# Temporal-Incremental Learning for Android Malware Detection

TIEZHU SUN*, University of Luxembourg, Luxembourg
NADIA DAOUDI, Luxembourg Institute of Science and Technology, Luxembourg
WEIGUO PIAN, University of Luxembourg, Luxembourg
KISUB KIM*, Singapore Management University, Singapore
KEVIN ALLIX, Independent Researcher, France
TEGAWENDÉ F. BISSYANDÉ and JACQUES KLEIN, University of Luxembourg, Luxembourg

Malware classification is a specific and refined task within the broader malware detection problem. Effective classification aids in understanding attack techniques and developing robust defenses, ensuring application security and timely mitigation of software vulnerabilities. The dynamic nature of malware demands adaptive classification techniques that can handle the continuous emergence of new families. Traditionally, this is done by retraining models on all historical samples, which requires significant resources in terms of time and storage. An alternative approach is Class-Incremental Learning (CIL), which focuses on progressively learning new classes (malware families) while preserving knowledge from previous training steps. However, CIL assumes that each class appears only once in training and is not revisited, an assumption that does not hold for malware families, which often persist across multiple time intervals. This leads to shifts in the data distribution for the same family over time, a challenge that is not addressed by traditional CIL methods. We formulate this problem as Temporal-Incremental Malware Learning (TIML), which adapts to these shifts and effectively classifies new variants. To support this, we organize the MalNet dataset, consisting of over a million entries of Android malware data collected over a decade, in chronological order. We first adapt state-of-the-art CIL approaches to meet TIML's requirements, serving as baseline methods. Then, we propose a novel multimodal TIML approach that leverages multiple malware modalities for improved performance. Extensive evaluations show that our TIML approaches outperform traditional CIL methods and demonstrate the feasibility of periodically updating malware classifiers at a low cost. This process is efficient and requires minimal storage and computational resources, with only a slight dip in performance compared to full retraining with historical data.

CCS Concepts: • **Software and its engineering** → **Software maintenance tools**; • **Security and privacy** → *Malware and its mitigation*; • **Computing methodologies** → *Learning paradigms*.

Additional Key Words and Phrases: Malware Classification, Incremental Learning

## 1 INTRODUCTION

The ever-evolving landscape of malware poses significant challenges for professionals and researchers in both the cybersecurity domain [1, 17, 64] and the software engineering field [5, 10, 62]. As malicious actors continuously craft newer and more sophisticated malware variants, traditional static detection and classification models often

*Corresponding authors.

Authors' addresses: Tiezhu Sun, tiezhu.sun@uni.lu, University of Luxembourg, Luxembourg; Nadia Daoudi, nadia.daoudi@list.lu, Luxembourg Institute of Science and Technology, Luxembourg; Weiguo PIAN, weiguo.pian@uni.lu, University of Luxembourg, Luxembourg; Kisub Kim, falconlk00@gmail.com, Singapore Management University, Singapore; Kevin Allix, kallix@kallix.net, Independent Researcher, France; Tegawendé F. Bissyandé, tegawende.bissyande@uni.lu; Jacques Klein, jacques.klein@uni.lu, University of Luxembourg, Luxembourg.


ACM 1557-7392/2024/11-ART
https://doi.org/10.1145/3702990

fall short, unable to adapt dynamically to these emerging threats [15, 32, 52]. In the era of rapidly advancing digital threats, there is a pressing need for dynamic, efficient, and adaptive solutions to recognize and categorize newly emerging malware families and adapt to shifts in data distribution within known malware families. A commonly adopted strategy to address this evolving challenge involves periodically retraining malware classifiers using the complete historical data repository during each update of the model [2, 8]. However, while this approach may initially seem effective, it imposes significant training time overhead and intensifies data storage demands. Indeed, the growing volume and complexity of accumulated data increase training time and storage requirements substantially, leading to escalating costs and data management complexities in terms of preprocessing, analysis, and learning iterations. Moreover, in many real-world settings, retaining historical data for full retraining is not feasible, due to privacy protection policies, data security concerns, or storage limitations. In such contexts, the full retraining approach, which requires access to all past data, becomes impractical if not impossible.

Class-incremental learning (CIL) has emerged as a potential solution to address these challenges associated with continuously evolving data streams [35, 42]. At its core, CIL aims to iteratively enhance the model's knowledge as new classes are introduced, relying mainly on data from current classes and minimizing the dependence on data from older classes. This paradigm ensures efficient use of computational resources and offers the potential for timely model adaptability. However, CIL operates on the assumption that each class, or in the context of our study, each malware family, appears uniquely in a distinct time interval, with no overlap across sequential time intervals. The concept is that once the model is trained on a malware family, it will not be re-trained on its newly emerged malware samples that will appear in future updates. Unfortunately, this fundamental assumption does not align with the reality of malware evolution, which involves two types of concept drift: ❶ the emergence of new malware families, and ❷ shifts in the data distribution of old families. Each type has its unique challenges, and their negative impacts on the effectiveness of malware classification models are detailed in Section 6.1. A malware family, rather than appearing in a fixed time period, often has an extended life cycle. This longevity means that even after a model is trained on a particular malware family (included in the training data of the current time interval), new samples from that family may appear in subsequent time intervals. Consequently, samples from the same malware family may exhibit significant data distribution shifts across different time periods. This complexity in malware dynamics makes conventional CIL approaches inadequate.

To address the aforementioned challenge, we introduce Temporal-Incremental Malware Learning (TIML). This learning problem is formulated to account for the aforementioned challenge and is thus tailored to the practical needs of evolving malware classification. TIML acknowledges the dynamic nature of malware families and their extended lifecycles, considering the presence of these families across multiple time intervals and adapting to variations in data distribution across instances. TIML builds upon foundational incremental learning principles, customized to address the complexities posed by malware classification. Figure 1 provides an illustration of time steps in the context of TIML, which depicts *when the model should be re-trained*. In the example depicted in this diagram, each time step indicates the endpoint of a time interval when the model requires (re)training, whether for initial training or due to a deterioration in performance that necessitates updates. Each time interval corresponds to the life spans of specific malware families, highlighting the need for model updates to integrate new malware samples and maintain robustness against evolving malware signatures.

In practical terms, the formulation of TIML allows for reasoning about efficient model updates. Indeed, it sets constraints on the need to retain acquired knowledge and adapt it in the case of concept drift without the need to train on the entire historical dataset. TIML therefore targets solutions that minimize resource and time requirements, leading to models that are ready for deployment in a timely and cost-effective manner. To better illustrate the reality of the problem, we consider a simple real-world scenario. Specifically, we assume that we have a malware classifier trained on samples from numerous families. The classifier can indeed efficiently assign a malware label to unseen malware samples that belong to families seen in the training process. However, after a few months, new malware emerged (i.e., samples exhibiting the two types of concept drift), which impacted

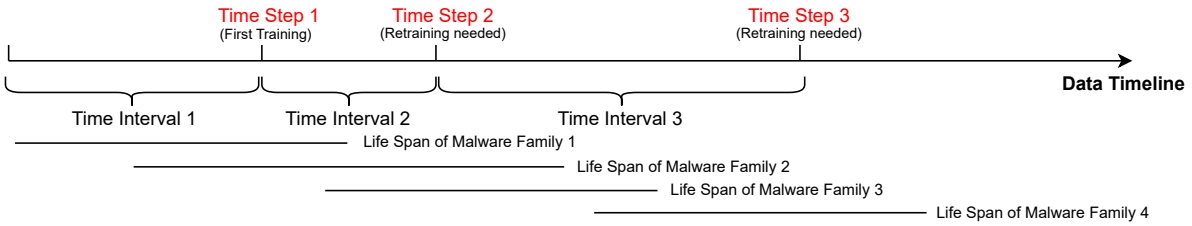


Fig. 1. Retraining schedule of the TIML framework, depicting he specific time steps when the model should be retrained based on the chronological emergence of malware families.



the performance of the malware classifier. Therefore, the model needs to be retrained on these new samples to preserve its performance. Retraining the model on the full dataset (i.e., newly collected samples and samples previously used to train the model) is impractical as it requires huge resources and time (e.g., the retraining can take more than one month based on our dataset). Consequently, updating the model based primarily on newly emerged samples seems to be a suitable solution. This strategy ensures that the model update is cheap as storing the full dataset from previous classes is not required. In addition, the training time is short, which would make the model ready for deployment in a timely manner.

To address the TIML problem, we first adapt several state-of-the-art CIL approaches to ensure their compatibility with this new framework, establishing them as baseline methods in our experiments. Then, we propose a novel multimodal TIML approach to capitalize on the detailed information available in MalNet [18] images and Malscan [61] call graphs. This dual-modality approach is designed to improve malware classification by combining the visual cues from images with the behavioral patterns from call graphs, providing a richer context for classifying both known and emerging malware threats with greater precision. To start our evaluation, the prerequisite is a robust and comprehensive dataset. For that, we consider MalNet [18], which is a benchmark dataset for malware classification containing over **1.2 million** Android malware from nearly seven hundred families. However, its current structure does not inherently meet the requirements of the TIML setting. Therefore, we carefully organized it according to the chronological order of the emergence of malware samples. Finally, we conduct comprehensive experiments from various perspectives to demonstrate that TIML-based malware classifiers can be dynamically updated with minimal storage and computational burdens. Our findings reveal the effectiveness of both the adapted TIML methods and our novel multimodal approach in classifying malware families within the TIML paradigm. The results indicate a slight dip in performance during subsequent time steps when compared to models retrained on the entire dataset, a consequence of the limited access to historical data. However, this decline is counterbalanced by the substantial reduction in training time and data storage requirements that TIML approaches offer compared to the comprehensive retraining method. To illustrate, the adapted TIML approach, "LwF with Exemplar," showcases a notable efficiency, reducing training time and data storage demands by 81% and 78% respectively, while only experiencing a modest 6.9 percentage points decline in accuracy. This underscores the operational efficiency of TIML approaches, highlighting their potential for real-world applications where resource optimization is crucial. Through these explorations, our ultimate goal is to pave the way for more robust, adaptive, and efficient malware classification systems in an increasingly volatile digital world.

The contributions of our study are as follows:

- We formulate TIML: Temporal-Incremental Malware Learning, an overlooked problem, which is introduced to dynamically accommodate shifts in data distributions of malware families and to efficiently classify emerging threats while reducing resource utilization.
- We provide a meticulously restructured **million-scale** dataset of Android malware tailored for the TIML paradigm, serving as a valuable resource for future research endeavors.
- We adapt four CIL approaches to address the TIML problem. The results show that the adapted approaches effectively classify malware families with limited resources.
- We introduce a novel multimodal TIML approach that enhances classification performance by effectively utilizing multiple malware data modalities.
- To facilitate replication, we share the dataset and the source code with the community at the following address: https://github.com/Trustworthy-Software/TIML

## 2 BACKGROUND

This section provides an overview of the context and challenges addressed in our study. We begin with a general background discussing the role of deep learning in malware classification and the phenomenon of concept drift. Following this, we delve into the technical background necessary for our proposed approaches.

### 2.1 General Background

This subsection explores the integration of deep learning techniques in malware classification and discusses the implications of concept drift in this domain.

*2.1.1 Deep Learning in Malware Classification.* Our study tackles a critical issue in software engineering: the evolution and classification of malware, a problem underscored by the dynamic nature of software threats and the necessity for robust defensive mechanisms. In the era of advanced artificial intelligence, the software engineering community has increasingly adopted deep learning techniques, resulting in significant advancements across various sub-domains, including malware classification. Specifically, recent seminal works such as Meta-MAMC [36], XMal [59], MalCertain [34], AMCDroid [38], and API2Vec++ [9] have introduced significant deep learning innovations to tackle the challenges of malware classification. These contributions highlight the vibrant and ongoing efforts within the software engineering community to harness deep learning for developing more effective solutions in malware defense. Our research contributes to this ongoing trend by addressing specific gaps and introducing novel methodologies that further enhance the capability of deep learning in the context of malware classification.

*2.1.2 Concept Drift.* Concept drift represents a significant challenge in malware classification [6, 20, 30], arising from the dynamic and evolving nature of malware threats. Conceptually, it refers to the changes in the underlying patterns of data that models are trained to predict. In the context of malware, there are primarily two types of concept drift:

- **Emergence of New Malware Families**: This type of drift occurs when entirely new categories of malware are developed. These new families often exhibit unique behaviors and characteristics that were not present in the training data, making them difficult for existing classifiers to detect without updates.
- **Shifts in Data Distribution of Old Families**: Even when the malware family remains the same, its manifestations can evolve. This type of drift involves subtle changes in the distributions of feature vectors of known malware families. Variations in these distributions can be due to factors like new attack vectors, enhanced obfuscation techniques, or other evolutionary changes in malware behavior. These changes can gradually or suddenly render previously effective detection models obsolete by altering the underlying data patterns that the models were trained to recognize.

Our study is dedicated to rigorously investigating the presence and impact of these two types of concept drift using the large-scale MalNet dataset, which contains 1.2 million real-world malware samples. This comprehensive data set provides a solid foundation for observing and analyzing how malware evolves over time. Following our investigation, we explore methodologies to efficiently and effectively update malware classifiers. This involves developing strategies that not only quickly adapt to new threats as they emerge but also recalibrate the model parameters to account for the subtle shifts in known malware distributions, thereby maintaining high levels of accuracy and responsiveness in malware detection systems.

## 2.2 Technical Background

In this section, we introduce two important concepts that are integral to our proposed approaches.

*2.2.1 Exemplar Set.* An Exemplar Set is a key concept in incremental learning, which involves selecting a representative subset of data from previously encountered classes to aid in the training of new models without revisiting the entire historical dataset [11, 66]. In the context of TIML, exemplar sets play a pivotal role by enabling the model to retain crucial information about older malware families while incorporating knowledge from newly emerging threats, thus preventing catastrophic forgetting. To align with the terminology used in the incremental learning literature, we refer to the process of training the model on a dataset at a specific time step as a *Task*.

**Exemplar Set Management:** With the evolving nature of the data stream, managing the exemplar set becomes crucial. Two predominant strategies emerge in standard incremental learning [27]. Transferring to malware family classification, we have:

(1) *Fixed Exemplars Per Family:* This approach maintains a constant count of exemplars for each family. Consequently, as the number of malware families increases, the size of the exemplar set expands. However, this leads to a linear increase in memory usage, making it unsustainable for real-world learning systems.
(2) *Fixed Total Exemplars:* This strategy focuses on maintaining a constant total number of exemplars across all families. Specifically, the model allocates $\left\lfloor \frac{M}{|\mathcal{Y}|} \right\rfloor$ exemplars per family, where $M$ is the fixed total number of exemplars and $|\mathcal{Y}|$ represents the total number of known malware families up to the most recent task. The $\lfloor \cdot \rfloor$ denotes the floor function. By standardizing the total memory used for exemplars, this approach ensures a consistent and manageable memory footprint, addressing storage constraints effectively.

In our research, we adopt the second strategy for exemplar set management.

**Exemplar Selection**: Exemplars are representative instances from each known family, chosen from the entire training set. A straightforward approach to determine the exemplars is to sample them randomly for each family, ensuring instance diversity. In contrast, another prevalent method in standard class-incremental learning is the herding technique [44, 57], which seeks to pinpoint the most representative instances for each class. Despite its advantage, this approach increases computational overhead and operates on the assumption that each class appears only once in a single time step—a premise not applicable to our TIML scenario. Consequently, we adopt the random sampling method.

*2.2.2 Loss Functions.* Before discussing our adapted and proposed approaches, we first formalize the two standard loss functions commonly used as sub-loss functions in incremental learning.

**Cross-entropy loss** [25], often used in classification tasks, measures the dissimilarity between the predicted probability distribution $P$ and the ground truth distribution $Q$. It is especially common in training neural networks for multi-class classification problems. Given:

- $P(y)$ as the predicted probability distribution,
- $Q(y)$ as the ground truth distribution

The cross-entropy loss is defined as:

$$\mathcal{L}_{CE}(P, Q) = -\sum_{y} Q(y) \log(P(y)) \tag{1}$$

**Kullback-Leibler (KL) Divergence** [33] is a measure of how one probability distribution diverges from an expected probability distribution. It is often used in scenarios involving generative models or when comparing two probability distributions. Given:

- $P(y)$ as the true probability distribution,
- $Q(y)$ as the approximate distribution,

The KL Divergence is defined as:

$$\mathcal{L}_{KL}(P||Q) = \sum_{y} P(y) \log\left(\frac{P(y)}{Q(y)}\right) \tag{2}$$

## 3 TEMPORAL-INCREMENTAL MALWARE LEARNING

Temporal-Incremental Malware Learning (TIML) is an incremental learning problem that fits with the practical use of evolving malware classification. We address TIML by dynamically responding to changes in data distributions among known malware families, while also proficiently identifying and categorizing newly emerging ones. To that end, we consider that the historical timeline of the data can be split into different *time steps*. A fundamental stipulation inherited from incremental learning is the limited data accessibility for model updates; specifically, the model training is limited to data from the current time step, supplemented by a restricted subset of samples from previous time steps—known as the exemplar set—when applicable. Figure 2 presents an overview of the Temporal-Incremental Malware Learning (TIML) problem formulation, with data variables detailed in the subsequent formal definition.



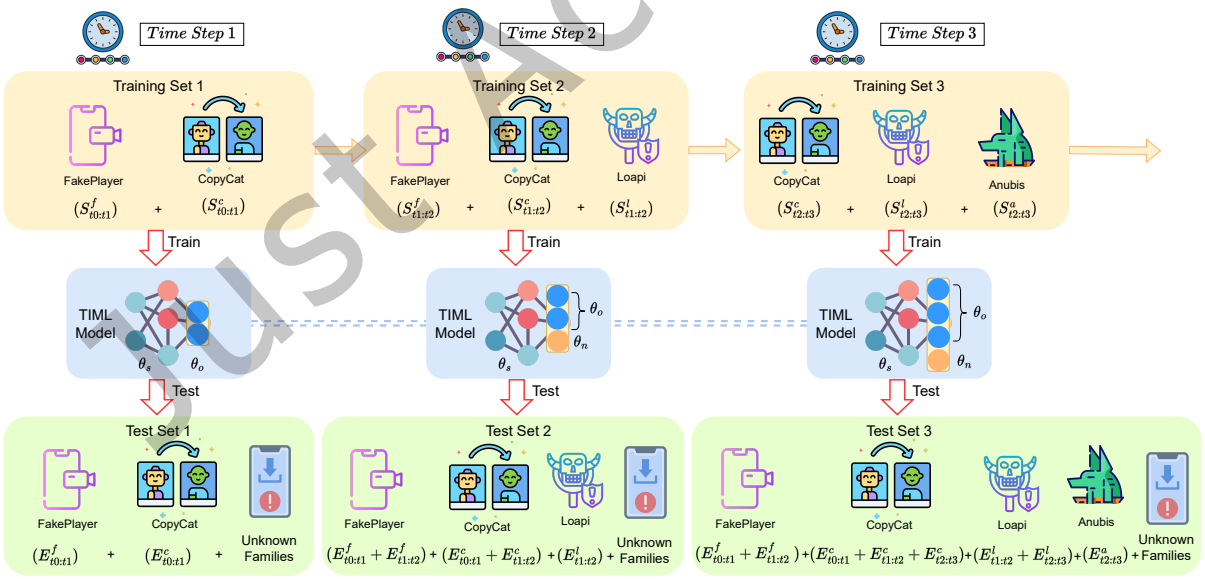


Fig. 2. Overview of Temporal-Incremental Malware Learning.

To precisely define the TIML problem, let us start by delineating a series of time steps:

$$T = \{T_1 : [t_0, t_1], T_2 : [t_1, t_2], \ldots, T_b : [t_{b-1}, t_b], \ldots, T_B : [t_{B-1}, t_B]\}. \tag{3}$$

Here, each time period $T_b$ represented as a distinct time interval $[t_{b-1}, t_b]$, corresponds to a unique duration for data accumulation. Training executed on the data from each time step $t_b$ is perceived as a distinct task. Given a dataset $S$, this results in a sequence of tasks $\{D^1, D^2, \ldots, D^b, \ldots, D^B\}$, where the subset $S_{t_{b-1}:t_b}$ corresponding to task $D^b$ encompasses $n_b$ training instances, expressed as $\{(x_i^b, y_i^b)\}_{i=1}^{n_b}$. Each instance $x_i^b$ belongs to malware family $y_i$ which is situated within the label space $Y_b$ for that task. Given that malware families typically have a life cycle, they may appear across multiple time steps. Consequently, consecutive label spaces from $S$, namely $\{Y_1, ..., Y_b\}$, could have overlapping malware families. We give the definition of TIML at a specific time step as follows:

Definition 1. *Given a specific time step $T_b$ (relating to the period $[t_{b-1}, t_b]$ within $T$), the following components and specifications apply:*

*(1) The training dataset during $T_b$, denoted as $S_{t_{b-1}:t_b}$ corresponding to task $D^b$, is comprised of:*
- *Samples originating from the newly introduced malware families: $N_{t_{b-1}:t_b}$.*
- *New samples from previously seen families: $O_{t_{b-1}:t_b}$.*

*(2) The dataset $S_{t_{b-1}:t_b}$ encompasses instances $\{(x_i^b, y_i^b)\}_{i=1}^{n_b}$, where $x_i^b$ indicates the malware feature vector and the corresponding label $y_i^b$ belongs to the label space $Y_b$, inclusive of both old and new malware families introduced only in current time step.*

*(3) For the instances contained in $N_{t_{b-1}:t_b}$, there are no overlapping malware families with the previous sequences $\{N_{t_0:t_1}, \ldots, N_{t_{b-2}:t_{b-1}}\}$.*

*(4) Samples located in $O_{t_{b-1}:t_b}$, and originating from the same unique malware family, are expected to conform to a distribution $F_{t_{b-1}:t_b}(x, y)$. Any shift in data distribution in the current time step's samples is observed if:*

$$F_{t_{b-l}:t_{b-l+1}}(x, y) \neq F_{t_{b-1}:t_b}(x, y), \text{ where } 2 \leqslant l \leqslant b. \tag{4}$$

*(5) For evaluation purposes during the $T_b$ time step, the testing data includes samples from all tasks up to the present: $D^{t_0:t_b} : \{D^1, \ldots, D^b\}$. This incorporates instances from the malware families in the current time step as well as all previously observed ones, i.e., $\{E_{t_0:t_1}, \ldots, E_{t_{b-1}:t_b}\}$. The overall label space is $\mathcal{Y}_b = Y_1 \cup \ldots Y_b$.*

*Consequently, the TIML problem requires a fitted model $f(x) : X \rightarrow \mathcal{Y}_b$, which minimizes the expected risk:*

$$f^* = \arg\min_{f \in \mathcal{H}} \mathbb{E}_{(x,y) \sim D_t^1 \cup \ldots D_t^b} \mathbb{I}(y \neq f(x)) \tag{5}$$

*where $\mathcal{H}$ is the hypothesis space, $\mathbb{I}(\cdot)$ is the indicator function which outputs 1 if the expression holds and 0 otherwise. $D_t^b$ denotes the data distribution of task $b$. A good TIML model satisfying Equation 5 can discriminate all malware families, which not only works well on new samples with two types of concept drift but also preserves the knowledge of former samples.*

Essentially, Temporal-Incremental Malware Learning (TIML) leverages training samples exclusively from the current time step, facilitating several key functionalities: It efficiently recognizes and classifies newly emerging malware families; it continuously adapts to distribution shifts within new samples of previously seen malware families; and it effectively preserves the knowledge from old samples of former families that are not impacted by distribution shifts. This paradigm requires that the model remains both current and robust, capable of handling the dynamic nature of malware threats without the need for comprehensive retraining on historical data. Nonetheless, in certain scenarios where more robust knowledge retention is necessary, this constraint is eased, allowing the model to maintain an exemplar set for enhanced adaptability and learning continuity.

## 4 TIML METHODOLOGY

In this section, we first provide our methodology of the data organization in Section 4.1. Then, we introduce the adapted TIML approaches in Section 4.2. In Section 4.3, we present our proposed multimodal TIML approach.

### 4.1 Data Organization

**Algorithm 1:** Algorithm describing the dataset organization

```
Input: dictAppDate, dictAppFamily, timeWindow, minNbrFamiliesInTask, minNbrSamples
Output: Subsets of malware apps organized according to TIML
minDate ← dictAppDate.values().min()
maxDate ← dictAppDate.values().max()
tasks ← ∅
currentMinDate ← minDate
Function constructTask (currentMinDate, currentMaxDate):
    task ← getAppsBetweenDates(dictAppDate, currentMinDate, currentMaxDate)
    task ← getFamiliesOfAppsInTask(task, dictAppFamily)
    familiesInTask ← getUniqueFamiliesInTask(task, minNbrSamples)
    return task, familiesInTask
while (currentMinDate ≤ maxDate) do
    currentMaxDate ← currentMinDate+timeWindow
    task, familiesInTask = constructTask(currentMinDate, currentMaxDate)
    while FamiliesInTask.length() ≤ minNbrFamiliesInTask do
        currentMaxDate ← currentMaxDate + 1
        task, familiesInTask = constructTask(currentMinDate, currentMaxDate)
    end
    currentMinDate ← currentMaxDate
    tasks.append(task)
end
```

In this section, we describe how we organize the data and find appropriate time intervals, aiming to respect the reality of malware families' emergence. We start by identifying the dates of the sample apps from Androzoo [4] and arranging them chronologically. We then proceed to systematically organize the collected samples for distinct tasks towards incrementally training the model. The general rule we use involves splitting the samples over a period of `timeWindow` months, starting from the earliest date associated with our data samples.

Before moving to identify the samples for the next task, it is crucial to ensure that a minimum number of malware families are available to train the model effectively for a given task. To this end, we use the threshold `minNbrFamiliesInTask`, which dictates the minimum number of families required in a task before proceeding to the next. If the number of families identified within a given time window is less than `minNbrFamiliesInTask`, we incrementally expand the time window by one month until this requirement is met. It's important to note that the end date of a task becomes the start date for the next. In addition, each malware family must meet a minimum sample threshold, denoted as `minNbrSamples`, to be included in the training set. A family with fewer than `minNbrSamples` is not considered during training and its samples are instead included in the testing set, labeled as "unknown". A family is counted towards `minNbrFamiliesInTask` only if it satisfies the `minNbrSamples` criterion.

These thresholds are essential for determining the appropriate time intervals for training tasks, given the irregular timing patterns of malware emergence. They ensure that we collect a sufficient number of samples and families within each interval, which is vital for optimizing training resources and maintaining the robustness of the TIML model. Retraining the model too frequently with only a few new malware samples would be inefficient and ineffective. This strategic approach to data collection and training ensures that our model is both efficient and effective, capable of learning robustly from sufficiently populated data sets without the inefficiencies associated with sparse data. We summarize our data organization method in Algorithm1.

## 4.2 TIML Approaches

In this section, we provide a formal definition of the baseline approaches considered in this study. Specifically, our four baselines are adaptations of three class-incremental learning techniques: LwF [37], iCaRL [44], and SS-IL [3]. Due to space constraints, we focus primarily on describing the adapted approaches, with a particular emphasis on the modifications to the loss functions.

While our manuscript centers on these loss function adaptations, it's crucial to note that our actual adaptations extend beyond this single aspect. These adaptations are pivotal for aligning the class-incremental learning techniques with the unique requirements of Temporal-Incremental Malware Learning (TIML), effectively addressing differences in data composition and optimizing the knowledge distillation process inherent to TIML. The choice to focus on loss functions was strategic; these modifications encapsulate the core changes made to the baseline approaches and are most indicative of the methodological shifts required for successful application within our TIML framework. For full details on the original methods and broader context of the adaptations, we refer readers to the respective foundational papers.

**Algorithm 2:** The general TIML algorithm

**Start with:**
- $\theta_s$: shared parameters
- $\theta_o$: task specific parameters for malware families introduced in old tasks
- $X_n, Y_n$: training data and ground truth from the new task
- $X_e, Y_e$: exemplar data and ground truth from the old tasks (optional)

**Initialize:**
- $Y_o \leftarrow Model(X_n, \theta_s, \theta_o)$ // compute old task output of old model for new data
- $\theta_n \leftarrow RandInit(|\theta_n|)$ // randomly initialize new parameters

**Train:**
- Define $\hat{Y}_o \equiv Model(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // old task output of new model for new data
- Define $\hat{Y}_n \equiv Model(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // new task output of new model for new data
- Define $\hat{Y}_e \equiv Model(X_e, \hat{\theta}_s, \hat{\theta}_o)$ // old task output of new model for exemplars (optional)
- $\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\arg\min}\ \mathcal{L}_{TIML}(Y_o, Y_n, Y_e; \hat{Y}_o, \hat{Y}_n, \hat{Y}_e)$ // depends on the specific method

*4.2.1 Adaptations for TIML.* We now elaborate on the adaptations brought to CIL approaches in order to fit with the TIML problem. Given a model with shared parameters $\theta_s$ and task-specific parameters $\theta_o$ (as shown in Figure 2), our goal is to ❶ add task-specific parameters $\theta_n$ for newly introduced malware families and ❷ learn parameters adapted to old and new tasks, using samples and labels only from the new task (and exemplars, if applicable). The general algorithm of TIML and the notations we will use in the following are outlined in Algorithm 2. Particularly, the training data $X_n, Y_n$ from the new task consists of two sources: ❶ seen malware families (i.e., samples belonging to malware families used to train the model in previous time steps) $X_n^{(seen)}, Y_n^{(seen)}$, and ❷ unseen malware families (i.e., samples from newly emerging malware families) $X_n^{(unseen)}, Y_n^{(unseen)}$. Following the experimental setting in MalNet [18], we employ ResNet-18 [26] as the backbone of each classifier. Note that the malware images in the MalNet dataset are created by converting Android app bytecode into images, mapping each byte to a pixel value.

The loss function for each baseline approach is defined as follows:

**Fine-tuning**: Serving as the lower bound among the baseline approaches, Fine-tuning solely utilizes new data from the current time step and employs a straightforward loss function, namely the cross-entropy loss. Contrary to the standard class-incremental learning scenario, we adapt Fine-tuning in the context of TIML to take into account the newly introduced samples belonging to previously seen malware families:

$$\mathcal{L}_{CE}(Y_n, \hat{Y}_n) = \mathcal{L}_{CE}(Y_n^{(seen)}, \hat{Y}_n^{(seen)}) + \mathcal{L}_{CE}(Y_n^{(unseen)}, \hat{Y}_n^{(unseen)}) \tag{6}$$

**Full Retraining**: As the upper bound, Full Retraining uses the same loss function as Fine-tuning but incorporates all historical data along with new data for training at each time step.

**LwF**: In contrast to Fine-tuning, LwF employs a task-specific knowledge distillation technique to retain knowledge from prior data. Specifically, it incorporates the KL divergence as an additional loss term to ensure the model's output distribution remains consistent with that of previous time steps:

$$\mathcal{L}_{LwF} = \mathcal{L}_{CE}(Y_n, \hat{Y}_n) + \lambda_o \sum_{t=1}^{b-1} \mathcal{L}_{KL}(\hat{Y}_o^{(D_t)} || Y_o^{(D_t)}) \tag{7}$$

where $b$ is the current time step and $D_b$ is its corresponding task, $\hat{Y}_o^{(D_t)}$ corresponds to the output on families that were firstly introduced in time step $t$, and $\lambda_o$ is a loss balance weight.

**LwF with Exemplars**: The original LwF methodology [37] does not incorporate exemplar data. However, in our experiments, we observed that integrating a small amount of exemplar samples can substantially enhance its performance. The corresponding loss function is defined as:

$$\mathcal{L}_{LwF-Exemp} = \mathcal{L}_{CE}(Y_n, \hat{Y}_n) + \mathcal{L}_{CE}(Y_e, \hat{Y}_e) + \lambda_o \sum_{t=1}^{b-1} \mathcal{L}_{KL}(\hat{Y}_o^{(D_t)} || Y_o^{(D_t)}) \tag{8}$$

**iCaRL**: The original iCaRL [44] introduces a herding-based method for selecting highly representative exemplars and employs a *nearest-mean-of-exemplar* rule for classification, both requiring stable data distributions across all the time steps for each family. Considering the potential shifts in malware family distributions, these techniques are inapplicable to TIML. Consequently, we opt for a random exemplar selection strategy and a conventional softmax-based classification approach. For knowledge distillation, we retain its overall KL loss term:

$$\mathcal{L}_{iCaRL} = \mathcal{L}_{CE}(Y_n, \hat{Y}_n) + \mathcal{L}_{CE}(Y_e, \hat{Y}_e) + \lambda_o \mathcal{L}_{KL}(\hat{Y}_o || Y_o) \tag{9}$$

**SS-IL**: This approach is designed to tackle a primary cause of catastrophic forgetting: the classification score bias introduced by the data imbalance between unseen classes and seen classes (stored in the exemplar-memory). SS-IL [3] introduces a separated softmax output layer to compute the cross-entropy for both seen and unseen classes independently, coupled with a task-specific knowledge distillation similar to LwF. Under the TIML definition, the seen malware families encompass not only those from the exemplars but also the new samples from the current time step that belong to previously seen malware families. The resultant loss function is formulated as:

$$\begin{aligned}\mathcal{L}_{SS-IL} =& \mathcal{L}_{CE}((Y_n^{(seen)})^{(D_1:D_{b-1})}, (\hat{Y}_n^{(seen)})^{(D_1:D_{b-1})}) + \mathcal{L}_{CE}((Y_n^{(unseen)})^{(D_b)}, (\hat{Y}_n^{(unseen)})^{(D_b)}) \\ &+ \mathcal{L}_{CE}(Y_e^{(D_1:D_{b-1})}, \hat{Y}_e^{(D_1:D_{b-1})}) + \lambda_o \sum_{t=1}^{b-1} \mathcal{L}_{KL}(\hat{Y}_o^{(D_t)} || Y_o^{(D_t)})\end{aligned} \tag{10}$$

## 4.3 Multimodal TIML

In our study, we focus on two primary malware features: bytecode images and graph centrality vectors. While each offers unique insights, previously adapted TIML methods were limited to using only one type of feature. However, we observed that combining these features could significantly enhance our understanding and performance in malware classification within the TIML framework. Consequently, we introduce a multimodal TIML approach that integrates both features, aiming for superior performance by leveraging the complementary strengths of bytecode images and graph centrality vectors.

The core motivation for adopting a multi-modal approach stems from the need to enhance the performance of TIML by enabling a more comprehensive understanding of malware behaviors. This is especially crucial in

TIML settings where the available training samples are significantly fewer than in full retraining scenarios. Maximizing the informational yield from each sample is essential to maintain robust model performance despite reduced training volumes. Our choice to combine bytecode images and function call graphs is driven by their complementary nature in representing malware characteristics:

- **Bytecode Images**: These provide a visual representation of the binary structure of the malware, offering insights into the compiled code without executing the malware. This modality helps in capturing structural patterns that are indicative of malicious intent or behavior.
- **Function Call Graphs**: These graphs map out the relationships and interactions between different functions within the malware code, offering a detailed view of the execution flow and potential points of malicious activity.

By combining these two modalities, our model benefits from a holistic view of each malware sample, leveraging structural insights from bytecode images and behavioral insights from function call graphs. This comprehensive input is designed to mitigate the impact of having fewer samples by enriching the information the model can learn from each individual sample.

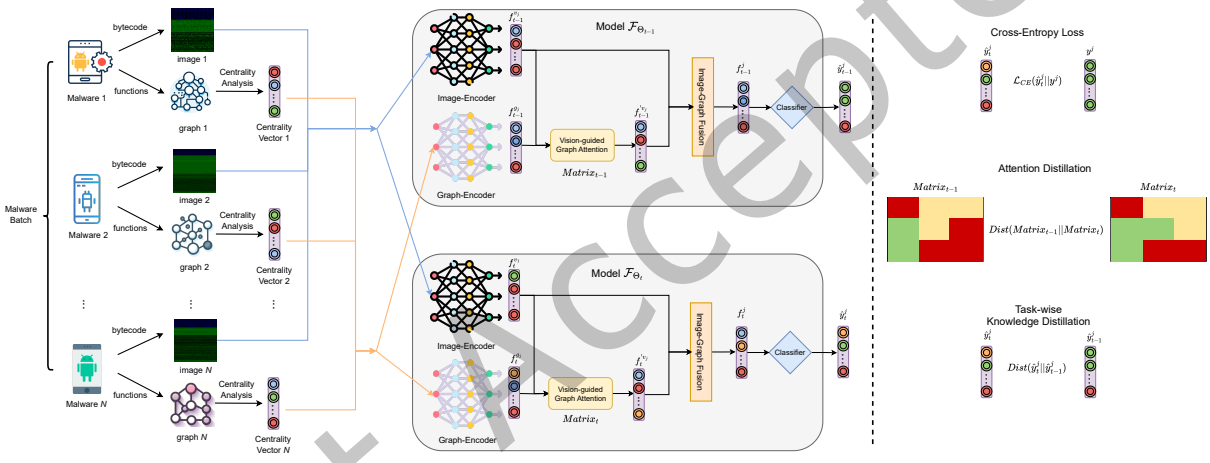


Fig. 3. Illustration of our proposed multimodal TIML approach.

*4.3.1 Overview.* As depicted in Figure 3, our multimodal TIML approach begins by extracting two primary features from each APK: the bytecode image and the centrality vector from the function call graph. Following the methodology outlined by MalNet [18], bytecode is extracted from the DEX files within the APK and converted into an image. The conversion is performed by mapping each byte in the DEX files to a corresponding pixel value in the image. Simultaneously, we utilize Malscan [61] to derive the centrality vector, which captures the critical interaction points within the function call graph.

Each of these features undergoes a separate but specialized processing pathway using custom feature extractors designed to produce high-level embeddings. The centrality vectors, comprising various base features, are particularly refined through a novel attention mechanism. This mechanism leverages the image embeddings as a guide, enhancing the focus on essential features within the vector that are most indicative of malware characteristics. This process is crucial as it allows for the centrality vector to be dynamically adjusted based on the insights provided by the visual patterns observed in the bytecode images.

The final stage of our approach involves the integration of these processed embeddings. As illustrated in the central portion of Figure 3, embeddings from both the bytecode image and the graph centrality vector are fused together. This fusion occurs in a fully connected layer designed to consolidate and capitalize on the complementary strengths of both modalities, enabling a more robust classification of the malware family. The integrated embeddings are then used to classify the APK into one of the known malware families.

*4.3.2 Model Architecture.* In the adapted TIML model architecture, ResNet-18 [26] is employed as the primary feature extractor, originally designed for real-world image classification within CIL methodologies. Recognizing the distinct characteristics of malware data, specifically bytecode images and graph centrality vectors, which lack the spatial and semantic depth of natural images, we opt for a more tailored, lightweight backbone in our multimodal TIML approach. This customized approach includes three 2D convolutional layers (kernel size=5, padding=2) for bytecode images, and three 1D convolutional layers (kernel size=7, padding=1) for graph centrality vectors, each followed by a linear layer to manage embedding dimensions.

The attention module plays a pivotal role, in processing the embeddings from both modalities. It adjusts the image embeddings for compatibility and computes attention weights by assessing the importance of different parts of vector embeddings relative to the overall guidance (i.e., image embedding). These weights are crucial for focusing the model's attention on the most pertinent data features. Subsequently, the module applies these weights to the vector embedding, yielding an "attended" vector that highlights information vital for malware classification, alongside the attention weights for further attention distillation in the training phase. Finally, we fuse the attended vector embedding with the image embedding through element-wise addition, directing the combined output to a fully connected layer for the ultimate task of malware family classification. This architecture leverages the distinct strengths of each data modality while ensuring a focused, efficient approach to malware identification.

*4.3.3 Loss Function.* The loss function for our multimodal TIML model is composed of three segments: cross-entropy for learning from new malware instances, task-specific KL divergence for distilling knowledge from prediction logits, and standard KL divergence to maintain attention-weight consistency. This comprehensive loss function ensures balanced learning and knowledge retention across updates and is defined as:

$$\mathcal{L}_{MM-TIML} = \mathcal{L}_{CE}(Y_n, \hat{Y}_n) + \mathcal{L}_{CE}(Y_e, \hat{Y}_e) + \lambda_o \sum_{t=1}^{b-1} \mathcal{L}_{KL}(\hat{Y}_o^{(D_t)} || Y_o^{(D_t)}) + \lambda_1 \mathcal{L}_{KL}(\hat{W}^{(D_t)} || W^{(D_t)}) \tag{11}$$

where $\hat{W}^{(D_t)}$ is the attention weights of old model on new data, i.e., $Matrix_{i-1}$ in Figure 3, and $W^{(D_t)}$ is the attention weights of new model on new data, i.e., $Matrix_i$ in Figure 3.

## 5 STUDY DESIGN

In this section, we first overview the research questions. Then, we present the details about the dataset and empirical setup. Last, we define the metrics we use for evaluation.

### 5.1 Research Questions

In this study, we aim to investigate the following research questions:

**RQ1** *Is concept drift a significant factor affecting malware classification?*
**RQ2** *How well do TIML approaches perform in malware classification?*
**RQ3** *How resilient are TIML approaches to catastrophic forgetting?*
**RQ4** *How effectively do TIML approaches optimize resource utilization?*

## 5.2 Dataset Description

We conducted our experiments on the MalNet dataset [18], which is uniquely suitable for our study's objectives. MalNet contains over 1.2 million malware images, originally sourced from AndroZoo [4], with each app's bytecode converted into colored images of size (256, 256). This dataset encompasses 696 malware families, labeled based on analyses from VirusTotal [54] and Euphony [29]. Euphony takes ViruTotal report containing up to 70 labels across a variety of antivirus vendors and unifies the labeling process by learning the patterns, structure, and lexicon of vendors over time. Importantly, MalNet covers a substantial time span of 10 years, providing essential temporal data that is critical for analyzing the evolution of malware threats.

The choice to exclusively utilize MalNet was driven by its comprehensive scale and depth, containing significantly more malware samples and families compared to other available datasets. For instance, when compared to Virus-MNIST, the next largest public dataset, MalNet offers approximately 24 times more malware samples and nearly 70 times more families. Furthermore, unlike Virus-MNIST, MalNet includes vital temporal information, such as the emergence dates of malware instances, making it the only dataset we identified that meets all critical requirements for Temporal-Incremental Malware Learning (TIML): large scale, extended duration, and detailed temporal information. These features make MalNet indispensable for studying the TIML problem effectively.

In our quest for a more comprehensive performance evaluation, we have also relied on the MalScan [61] features that we extracted from the same applications in the MalNet dataset. MalScan considers the function call graph of Android apps as a social network to conduct a centrality analysis on the graph. We rely on the *concatenate* feature type of MalScan as it includes the concatenation of its four base features: degree, katz, closeness, and harmonic. In our novel multi-modal TIML approach, we utilize concatenated long vectors as input for MalScan. However, since the four adapted TIML approaches were originally designed to take only images as input, we follow previous work [60] that converted MalScan feature vectors to images, allowing us to use the same model architecture for MalNet images. Since MalScan feature vectors have a size of 87 944, we pad them with zeros and convert them to images of size (176, 176).

Since some approaches make use of exemplars to not forget the knowledge learned from previous tasks, we rely on the *Fixed Total Exemplars* technique as we explained in Section 2. Specifically, we set $M$ to 10 000, which means that the total number of exemplars we keep from previous tasks cannot exceed 10 000.

## 5.3 Empirical Settings

All of our experiments are conducted on MalNet and MalScan features, and our experimental configuration is derived from thorough and comprehensive experimentation. We set our dataset organization thresholds, i.e., `timeWindow`, `minNbrFamiliesInTask`, and `minNbrSamples`, to 4, 4, and 20 respectively, based on the constraints described in Section 4.1. Note that these parameters are the inputs of Algorithm 1, meaning that a time window can be 4 months or more, depending on the output of Algorithm 1 (which, as a reminder, can incrementally extend a time window if there are not enough malware samples and families - cf. Section 4.1). Moreover, these thresholds ensure that there are a minimum of 4 families per task and that each family has at least 20 samples.

After identifying the malware families in a given task, we systematically divide the samples of each family into training (80%) and test (20%). In the preliminary study phase, 70% of the training portion is used for actual training, while the remaining 10% is allocated as the validation set. This is done to fine-tune the hyperparameters and prevent overfitting. In the second phase, where we perform the final evaluation on the full dataset, we use the entire 80% training portion for training without reserving any validation set, as the hyperparameters have already been optimized. This structured division ensures robust training during the preliminary phase and maximizes the use of data during the final evaluation phase. Moreover, a portion (20%) is always reserved as a test set. This strategy enables the evaluation of TIML's performance on unseen samples, allowing us to assess the model's effectiveness in recognizing new, previously unseen malware instances within the same time frame.

The randomness in our evaluation arises from the use of random seeds. To mitigate potential biases and ensure robustness, we conduct each experiment five times with different seeds and report the average results. We provide statistical analysis in the Appendix. This method is in line with general practices in the literature and helps stabilize the evaluation metrics.

It is imperative to note that families comprising fewer than 20 samples within a specific task will be uniformly integrated into the test set and assigned the categorical label "unknown". Since these families are not included in the training, the malware classifier does not learn their characteristics. Consequently, we consider two evaluation scenarios: ❶ we test the malware classifier only on the families that have been learned during training, which implies removing the "unknown" families in this case. We refer to this scenario as *unknown families excluded*. ❷ we consider the "unknown" families as new emerging malware families since only a few of their samples are available. In this case, they are not well-established families and are treated as the other samples in the test. Assuming that the labels of all the samples in the test set are known, the malware classifier is expected to incorrectly classify these unknown samples into a known malware family. We refer to this scenario as *Unknown families included*.

Each adapted TIML approach employs ResNet-18 [26] as the foundational network architecture for the classifier, whereas the proposed multimodal TIML approach incorporates the newly designed simple backbone outlined in Section 4.3.2. We train the models using a *learning rate* of `1e-4` and a *number of epochs* equals to `100`. For the case of *SS-IL*, we trained it for `150` epochs as it used a lower *learning rate* of `1e-5` in the original setting. We set the *batch size* to `128` and the *weight decay* to `1e-4`. All the experiments were conducted on a single Tesla V-100 GPU with 32GB of memory on an NVIDIA DGX Station. For more details about our implementation, please refer to our replication package.

## 5.4 Evaluation Metrics

An approach that addresses the TIML problem is effective when it ensures that the amount of resources needed to achieve standard performance is limited. The goal of our evaluation is therefore to assess the extent to which TIML approaches can reduce resource utilization (i.e., training time and data storage) while achieving high performance (i.e., accuracy and forgetting score).

**The Average Training Time**: It measures the average duration required to train the model for each time step. It is defined as follows:

$$Avg.T.Time = \frac{1}{T}\sum_{t=1}^{T} d_t, \tag{12}$$

where $d_t$ indicates the duration of training at the current time step.

**The Maximum Data Storage**: To assess the data storage, we consider the Maximum Data Storage across all time steps, taking into account that storage space can be reused across different time steps. We formalize it as follows:

$$Max.D.Storage = \max_{t\in\{1,\ldots,T\}} (s_t + e_t), \tag{13}$$

where $s_t$ represents the storage space of training data and $e_t$ corresponds to the storage space of exemplar data at each time step.

We conduct the performance evaluation of the various approaches using Mean Accuracy and Average Forgetting metrics that are widely recognized in incremental learning research.

**Mean Accuracy**: It represents the cumulative average testing accuracy across all malware families encountered up to the current time step, and is mathematically defined as follows:

$$MeanAcc. = \frac{1}{T}\sum_{t=1}^{T} a_t, \tag{14}$$

where $a_t$ denotes testing accuracy of all seen malware families until the current time step $t$.

**Average Forgetting**: It serves as a metric to quantify the degree of catastrophic forgetting of previously acquired knowledge over time. Catastrophic forgetting is a phenomenon where a model, much like humans, loses previously acquired knowledge. This occurs when the model is trained on new data and consequently forgets what it had learned from older data. This is a crucial issue in incremental learning, as it limits the model's ability to adapt to new information without losing existing knowledge. Addressing catastrophic forgetting is essential for developing models that can effectively learn and evolve over time in dynamic environments. The average forgetting score is formalized as:

$$Avg.Forget. = \frac{1}{T-1}\sum_{t=2}^{T}\left(\frac{1}{t-1}\sum_{i=1}^{t-1}\max_{\tau\in\{1,\ldots,t-1\}}(a_{\tau,i}-a_{t,i})\right), \tag{15}$$

where $a_{\tau,i}$ is the testing accuracy on evaluation data of $\tau$-th time step.

## 6 EXPERIMENTAL RESULTS

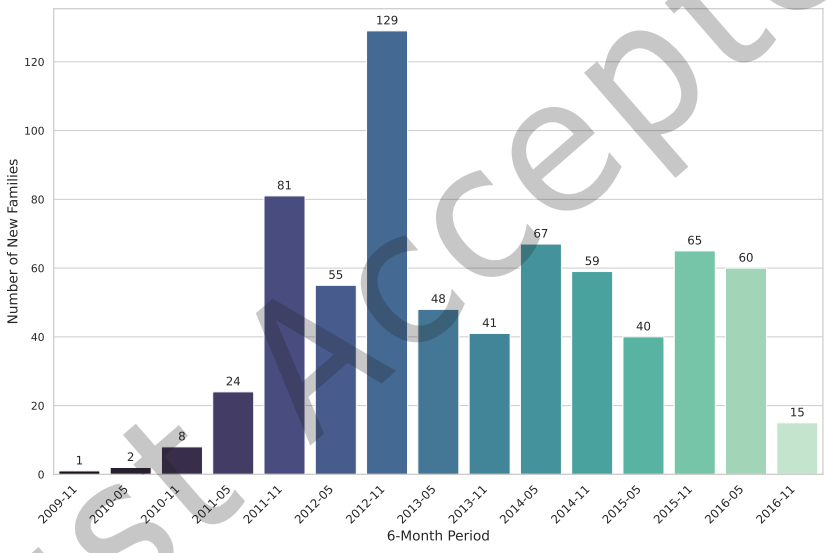


Fig. 4. Distribution of newly emerging malware families across 6-month time steps.

In this section, we present the evaluation results of the different approaches based on two malware features, i.e., bytecode image and graph vector. Explicitly, we name the features the same as the names of the approaches, i.e., MalNet [18] and MalScan [61]. This section also answers our research questions.

### 6.1 **RQ1** Is concept drift a significant factor affecting malware classification?

Incremental learning starts gaining attention and significance in malware classification due to the high cost of model retraining, which is required to tackle the concept drift of malware over time. Concept drift can affect the performance of a model that was trained on older data, making it less effective in identifying new or evolved malware samples. Therefore, with the first research question, we seek to rigorously investigate the presence and

impact of concept drift using the large-scale MalNet dataset that contains 1.2 million real-world malware. This analysis will support our primary motivation for the exploration of Temporal-Incremental Malware Learning.

We identified two primary forms of concept drift over time: ❶ the emergence of new malware families, and ❷ shifts in the distribution of old families. Specifically, we first obtained the emergence date from AndroZoo [4] for each malware instance. We then identified the date of the first malware sample within each family as the emergence date for that family. Using this method, we calculated the number of new malware families that emerged each half-year. As depicted in Figure 4, new malware families have been consistently emerging each year since 2009, which follows the inaugural year of Android. A noticeable increase in new families occurred between 2011 and 2012, which is consistent with the peak popularity of the Android platform. The emergence of new families forces the updating of existing classification models to maintain their performance. Concurrently, our analysis indicates that individual malware families exhibit longevity and persist for several years. Figure 5 illustrates the lifespan of the 20 largest malware families, based on the number of samples, providing valuable insights into their respective life cycles. This phenomenon of enduring presence is also observable in the vast majority of other malware families, raising an important question: Do the distributions of malware samples within existing families change over time? In other words, does a classifier's performance degrade when encountering new instances from families it was previously trained to recognize?

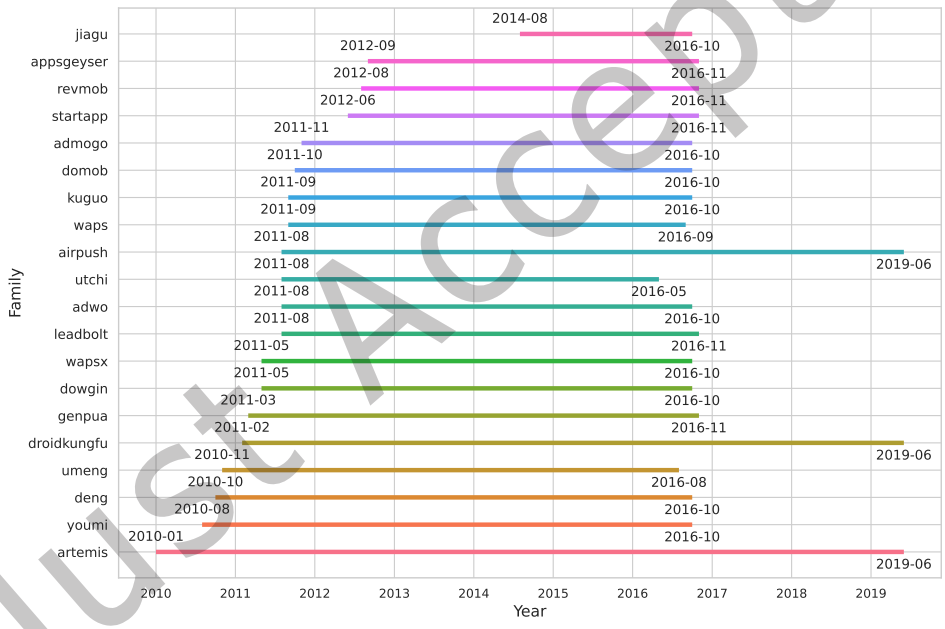


Fig. 5. Life span of the 20 largest malware families, showing their emergence and last-seen dates over time.

We conduct an experiment to examine the distributional shifts within malware families over time. To this end, we train a model on samples from eight malware families that appeared before 2012 and subsequently test it on samples from the same families but emerging between 2012 and 2017. This experimental setup allows us to directly observe the changes in malware characteristics over time. As depicted in Figure 6, we notice a consistent decline in mean accuracy when applying the model to newer samples, using two distinct input features—MalNet and MalScan. This decline demonstrates what we refer to as "temporal evolution" within malware families.

Specifically, the term describes how malware samples, even from the same families, exhibit shifts in their feature distributions over time, rendering models trained on earlier data less effective at recognizing newer variants. This phenomenon not only confirms the existence of temporal evolution but also underscores the necessity for our research into Temporal Incremental Malware Learning (TIML). Our TIML approaches are specifically designed to adapt to these temporal shifts, enabling the malware classification model to incrementally update itself and thereby reduce retraining costs while maintaining high levels of accuracy. Further elaboration on our methods and their implications in addressing these challenges is detailed in the answers to subsequent research questions.

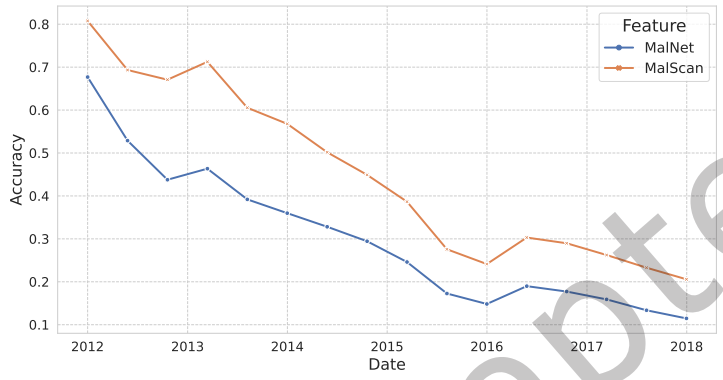


Fig. 6. Performance drop curve of models trained on malware families appeared before 2012 and evaluated on samples from the same families but appeared after 2012.

**RQ1 Answer:** Concept drift negatively impacts the predictive capabilities of malware classification models, confirming that it is a significant factor in malware family classification.

## 6.2 **RQ2** How well do TIML approaches perform in malware classification?

In this section, we first introduce a preliminary study that we conducted to validate our hypothesis that conventional CIL methods fall short within the TIML framework. This investigation underpins the imperative for developing tailored TIML strategies. Subsequently, we evaluate the performance of the four adapted TIML methods and our newly developed multimodal TIML approach.

Table 1. Comparison of accuracy between the adapted TIML approaches and their original CIL counterparts.

| Method | Adapted TIML Accuracy | CIL Accuracy |
|---|---|---|
| LwF | 49.68% | 27.99% |
| iCaRL | 58.15% | 23.13% |
| SS-IL | 54.58% | 21.65% |

*6.2.1 Preliminary Study.* In our preliminary investigation, we utilized the official small version of the MalNet dataset, which comprises 87 430 malware samples. This exploratory phase was crucial for assessing the viability of CIL methods within the TIML framework. Our findings are significant, highlighting marked improvements

in the performance of various TIML adaptations compared to their CIL counterparts. Specifically, as shown in Table 1, the adapted TIML version of LwF demonstrated an accuracy of 49.68%, markedly superior to its CIL version of 27.99%. Similarly, the adapted TIML version of iCaRL achieved an accuracy of 58.15%, a substantial improvement over its CIL version of 23.13%. Lastly, the adapted TIML version of the SS-IL method showed an accuracy of 54.58%, significantly outperforming its CIL version of 21.65%. These results underscore the potential benefits and necessity of developing specialized TIML approaches, as traditional CIL methods may not fully leverage the dynamics and challenges inherent in the TIML paradigm.

*6.2.2 Performance Analysis of TIML Approaches.* We first include three baseline TIML approaches. The Random Prediction model serves as our foundational baseline, where a family label is attributed to a malware sample based purely on randomness. Any approach surpassing the Random Prediction model in terms of accuracy indicates its capability to grasp the temporally incremental knowledge of evolving malware. Fine-tuning is a prevalent choice for model updates. In this method, only newly emerged data from the current time step is employed to fine-tune a model originally trained on older data. Given its narrow data scope within the TIML setting and the absence of tailored techniques for the TIML challenges, we consider it as a lower-bound approach. Conversely, the Full Retraining strategy makes use of the entire data seen up to the current time step, establishing itself as an upper-bound approach. The objective of the adapted TIML approaches and our new multimodal approach is to get the performance closer to Full Retraining while primarily relying only on the data from the current time step, similar to the Fine-tuning method, but potentially augmented with a minimal set of exemplar samples from previous data.

Table 2. Performance comparison of different approaches based on two input features: MalNet and MalScan.

| | *Unknown families excluded* | | | | *Unknown families included* | |
|---|---|---|---|---|---|---|
| | Mean Accuracy (%) | | Average Forgetting | | Mean Accuracy (%) | |
| Approach | MalNet | MalScan | MalNet | MalScan | MalNet | MalScan |
| Random Prediction | 1.72 | 1.72 | - | - | 1.28 | 1.28 |
| Fine-tuning * | 51.63 | 64.66 | 13.25 | 18.59 | 35.46 | 41.43 |
| LwF | 52.82 | 65.69 | 12.48 | 18.09 | 37.86 | 45.67 |
| SS-IL | 51.73 | 67.14 | 8.24 | 8.73 | 35.10 | 42.30 |
| iCaRL | 53.57 | 68.57 | 8.79 | 8.57 | 36.17 | 44.34 |
| LwF with Exemplars | 56.28 | 69.74 | 8.07 | 8.13 | 39.53 | 44.95 |
| Full Retraining * | 63.23 | 75.08 | - | - | 45.35 | 54.91 |
| MM-TIML | 70.53 | | 7.66 | | 45.08 | |

* Fine-tunning and Full Re-training are referred as Lower and Upper Bound, respectively.

Table 2 provides an overview of the performance comparison among the various approaches based on two input features (i.e., MalNet and MalScan), in which the Mean Accuracy is an average value across all time steps. Specifically, we report the results for the two scenarios we introduced in Section 5.3, which aim to evaluate the performance of the studied approaches in the absence and presence of "unknown" families.

From Table 2, we observe that Random Prediction has a very low performance which offers insight into the inherent complexity and difficulty of the TIML problem. TIML approaches strive to bridge the performance gap between Fine-tuning and Full Retraining, aiming to achieve performance levels closer to Full Retraining while utilizing a constrained amount of historical data as fine-tuning does. The results in Table 2 show that the adapted TIML approaches indeed report a good performance in classifying malware families. As we can see,

MalScan's graph vectors demonstrate greater effectiveness than MalNet images, thus our MM-TIML positions the lower bound and upper bound of MalScan as the performance boundaries. Especially, LwF with Exemplars stands out with the highest Mean Accuracy compared to other adapted TIML methods. Notably, our newly proposed multimodal TIML approach (i.e., MM-TIML) significantly outperforms all the single-feature-based methods. This success can be attributed to its innovative integration of diverse data modalities, which provides a more holistic understanding of malware characteristics. This trend is further confirmed by Figure 7, which illustrates the performance evolution of each approach across individual time steps. In the first time-steps, the performance of these TIML approaches was very close to the Upper Bound. In the last time-steps, we observe that the performance has slightly decreased for the TIML approaches compared to the Upper Bound, which is justifiable given that they do not have access to data from previous time-steps. Specifically, as depicted in Figure 7, the SS-IL exhibits subpar performance during the majority of the initial time steps. This is primarily due to the fundamental assumption underlying its original version, which assumes that the training samples of older malware families within each time step are solely derived from exemplars. However, this assumption does not align with the realities of the TIML scenario. Consequently, the adaptation process is unable to fully capitalize on the strengths of its original design.

Finally, we highlight that the performance of Temporal-Incremental Malware Learning (TIML) approaches should be understood within the specific context of incremental learning, which inherently utilizes significantly limited data for retraining the model. Comparing TIML performance to Full Retraining may initially suggest a drop in accuracy; however, such comparisons do not account for the operational constraints inherent to TIML. Full Retraining, representing an ideal upper bound, is often impractical due to data retention constraints such as privacy policies, data security, or storage limitations, especially in real-world settings where retaining comprehensive historical data is not feasible. In contrast, TIML approaches, requiring only current data increments, offer a more viable and often the only feasible option. Our evaluations demonstrate that TIML approaches not only adapt effectively within these constraints but also underscore the importance of this research direction, encouraging further investigation and advancements in this field.



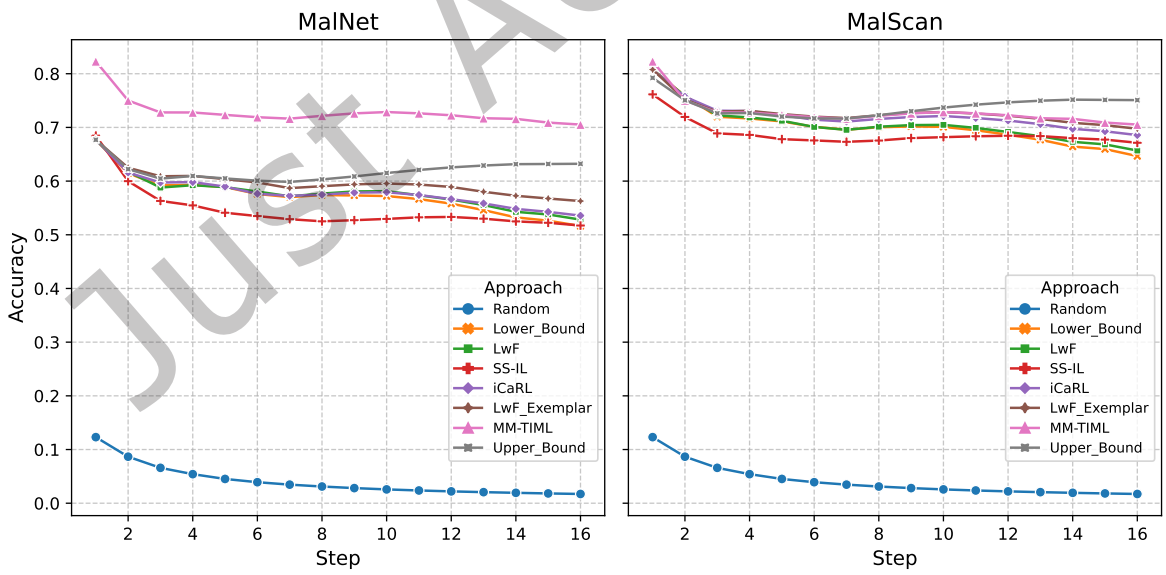


Fig. 7. Performance curves of various approaches across consecutive time steps.

**RQ2 Answer:** TIML approaches maintain strong performance in classifying malware families, despite a slight decrease compared to full retraining. This performance dip is primarily due to TIML's limited access to historical data, aligning with its design to operate effectively under practical constraints where extensive data retention is not feasible.

### 6.3 **RQ3** How resilient are TIML approaches to catastrophic forgetting?

In this section, we explore how the TIML approaches deal with catastrophic forgetting, specifically assessing how they retain or lose knowledge learned from previous instances over time. As outlined in Section 5.4, the Average Forgetting score is employed to estimate the magnitude of catastrophic forgetting. The overall average forgetting score for each approach across all time steps can be found in Table 2. We also report the detailed forgetting scores across individual time steps in Figure 8. For the case when "unknown" families are included, it is meaningless to calculate the forgetting score since its value is zero for the "unknown" families. In instances where the forgetting score is low, it is crucial to note that the model effectively retains knowledge from historical data. Conversely, a high forgetting score indicates the model's difficulty in recalling information from preceding time-steps. The zero forgetting score on "unknown" families is misleading as it suggests that the model does not forget the knowledge from the "unknown" families even though these families were not included in the training. Consequently, the forgetting scores are omitted when the "unknown" families are included.

It is worth mentioning that the Random Prediction method derives its results purely from randomness; without any data training, there is no acquired knowledge to forget. In contrast, the Full Retraining approach, considered as the upper bound, utilizes all historical data in every time step. This comprehensive data usage allows it to continually revisit previously learned knowledge, ensuring no information loss. As a result, measuring its degree of catastrophic forgetting is irrelevant. Therefore, the forgetting scores for these two baselines are omitted. We note that a low forgetting score does not necessarily imply a high Mean Accuracy. "Forgetting" primarily focuses on the performance related to old samples, overlooking the samples newly introduced in the current step, while accuracy evaluates the classifier's performance on all samples accumulated up to the current time step. Furthermore, forgetting score, being a relative metric underscores the difference between old and new accuracy but is unable to offer insights into the absolute accuracy levels.

Overall, the forgetting scores for the studied approaches are continuously increasing. We observe that the four approaches: LwF with Exemplar, iCaRL, SS-IL, and MM-TIML exhibit a decreasing trend in forgetting up to step 4. This is largely attributed to the exemplar set that encompasses nearly all historical data during these steps. We also observe that our proposed MM-TIML depicts the lowest forgetting scores compared to the other baselines. The Lower Bound, as expected, has the highest forgetting score as it does not employ any mechanism to retain previous knowledge.

**RQ3 Answer:** TIML approaches exhibit signs of forgetting, with our proposed MM-TIML demonstrating the strongest retention of previous knowledge.

### 6.4 **RQ4** How effectively do TIML approaches optimize resource utilization?

Having explored the evolution of malware over time and the performance of our TIML approaches in previous sections, we now delve into the resources saved by these methods. This analysis is pivotal for justifying our initial motivation and affirming the added value of these approaches. We focus on two primary metrics: computational efficiency, represented by training time, and maintenance cost, quantified by data storage space. We report their values in Table 3 and illustrate the evolution of these two metrics for each approach across time steps in Figure 9. Particularly, the storage curves of lower bound and LwF are overlapped as they only use the same newly added

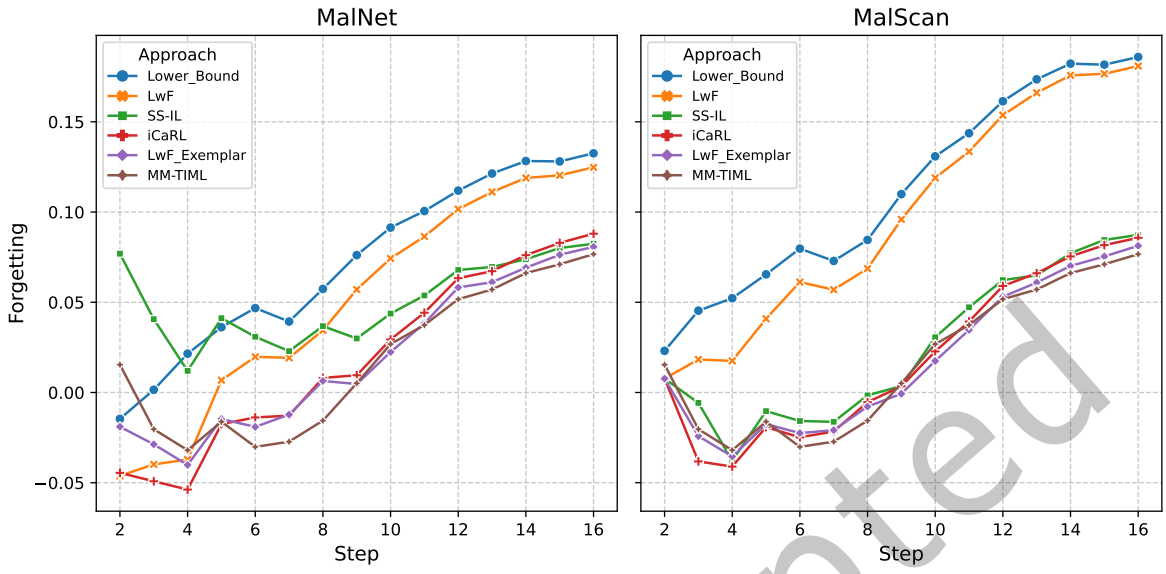


Fig. 8. Forgetting curves of various approaches across consecutive time steps.

data at each step; LwF with Exemplar, iCaRL, and SS-IL are overlapped as they use both the same new data and the same exemplars.

Table 3. Resource cost of different approaches based on two input features: MalNet and MalScan. Note: A. Time = Average Training Time, T. Time = Total Training Time, D. Storage = Maximum Data Storage.

| | A. Time (H) | | T. Time (H) | | D. Storage (G) | |
|---|---|---|---|---|---|---|
| Approach | MalNet | MalScan | MalNet | MalScan | MalNet | MalScan |
| Fine-tuning * | 11.80 | 4.99 | 188.80 | 79.84 | 10.78 | 7.32 |
| LwF | 11.44 | 5.90 | 183.04 | 94.40 | 10.78 | 7.32 |
| SS-IL | 20.63 | 11.95 | 330.08 | 191.20 | 13.04 | 8.09 |
| iCaRL | 14.84 | 7.05 | 237.44 | 112.80 | 13.04 | 8.09 |
| LwF with Exemplars | 12.91 | 8.85 | 206.56 | 141.60 | 13.04 | 8.09 |
| Full Re-training * | 67.52 | 27.59 | 1080.32 | 441.44 | 58.33 | 36.22 |
| MM-TIML | 12.63 | | 202.08 | | 21.13 | |

* Fine-tunning and Full Re-training are referred as Lower and Upper Bound, respectively.

Overall, the training time and the data storage of TIML approaches are significantly lower than the Upper Bound. For instance, in the case of MalNet features, the Upper Bound requires a total training time of 1080.32 (H) (i.e., 45 days) while MM-TIML only needs 202.08 (H) (i.e., 8.4 days) to finish the training process. Additionally, to store the training data, the Upper Bound requires maximum data storage of 58.33 (GB) compared to 21.13 (GB) for MM-TIML. As the MM-TIML incorporates both modalities, it shows a modest result in data storage requirements compared to the methods based on a single modality. Despite it is not the most efficient way for the storage requirement, it only consumes approximately 36.22% of what Upper Bound needs, which indicates that

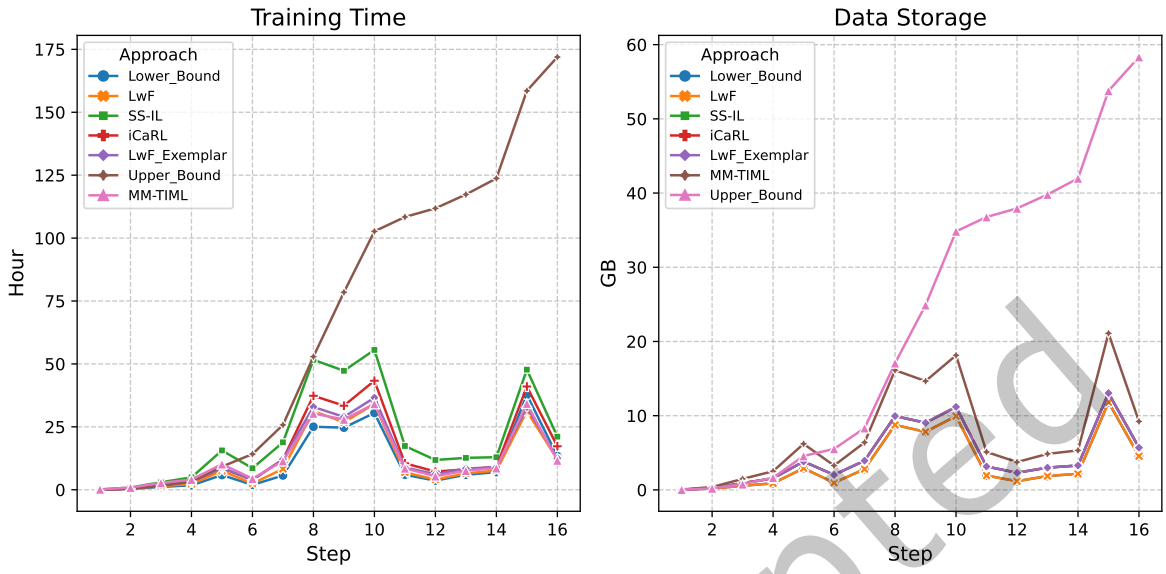


Fig. 9. Training time and data storage comparison of different approaches, based on MalNet.

our strategy balances enhanced malware learning capabilities with efficient use of storage resources. The results in Figure 9 also show how fast the training time and storage of the Upper Bound is growing compared to TIML approaches. This pattern suggests that the gap between the training time and storage required for the Upper Bound and TIML approaches will be larger and larger when updating the model in future time-steps.

**RQ4 Answer:** TIML approaches require significantly lower training times and data storage compared to the Upper Bound, with the observed trend indicating a widening gap over future model updates.

## 7 RELATED WORK

In this section, we delve into the foundational literature underpinning our study. We begin with a review of incremental learning, the central concept of our research. Subsequently, we explore the works focused on malware classification. Last, we present state-of-the-art works that have tried to integrate incremental learning techniques within the realm of malware classification.

### 7.1 Incremental Learning

In traditional deep learning [26, 28, 50, 51, 53, 65], precollected datasets are predominantly used. With the emergence of streaming data, there are inherent challenges such as storage constraints [23, 24], privacy issues [7, 11], and elevated re-training costs [13, 19]. These challenges underscore the necessity for the development of Incremental Learning (IL) strategies, which accommodate dynamic and evolving data sources while minimizing the storage and computational overhead. A central challenge in IL is ”catastrophic forgetting”, where training on only new data might overwrite previous knowledge.

Incremental learning can be broadly categorized into three types, as discussed in various literature sources [11, 66]: Class-Incremental Learning (CIL), Domain-Incremental Learning (DIL), and Task-Incremental Learning (TIL). Among these, CIL has been the most extensively explored type [3, 14, 31, 40, 41, 44]. As a revolutionary CIL method,

”Learning without Forgetting” (LwF) [37] addresses catastrophic forgetting through knowledge distillation, allowing for the integration of new classes into models while preserving previously acquired knowledge. This avoids the dependence on the original dataset and mitigates the cost of retraining. Following LwF, iCaRL [44] pioneered the trend towards exemplar-based methods, which soon gained widespread popularity. It introduced an exemplar-based methodology, employing nearest-mean-of-exemplars for classification and representation learning. Following on this direction, PODNet [14] introduced a distillation-based strategy that relies on spatial-based loss to curb representation forgetting. Meanwhile, SS-IL [3] identified that over-reliance on pure softmax can skew predictions in CIL. To address this, they introduced the Separated Softmax layer, which yielded significant improvements in performance. Recently, AFC [31] presented an innovative regularization technique for knowledge distillation. Their method prioritized the preservation of crucial features, ensuring minimal changes while assimilating new classes.

In the context of incremental learning, a ”task” refers to a distinct learning objective or problem, often characterized by a unique set of classes or data distributions. TIL shares similarities with CIL in terms of managing newly arriving classes in tasks. The key difference manifests during the inference phase: While CIL requires classification across all classes, TIL restricts classification to the classes within the pertinent task, eliminating the need for cross-task discrimination. This makes TIL a more streamlined version, often seen as a subset of CIL [11, 66]. On the other hand, DIL is tailored for situations characterized by concept drift or distribution changes [19, 39, 55]. In these cases, new tasks come from varied domains but retain a consistent label space. Some DIL strategies [16, 56, 63] take inspiration from CIL methodologies. Since our TIML approaches primarily derive from CIL methods, we opt not to delve further into the details of TIL and DIL in this section.

### 7.2 Malware Classification

A comprehensive understanding of malware classification is essential to advance the field of Android security, particularly through the application of Temporal-Incremental Malware Learning. This section explores the significant challenge of not only detecting but accurately categorizing malware into specific families, which is critical for deploying effective, targeted responses against various types of malware threats. In the field of Android security, malware detection and malware families are two critical challenges. Malware detection [5, 10, 21, 47–49] focuses on distinguishing malicious Android applications from benign ones. In contrast, malware family classification categorizes these detected malware based on shared characteristics or operational behaviors.

In this section, our spotlight is on malware family classification, given its pertinence to our work. Over the years, several strategies have emerged to classify Android malware into distinct families. DroidSieve [46] integrates static features such as Certificates and Permissions, utilizing the Extra Tree algorithm [22]. Andro-Simnet [32] delves into hybrid features like API call sequences coupled with Permissions and further enhances it through Social Network analysis. DroidLegacy [12], on the other hand, bases its classification on API call-based signatures. Recently, there has been a shift towards leveraging image representations of bytecode for malware family classification [15, 52]. As an illustrative example, sections of the DEX file have been transformed into images, enabling the extraction of nuanced texture and color features [15]. These features, combined with plain-text traits, are subsequently processed by a multiple kernel learning classifier. In the same direction, MalNet [18] database has been released, which is a collection of over 1.2 million malware images spread across 696 families, which indeed facilitates future research in the realm of malware classification.

### 7.3 Incremental Learning in Malware Classification

The application of incremental learning to malware classification remains underexplored, and no comprehensive studies have addressed this topic. A critical limitation in the existing related studies is their inability to emulate real-world malware evolution, given their simplified and unrealistic assumptions. For instance, some studies relied

on a standard Class-Incremetal Learning (CIL) assumption. Li et al. [35] devised a CIL methodology based on multi-class SVM [58], and tested its efficacy on a small dataset. Compared to TIML, this approach does not utilize deep learning models, which are crucial for effectively implementing knowledge distillation—a key component of incremental learning. Moreover, their evaluation was limited to small datasets, insufficient to fully explore the complexities and evolving nature of malware threats. Qiang et al. [42] introduced a few-shot learning approach under the CIL assumption, facilitating the dynamic adaptation of pre-trained models to previously unknown families with minimal samples. While their method addresses the challenge of learning from limited data, it does not account for the distribution shifts in malware feature vectors within the same malware families, which is a critical aspect of what TIML effectively manages. Renjith et al. [45] investigated incremental learning for on-device Android malware detection. However, their study was limited to binary classification (i.e., predicting whether a given app is malware) and their dataset was not organized in temporal order, failing to capture the real-world evolution of malware.

Rahman et al. [43], in their attempt to explore three different IL scenarios, faced bottlenecks in performance due to various limitations. For instance, their emulation of CIL involved a straightforward addition of new malware classes in successive steps, without taking into account the authentic lifecycle of malware in the real world. Furthermore, their reliance on basic ML features to train DL models on limited datasets (in the order of thousands) inevitably hindered the capabilities of the DL models and the potential of IL techniques. Recently, Chen et al. [8] delved into continuous learning applied to malware detection, addressing the concept drift issue prevalent in Android malware classifiers. They focus on reducing human efforts in the process of labeling emerging malware by selecting representative samples with their techniques. However, they still need to retrain the classifier on the full historical dataset. In contrast, TIML aims to retrain the classifier using only new malware without a historical dataset to reduce the training and storage resources.

## 8 CONCLUSION

The continuous evolution of Android malware makes the development of effective classification models crucial. After training, malware classifiers must be regularly updated to address concept drift. However, this updating process is resource-intensive, requiring significant storage and time. In this work, we introduced Temporal-Incremental Malware Learning (TIML) and corresponding approaches for incrementally updating malware classifiers. These methods effectively classify emerging malware families while adapting to the evolving data distributions of existing families. TIML approaches primarily leverage current data samples to update classifiers, reducing resource consumption. We adapted CIL approaches within the TIML framework and proposed a novel multimodal TIML approach to further enhance classification performance. Our results demonstrate that training classifiers under TIML achieve promising performance while significantly lowering time and storage requirements.

Additionally, TIML's potential extends beyond the Android platform. Its effectiveness in other contexts depends on access to large-scale, chronologically ordered datasets and the development of representation learning techniques suited to the specific software characteristics of each platform.

## ACKNOWLEDGMENTS

This research was funded in whole or in part by the Luxembourg National Research Fund (FNR), grant references 16344458 (REPROCESS), 18154263 (UNLOCK) and 17046335 (AFR PhD grant).

## REFERENCES

[1] Ikram B Abdel Ouahab, Mohammed Bouhorma, Lotfi El Aachak, and Anouar Abdelhakim Boudhir. 2022. Towards a new cyberdefense generation: proposition of an intelligent cybersecurity framework for malware attacks. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)* 15, 8 (2022), 1026–1042.

[2] Saket Acharya, Umashankar Rawat, and Roheet Bhatnagar. 2022. A low computational cost method for mobile malware detection using transfer learning and familial classification using topic modelling. *Applied Computational Intelligence and Soft Computing* 2022 (2022), 1–22.

[3] Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. 2021. Ss-il: Separated softmax for incremental learning. In *Proceedings of the IEEE/CVF International conference on computer vision*. 844–853.

[4] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508

[5] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

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

[7] Mahawaga Arachchige Pathum Chamikara, Peter Bertók, Dongxi Liu, Seyit Camtepe, and Ibrahim Khalil. 2018. Efficient data perturbation for privacy preserving and accurate data stream mining. *Pervasive and Mobile Computing* 48 (2018), 1–19.

[8] Yizheng Chen, Zhoujie Ding, and David Wagner. 2023. Continuous learning for android malware detection. In *32nd USENIX Security Symposium (USENIX Security 23)*. 1127–1144.

[9] Lei Cui, Junnan Yin, Jiancong Cui, Yuede Ji, Peng Liu, Zhiyu Hao, and Xiaochun Yun. 2024. API2Vec++: Boosting API Sequence Representation for Malware Detection and Classification. *IEEE Transactions on Software Engineering* (2024).

[10] Nadia Daoudi, Jordan Samhi, Abdoul Kader Kabore, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2021. Dexray: a simple, yet effective deep learning approach to android malware detection based on image representation of bytecode. In *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2*. Springer, 81–106.

[11] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence* 44, 7 (2021), 3366–3385.

[12] Luke Deshotels, Vivek Notani, and Arun Lakhotia. 2014. Droidlegacy: Automated familial classification of android malware. In *Proceedings of ACM SIGPLAN on program protection and reverse engineering workshop 2014*. 1–12.

[13] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. 2015. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine* 10, 4 (2015), 12–25.

[14] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. 2020. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*. Springer, 86–102.

[15] Yong Fang, Yangchen Gao, FAN Jing, and LEI Zhang. 2020. Android malware familial classification based on dex file section features. *IEEE Access* 8 (2020), 10614–10627.

[16] Enrico Fini, Victor G Turrisi Da Costa, Xavier Alameda-Pineda, Elisa Ricci, Karteek Alahari, and Julien Mairal. 2022. Self-supervised models are continual learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9621–9630.

[17] James B Fraley and James Cannady. 2017. The promise of machine learning in cybersecurity. In *SoutheastCon 2017*. IEEE, 1–6.

[18] Scott Freitas, Rahul Duggal, and Duen Horng Chau. 2022. MalNet: A large-scale image database of malicious software. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3948–3952.

[19] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

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

[21] Joshua Garcia, Mahmoud Hammad, and Sam Malek. 2018. Lightweight, obfuscation-resilient detection and family identification of android malware. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 3 (2018), 1–29.

[22] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine learning* 63 (2006), 3–42.

[23] Lukasz Golab and M Tamer Özsu. 2003. Issues in data stream management. *ACM Sigmod Record* 32, 2 (2003), 5–14.

[24] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 1–36.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

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

[27] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. 2019. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 831–839.

[28] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Q Weinberger. 2019. Convolutional networks with dense connectivity. *IEEE transactions on pattern analysis and machine intelligence* 44, 12 (2019), 8704–8716.

[29] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F. Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. 2017. Euphony: Harmonious Unification of Cacophonous Anti-Virus Vendor Labels for Android Malware. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 425–435. https://doi.org/10.1109/MSR.2017.57

[30] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *26th USENIX security symposium (USENIX security 17)*. 625–642.

[31] Minsoo Kang, Jaeyoo Park, and Bohyung Han. 2022. Class-incremental learning by knowledge distillation with adaptive feature consolidation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16071–16080.

[32] Hye Min Kim, Hyun Min Song, Jae Woo Seo, and Huy Kang Kim. 2018. Andro-simnet: Android malware family classification using social network analysis. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 1–8.

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

[34] Haodong Li, Guosheng Xu, Liu Wang, Xusheng Xiao, Xiapu Luo, Guoai Xu, and Haoyu Wang. 2024. MalCertain: Enhancing Deep Neural Network Based Android Malware Detection by Tackling Prediction Uncertainty. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[35] Jingmei Li, Di Xue, Weifei Wu, and Jiaxiang Wang. 2020. Incremental learning for malware classification in small datasets. *Security and Communication Networks* 2020 (2020), 1–12.

[36] Yao Li, Dawei Yuan, Tao Zhang, Haipeng Cai, David Lo, Cuiyun Gao, Xiapu Luo, and He Jiang. 2024. Meta-Learning for Multi-Family Android Malware Classification. *ACM Transactions on Software Engineering and Methodology* (2024).

[37] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence* 40, 12 (2017), 2935–2947.

[38] Zhijie Liu, Liang Feng Zhang, and Yutian Tang. 2023. Enhancing Malware Detection for Android Apps: Detecting Fine-Granularity Malicious Components. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1212–1224.

[39] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering* 31, 12 (2018), 2346–2363.

[40] Shentong Mo, Weiguo Pian, and Yapeng Tian. 2023. Class-Incremental Grouping Network for Continual Audio-Visual Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

[41] Weiguo Pian, Shentong Mo, Yunhui Guo, and Yapeng Tian. 2023. Audio-Visual Class-Incremental Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

[42] Qian Qiang, Mian Cheng, Yang Hu, Yuan Zhou, Jiawei Sun, Yu Ding, Zisen Qi, and Fei Jiao. 2022. An Incremental Malware Classification Approach Based on Few-Shot Learning. In *ICC 2022-IEEE International Conference on Communications*. IEEE, 2682–2687.

[43] Mohammad Saidur Rahman, Scott Coull, and Matthew Wright. 2022. On the Limitations of Continual Learning for Malware Classification. In *Conference on Lifelong Learning Agents*. PMLR, 564–582.

[44] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2001–2010.

[45] G Renjith and S Aji. 2022. On-device Resilient Android Malware Detection using Incremental Learning. *Procedia Computer Science* 215 (2022), 929–936.

[46] Guillermo Suarez-Tangil, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. 2017. Droidsieve: Fast and accurate classification of obfuscated android malware. In *Proceedings of the seventh ACM on conference on data and application security and privacy*. 309–320.

[47] Tiezhu Sun, Kevin Allix, Kisub Kim, Xin Zhou, Dongsun Kim, David Lo, Tegawendé F Bissyandé, and Jacques Klein. 2023. DexBERT: Effective, Task-Agnostic and Fine-grained Representation Learning of Android Bytecode. *IEEE Transactions on Software Engineering* (2023).

[48] Tiezhu Sun, Nadia Daoudi, Kevin Allix, and Tegawendé F Bissyandé. 2021. Android malware detection: looking beyond dalvik bytecode. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 34–39.

[49] Tiezhu Sun, Nadia Daoudi, Kisub Kim, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2024. DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware. *arXiv preprint arXiv:2408.16353* (2024).

[50] Tiezhu Sun, Weiguo Pian, Nadia Daoudi, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2024. LaFiCMIL: Rethinking Large File Classification from the Perspective of Correlated Multiple Instance Learning. In *International Conference on Applications of Natural Language to Information Systems*. Springer, 62–77.

[51] Tiezhu Sun, Wei Zhang, Zhijie Wang, Lin Ma, and Zequn Jie. 2018. Image-level to Pixel-wise Labeling: From Theory to Practice.. In *IJCAI*. 928–934.

[52] Yuxia Sun, Yanjia Chen, Yuchang Pan, and Lingyu Wu. 2019. Android malware family classification based on deep learning of code images. *IAENG International Journal of Computer Science* 46, 4 (2019), 524–533.

[53] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. 2020. FCOS: A simple and strong anchor-free object detector. *IEEE transactions on pattern analysis and machine intelligence* 44, 4 (2020), 1922–1933.

[54] VirusTotal. . https://www.virustotal.com. [Online; accessed 21-September-2023].

[55] Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. 2022. S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning. *Advances in Neural Information Processing Systems* 35 (2022), 5682–5695.
[56] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. 2022. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 139–149.
[57] Max Welling. 2009. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 1121–1128.
[58] Jason Weston, Chris Watkins, et al. 1999. Support vector machines for multi-class pattern recognition.. In *Esann*, Vol. 99. 219–224.
[59] Bozhi Wu, Sen Chen, Cuiyun Gao, Lingling Fan, Yang Liu, Weiping Wen, and Michael R Lyu. 2021. Why an android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 2 (2021), 1–29.
[60] Yueming Wu, Shihan Dou, Deqing Zou, Wei Yang, Weizhong Qiang, and Hai Jin. 2022. Contrastive Learning for Robust Android Malware Familial Classification. *IEEE Transactions on Dependable and Secure Computing* (2022).
[61] Y. Wu, X. Li, D. Zou, W. Yang, X. Zhang, and H. Jin. 2019. MalScan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 139–150. https://doi.org/10.1109/ASE.2019.00023
[62] Guoqing Xiao, Jingning Li, Yuedan Chen, and Kenli Li. 2020. MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *J. Parallel and Distrib. Comput.* 141 (2020), 49–58.
[63] Jiangwei Xie, Shipeng Yan, and Xuming He. 2022. General incremental learning with domain-aware categorical representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14351–14360.
[64] Senming Yan, Jing Ren, Wei Wang, Limin Sun, Wei Zhang, and Quan Yu. 2022. A Survey of Adversarial Attack and Defense Methods for Malware Classification in Cyber Security. *IEEE Communications Surveys & Tutorials* (2022).
[65] Mingxin Zhang, Zhijie Wang, Tiezhu Sun, and Xiaolei Li. 2019. Salient object detection by pyramid networks with gating. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 1791–1796.
[66] Da-Wei Zhou, Qi-Wei Wang, Zhi-Hong Qi, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. 2023. Deep class-incremental learning: A survey. *arXiv preprint arXiv:2302.03648* (2023).

# APPENDIX

## Hyper-parameter Sensitivity

Table 4. Summary of experimental results displaying the impact of varying mini batch sizes, learning rates, and weight decay settings on mean accuracy of model LwF.

| Exp Index | Mini Batch Size | Learning Rate | Weight Decay | Mean Accuracy (%) |
|---|---|---|---|---|
| 0 | 128 | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ | 49.68 |
| 1 | 128 | $1 \times 10^{-4}$ | $1 \times 10^{-3}$ | 47.18 |
| 2 | 128 | $1 \times 10^{-3}$ | $1 \times 10^{-4}$ | 42.76 |
| 3 | 128 | $1 \times 10^{-5}$ | $1 \times 10^{-4}$ | 48.22 |
| 4 | 256 | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ | 48.38 |

During the preliminary study presented in Section 6.2.1, we conducted a comprehensive analysis of various hyper-parameters using the same dataset to ascertain their optimal combination for the models' performance. In this section, we provide detailed empirical experimental results and our findings from these investigations.

We initially used the Learning without Forgetting (LwF) to establish three fundamental hyper-parameters: mini-batch size, learning rate, and weight decay. As all the baseline approaches utilize the same backbone architecture (i.e., ResNet-18) and operate within the same incremental learning paradigm, these parameters were determined as the common optimal choices for other baseline approaches as well. However, due to the special strategy employed by SS-IL regarding the use of exemplars, we conducted additional experiments specifically for this approach to identify the most suitable hyper-parameters tailored to its requirements. Tables 4 and 5

Table 5. Summary of experimental results displaying the impact of varying mini batch sizes, learning rates, and weight decay settings on mean accuracy of model SS-IL.

| Exp Index | Mini Batch Size | Learning Rate | Weight Decay | Mean Accuracy (%) |
|---|---|---|---|---|
| 5 | 128 | $1 \times 10^{-5}$ | $1 \times 10^{-4}$ | 54.58 |
| 6 | 128 | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ | 38.81 |
| 7 | 128 | $1 \times 10^{-3}$ | $1 \times 10^{-4}$ | 26.67 |
| 8 | 128 | $1 \times 10^{-5}$ | $1 \times 10^{-3}$ | 54.70 |

summarize the impact of varying mini-batch sizes, learning rates, and weight decay settings on the mean accuracy for the LwF and SS-IL models. These experiments highlight the sensitivity of these models to different parameter configurations. Specifically, the LwF model demonstrates relatively stable performance across various settings, whereas the SS-IL model exhibits a pronounced preference for lower learning rates, underlining its particular sensitivity to this parameter.

Table 6. Comparison of mean accuracy across different values of lambda for LwF with Exemplar and iCaRL methods.

| LwF with Exemplar | | | iCaRL | | |
|---|---|---|---|---|---|
| Exp Index | Lambda | Mean Accuracy (%) | Exp Index | Lambda | Mean Accuracy (%) |
| 9 | 0.05 | 59.61 | 13 | 0.05 | 58.68 |
| 10 | 0.1 | 60.05 | 14 | 0.1 | 58.76 |
| 11 | 0.5 | 59.10 | 15 | 0.5 | 57.93 |
| 12 | 1 | 58.48 | 16 | 1 | 58.15 |

Table 7. Comparison of mean accuracy across different values of lambda for SS-IL and MM-TIML methods.

| SS-IL | | | MM-TIML | | | |
|---|---|---|---|---|---|---|
| Exp Index | Lambda | Mean Accuracy (%) | Exp Index | Lambda 0 | Lambda 1 | Mean Accuracy (%) |
| 17 | 0.05 | 55.19 | 21 | 0.1 | 0.1 | 66.32 |
| 18 | 0.1 | 55.18 | 22 | 0.5 | 0.1 | 65.86 |
| 19 | 0.5 | 55.72 | 23 | 0.1 | 0.5 | 66.01 |
| 20 | 1 | 54.58 | 24 | 0.5 | 0.5 | 65.94 |

Following the establishment of the three fundamental hyper-parameters, we engaged in further comparative experiments to fine-tune the parameter $\lambda$ for each approach. $\lambda$ refers to the trade-off weight in knowledge distillation techniques, impacting how previous knowledge is preserved while learning new information, which is critical in their respective loss functions. Note that the Fine-tuning and Full Retraining baselines do not include this parameter, as they do not utilize the knowledge distillation technique. Table 6 and 7 illustrate the impact of the parameter $\lambda$ on the performance of different approaches. The results indicate that both LwF and iCaRL maintain relatively stable performance across a range of $\lambda$ values, as shown in Table 6. Furthermore, as depicted in Table 7, the SS-IL and MM-TIML approaches demonstrate consistent performance across different settings of $\lambda$, with SS-IL particularly showing stable outcomes even with significant variations in $\lambda$. This consistency demonstrates the robustness of the $\lambda$ parameterization of these models and particularly highlights the adaptability of SS-IL to different settings of the knowledge distillation parameter, in stark contrast to its sensitivity to the learning rate.

To conclude, while subtle changes in parameters resulted in corresponding shifts in performance, the overall performance of most of our TIML approaches remained relatively stable across different configurations. We selected the optimal parameter combination for the final experiments conducted on the full dataset to ensure maximum efficacy. It is important to note that the scope of the empirical experiments conducted was far more extensive than what is detailed here. Only the most critical experiments that significantly influenced our understanding and development of the TIML framework are reported in this appendix.

## Statistical Analysis

Following the hyper-parameter sensitivity investigation in the previous section, we perform a statistical analysis of our MM-TIML approach and three key baseline approaches using the same dataset and features under varying random seed conditions. This analysis uses the optimal hyper-parameter settings determined in the previous section and evaluates the methods across five different random seeds to ensure robustness and consistency in performance.

Table 8. Summary of experimental results displaying mean accuracy (%) across different methods and random seeds.

| Methods | Seed 0 | Seed 42 | Seed 123 | Seed 999 | Seed 2023 |
|---|---|---|---|---|---|
| LwF with Exemplar | 59.58 | 60.05 | 59.13 | 58.85 | 59.42 |
| iCaRL | 57.96 | 58.76 | 58.24 | 57.39 | 58.33 |
| SS-IL | 55.74 | 55.72 | 55.61 | 55.58 | 55.94 |
| MM-TIML | 65.88 | 66.32 | 65.31 | 66.12 | 66.05 |

Table 9. Mean accuracy (%) and standard deviation across different methods over five random seeds.

| Methods | Mean Accuracy (%) | Std Dev (%) |
|---|---|---|
| LwF with Exemplar | 59.41 | 0.46 |
| iCaRL | 58.14 | 0.51 |
| SS-IL | 55.72 | 0.14 |
| MM-TIML | 65.94 | 0.38 |

Table 10. Paired t-Test results between MM-TIML and baseline methods. The results display the differences per seed, mean difference, standard deviation of differences, t-statistic, degrees of freedom (df), and p-value.

| Comparison | Differences per Seed | Mean Difference (%) | Std Dev (%) | t-Statistic | df | p-Value |
|---|---|---|---|---|---|---|
| MM-TIML vs. LwF-w-E | [6.30, 6.27, 6.18, 7.27, 6.63] | 6.53 | 0.45 | 32.68 | 4 | < 0.0001 |
| MM-TIML vs. iCaRL | [7.92, 7.56, 7.07, 8.73, 7.72] | 7.80 | 0.61 | 28.69 | 4 | < 0.0001 |
| MM-TIML vs. SS-IL | [10.14, 10.60, 9.70, 10.54, 10.11] | 10.22 | 0.37 | 62.43 | 4 | < 0.0001 |

Table 8 shows the mean accuracy of the four methods for each random seed. The results demonstrate that MM-TIML consistently outperforms the other approaches across all five seeds. LwF with Exemplar, iCaRL, and SS-IL, on the other hand, exhibit lower mean accuracies, with SS-IL achieving the lowest values. The variability in performance across different seeds is relatively small, suggesting stable model behavior. Table 9 summarizes the overall mean accuracy and standard deviation of the methods across the five random seeds. MM-TIML continues

to exhibit the highest mean accuracy. The standard deviations are low across all models, indicating that the methods are relatively stable across different seeds. Table 10 presents the paired t-test results between MM-TIML and the baseline approaches (LwF with Exemplar, iCaRL, and SS-IL). The mean difference per seed is calculated along with the standard deviation, t-statistic, degrees of freedom, and p-values.

- The comparison between MM-TIML and LwF with Exemplar shows a mean difference of 6.53% in favor of MM-TIML, with a t-statistic of 32.68 and a highly significant p-value of less than 0.0001.
- For MM-TIML vs. iCaRL, the mean difference is 7.80%, with a t-statistic of 28.69, again demonstrating a highly significant result (p-value $< 0.0001$).
- Finally, MM-TIML vs. SS-IL yields the largest mean difference of 10.22%, with a t-statistic of 62.43 and a p-value less than 0.0001, strongly indicating that MM-TIML significantly outperforms SS-IL.

The statistical analysis confirms that MM-TIML significantly outperforms the other methods in terms of mean accuracy across different random seeds. The results are consistent and robust, as indicated by the low standard deviations and significant t-test results. The performance improvements of MM-TIML over the baseline methods are statistically significant, reinforcing the effectiveness of our proposed approach.