with $\{500, 1000\} \in FReS - FIFO$ and $\{500, 1000\} \in FRS$, specially in the later tasks such as July, August, October, and December. $\overline{AT}$ and $\widehat{AT}$ show a more clear quantitative standing of the capacity of each of these configurations. $\overline{AT} = \{94.8, 95.2\}$ and $\widehat{AT} = \{94.3, 94.8\}$ for $\{500, 1000\} \in FReS - LIFO$, whereas $\overline{AT} = \{95.1, 95.3\}$ and $\widehat{AT} = \{94.8, 94.7\}$ for $\{500, 1000\} \in FReS - FIFO$. Both of the FReS-FIFO configurations are providing similar performance compared to FRS. Both of the FReS-LIFO configurations, on the other hand, are relatively performing slightly lower than FRS and FReS-FIFO. Even though, the score difference is very low, we still would like to choose the performant one for these investigations.

From the next set of investigations, we only consider 500 samples per family ($500/f$) as we would like to improve the performance of continual learning with as few samples as possible. As we can see $500/f$ for FReS-FIFO is performing similar to FRS-500/f, we move forward with this configuration. Even though $1000 - f$ for FReS-FIFO is also performing similar to FRS-1000/f, the number of replay samples is higher than $500/f$ configuration and our goal is to choose the cheaper one. Hence we aim to improve the accuracy with $500/f$ sample selection even further.

A.3.3 HS

As we have explained in the previous section, a density based clustering such as HDBSCAN can further form sub-clusters within a family based data distribution which in turn can be better to use as a reference to select samples within a family. Before we start running experiments with default and varying parameters of HDBSCAN, we first endeavor to find out whether HDBSCAN can actually provide sub-clusters within a family distribution and whether they are visually distinguishable. As this is the primary motivation and hypothesis, we would like to loosely confirm our intuition and whether this show us some direction or not.

To perform this analysis, (i) first we randomly choose three families – *emotet*, *installmonster*, and *zusy* among the top 100 malware families selected based on their frequencies, (ii) then we utilize HDBSCAN to produce sub-cluster and their associated sub-cluster labels for con-

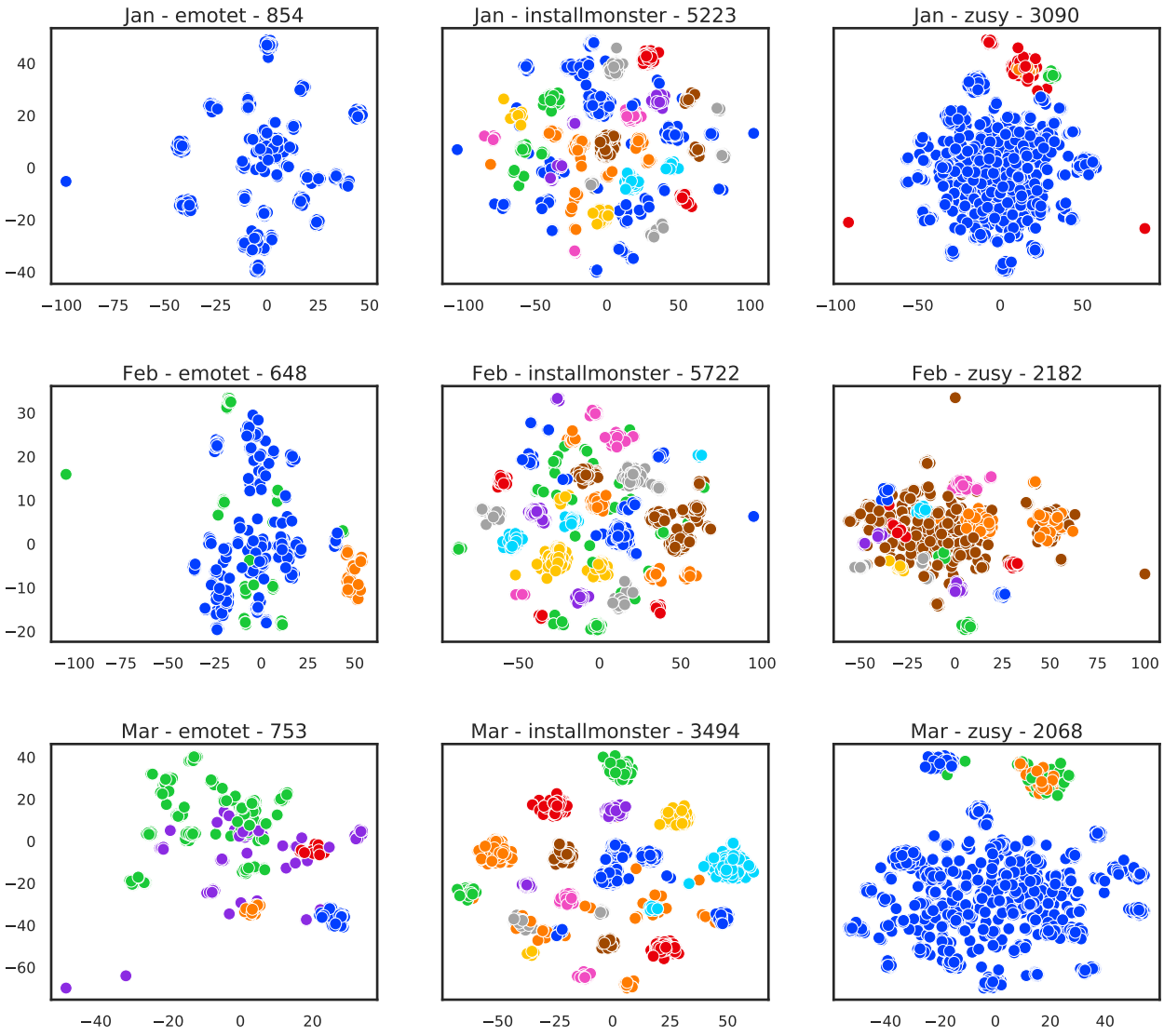


Figure A.3: Family data distribution in tSNE projection and colors represent the different cluster labels produced by HDBSCAN.

secutive three months (i.e., January, February, and March), (iii) afterwards, we get the 2D tSNE projections of the family data distribution, and (iv) finally, we label the points of the tSNE projection based on the sub-cluster labels produced by HDBSCAN. We show the visualization of this analysis in Figure A.3. Note that colors in each of the sub-figure are independent and similar color in different sub-figure does not indicate the same cluster.

The visualization initially confirms our hypothesis and we can, with some confidence, distinguish different clusters in each of the family based distribution if different task month. This

Table A.2: HS results with different HDBSCAN parameters. **HS-500/f** represents default HDBSCAN, and **HS-500/f (X-Y)** represents tuned HDBSCAN where X : `min-cluster-size` and Y : `min-samples`.

Metrics	HS-500/f	HS-500/f (10-5)	HS-500/f (30-2)	HS-500/f (50-2)	FRS-500/f
$\overline{AT}$	94.0	**94.7**	94.6	94.6	**95.1**
$\widehat{AT}$	92.9	**94.2**	92.7	92.7	**93.6**

provides us an indication that HDBSCAN based sample selection (HS) might be a better candidate for sample selection.

As we have mentioned before in Section A.2.3 that there are two important parameters of HDBSCAN – `min-cluster-size` and `min-samples`. In addition to the default HDBSCAN configuration, we selectively tune these two parameters to perform another three sets of experiments. We perform experiments with increasing `min-cluster-size` which include $\{10, 30, 50\}$ and reducing the `min-samples` to $\{2, 5\}$. While there can be any number of combinations of parameters that can be utilized to perform the experiment, we see from our results that increasing the `min-cluster-size` significantly does not contribute to more percentage in the accuracy and reducing the `min-samples` also does not help to boost the performance. Table A.2 shows the results of each of these four experiments. Note that we only perform experiments with 500-samples per family for HS. We also compare the performance of these set of experiments with FRS-500/f.

We can see from the table that the best performant HS configuration is with `min-cluster-size` $=$ 10 and `min-samples` $= 5$ yielding $\overline{AT} = 94.7$ and $\widehat{AT} = 94.5$. To our surprise, however, none of the HS configurations perform either on par or exceed FRS-500/f configuration. These results indicate that HDBSCAN based sampling may be failing to capture a better discriminative and representative samples.

Table A.3: Summary of the Experiments. The average accuracy over all tasks $\overline{AT}$ and the minimum accuracy among the tasks $\widehat{AT}$ for the sets of experiments. Bold indicates performances which are higher than that of at least GRS-50%.

Approach	**Method**	**Domain-IL**	
		$\overline{AT}$	$\widehat{AT}$
Baselines	None	92.2±1.0	91.8±1.1
	GRS-20%	94.4	93.8±1.1
	GRS-50%	**95.0±1.0**	**94.2±1.2**
	Joint	**95.9**	**93.7**
Regularization	EWC	92.8	90.0
	EWC-O	93.1	91.5
	SI	93.0	91.1
Replay	LwF	93.2	91.7
	GR	93.2	91.6
	GR-D	93.2	91.7
	RtF	93.1	91.1
	BI-R	93.4	91.6
Replay + Exemplars	ER	75.9	65±4.5
	A-GEM	77.5	67.4
Preliminary diversity aware replay (ours)	FRS-200/f	94.9	93.5±1.8
	FRS-500/f	**95.1**	**94.0±1.2**
	FRS-1000/f	**95.3**	**93.9±1.6**
	FReS-FIFO-500/f	**95.1**	**94.8±1.0**
	FReS-FIFO-1000/f	**95.3**±1.0	94.7 ± 1.5
	FReS-LIFO-500/f	94.8	94.3±1.0
	FReS-LIFO-1000/f	**95.2**	94.8 ± 1.0
	HS-500/f (10-5)	94.7	94.2

A.4 Analysis of the Preliminary Replay Buffer Techniques

We provide a summary of results from all experiments conducted for the replay sample selection configurations discussed in this appendix, shown in Table A.3. Specifically, we explored various techniques for selecting replay samples that capture the most discriminative and representative samples from family-based data distributions. Our investigation compares the performance of these techniques with GRS-50% and Joint configurations. In total, we experimented with over 10 variants of replay sample selection, categorized into FRS, FReS, and HS. Table A.3 showcases only the effective diversity-aware replay techniques that achieved average accuracy over all tasks ($\overline{AT}$) either on par with or exceeding FRS-50%.

Our exploration of FRS to select samples from a family based distribution which yielded relatively better performance with similar memory budget compared to global reservoir sampling. In the table, we can see FRS-500/f and FRS-1000/f configurations of this setting as performant with more than 95% accuracy. Then our exploration on utilizing either the latest or the oldest samples for the memory buffer proved to be also effective as well. We can see from the table that 3 out of four configurations of this setting (i.e., FReS-FIFO-500/f, FReS-FIFO-1000/f, and FReS-LIFO-1000/f) provide more than 95% accuracy. Realizing that these two settings still may lack to capture the most discriminative and representative samples, we expand our exploration into clustering based technique namely HDBSCAN technique the sample selection. Our results show that HDBSCAN based sample selection (HS) is relatively less effective.

In summary, our preliminary work demonstrates that state-of-the-art performance can be achieved using a diversity-aware replay-based continual learning technique for malware classification. This initial investigation provided valuable insights that informed the development of a more advanced diversity-aware replay continual learning technique for malware classification, as presented in Chapter 5.