algorithms

MDPI

Article

ProtoMal: Prototype-Guided Dual-Branch Continual Learning for Robust Android Malware Detection

Xuan Zhang [1], Aihua Zhang [1,*], Maode Ma [2], Yuanjie Bo [1], Yiying Zhang [1] and Yanan Zhang [1]

[1] Department of Artificial Intelligence, Tianjin University of Science and Technology, Tianjin 300457, China; 15122135122@163.com (X.Z.); 19862516948@163.com (Y.B.); yiyingzhang@tust.edu.cn (Y.Z.); ynzhang@tust.edu.cn (Y.Z.)

[2] Faculty of Computer Science and Artificial Intelligence, Shenzhen University of Advanced Technology, Shenzhen 518055, China; mamaode@suat-sz.edu.cn

* Correspondence: zah@tust.edu.cn

Abstract

Traditional Android malware detection systems struggle to adapt to evolving threats without sacrificing performance on legacy families. To address this, we present ProtoMal, a dual-branch continual learning framework that achieves a fine-grained balance between stability and plasticity. The framework utilizes a frozen old branch for knowledge preservation and a trainable new branch for novel threat acquisition. A key contribution is our robust median-based prototype learning mechanism, which leverages centroids and outlier filtering to handle the high intra-class variability and label noise inherent in malware datasets. Experimental results across three large-scale benchmarks AMD, VirusShare, and VirusShareYears demonstrate that ProtoMal significantly curtails performance degradation and achieves highly competitive average accuracy. Most notably, the proposed framework demonstrates highly competitive model stability and yields robust anti-forgetting capabilities alongside current state-of-the-art incremental learning paradigms, maintaining particular resilience under severe concept drift.

Keywords: malware; class-incremental learning; deep learning

1. Introduction

Malicious software, also known as malware [1], typically interrupts, damages computer systems, or accesses private systems without authorization. Cybercriminals use it to infect devices to steal data, gain access to banking credentials or personal information, or even extort money from victims [2]. Meanwhile, to evade detection and analysis, malicious software often employs code obfuscation techniques, which alter the structure of the code while retaining its original functionality.

Current Android malware detection methods, whether based on static or dynamic analysis, typically rely on malware samples collected within a specific timeframe for training, thus constructing static classification models. Once deployed, the parameters and decision boundaries of these models often remain fixed, lacking continuous learning and update mechanisms. Consequently, when confronted with continuously evolving new malware families or variants of existing ones, the original classifier may experience degraded performance or even misclassification due to its inability to learn newly emerging features or behavioral patterns.

Therefore, security applications require continuous learning capabilities rather than relying on a single, fixed classification model. However, during continuous learning, as

Academic Editor: Carolina Del-Valle-Sot

Received: 13 April 2026
Revised: 19 May 2026
Accepted: 26 May 2026
Published: 4 June 2026

new malware categories constantly emerge, the model gradually adapts to new family features, subsequently leading to catastrophic forgetting [3].

Class-incremental learning (CIL), a representative paradigm of continual learning, is widely applied to address the challenge posed by the continuous emergence of malware families [4]. In the context of combating malware challenges, CIL has established itself as a pivotal and frequently employed technique within contemporary research. The fundamental strength of this approach is its capacity to harness already trained base-class models as a foundation, thereby facilitating a swift expansion of recognition capabilities for emergent malware families through a limited number of iterations. This, in turn, leads to a substantial optimization of learning efficiency and a reduction in computational resource expenditure. Crucially, this methodology proficiently addresses the inherent dilemma of models needing to continuously adapt to novel threats while simultaneously preserving previously acquired knowledge.

To address this challenge, this paper introduces ProtoMal, a prototype-guided dual-branch class-incremental learning framework, specifically designed to mitigate catastrophic forgetting in continuous malware detection. The name "ProtoMal" reflects its core design principle: leveraging robust class prototypes to guide the model in learning malware patterns incrementally, ensuring both the retention of previously learned families and the rapid adaptation to emerging threats. The method constructs a dual-path architecture consisting of a frozen old branch and a trainable new branch, and employs a weighted fusion strategy to dynamically balance the contributions of old and new knowledge. Concurrently, it introduces centroid-based prototype learning to replace the traditional mean-based approach, thereby enhancing classifier robustness by eliminating outlier samples. The main contributions are summarized as follows:

- Weighted Fusion Strategy: By introducing an adjustable parameter $\alpha \in [0,1]$, this strategy dynamically controls the feature contribution of the frozen old branch and the trainable new branch, achieving a flexible balance between stability and plasticity. Empirical analysis demonstrates that setting $\alpha = 0.05$ provides an optimal trade-off: it effectively minimizes performance degradation while maintaining high adaptability to newly introduced malware families, significantly reducing parameter overhead compared to standard architectural expansion strategies.
- Centroid-based Prototype Learning: Before calculating class prototypes, this method identifies and filters out the 10% of samples furthest from the median, effectively suppressing the negative impact of outliers and noisy labels. Experiments demonstrate that under 10% label noise, the centroid method reduces the accuracy drop from 8.2% to 2.1% and the standard deviation from 2.7% to 0.8%, significantly improving robustness and stability.
- Comprehensive experiments were conducted on the AMD, VirusShare, and VirusShareYears datasets to validate the effectiveness of the proposed method. Compared with existing state-of-the-art methods, ProtoMal achieves highly competitive average accuracy while demonstrating superior anti-forgetting capabilities and robust cross-session stability, providing a resilient and reliable technical solution for practical, dynamic malware detection.

2. Related Work

2.1. Traditional Malware Detection

Malware detection classifies files as malicious (malware) or benign using static analysis to examine code, dynamic analysis to observe execution behavior, and hybrid analysis to combine both methods for enhanced accuracy [5].

Static analysis examines code without execution. To address the challenges of high-dimensional feature spaces and detection latency in real-time malware URL identification, a novel bio-inspired optimization framework synergizes Harris Hawks Optimization (HHO) and the Bat Algorithm (BA) for feature selection using a dual union-intersection mechanism, while integrating Grid Search-tuned tree-based classifiers to balance maximum detection accuracy and computational efficiency for adaptable cybersecurity deployment [6]. The MI-GAN framework [7] utilizes Generative Adversarial Networks (GANs) to synthesize high-quality malware images. This effectively mitigates dataset class imbalance and achieves high-precision classification of malware families. To avoid feature oversight vulnerabilities in Android malware detection, a novel framework comprehensively evaluates application permissions using an expansive multi-metric ensemble, while deploying a statistical ranking fusion mechanism that synergizes diverse feature selection methodologies to capture high-impact attributes [8].

Conversely, dynamic analysis observes the execution behavior of malware. The Dynamic Evolution Graph Convolutional Network (DEGCN) [9] utilizes multi-scale API graph sequences to model the API-level and global graph-level temporal correlations of software behavior. This effectively captures dynamic evolution patterns through the integration of a Graph Encoding-based Gated Recurrent Unit (GGRU).

Hybrid analysis, integrating static and dynamic features, has become a significant direction for more precisely identifying highly variable malware. This method allows for a more comprehensive characterization of malware attributes. For instance, the FGL_Droid method [10] achieves high accuracy in Android malware detection by converting lengthy dynamic API call sequences into function call graphs that preserve execution order information, and by fusing permission features. By employing a selective strategy to filter crucial features, a hybrid framework [11] constructs an improved HHO-driven neural network that synergizes static and dynamic behavioral traits to effectively enhance malware identification and family categorization.

Despite the partial effectiveness of traditional malware analysis methods, the evolving nature of malware means that fixed models are markedly inadequate in recognizing new sample categories, resulting in a degradation of their detection capacity.

2.2. Continuous Evolution of Malware Detection

Earlier studies have highlighted the critical importance of evaluating malware excluded from the model training process.

As a foundational step in drift detection, the Transcend framework [12] actively detects concept drift in deployed malware classification models via a p-value-based statistical evaluation. It identifies early signs of model degradation without ground truth labels, filters unreliable predictions, and thus significantly boosts classification system reliability in dynamic threat environments.

Building upon the premise of autonomous adaptation, DroidEvolver [13] signifying autonomous evolvement in Android ecosystems, counters continuous concept drift by combining online learning and dynamic feature sets with a pseudo-labeling mechanism to eliminate manual retraining; its robust successor, DroidEvolver++ [14], with the '++' suffix denoting enhanced reliability, overcomes the subsequent self-poisoning problem of defective pseudo-labels by integrating strict quality control to prevent catastrophic performance degradation.

To further accelerate this adaptive process, a novel framework termed Optimized Malware Detection through Real-Time and Adaptive Security (OMD-RAS) [15] extracts comprehensive malware behaviors through combined static and dynamic analyses for optimization, while employing a fast-training Extreme Learning Machine integrated with a

continuous learning mechanism to enable swift real-time detection and dynamically adapt to new malware variants with exceptional precision.

However, continuous adaptation inherently introduces the challenge of catastrophic forgetting. To mitigate this, the proposed Generative Replay-based constraint ongoing adaptability [16] counters catastrophic forgetting amid rapidly evolving malware threats by integrating Generative Adversarial Networks (GANs) equipped with a feature-matching loss and an innovative hidden-representation-based sample selection scheme to seamlessly retrain the primary model. Addressing this challenge from complementary algorithmic perspectives, the Self-Paced Class-Incremental Learning (SPCIL) method [17], denoting its capability to autonomously regulate the learning pace while sequentially assimilating novel categories, counters catastrophic forgetting and seamlessly integrates historical and newly emerging malware by synergistically combining a sparse dual loss with a standard sparse loss mechanism. The MADAR framework [18] significantly boosts continual learning performance in malware classification. By specifically accounting for malware data distribution, it effectively tackles concept drift and catastrophic forgetting.

Ultimately, beyond natural malware evolution, models must withstand deliberate adversarial manipulations. Consequently, the Dual-Opponent Generative Adversarial Network (DOpGAN) [19], with its dual-opponent nomenclature denoting a grey-box attack architecture designed to challenge baseline opcode-modification defenses, exposes Android detection vulnerabilities by subtly altering opcode distributions to generate deceptive benign-looking malware, thereby yielding invaluable adversarial assets to retrain and fortify systems against evolving evasion threats.

2.3. Insights from CIL in Other Domains

Driven by the evolution of malware detection from static classification to continuous adaptation, CIL has established itself as a critical paradigm for learning new malware families sequentially. To mitigate the inherent stability–plasticity dilemma, current literature generally approaches this challenge from four distinct perspectives: data replay, optimization constraints, architectural efficiency, and representation learning.

From the data perspective, replay-based strategies preserve historical knowledge directly by maintaining or generating past samples. Breaking the sub-optimal adaptivity bottleneck of analytic learning combined with pre-trained models, the Momentum-based Analytical Learning (MoAL) framework leverages momentum-based adapter weight interpolation to acquire new classes effectively and utilizes a knowledge rumination mechanism to revisit and consolidate past knowledge without catastrophic forgetting [20].

Alternatively, from an optimization standpoint, regularization-based methods constrain model updates to protect previous knowledge without relying on stored data. To address the stability–plasticity dilemma in few-shot class incremental learning, a recent approach integrates graph neural networks and physics-inspired energy constraints with a parameter-efficient CLIP backbone to enhance cross-modal alignment and mitigate catastrophic forgetting [21]. A novel proactive soft-orthogonal regulation strategy mitigates the compounded effects of head-class bias and catastrophic forgetting in long-tailed continual learning by pre-allocating embedding space for future tasks, thereby preserving semantic continuity and ensuring the robust integration of vulnerable tail classes [22]. Similarly, by directly reweighting skewed gradients to correct biased classifier updates, a robust framework neutralizes both intra- and inter-phase data imbalances, further integrating a distribution-aware knowledge distillation loss to proportionally protect instance-rich categories from severe performance degradation during incremental phases [23].

From an architectural efficiency perspective, parameter isolation and efficient fine-tuning techniques aim to minimize computational overhead and structural bloat during

continuous updates. In contrast to computationally expensive prompt pool methods that increase sequence lengths, a recently proposed prompt-based approach utilizes a single shared prompt set to directly alter the CLS token's attention, thereby drastically reducing inference costs and parameters while maintaining strong incremental performance [24]. Driven by the necessity for strict parameter efficiency in streaming scenarios, a unified strategy accommodates new data streams by appending extremely lightweight declarative parameters to a fixed-capacity backbone, combining a plastic extractor with an analytical classifier to ensure competitive accuracy and task-order robustness [25].

Alongside these approaches, from the perspective of representation learning, feature decoupling and prototype calibration focus on refining decision boundaries and mitigating inter-class interference. To avoid structural overhead and catastrophic forgetting in vision-language continual learning, a novel framework termed Bridge-layer Orthogonal Fusion for Adaptation (BOFA) restricts adaptation to CLIP's native bridge layer using an orthogonal low-rank fusion mechanism, while constructing cross-modal hybrid prototypes that synergize stable textual and dynamic visual features to enhance classification robustness [26]. Mitigating the severe fluctuations caused by class arrival sequences, the Dynamic Class Conflict Isolation for Incremental Learning (DCCIL) approach avoids inter-class interference by utilizing graph coloring to dynamically cluster similar fault categories into distinct pseudo-tasks with independent classifiers, ensuring computational efficiency and robust historical knowledge retention [27].

The direct application of standard CIL to malware detection is often limited by feature overlap and resource constraints, making conventional approaches insufficient. Therefore, this work introduces a specialized CIL framework designed to maintain clear decision boundaries and achieve effective malware classification in dynamic environments.

3. Methodology

In response to the dynamic evolutionary threats posed by malware, this research proposes and meticulously implements an advanced incremental learning framework. The primary objective of this framework is to facilitate the continuous acquisition of knowledge regarding nascent malware families by classification models, concurrently mitigating the critical challenge of catastrophic forgetting in continuous learning paradigms. The foundational methodology is structured around three key pillars: the transformation of raw malware samples into visually salient grayscale images, the establishment of a staged fine-tuning incremental learning workflow, and the integration of an innovative dual-branch network architecture designed to assimilate both prior and emergent knowledge, further enhanced by prototype calibration for classifier optimization.

3.1. Malware Binary to Image Conversion

In order to fully utilize the robust feature extraction power of Convolutional Neural Networks (CNNs) for image recognition, this study transforms unstructured malware binaries into structured two-dimensional grayscale images.

As shown in Figure 1, our study utilizes a byte stream mapping approach for malware visualization, transforming binary files into grayscale images for subsequent visual analysis. The methodology involves several steps: Initially, the malicious binary file is conceptualized as a one-dimensional byte stream, $B = \{b_1, b_2, \ldots, b_L\}$, where L represents the total byte count. Subsequently, each byte b_i is interpreted as an 8-bit unsigned integer, with its value $v_i \in [0, 255]$ directly corresponding to a pixel intensity in the grayscale image. Finally, this one-dimensional intensity vector, $\boldsymbol{V} = (v_1, v_2, \ldots, v_L)$, is reshaped into a two-dimensional grayscale image matrix $\boldsymbol{I}$. This reshaping is achieved by defining a fixed width W (typically 256) and calculating the height $H = \lceil L/W \rceil$. If L is not perfectly divisible by W, the stream

is padded with zero values to attain a total length of $W \times H$. The pixel intensity at image coordinates (x, y) is determined by the formula:

$$\boldsymbol{I}(x,y) = \begin{cases} v_{y \cdot W + x + 1} & \text{if } y \cdot W + x + 1 \leq L \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $0 \leq x < W, 0 \leq y < H$.

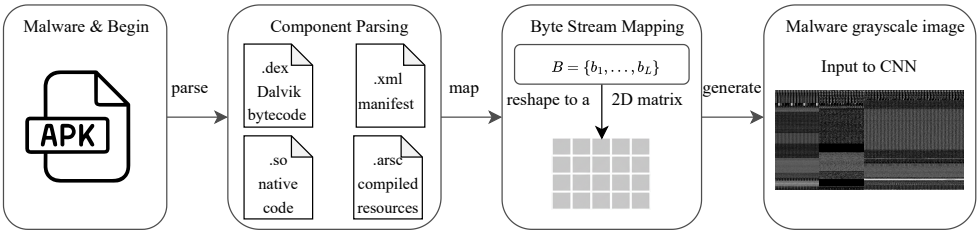

Figure 1. The process of generating grayscale images.

During the parsing phase, four distinct component types are extracted, each possessing a pivotal function: The .Dex files are instrumental, as they contain the application's Dalvik bytecode, thereby revealing the characteristics of its core logical code. The .xml manifest files document essential configuration data, including application permissions and component declarations. The .arsc resource files are repositories for compiled resources, such as strings and layout definitions. Furthermore, the .so library files encapsulate natively compiled code, which can be indicative of underlying malicious behaviors.

It is important to note that the extraction of specific file components, including .dex, .xml, arsc, and .so, is specifically tailored for Android Application Packages. Consequently, the feature representation pipeline constructed in this study is inherently designed for and evaluated on Android malware environments.

3.2. Incremental Learning Framework for Malware Classification Based on Fine-Tuning

Our model operates in a CIL scenario. The entire learning process is divided into a sequence of tasks $\{T_0, T_1, \ldots, T_N\}$, where T_0 contains base malware families, and subsequent tasks $T_K (0 < K < N)$ introduce new malware families. Let C_k be the number of classes in the k-th task. The total number of known classes is $C_K^{\text{total}} = C_0 + \sum_{i=1}^{k} C_i$.

3.2.1. Initial Task (T_0) for the Construction and Solidification of the Basic Knowledge Model

The initial task T_0 serves as the foundation and starting point for the entire incremental learning framework. This stage is based on a pre-collected training dataset, which comprises multiple base malware families representative of current mainstream malware types. By thoroughly training on this foundational dataset, we establish a robust feature extractor and classifier, both of which will collectively form the core basis for all subsequent incremental learning tasks.

Centroid-based Prototype Learning is adopted for feature modeling to mitigate the impact of non-Gaussian distributions, extreme outliers, and label noise typical in malware. In contrast to conventional mean-based prototypes, this method establishes class centers by computing the median or centroid of a filtered feature set, thereby effectively neutralizing the bias typically induced by irregular samples. By ensuring that prototypes capture the essential distribution of each class, the method provides principled support for the integration of new and old knowledge during incremental learning, inherently reducing the risk of catastrophic forgetting. After generating centroid prototypes, the initial model is frozen as an old branch to preserve prior knowledge. This ensures stability for old classes while enabling efficient fusion with new features for continuous incremental learning.

In the initial task, the model is trained on base malware families. Given the training dataset $D_0 = \{(x_i, y_i)\}_{i=1}^{n_0}$, which contains n_0 samples from C_0 base classes, each sample $\boldsymbol{x_i}$ is a grayscale image converted from a malware binary file, and $y_i \in \{0, 1, \ldots, C_0 - 1\}$ is its corresponding class label. The model completes this classification task by learning a feature extractor f_θ and classifier head parameters θ.

For the initial task, training is conducted by leveraging the standard cross-entropy loss function, whereby model parameters are optimized through the maximization of the probability assigned to the correct class. $\mathcal{L}_{\text{CE}}$ is expressed as:

$$\mathcal{L}_{CE} = -\frac{1}{n_0} \sum_{i=1}^{n_0} \sum_{c=0}^{C_0-1} y_{i,c} \log(\hat{\boldsymbol{y}}_{i,c}) \tag{2}$$

where $y_{i,c}$ is the one-hot representation of the ground-truth label y_i, and $\hat{y}_{i,c}$ is the cth element of the predicted probability vector $\hat{\boldsymbol{y}}$.

After training the base model, we perform a post-processing step to compute median centroid prototypes for base classes. The median method offers greater robustness than the mean when applied to malware datasets with noise, outliers, or polymorphic variants. Its key strength is its resistance to extreme values, ensuring that highly divergent feature vectors do not significantly shift the prototype's location.

The first step is to extract features from the samples belonging to the C_0 classes in the base dataset D_0, which results in a set of feature vectors, denoted as F_C. We compute the element-wise median of the feature set F_C to establish an initial, outlier-robust class center, denoted as $\tilde{\mu}_c = \text{Median}(F_C)$. We then detect outliers based on $\text{Median}(F_C)$ by computing the Euclidean distance $d_i = \|f_i - \tilde{\mu}_c\|_2$ for each feature vector to the center. Following established empirical conventions in robust feature estimation, the 90th percentile of these distances is selected as the default outlier threshold. This standard 10% filtering rate is widely recognized for effectively balancing the removal of extreme mutated variants with the preservation of valuable intra-class diversity. Samples with distances greater than this threshold are deemed outliers and are subsequently discarded. This filtering purifies the feature set for prototype computation, resulting in a clean set of inlier features, F'_C. This process is summarized as follows:

$$F'_C = \{f_i \in F_C \mid d_i \leq \text{Percentile}(d_j, 90)\} \tag{3}$$

where $\text{Percentile}(d_j, 90)$ represents the 90th percentile of the distance distribution. The index j is used to iterate over all samples in F_C to compute this distribution and its corresponding threshold, while i refers to an individual sample f_i being evaluated against this threshold.

Upon obtaining the purified feature set F'_C, we once again apply the element-wise median computation to it, yielding the final median centroid prototype p_c.

$$p_c = \text{Median}(F'_C) \tag{4}$$

The prototype p_c is derived from the sample population most representative of the class core, making it inherently robust to outliers in the initial dataset and more adept at accurately capturing the central tendency of non-Gaussian or skewed distributions. To ensure scale consistency for the classification task, this prototype is subsequently L_2-normalized, transforming it into a unit vector that serves as the initial classifier weight for the class.

The median centroid prototype generated by our method offers greater stability and representativeness than traditional mean prototypes, especially in challenging scenarios with high malware variance and noisy labels.

Upon completion of the centroid prototype generation, the parameters of the initial model are rendered immutable. Consequently, throughout all subsequent incremental tasks, the initial feature extractor's parameters remain fixed and are not updated during the learning of new categories. This frozen initial model, together with its corresponding classifier weights, is formally defined as the 'old branch.' This branch assumes a pivotal role in facilitating feature fusion and ensuring knowledge preservation during the incremental learning phase.

3.2.2. Incremental Task ($T_k, k > 0$) in Dual-Branch Architecture and Weighted Fusion

When new malware families emerge and are detected, the task is to learn C_k new classes while simultaneously preserving prior knowledge. This encapsulates the core challenge of incremental learning: how to accommodate new data without catastrophically forgetting previously learned information.

The total number of known categories is given by $C_k^{\text{total}} = C_0 + \sum_{i=1}^{k} C_i$. For the new malware families emerging in task T_k, the corresponding classifier weights are directly initialized using their centroid prototypes. To address this incremental learning requirement, we introduce an innovative dual-branch architecture featuring self-adjusting weights. The dual-branch architecture utilizes two parallel feature extractors to simultaneously preserve old knowledge and adapt to new malware, effectively balancing stability with adaptability. The dual-branch architecture consists of the following two components:

- Old Branch: A frozen feature extractor $f_{\theta^{(0)}}$, trained on task T_0. Its parameters are locked to retain the initial feature extraction ability, representing a stable knowledge base for all known malware. The feature vector from the old branch is: $z_{\text{old}} = f_{\theta^{(0)}}(x) \in \mathbb{R}^d$. z_{old}
- New Branch: A trainable feature extractor $f_{\theta^{(k)}}$, whose parameters are updated to learn new malware features. It is initialized with the old branch's weights to maintain a connection between the branches and avoid starting from scratch. The feature vector from the new branch is: $z_{\text{new}} = f_{\theta^{(k)}}(x) \in \mathbb{R}^d$.

z_{old} is the gradient-free feature from the old branch, while z_{new} is the trainable feature from the new branch, updated through backpropagation. Since both vectors exist in the same d-dimensional space, their fusion is meaningful.

Instead of simple concatenation or addition, we use a weighted fusion mechanism, controlled by a parameter $\alpha \in [0, 1]$. It dynamically balances the contributions of old and new knowledge according to their feature distributions during incremental learning:

$$z_{\text{f}} = \alpha \cdot z_{\text{new}} + (1 - \alpha) \cdot z_{\text{old}} \tag{5}$$

The weighted fusion mechanism is fundamentally grounded in the Stability–Plasticity Dilemma, which stipulates that an incremental model must simultaneously maintain stability to preserve previously acquired knowledge and exhibit plasticity to rapidly accommodate new information. The parameter α directly governs the stability–plasticity balance of the incremental model: as α approaches 1.0, the model is dominated by z_{new}, prioritizing the rapid learning of feature patterns from novel malware; conversely, as α approaches 0.0, it is dominated by z_{old}, emphasizing the retention of knowledge about old malware families.

3.3. CIL Testing Phase

After completing the pre-training and incremental phases, the entire learning framework forms a complete training process. This subsection elaborates on how the ProtoMal method efficiently learns new knowledge while stably retaining old knowledge, ultimately achieving the accurate classification of malware. The entire process is illustrated in Figure 2.

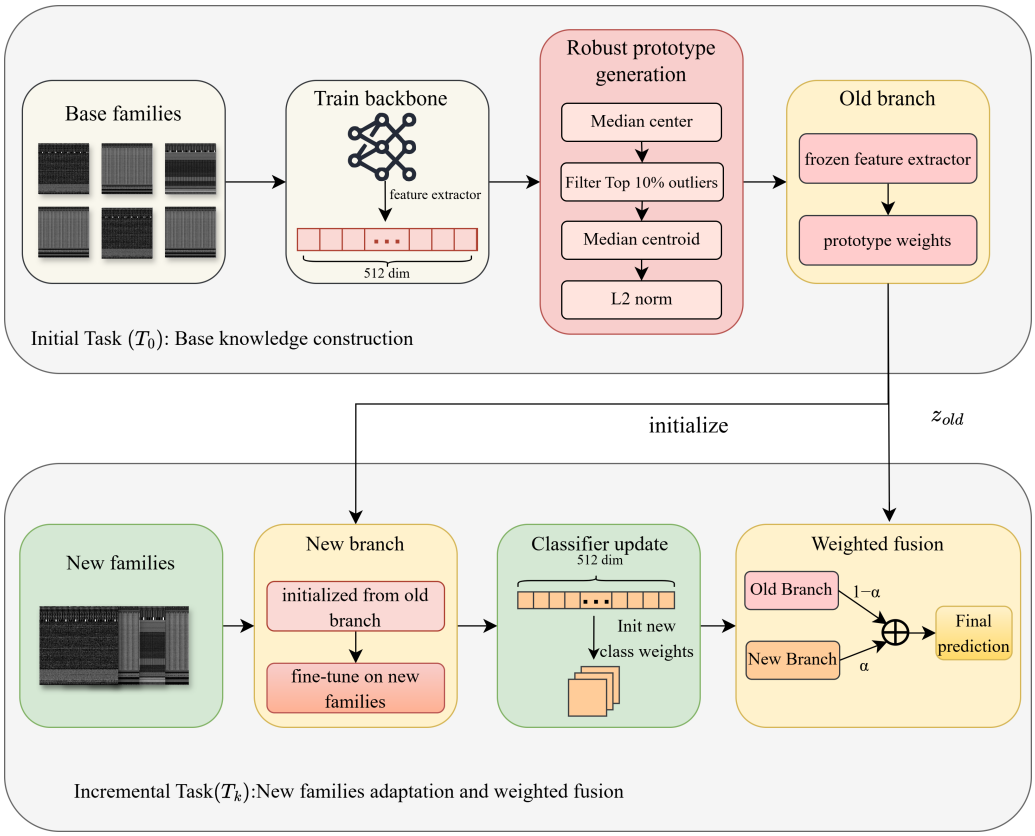

Figure 2. The structure of ProtoMal.

The initial task T_0, serves as the foundation and starting point for the entire incremental learning framework. It performs predictions for the base classes using the cross-entropy loss function as shown in Equation (2). Then, a median centroid prototype is calculated for each base class, as defined in Equations (3) and (4). Compared to the traditional mean prototype, this median-of-centroids prototype exhibits superior stability and representativeness in complex scenarios, such as those involving high intra-family variance within malware families and the presence of noisy data labels. The centroid prototypes are generated, all parameters of the initial model are frozen. This implies that in all subsequent incremental tasks, the parameters of the initial feature extractor will remain fixed and will not be modified by the learning of new classes.

When new malware families emerge and are discovered, it becomes necessary to learn C_k new classes while preserving prior knowledge. To this end, this paper innovatively employs a dual-branch structure with dynamically adjustable weights, allowing for the use of different weights for different datasets. In this architecture, the parameters of the old branch are completely frozen, and no gradients are computed for them. The new branch is initialized with the weights of the old branch, and its parameters are updated via backpropagation to learn the feature representations of the new malware families. The feature vectors from both branches reside in the same d-dimensional space and can be effectively fused by adjusting a parameter α. The training loss for each incremental task is formulated as follows:

$$L_k = -\frac{1}{n_k}\sum_{i=1}^{n_k}\sum_{c=0}^{C_k^{total}-1} y_{i,c} log(\hat{y}_{i,c}) \quad (6)$$

where n_k denotes the number of samples in the k-th task, $y_{i,c}$ is the real one-hot label indicator, and $\hat{y}_{i,c}$ is the predicted probability for class c after the fused features are passed through the classification layer.

Finally, for a given test sample x_test, features are extracted through the two branches. Based on the discriminative information learned during the dual-branch training, the cosine similarity between the fused feature vector and the prototype weights of the fully connected (FC) layer is computed as follows:

$$s_c = \frac{z_f \cdot p_c}{\|z_f\| \|p_c\|} \tag{7}$$

The logit values for all classes are then passed through the softmax function to compute a probability distribution, and the class with the maximum probability is selected as the final prediction:

$$\hat{y} = \arg\max_c \frac{\exp(s_c)}{\sum_{j=1}^{C_{\text{total}}^k} \exp(s_j)} \tag{8}$$

Ultimately, the ProtoMal framework achieves comprehensive and accurate classification of Android malware, spanning from known legacy families to novel emerging threats.

4. Experiments

4.1. Setup

To empirically evaluate the efficacy of the proposed framework, we conducted extensive experiments on three specially constructed Android malware benchmark datasets. In this section, three malware datasets and the methods used to adapt them to CIL scenarios for malware detection are introduced. Table 1 provides a detailed introduction to the malicious software datasets of AMD, VirusShareYears and VirusShare.

Table 1. Details of AMD, VirusShareYears and VirusShare datasets.

Dataset	Base Session	Incremental Session	
	Classes	Classes	Incremental Form
AMD	30	5	one class
VirusShareYears	40	10	one class
VirusShare	110	10	one class

The AMD is derived from the Android Malware Dataset. This dataset contains 35 categories. Among them, 30 categories were designated as base classes, and the remaining 5 categories were classified as incremental categories. These 5 incremental categories were further divided into 5 sessions.

VirusShare is a large-scale dataset originating from the VirusShare repository. This dataset contains 120 categories. Among them, 110 categories were designated as base classes, and the remaining 10 categories were classified as incremental categories. These 10 incremental categories were further divided into 10 sessions.

VirusShareYears is built from the VirusShare repository and is used to simulate the concept drift of time. This dataset contains 50 categories. Among them, 40 categories were designated as base classes, and the remaining 10 categories were classified as incremental categories. These 10 incremental categories were further divided into 10 sessions.

4.1.1. Evaluation Metrics

The proposed method was comprehensively evaluated using three key metrics:

i Accuracy in Each Session. It is used to evaluate the classification ability of the model in classifying the already learned categories and the newly introduced categories after each incremental step. Incremental learning assesses the model performance

through phased learning, mainly focusing on three indicators: the accuracy of the new categories, the accuracy of the old categories, and the overall accuracy.

ii Performance Degradation (PD). PD is an indicator for measuring the decline in the classification performance of the model for old categories in incremental learning, reflecting the degree of forgetting of old knowledge when learning new categories. Where $PD = \frac{1}{T}\sum_{t=1}^{T}(Acc_0 - Acc_t)$, Acc_0 is the initial accuracy of the base category, Acc_t is the accuracy of the base category after the t-th increment, and T is the total number of increments. The smaller the PD value, the stronger the model's ability to resist forgetting.

iii Average Accuracy. It is a key metric for evaluating the overall performance of the model in the incremental learning task. Its formula is $\mathcal{A}_{avg} = \frac{1}{T+1}\sum_{t=0}^{T} Acc_t$, where Acc_t represents the accuracy after the t-th incremental session, and T is the total number of incremental sessions. The higher this metric is, the more stable and reliable the classification performance of the model in incremental learning is.

4.1.2. Baseline

This study utilizes SimpleCIL [28] as the primary non-exemplar baseline method and achieves the incremental update of the detection model through the prototype-based classification of newly introduced malware categories. To systematically verify the effectiveness of the method, we selected five comparison methods: the iCaRL [29] method based on sample replay and knowledge distillation, the BiC [30] method which addresses the data imbalance problem through a bias correction layer, the FOSTER [31] method based on feature boosting and compression, the LwF [32] method based on knowledge distillation, and the Replay [33] method based on sample replay.

The experimental design covers key directions such as static prototype adaptation, bias rectification, and dynamic architecture expansion. Even without storing any historical malware samples, our ProtoMal demonstrates excellent anti-forgetting ability in complex scenarios such as the tracking of evolving malware families, and achieves a significant improvement in generalization ability, providing a secure and robust solution for dynamic malware detection.

4.1.3. Experimental Details

We conducted experiments on class-incremental learning (CIL) tasks, using ResNet-34 as the backbone network. This specific architecture was deliberately selected to facilitate a rigorous and standardized comparison with existing state-of-the-art CIL baselines. Additionally, our preliminary empirical evaluations confirmed that it strikes an optimal balance between computational efficiency and feature extraction capacity for processing malware grayscale images. The overall framework was implemented in PyTorch version 2.4.1 and optimized via Stochastic Gradient Descent (SGD) with momentum. In the pre-training phase, the model was trained for 200 epochs on the base classes across the AMD, VirusShare, and VirusShareYears datasets. The initial learning rate and the incremental learning rate were set to 0.1 and 1×10^{-5} for all three benchmarks, respectively. For all datasets, the learning rate was decayed to 0.1 times its original value at the 40th and 70th epochs during the initial training stage. In the incremental learning process, a weighted fusion strategy with a parameter $\alpha = 0.05$ and median-based centroid prototypes were employed to enhance the model's stability and robustness against catastrophic forgetting.

Accuracy served as a critical metric for evaluating malware detection systems. Achieving high accuracy lessens the burden of expert secondary verification and facilitates the rapid identification of malicious attacks in high-traffic environments, enabling appropriate defensive responses. Nonetheless, the inherent complexities of malware, including coding

variations and prevalent obfuscation techniques, result in minimal inter-class separation. These factors constitute a formidable barrier to maintaining high detection performance. Within the framework of class-incremental learning (CIL), these challenges become even more pronounced. Incremental detection models frequently encounter significant obstacles, particularly catastrophic forgetting, which results in a severe decline in the model's ability to recognize previously learned malware families when adapting to new threats. Consequently, the development of more adaptive and robust solutions for continuous malware identification is imperative. In order to rigorously evaluate catastrophic forgetting under severe resource constraints, we constructed a single-class incremental CIL scenario, where each incremental session introduces one novel malware family to evaluate the model's adaptability and resistance to catastrophic forgetting. In this setting, the model leverages the available samples of the new class to generate its representative class prototype for incremental adaptation. As shown in Tables 2–4, highly competitive accuracy is demonstrated, and superior long-term stability against catastrophic forgetting is consistently maintained across the incremental sessions by our proposed framework, ProtoMal.

In real-world cybersecurity environments, novel malware families and variants continuously evolve and emerge sequentially. Rather than waiting for multiple new threat categories to accumulate before updating the model, security systems must rapidly adapt to each new threat independently as it is discovered. To rigorously evaluate the framework's resistance to extreme catastrophic forgetting under this highly imbalanced sequential learning constraint, we constructed a strict single-class incremental session protocol. In this scenario, each session introduces exactly one novel malware family. ProtoMal leverages the available incoming data stream for this specific class to train robust, median-based prototypes, ensuring clear decision boundaries and stable integration of new knowledge without requiring simultaneous multi-class updates. As shown in Tables 2–4, highly competitive accuracy and robust, sustained performance stability across the incremental sessions are exhibited by our proposed framework, ProtoMal.

Table 2. Comparison of ProtoMal with existing methods on the AMD Dataset.

Method	Accuracy in Each Session (%)						Avg	Imprv.
	0	1	2	3	4	5		
SimpleCIL	90.72	88.97	87.13	85.98	81.36	80.26	85.74	+1.25
iCaRL	91.62	90.77	90.13	89.52	85.14	80.64	87.97	−0.98
BiC	90.18	90.04	89.88	88.35	84.35	80.27	87.18	−0.19
Foster	91.13	91.13	66.91	66.00	59.13	59.92	73.04	+13.95
LwF	91.20	79.60	75.80	71.90	64.80	60.40	73.95	+13.04
Replay	90.77	80.35	80.16	79.61	66.92	70.75	78.09	+8.90
ProtoMal	90.65	90.09	88.63	87.71	83.12	81.77	86.99	-

Table 3. Comparison of ProtoMal with existing methods on the VirusShare Dataset.

Method	Accuracy in Each Session (%)											Avg	Imprv.
	0	1	2	3	4	5	6	7	8	9	10		
SimpleCIL	79.73	76.97	76.32	75.92	75.62	74.82	74.52	74.31	73.59	73.49	72.78	74.82	+2.33
iCaRL	80.35	75.85	73.95	72.29	70.96	65.09	60.89	57.71	55.15	53.15	66.54	66.54	+10.61
BIC	79.72	79.47	79.39	78.15	78.13	76.85	59.52	60.74	59.66	59.33	59.03	69.99	+7.16
Foster	78.90	74.80	71.60	69.20	66.70	63.50	61.80	60.90	60.10	58.90	56.60	64.80	+12.35
LwF	79.40	75.20	72.80	70.30	67.90	65.10	63.20	62.10	61.00	58.90	56.70	65.60	+11.55
Replay	79.80	76.30	73.90	71.80	69.50	66.80	64.90	63.60	62.20	60.00	57.70	66.40	+10.75
ProtoMal	79.58	78.09	77.94	77.59	77.46	77.23	76.94	76.77	76.01	75.88	75.15	77.15	-

Table 4. Comparison of ProtoMal with existing methods on the VirusShareYears Dataset.

Method	Accuracy in Each Session (%)											Avg	Imprv.
	0	1	2	3	4	5	6	7	8	9	10		
SimpleCIL	73.85	67.44	67.47	66.57	66.51	63.79	62.72	62.72	62.23	62.23	59.67	65.04	+1.68
iCaRL	75.73	67.57	67.19	62.69	62.23	34.55	33.78	34.54	35.38	37.02	39.21	49.99	+16.73
BiC	73.86	72.65	72.53	71.39	70.21	64.64	59.50	59.55	59.69	59.35	58.95	65.76	+0.96
Foster	80.13	79.95	43.11	41.08	40.92	40.24	39.70	39.72	37.43	38.75	38.33	49.03	+17.69
LwF	75.20	59.50	58.30	57.20	56.40	54.10	52.80	51.90	50.80	49.70	48.60	55.86	+10.86
Replay	75.50	59.00	58.40	57.90	57.10	55.00	53.80	53.00	51.70	50.60	49.40	56.49	+10.23
ProtoMal	74.27	69.14	69.17	68.54	68.30	66.35	64.79	64.67	64.22	64.27	60.24	66.72	-

The comprehensive results of these experiments are visually presented in Figure 3, offering intuitive insights into performance trends, while the corresponding detailed numerical results are meticulously tabulated in Tables 2–4, providing a quantitative basis for our findings.

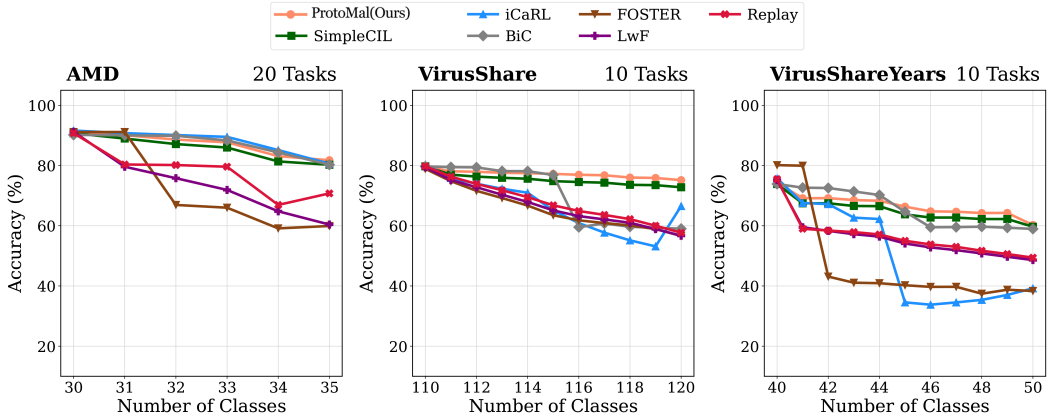

Figure 3. Test accuracy curves of different methods on the AMD, VirusShare, and VirusShareYear datasets. The plots illustrate the changes in classification accuracy as the number of classes increases during the incremental learning process.

4.2. Comparison with the State-of-the-Art Methods

As shown in Table 3, taking the initial session (Session 0) as an example, ProtoMal achieves an accuracy rate of 79.58%, which demonstrates competitive performance compared to established methods. It is notable that in subsequent sessions (from Session 1 to Session 10), ProtoMal maintains a highly stable accuracy rate of over 75%, while other methods exhibit significant catastrophic forgetting phenomena as more malware families are introduced. For instance, in Session 10, the accuracy rate of BiC drops sharply to 59.03%, further highlighting ProtoMal's advantage in long-term continual learning.

Indeed, ProtoMal showcases exceptional robustness, evidenced by its average accuracy of 77.15% on the VirusShare dataset—a significant lead over Foster (64.8%), LwF (65.6%), Replay (66.4%), iCaRL (66.54%), and BiC (69.99%) by 12.35%, 11.55%, 10.75%, 10.61%, and 7.16%, respectively. Furthermore, by achieving the highest performance improvement over all compared methods on this benchmark, ProtoMal demonstrates its unique capacity to both retain prior knowledge and adeptly adapt to new tasks, effectively navigating the stability–plasticity challenge that hinders baseline methods.

As presented in Table 4, ProtoMal demonstrates superior performance under temporal concept drift. Achieving an average accuracy of 66.72%, it outperforms all baseline methods, including BiC (65.76%) and SimpleCIL (65.04%). Its remarkable cross-session stability (68.54% to 64.27% from Session 3 to Session 9) effectively mitigates the impact of evolving malware variants over time. Similarly, as presented in Table 2 for the AMD dataset,

competitive accuracy, stronger anti-forgetting capability, and cross-session stability are consistently demonstrated by our proposed framework, ProtoMal. Although achieving an average accuracy of 86.99%, which does not represent the highest absolute accuracy when compared to iCaRL and BiC, exceptional resilience against catastrophic forgetting is maintained by the model. Remarkable cross-session stability is exhibited, with accuracy effectively sustained between 87.71% in Session 3 and 81.77% in Session 5, successfully mitigating representation drift issues. This stability underscores a robust long-term learning efficiency, driven by the weighted fusion strategy. Furthermore, stable class representations across incremental stages are ensured by the innovative centroid-based prototype learning.

4.3. Analysis Across Different Sessions

The stability of accuracy during sessions significantly reflects the model's ability to resist catastrophic forgetting during the incremental learning process. Specifically, Top-1 accuracy measures the model's ability to detect malware for the first time, while Top-5 accuracy measures its ability to identify malware within the first five detections. Additionally, the performance degradation rate (PD) further quantifies the extent to which the model's performance declines as the incremental learning process progresses.

Table 5 presents the performance of the model for each session on the AMD dataset. It can be observed that as the number of sessions increases, the Top-1 accuracy gradually decreases from 0.91 in Session 0 to 0.82 in Session 5, with an average Top-1 accuracy of 0.87. The Top-5 accuracy gradually decreases from 0.91 in Session 0 to 0.84 in Session 5, with an average Top-5 accuracy of 0.89. The PD gradually increases from 0.56% in Session 1 to 8.88% in Session 5, with an average PD of 4.38%.

Table 5. Accuracy and Performance Degradation Across Sessions on AMD Dataset.

Session	T1 Accuracy	T5 Accuracy	PD (%)
0	0.91	0.91	base
1	0.90	0.90	0.56
2	0.89	0.90	2.02
3	0.88	0.89	2.94
4	0.83	0.87	7.53
5	0.82	0.84	8.88
AVG	0.87	0.89	4.38

Table 6 presents the performance of the model for each session on the VirusShare dataset. It can be observed that as the number of sessions increases, the Top-1 accuracy gradually decreases from 0.79 in Session 0 to 0.75 in Session 10, with an average Top-1 accuracy of 0.77. The Top-5 accuracy gradually decreases from 0.89 in Session 0 to 0.83 in Session 10, with an average Top-5 accuracy of 0.86. The PD gradually increases from 1.49% in Session 1 to 4.43% in Session 10, with an average PD of 2.67%.

Table 7 presents the performance of the model for each session on the VirusSh-areYears dataset. The Top-1 accuracy rate of Session 0 is 0.74, and the Top-5 accuracy rate is 0.92. As the sessions progress, the Top-1 accuracy rate gradually drops to 0.60 of Session 10, and the Top-5 accuracy rate also gradually decreases to 0.75 of Session 10. The performance degradation rate increases from 5.13% of Session 1 to 14.03% of Session 10, with an average PD of 8.30%.

By comparing Tables 5–7, it can be observed that in the data set of in Table 5, the Top-1 and Top-5 accuracy rates of the initial session are relatively higher. However, the growth trend of the performance degradation rate is similar across all benchmarks. Nevertheless, the average PD of Table 7 is higher than that of Tables 5 and 6, which is primarily due to the

significant temporal concept drift present in the VirusShareYears dataset. On large-scale datasets that are highly complex and have extremely imbalanced sample distributions, the method proposed in this paper significantly improves the generalization performance and stability of the model.

Table 6. Accuracy and Performance Degradation of ProtoMal Across 10 Incremental Sessions on VirusShare Dataset.

Session	T1 Accuracy	T5 Accuracy	PD (%)
0	0.7958	0.8910	base
1	0.7809	0.8876	1.49
2	0.7794	0.8764	1.64
3	0.7759	0.8731	1.99
4	0.7746	0.8675	2.12
5	0.7723	0.8557	2.35
6	0.7694	0.8509	2.64
7	0.7677	0.8493	2.81
8	0.7601	0.8412	3.57
9	0.7588	0.8374	3.70
10	0.7515	0.8338	4.43
AVG	0.7715	0.8602	2.67

Table 7. Accuracy and Performance Degradation of ProtoMal Across Sessions on VirusShareYears Dataset.

Session	T1 Accuracy	T5 Accuracy	PD (%)
0	0.7427	0.9153	base
1	0.6914	0.8976	5.13
2	0.6917	0.8878	5.10
3	0.6854	0.8665	5.73
4	0.6830	0.8368	5.97
5	0.6635	0.8025	7.92
6	0.6479	0.7723	9.48
7	0.6467	0.7717	9.60
8	0.6422	0.7627	10.05
9	0.6427	0.7607	10.00
10	0.6024	0.7537	14.03
AVG	0.6672	0.8207	8.30

Table 8 presents the comparison of different methods across the AMD, VirusShare, and VirusShareYears benchmarks. From the table, highly competitive average Performance Degradation scores are achieved on the VirusShare and VirusShareYears datasets, with values of 2.67% and 8.30% respectively, while demonstrating competitive accuracy and stability on the AMD dataset with a PD of 4.38%. In contrast, larger performance degradation under incremental learning is generally exhibited by other methods including SimpleCIL, iCaRL, BiC, and FOSTER. For instance, FOSTER reaches a PD score of 22.51% on the AMD dataset, and iCaRL reaches a PD score of 28.31% on the VirusShareYears dataset. These numerical results forcefully support that highly competitive anti-forgetting capability on baseline datasets, alongside superior cross-session stability under severe temporal drift, are prioritized and achieved by the continual learning framework, ProtoMal.

Table 8. Comparison of different methods by Average PD (%) metrics across three benchmarks.

Method	AMD (%)	VirusShare (%)	VirusShareYears (%)
SimpleCIL	5.98	4.90	9.72
iCaRL	4.38	15.19	28.31
BiC	3.60	10.69	9.01
FOSTER	22.51	14.49	36.21
LwF	20.70	14.08	21.27
Replay	15.21	13.13	20.91
ProtoMal (Ours)	4.38	2.67	8.30

4.4. Ablation Studies and Analysis

To thoroughly evaluate the contributions of each key component within ProtoMal, this subsection details the impact of the weighted fusion strategy, median-based centroid calculation, and outlier filtering mechanism on the final results. All experiments utilized the same ResNet-34 model architecture and training data on the VirusShareYears dataset, with detailed results presented in Table 9.

Table 9. Performance comparison of different components across training sessions on VirusShareYears.

Methods	Sessions (VirusShareYears Dataset)											Avg.
	0	1	2	3	4	5	6	7	8	9	10	
Baseline (None)	74.19	63.97	63.91	62.05	60.99	55.10	53.74	53.59	53.27	53.15	32.30	56.93
+ Strategy (Add)	74.23	58.44	58.45	57.94	55.90	54.79	53.41	53.29	52.72	52.75	48.38	56.39
+ Without Outlier Filtering	74.23	64.25	64.15	63.57	61.20	59.90	58.38	58.22	57.85	57.81	35.08	59.51
ProtoMal (Ours)	74.27	69.14	69.17	68.54	68.30	66.35	64.79	64.67	64.22	64.27	60.24	66.72

4.4.1. Weighted Fusion Strategy

We first evaluated the impact of our proposed Weighted Fusion Strategy on model performance. The ProtoMal method integrates this weighted fusion mechanism to achieve a more adaptive and balanced feature representation between known malware families and emerging threats. We compared it with two baseline integration strategies: '+Concatenation' (merging features from both branches through vector concatenation) and '+Addition' (combining features through simple element-wise summation).

As clearly shown in Table 9, ProtoMal consistently demonstrates superior performance across all incremental sessions and in terms of average accuracy. ProtoMal achieved the highest average accuracy of 66.72%, significantly surpassing '+Addition' at 56.39% and the baseline without fusion at 56.93%. Specifically, compared to the strategy that employs simple additive fusion, ProtoMal improved the average accuracy by 10.33%.

In the early incremental training stages (Session 0), while the '+Addition' strategy reached 74.23%, ProtoMal still maintained a lead with 74.27% performance. However, the crucial distinction lies in the model's long-term stability. As incremental sessions progress, integration strategies not employing the weighted fusion mechanism experience a more pronounced performance degradation. For instance, by Session 10, ProtoMal still maintains an accuracy of 60.24%, whereas '+Addition' drops to 48.38%, and the baseline without fusion further declines to 32.30%. This clear performance gap demonstrates the effectiveness of the weighted fusion strategy in dynamically balancing the contributions of prior and emergent knowledge via the adjustable parameter α. It achieves a more robust feature integration between the stably frozen branch and the plastic training branch, effectively mitigating the representation drift that usually occurs as the number of incremental training

tasks increases. This capability is crucial for the long-term stability and accuracy of the model in real-world malware detection scenarios where new families continuously emerge.

4.4.2. Effect of the Weight Parameter α

To further investigate the sensitivity of the weighted fusion mechanism to parameter changes, we conducted an experiment on the VirusShareYears dataset by adjusting the weight parameter α. Since α controls the relative contribution of plastic and frozen branches during feature fusion, this experiment aims to elucidate how different parameter settings affect the balance between adaptability and knowledge retention of the model.

As shown in Table 10, the model's performance exhibits a clear trend with changes in α. When α gradually decreases from 0.40 to 0.05, the average accuracy improves from 60.37% to 66.72%, while the performance degradation rate drops from 6.74 to 1.66. This result indicates that assigning a relatively small weight to the newly updated branches helps to more effectively retain previously learned knowledge while maintaining adaptability to newly introduced malware families.

Table 10. Effect of different weight parameter settings on Accuracy and PD on the VirusShareYears benchmark.

Weight Parameter (α)	Accuracy (%)	PD
0.40	60.37	6.74
0.30	63.58	4.86
0.20	64.52	3.87
0.10	66.31	2.54
0.05	66.72	1.66
0.04	66.68	2.02
0.03	66.63	2.21
0.02	49.84	11.15

However, when α is further reduced to 0.02, the model performance deteriorates sharply, with the average accuracy dropping to 49.84% and the performance degradation rate increasing to 11.15. This phenomenon suggests that an excessively small α overemphasizes the frozen old branch and weakens the plastic branch's ability to absorb discriminative information from emerging malware families. In contrast, overly large values of α bias the model toward recently learned representations, leading to weaker retention of old knowledge.An evaluation of adjacent values reveals that setting α to 0.05 yields a higher average accuracy of 66.72% compared to the 66.68% achieved when α is set to 0.04.

Furthermore, this optimal setting achieves a lower performance degradation rate of 1.66 compared to the rate of 2.02 observed at an α of 0.04. Consequently, $\alpha = 0.05$ was selected as the default parameter, as it provides the optimal trade-off between knowledge retention, indicated by a low performance degradation rate, and adaptation to novel classes. Extreme parameter values, whether excessively small such as an α of 0.02, or overly large such as an α of 0.40, inherently disrupt this delicate balance.

4.4.3. Prototype Calculation and Sensitivity Analysis of Filtering Thresholds

Ablation experiments evaluate the robustness of feature modeling by comparing the effects of centroid-based prototype learning against traditional mean-based approaches. Under identical training and evaluation protocols, the prototype generation strategy of the proposed method is compared with other standard category representation methods such as mean representation. The results demonstrate that calculating robust family prototypes by filtering outlier samples achieves a superior fit for the non-Gaussian distribution of malware characteristics. In contrast, methods that directly compute centers based on all samples fail

to account for the deleterious effects of polymorphic variants and labeling uncertainty. By suppressing interference from data noise and enhancing overall performance, this strategy, which is more suitable for class-incremental malware detection, is integrated into ProtoMal.

To further investigate the specific impact of outlier filtering rules on the stability of continual learning, this study conducts a detailed sensitivity analysis regarding different data retention percentiles. The specific tests encompass the 95th, 90th, and 85th percentiles of data retention, which correspond to outlier filtering ratios of 5%, 10%, and 15%, respectively. All sensitivity analysis experiments are evaluated using the full training dataset. Additionally, memory calculation is not considered within the scope of the practicality evaluation metrics for this research. The experimental results are presented in Tables 11–13.

A clear performance trend is observable from the experimental results. When employing the 95th retention percentile, which filters a small 5% portion of outliers, the average accuracy of the model fails to reach an optimal level across different datasets. This is attributed to the relatively conservative filtering intensity, which leaves highly deceptive label noise and extreme mutated samples within the dataset, thereby interfering with the accurate capture of core family features by the centroid prototype. Conversely, when the retention percentile is reduced to 85, representing an increased filtering intensity of 15%, the average accuracy of the model exhibits a significant decline. While such excessive filtering completely removes anomalous noise, it simultaneously discards normal and valuable intra-class diversity features, severely weakening the generalization ability of the classifier when encountering emerging variants.

The experimental data robustly demonstrate that the threshold setting achieving the optimal balance between effective feature space purification and retention of critical intra-class differences is applied in ProtoMal. This approach, combining a meticulously designed weighted fusion strategy with robust centroid-based prototype learning, not only improves the initial accuracy of the model but also effectively mitigates performance degradation as incremental sessions increase. For ensuring long-term stability and accuracy in scenarios where malware threats continuously evolve, this scheme represents core value, and its practical manifestation is ProtoMal.

Table 11. Sensitivity analysis of different filtering percentiles on the AMD Dataset expressed as accuracy percentages.

Retention Percentile	S0	S1	S2	S3	S4	S5	Avg
95th (5% Filter)	89.45	88.61	87.13	85.50	80.22	78.48	84.89
90th (10% Filter)	90.65	90.09	88.63	87.71	83.12	81.77	86.99
85th (15% Filter)	88.13	87.20	85.27	83.06	78.39	76.09	83.03

Table 12. Sensitivity analysis of different filtering percentiles on the VirusShare Dataset expressed as accuracy percentages.

Retention Percentile	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Avg
95th (5% Filter)	78.10	76.23	75.77	75.06	74.84	74.38	73.72	73.55	72.31	72.05	71.09	74.28
90th (10% Filter)	79.58	78.09	77.94	77.59	77.46	77.23	76.94	76.77	76.01	75.88	75.15	77.15
85th (15% Filter)	76.49	74.76	74.51	73.81	73.44	72.81	72.06	71.75	70.82	70.53	69.11	72.74

Table 13. Sensitivity analysis of different filtering percentiles on the VirusShareYears Dataset expressed as accuracy percentages.

Retention Percentile	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Avg
95th (5% Filter)	72.82	67.26	67.19	66.28	65.76	63.61	62.07	61.78	61.09	60.97	56.59	64.13
90th (10% Filter)	74.27	69.14	69.17	68.54	68.30	66.35	64.79	64.67	64.22	64.27	60.24	66.72
85th (15% Filter)	71.55	65.56	65.29	64.42	64.16	61.78	60.27	59.89	59.44	59.26	54.80	62.39

4.4.4. Outlier Filtering Mechanism

To verify the impact of the outlier filtering mechanism on the stability of continuous learning, this paper designed a control variable experiment to compare and analyze the performance differences between ProtoMal and the version without filtering (+Without Outlier Filtering) on the VirusShareYears dataset. As shown in Table 9, removing the outlier filtering mechanism led to a significant 7.21% decrease in the average detection accuracy of the model.

The fixed 10% filtering threshold (90th percentile) was empirically selected as it effectively purifies feature representations by discarding extreme mutated variants without discarding useful intra-class diversity.

Further analysis of the performance fluctuations in each session revealed that the outlier filtering mechanism significantly enhances robustness across two key dimensions. In the incremental learning stages, outlier filtering reduced the variance of accuracy from 2.7% to 0.8% ($\sigma = 0.8$ vs. $\sigma = 2.7$), indicating that it effectively purifies the feature representations by discarding mutated variants and annotation uncertainties, effectively suppressing representational drift. Collectively, these experimental results underscore that outlier filtering is indispensable for maintaining the stability of incremental learning in complex and adversarial malware environments.

5. Conclusions

In this study, we introduced ProtoMal, a robust continual learning framework designed for adaptive Android malware detection in dynamic threat environments. Conventional static classifiers are prone to catastrophic forgetting and fail to recognize emerging malware families due to concept drift and feature distribution shifts. ProtoMal addresses these challenges by combining a weighted dual-branch architecture with median centroid-based prototype learning. The dual-branch design decouples knowledge retention and adaptation, while the parameter α dynamically balances stability and plasticity. Additionally, using centroid-based prototypes with outlier filtering reduces the impact of highly polymorphic samples and irregular feature distributions, producing highly representative class centers.

Extensive experiments on the AMD, VirusShare, and VirusShareYears benchmarks demonstrate that competitive accuracy, stronger anti-forgetting capability, and cross-session stability are consistently achieved by the proposed continual learning framework, ProtoMal. Rather than merely maximizing raw average accuracy, the numerical evidence across all datasets forcefully supports that significantly lower performance degradation and superior cross-session stability under severe temporal drift and highly imbalanced class distributions are established by the framework. Ablation studies confirm that both the weighted fusion mechanism and the centroid-based prototype strategy are critical for maintaining long-term performance and mitigating catastrophic forgetting. Sensitivity analysis further shows that an appropriate empirical balance of α is essential for optimal incremental learning, effectively managing the stability–plasticity trade-off in dynamic malware environments. Importantly, ProtoMal achieves these gains without storing historical samples, offering practical efficiency and privacy advantages for real-world deployments. Furthermore,

computational evaluations confirm that the same inference latency as standard static baselines is maintained, successfully avoiding the doubled structural overhead inherent in dynamic architectures like FOSTER, thereby ensuring its practicality for high-traffic real-world deployments. Note that memory usage is not considered as a primary metric for the model's practicality claims.

While ProtoMal demonstrates strong anti-forgetting capabilities, certain limitations highlight avenues for future research. Currently, the feature extraction pipeline is tailored specifically to Android applications (parsing .dex, .xml, .arsc, and .so components), leaving its transferability to cross-platform threats like Windows PE or Linux ELF binaries unexplored. Furthermore, while the strict 1-way single-class incremental protocol effectively stress-tested catastrophic forgetting, future evaluations must encompass broader multi-class scenarios to fully validate the framework's generalizability. Algorithmically, relying on a fixed parameter α and a static 10% outlier filtering threshold may constrain the model's flexibility during irregular class distribution shifts; dynamically adapting these values remains a key objective. Lastly, to facilitate edge-device deployment, future work will explore lightweight architectures and optimize overall training efficiency, building upon this foundational study where ResNet-34 was utilized exclusively for fair benchmarking against existing baselines.

Looking forward, several avenues can enhance the framework. Multimodal integration of dynamic behavioral features, network traffic, and system call sequences can provide richer malware representations and extend applicability beyond Android environments. Improving interpretability through visualization and explainable AI can increase trust in automated detection, while federated incremental learning could enable collaborative threat intelligence without compromising privacy. Further exploration of adaptive parameter tuning for α and the filtering threshold, meta-learning strategies, and comprehensive profiling of computational costs will be prioritized to optimize the framework for high-traffic real-world deployments.

In conclusion, ProtoMal represents a significant step forward in continual learning-driven malware detection, offering a scalable, robust, and interpretable solution that effectively mitigates catastrophic forgetting while accommodating new threats. Its combination of weighted dual-branch architecture and median centroid-based prototype learning provides both theoretical insight and practical utility, establishing a foundation for future advances in proactive and intelligent cybersecurity systems.

Author Contributions: Conceptualization, X.Z.; methodology, A.Z.; software, X.Z.; validation, X.Z.; formal analysis, X.Z.; investigation, Y.B.; resources, X.Z.; data curation, X.Z.; writing—original draft preparation, X.Z.; writing—review and editing, Y.Z. (Yanan Zhang); visualization, Y.B.; supervision, M.M.; project administration, Y.Z. (Yiying Zhang); funding acquisition, X.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: During the preparation of this manuscript, the authors used Gemini 3.1pro to improve the English phrasing and grammar. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Deldar, F.; Abadi, M. Deep learning for zero-day malware detection and classification: A survey. *ACM Comput. Surv.* **2023**, *56*, 1–37. [CrossRef]
2. What Is Malware? Available online: https://www.microsoft.com/zh-cn/security/business/security-101/what-is-malware (accessed on 25 May 2026).
3. Gong, S.; Zhong, H. Incremental learning of code authors over time. *J. Syst. Softw.* **2025**, *230*, 112527. [CrossRef]
4. Ganesan, S.; Ravi, V.; Krichen, M.; V, S.; Alroobaea, R.; KP, S. Robust Malware Detection using Residual Attention Network. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 10–12 January 2021; pp. 1–6. [CrossRef]
5. Aboaoja, F.A.; Zainal, A.; Ghaleb, F.A.; Al-rimy, B.A.S.; Eisa, T.A.E.; Elnour, A.A.H. Malware Detection Issues, Challenges, and Future Directions: A Survey. *Appl. Sci.* **2022**, *12*, 8482. [CrossRef]
6. Abualhaj, M.M.; Al-Mimi, H.; Al-Zyoud, M.; Al-Khatib, S.N.; Daoud, M.S.; Al-Aqrabi, H.; Anbar, M.; Shalaldeh, A. A bio inspired hybrid optimization framework for efficient real time malware detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2026**, *16*, 4542. [CrossRef] [PubMed]
7. Sharma, O.; Sharma, A.; Kalia, A. MIGAN: GAN for facilitating malware image synthesis with improved malware classification on novel dataset. *Expert Syst. Appl.* **2024**, *241*, 122678. [CrossRef]
8. Jain, S.; Goyal, H.; Arora, A.; Kumar, D. EnFeSTDroid: Ensembled feature selection techniques based Android malware detection. *Comput. Electr. Eng.* **2026**, *129*, 110763. [CrossRef]
9. Zhang, Z.; Li, Y.; Wang, W.; Song, H.; Dong, H. Malware detection with dynamic evolving graph convolutional networks. *Int. J. Intell. Syst.* **2022**, *37*, 7261–7280. [CrossRef]
10. Wang, W.; Ren, C.; Song, H.; Zhang, S.; Liu, P. Fgl_droid: An efficient android malware detection method based on hybrid analysis. *Secur. Commun. Netw.* **2022**, *2022*, 8398591. [CrossRef]
11. Taher, F.; AlFandi, O.; Al-kfairy, M.; Al Hamadi, H.; Alrabaee, S. DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection. *Appl. Sci.* **2023**, *13*, 7720. [CrossRef]
12. Jordaney, R.; Sharad, K.; Dash, S.K.; Wang, Z.; Papini, D.; Nouretdinov, I.; Cavallaro, L. Transcend: Detecting Concept Drift in Malware Classification Models. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 625–642.
13. Xu, K.; Li, Y.; Deng, R.; Chen, K.; Xu, J. DroidEvolver: Self-Evolving Android Malware Detection System. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, 17–19 June 2019; pp. 47–62. [CrossRef]
14. Kan, Z.; Pendlebury, F.; Pierazzi, F.; Cavallaro, L. Investigating Labelless Drift Adaptation for Malware Detection. In Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, AISec'21, Virtual Event, 15 November 2021; pp. 123–134. [CrossRef]
15. Mohammad, F.; Al-Ahmadi, S.; Al-Muhtadi, J. OMD-RAS: Optimizing Malware Detection through Comprehensive Approach to Real-Time and Adaptive Security. *Comput. Mater. Contin.* **2025**, *84*, 5995–6014. [CrossRef]
16. Chen, Z.; Zhang, Z.; Kan, Z.; Yang, L.; Cortellazzi, J.; Pendlebury, F.; Pierazzi, F.; Cavallaro, L.; Wang, G. Is It Overkill? Analyzing Feature-Space Concept Drift in Malware Detectors. In Proceedings of the 2023 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 25 May 2023; pp. 21–28. [CrossRef]
17. Xu, X.; Zhang, X.; Zhang, Q.; Wang, Y.; Adebisi, B.; Ohtsuki, T.; Sari, H.; Gui, G. Advancing Malware Detection in Network Traffic With Self-Paced Class Incremental Learning. *IEEE Internet Things J.* **2024**, *11*, 21816–21826. [CrossRef]
18. Rahman, M.S.; Coull, S.; Yu, Q.; Wright, M. MADAR: Efficient continual learning for malware analysis with diversity-aware replay. *arXiv* **2025**, arXiv:2502.05760.
19. Ahmad, N.; Saleem Rana, A.; Jalil Hadi, H.; Bashir Hussain, F.; Chakrabarti, P.; Alshara, M.A.; Chakrabarti, T. GEAAD: Generating evasive adversarial attacks against android malware defense. *Sci. Rep.* **2025**, *15*, 11867. [CrossRef] [PubMed]
20. Gao, Z.; Jia, W.; Zhang, X.; Zhou, D.; Xu, K.; Dawei, F.; Dou, Y.; Mao, X.; Wang, H. Knowledge Memorization and Rumination for Pre-trained Model-based Class-Incremental Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 11–15 June 2025; pp. 20523–20533.
21. Ma, Y.; Liu, Y.; Du, B. A Few-Shot Class Incremental Learning Method Using Graph Neural Networks. *IEEE Trans. Image Process.* **2026**, *35*, 1337–1349. [CrossRef] [PubMed]
22. Fu, Z.; Zhang, Z.; Liao, S.; Huang, Z.; Chen, Z.; Shen, T. PSR: Proactive soft-orthogonal regulation for long-tailed class-incremental learning. *Pattern Recognit.* **2026**, *176*, 113207. [CrossRef]
23. He, J. Gradient Reweighting: Towards Imbalanced Class-Incremental Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 17–21 June 2024; pp. 16668–16677.
24. Chen, H.; Wang, P.; Zhou, Z.; Zhang, X.; Wu, Z.; Jiang, Y.G. Achieving More with Less: Additive Prompt Tuning for Rehearsal-Free Class-Incremental Learning. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Honolulu, HI, USA, 19–20 October 2025; pp. 340–349.

25. Li, D.; Zeng, Z.; Dai, W.; Suganthan, P.N. Complementary Learning Subnetworks Towards Parameter-Efficient Class-Incremental Learning. *IEEE Trans. Knowl. Data Eng.* **2025**, *37*, 3240–3252. [CrossRef]
26. Li, L.; Hu, T.; Zhou, D.W.; Yang, J.Q.; Ye, H.j.; Zhan, D.C. BOFA: Bridge-Layer Orthogonal Low-Rank Fusion for CLIP-Based Class-Incremental Learning. *Proc. AAAI Conf. Artif. Intell.* **2026**, *40*, 22967–22975. [CrossRef]
27. Wang, C.; Wang, Y.; Xia, R.; Zheng, F.; Wang, Y.; Li, M.; Zhao, Y.; Yang, H. DCCIL: Mitigating class conflicts in incremental learning through dynamic isolation for intelligent fault diagnosis. *Knowl.-Based Syst.* **2026**, *337*, 115417. [CrossRef]
28. Zhou, D.W.; Wang, Q.W.; Ye, H.J.; Zhan, D.C.; Liu, Z.H. SimpleCIL: Simple Class-Incremental Learning for Seeing New Classes. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2023; pp. 2692–2701.
29. Rebuffi, S.A.; Kolesnikov, A.; Sperl, G.; Lampert, C.H. iCaRL: Incremental Classifier and Representation Learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 2001–2010.
30. Wu, Y.; Chen, Y.; Wang, L.; Ye, Y.; Liu, Z.; Guo, Y.; Fu, Y. Large Scale Incremental Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 374–382.
31. Wang, F.Y.; Zhou, D.W.; Ye, H.J.; Zhan, D.C. FOSTER: Feature Boosting and Compression for Class-Incremental Learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 398–414.
32. Li, Z.; Hoiem, D. Learning without forgetting. In *Proceedings of the European Conference on Computer Vision (ECCV)*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 614–629.
33. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.