Deep Learning from Imperfectly Labeled Malware Data

Fahad Alotaibi
f.alotaibi21@imperial.ac.uk
Department of Computing
Imperial College London
London, United Kingdom

Euan Goodbrand
euan.goodbrand23@imperial.ac.uk
Department of Computing
Imperial College London
London, United Kingdom

Sergio Maffeis
sergio.maffeis@imperial.ac.uk
Department of Computing
Imperial College London
London, United Kingdom

Abstract

Deep learning approaches have achieved remarkable performance in malware classification and detection. However, their success relies on the availability of large, accurately labeled datasets: a critical yet challenging requirement in the malware domain. In practice, most malware datasets are automatically labeled using outputs from antivirus engines, a process that often introduces significant label noise. Such imperfections can severely degrade the performance and generalizability of deep learning models.

To address this challenge, we introduce SLB, a framework designed to robustly train deep learning–based malware systems while simultaneously refining dataset labels. SLB begins by partitioning the dataset into two subsets: a clean set containing samples with reliable labels, and a noisy set with samples that may be mislabeled, to which pseudo labels are assigned. As training progresses, SLB continuously monitors the model's predictions to dynamically update both sets. Specifically, samples in the noisy set that consistently receive predictions aligning with their (observed or pseudo) labels are promoted to the clean set, whereas samples in the clean set that exhibit unstable predictions are reclassified as noisy. This iterative process not only enhances model performance but also progressively corrects labeling errors.

We evaluated SLB on multiple security datasets with both synthetic and real-world label noise across various deep learning architectures and ML algorithms. Experimental results show that SLB significantly improves malware detection performance and reduces overall noise. For example, on the Android binary dataset with 25% injected label noise, SLB reduced the noise to below 1.5% while increasing the macro F1 score from 74.51% to 96.03% and the accuracy score from 87.66% to 98.68%.

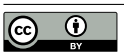
CCS Concepts

• **Security and privacy → Malware and its mitigation**; • **Computing methodologies → Neural networks**.

Keywords

Malware datasets, incorrect labels, supervised learning from noisy labels

ACM Reference Format:
Fahad Alotaibi, Euan Goodbrand, and Sergio Maffeis. 2025. Deep Learning from Imperfectly Labeled Malware Data. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765197



CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765197

1 Introduction

Malware detection and classification have become pressing challenges in the cybersecurity landscape. In 2024 alone, AV-Test[1] identified more than 100 million new malware samples, illustrating both the rapid proliferation of malicious software and the complexity of detecting it in a timely manner. While traditional rule-based techniques, such as signature matching, still play a role, the sheer volume and polymorphism of contemporary threats demand more adaptable strategies. Consequently, researchers have turned to deep learning methods that leverage large, labeled malware datasets to train models capable of identifying malicious variants with high accuracy [18, 55]. These methods also allow for relatively efficient updates using only modest amounts of new data, thereby keeping pace with the fast-evolving nature of modern malware [7, 64].

Despite the growing success of deep learning, constructing large, accurately labeled datasets for malware classification remains a significant challenge. Ideally, every sample would be carefully validated by multiple security experts to ensure high-quality annotations, but this is infeasible when handling tens or hundreds of thousands of samples. As a result, the community often relies on automated labeling from Anti-Virus (AV) engines, with VirusTotal (VT), an aggregator of multiple AV reports, being a popular choice. To efficiently build large datasets, these reports are typically combined using threshold-based heuristics [1, 5–7, 34, 36]. Unfortunately, such heuristics can be error-prone because VT results for the same file can change over time: a file initially flagged as benign or grayware may later be reclassified as malicious, or it may be assigned to a different family as more information emerges [16, 29, 58, 61]. As a result, relying on these reports produces large but imperfect datasets, where some samples inevitably end up with incorrect labels, collectively referred to as label noise [16, 58, 61].

Imperfect datasets introduce two major complications for Machine Learning (ML) experiments. First, when learning-based models are trained on data with incorrect labels, they tend to overfit these errors, which reduces their performance on real-world data (*i.e.*, the data encountered when the model is deployed in practice) [29, 61]. Second, evaluating models on validation or holdout sets that contain labeling errors yields misleading performance metrics. Consequently, researchers may prematurely embrace approaches that appear effective based on these flawed evaluations, even though such methods may simply be learning noise rather than meaningful patterns [58].

[1] https://portal.av-atlas.org/malware

To address label noise in malware datasets, two principal research directions have emerged. The first focuses on revising the label combination process itself. Researchers might filter out less reputable AV engines [1], collect multiple reports over extended periods for greater consistency [5, 36], or apply secondary ML tools to VT outputs to infer more reliable labels [40, 52]. Although these techniques may help reduce the proportion of mislabeled samples, each has inherent drawbacks. Extended data collection, for instance, may delay the release of newly gathered datasets by months or even years, limiting their immediate usefulness. Meanwhile, filtering out specific AV engines is complicated by the correlated behavior among AV products [58, 61].

The second direction addresses label noise during the training process. This research area is relatively new, and to the best of our knowledge, only three efforts [58, 61, 62] specifically target label noise in malware datasets. Two of these [58, 62] focus on binary Android detection and introduce additional constraints. `MalWhiteOut` [58] requires prior domain knowledge, whereas Differential Training [62] demands multiple sequential training phases that are often impractical in real-world contexts. The only approach that generalizes to detection and multi-class classification across a broader range of malware datasets is `MORSE` [61], which adopts a Semi-Supervised Learning (SSL) strategy. Technically, it trains the supervised loss on a small curated clean subset and applies consistency regularization to the remaining data. Although this setup reduces reliance on large amounts of perfectly labeled data, the limited size of the strongly supervised subset can restrict a deep model's ability to learn robust patterns—particularly since larger labeled datasets generally improve performance [48]. This highlights the need for methods that can leverage imperfectly labeled malware data more effectively without drastically shrinking the supervised training set.

In this work, we introduce Selective Label Bootstrapping (`SLB`), a technique designed to mitigate label noise by progressively correcting and expanding an automatically identified clean subset. `SLB` operates in two stages. First, an ensemble of training epochs partitions the dataset into (1) a high-confidence clean subset, containing samples whose predictions consistently align with their observed (given) labels, and (2) a noisy subset containing ambiguous or inconsistently labeled samples. The noisy subset is assigned pseudo-labels to address its uncertain annotations. Second, `SLB` performs continuous label revision throughout training. Samples in the noisy set that repeatedly match either their observed label or pseudo-label are promoted to the clean subset, occasionally flipping labels if necessary. Conversely, samples in the clean set that exhibit inconsistent predictions are reclassified as noisy. This iterative correction mechanism steadily denoises the dataset, expands the clean set, and reduces overall label noise. At the same time, the classifier trained in parallel becomes increasingly resilient to noise and can be used directly for multi-class classification or detection tasks.

We evaluated `SLB` on five diverse datasets, including VirusShare 2018 [27], Malware PE [61], and APIGraph [7], covering both real-world (VT-based) and synthetic (randomly generated) label noise at varying rates. At high noise levels (70% synthetic), `SLB` improved the baseline model's macro F1 score by 13.55% on VirusShare 2018 and 26.32% on Malware PE. Even under a real-world noise rate of 31.44% in VirusShare 2018, `SLB` achieved a 10.12% macro F1 improvement, while having minimal effect in noise-free settings. Moreover, `SLB` outperformed state-of-the-art label-noise handling techniques for malware data, `MORSE` [61] and Differential Training [62], while requiring fewer computational resources.

Beyond improving classification robustness, `SLB` substantially revises dataset labels. For example, it reduced 25% injected noise in APIGraph to below 1.5% and lowered 30% noise in Malware PE to under 6%. This label refinement also boosts classical ML models: on Malware PE with 70% noise, Random Forest (RF) [4] accuracy increased from 59.36% to 86.33%. Furthermore, we evaluated `SLB` with advanced deep architectures (*e.g.*, ResNet18 [13]), conducted ablation studies of its core components, and performed a hyperparameter sensitivity analysis. Collectively, these experiments demonstrate that `SLB` generalizes across diverse models, each of its components is crucial to overall effectiveness, and it remains robust even under suboptimal hyperparameter settings.

In summary, the contributions of this work are as follows:

(1) We propose an ensemble-based data cleaning technique that aggregates predictions across multiple epochs to identify correctly labeled samples. This method partitions the dataset into clean and noisy subsets, assigning high-quality pseudo-labels to uncertain samples.
(2) We introduce a continuous revision strategy that dynamically updates sample labels during training. This strategy leverages both observed labels and assigned pseudo-labels, employing a label-flipping mechanism to promote samples from the noisy set to the clean set.
(3) We integrate these components into a single framework, `SLB`, which continuously revises labels, reduces noise in malware datasets, and simultaneously trains a robust classifier. To encourage reproducibility and further research, we release our code and datasets.[2]
(4) We extensively evaluate `SLB` across diverse deep learning architectures (*e.g.*, ResNet18), datasets (*e.g.*, Android and Windows), and ML algorithms (*e.g.*, RF) under both real-world and synthetic noise at varying levels. Our results demonstrate that `SLB` substantially improves model robustness and generalization while yielding cleaner datasets. It outperforms existing binary and multi-class label-noise handling methods for malware and offers greater cost efficiency.

2 Background

This section provides essential background information. First, we detail malware data collection and labeling methods, outlining their capabilities, limitations, and challenges (§2.1). Second, we review label noise handling techniques from non-security domains and their applicability to malware detection (§2.2).

2.1 Malware Datasets Construction

Accessing large volumes of malicious and benign software is relatively easy due to the abundance of online repositories and distribution platforms (*e.g.*, AndroZoo[3], VT[4], and VirusShare[5]). For instance, as of April 14, 2025, AndroZoo alone hosts more than 25

[2] https://doi.org/10.5281/zenodo.16924658
[3] https://androzoo.uni.lu
[4] https://www.virustotal.com
[5] https://www.virusshare.com

million APK files. However, the primary challenge lies in accurately labeling these software samples, a crucial step for training effective ML models for malware detection and classification. The literature identifies two main approaches to labeling: manual labeling and VT-based labeling.

Manual labeling is the most reliable method, where security experts analyze each sample individually to assign its binary or multi-class label. This can be achieved through dynamic and statistical analysis of the sample's code, behavior, and other characteristics, or by reviewing previously published security reports and threat intelligence data [16]. Despite its high accuracy, manual labeling is slow, costly, and labor-intensive, making it impractical for large-scale datasets. Studies estimate that labeling a single previously unseen malware sample takes an average of 10 hours, emphasizing the substantial time and resource demands of this approach [28].

VT-based labeling is the most widely used method due to its scalability and ease of implementation. A software sample or its hash is submitted to the VT platform, which returns detection results from multiple AV engines. Security experts then assign a label based on predefined thresholds [1, 5, 19, 26, 34, 53, 56], often guided by intuition or insights from previous dynamic analysis studies [57, 69]. For instance, a sample is labeled as malware if at least four AV engines detect it as malicious, while it is considered benign if none flag it as such [34]. Despite its scalability, VT-based labeling introduces label noise, leading to imperfectly labeled datasets. For example, Joyce *et al.* [16] analyzed discrepancies between labels obtained from previously published security reports and VT-based labeling, finding that only 62.10% of binary labels and 46.78% of multi-class labels matched.

2.2 Learning from Imperfectly Labeled data

Deep learning-based models require large and diverse datasets to achieve strong generalization. However, manual labeling at scale is costly and time-consuming, making it impractical for many fields beyond security, including computer vision and natural language processing. To address this challenge, researchers have explored alternative approaches to label large datasets. For example, a common strategy is using crowdsourcing platforms such as Amazon Mechanical Turk (MTurk), where multiple human annotators contribute to dataset labeling [59, 60]. Although this method provides scalability, it introduces variability in label quality due to differences in annotator expertise, leading to potential human errors [14, 59]. Wei *et al.* studied this issue by using MTurk to label CIFAR-10, assigning each sample to three annotators [59]. Their analysis revealed that even with majority voting, the dataset contained 9.03% label noise, highlighting the challenges of ensuring annotation accuracy.

Due to these challenges, extensive research efforts have focused on developing methods to mitigate label noise during model training across various domains. Figure 1 illustrates the most common strategies, which are sample selection and label correction.

Sample selection identifies a subset of training data, known as the clean set, that is likely free of label noise. The model is then trained using only this subset, computing the supervised loss exclusively on the selected samples. A common approach to constructing the clean set is to use the *small-loss trick*, which assumes that samples with low-loss values are more likely to be correctly labeled [12, 65].

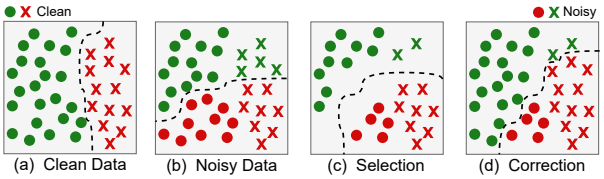


Figure 1: Learning from Label Noise.

Based on this principle, the model selects low-loss samples and trains only on them. Figure 1c illustrates this process, comparing it to a training set that is noise-free (Figure 1a) and a noisy training dataset (Figure 1b). As observed, the selection process favors easier samples—those in densely populated regions and far from the decision boundary. Although this approach mitigates the effect of label noise, it can result in an oversimplified decision boundary by overlooking hard-to-classify, correctly labeled samples that are potentially highly informative. To address this limitation, subsequent research has incorporated non-selected samples into training by down-weighting their influence [38, 44] or applying consistency regularization techniques, such as Mean Teacher [51] and FixMatch [45], to retain useful information [30, 66].

Label correction leverages the entire training dataset by revising incorrectly labeled samples rather than discarding them (Figure 1d). The goal is to dynamically adjust incorrect labels during training, improving data quality and model reliability. Unlike sample selection, which filters out suspected mislabeled data (Figure 1c), label correction retains all data points, allowing more samples to contribute to supervised learning and potentially enhancing model performance. Reed *et al.* [37] introduced a bootstrapping method that, in each training epoch, blends the original labels with the model's current predictions to gradually correct mislabeled data. Lu and He [23] refined this approach using self-ensembling, which blends the original labels with predictions aggregated over multiple training epochs, resulting in more robust and reliable label corrections. However, because these methods revise every sample, they risk changing labels that were originally correct. To address this, Zheng *et al.* [68] proposed error-bounded correction, modifying labels only when the model is sufficiently confident, reducing the risk of incorrect revising.

Applications to malware data. Despite their success in non-security domains, these methods often struggle when applied directly to noisy malware datasets. Wang *et al.* [58] emphasized that these methods require further revisions to achieve meaningful results on malware datasets. Wu *et al.* [61] expanded on this by systematically evaluating eight label noise handling methods on malware datasets and found that they underperformed. They attributed this to three key reasons. First, real-world malware datasets contain a significantly higher proportion of incorrect labels compared to other datasets. Second, noise is distributed unevenly across classes, with some being disproportionately affected. Third, malware datasets often suffer from severe class imbalances, a challenge that these methods are not inherently designed to handle effectively.

3 SLB

In this Section, we introduce SLB, a framework designed to robustly train malware classifiers under label noise while simultaneously producing a cleaner dataset. We begin by formally defining the problem setup (§3.1), then provide a high-level overview of the framework (§3.2). Finally, we describe the two main components of SLB: *Data Split* (§3.3) and *Continuous Revision* (§3.4).

3.1 Problem Definition

We consider a malware dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where each of the N software samples is represented by a feature vector $x_i \in \mathbb{R}^d$. Depending on the analysis technique, these features can be extracted from static properties (*e.g.*, strings, imports, section metadata) or from dynamic behaviors (*e.g.*, system call traces). Each sample belongs to one of K classes, denoted by y_i. In a multi-class *malware classification* setting, these classes typically include multiple malicious families plus a benign class, whereas in a *malware detection* scenario, the task might be binary (benign vs. malicious).

In practice, large-scale labeling often relies on combining detection outputs from multiple AV engines to form the observed label $\tilde{y}_i$. However, discrepancies and errors in these AV outputs can create a significant fraction of incorrect labels, such that $\tilde{y}_i \neq y_i$. When a Deep Neural Network (DNN) is trained on $\tilde{\mathcal{D}} = \{(x_i, \tilde{y}_i)\}_{i=1}^N$ using the cross-entropy loss

$$L_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \log p_i(\tilde{y}_i) \tag{1}$$

where $p_i(\tilde{y}_i)$ is the softmax probability of $\tilde{y}_i$, it may overfit to mislabeled samples, degrading detection and classification performance in real-world scenarios [58, 61].

Our objective is twofold. First, we aim to learn a robust classifier

$$\hat{f}_\theta : \mathbb{R}^d \rightarrow \{1, \ldots, K\}$$

that can effectively mitigate the impact of label noise and generalize well on unseen data. Second, we seek to produce a revised dataset

$$\hat{\mathcal{D}} = \{(x_i, \hat{y}_i)\}_{i=1}^N$$

whose labels $\hat{y}_i$ closely approximate the true labels y_i, formally:

$$\hat{y}_i \approx y_i.$$

By systematically identifying and handling mislabeled samples, we preserve the valuable diversity of the malware dataset, strengthening the classifier's robustness and generalization while also leaving behind a dataset with substantially fewer incorrect labels.

3.2 Overview of the SLB Framework

The SLB framework is built on two key ideas that help combat label noise in malware datasets. The first idea leverages the tendency of DNNs to learn simpler, correctly labeled examples more quickly than mislabeled or complex ones [2, 12]. By tracking the consistency of each sample's predictions over multiple training epochs, we derive an initial partition of the dataset $\tilde{\mathcal{D}}$ into a high-confidence clean set $\mathcal{D}^c$ and a noisy set $\mathcal{D}^n$. For samples deemed noisy, we also generate a pseudo-label that may later replace the originally observed label.

Algorithm 1 Data Split

Require: Noisy dataset $\tilde{\mathcal{D}} = \{(x_i, \tilde{y}_i)\}_{i=1}^N$, epochs e, class-balanced loss hyperparameter β

Initialize model f_θ^{DS}
for $t = 1$ to e **do** ▷ Model training
  Train f_θ^{DS} on $\tilde{\mathcal{D}}$ using the loss:
  $L_{\text{CB}}^{\text{noisy}} = -\frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{EN}_{\tilde{y}_i}} \log\big(p_i(\tilde{y}_i)\big)$, where $\text{EN}_{\tilde{y}_i} = \frac{1-\beta^{n_{\tilde{y}_i}}}{1-\beta}$
  Save predicted label $\hat{y}_{i,t}$ for each sample x_i in $\mathbf{P}_i$
end for
$\mathcal{D}^c \leftarrow \emptyset, \quad \mathcal{D}^n \leftarrow \emptyset$
for $i = 1$ to N **do** ▷ Data partitioning
  Compute consistency ratio:
  $r_i = \frac{|\{\, \hat{y} \in \mathbf{P}_i : \hat{y} = \tilde{y}_i \,\}|}{e}$
  if $r_i = 1$ **then**
    Mark x_i as clean:
    $\mathcal{D}^c \leftarrow \mathcal{D}^c \cup \{(x_i, \tilde{y}_i)\}$
  else
    Derive pseudo-label:
    $y_i^{\text{maj}} \leftarrow \text{majority}(\mathbf{P}_i)$
    Mark x_i as noisy:
    $\mathcal{D}^n \leftarrow \mathcal{D}^n \cup \{(x_i, \tilde{y}_i, y_i^{\text{maj}})\}$
  end if
end for
return $\mathcal{D}^c$ and $\mathcal{D}^n$

The second idea addresses the fact that relying on a single, static split can inadvertently discard genuinely correct but harder-to-learn samples, while still keeping mislabeled examples in the clean set. To resolve these issues, SLB incorporates a continuous revision stage that periodically re-checks each sample's predictions using an Exponential Moving Average (EMA). During revision, samples can move between $\mathcal{D}^c$ and $\mathcal{D}^n$, and their assigned label can switch between the observed label and the pseudo-label, depending on the model's evolving confidence.

3.3 Data Split

The first component of SLB is the *Data Split*, where we divide the observed dataset

$$\tilde{\mathcal{D}} = \{(x_i, \tilde{y}_i)\}_{i=1}^N$$

into a clean set $\mathcal{D}^c$ and a noisy set $\mathcal{D}^n$. One widely used strategy for this split is the *small-loss trick* [12, 61]. In this approach, the model trains for a brief warm-up period, and the final-epoch training losses are used to rank samples. Those with the lowest loss (*e.g.*, the bottom 15% [61]) are designated as clean.

This simple strategy suffers from several limitations. Setting the threshold is difficult without access to a clean validation set or prior knowledge of the noise rate. Additionally, it struggles to handle class-specific noise effectively, particularly when certain classes are underrepresented or frequently mislabelled. Moreover, it is highly sensitive to the choice of warm-up epoch; if the model has not stabilized or begins to overfit during this period, it can lead to substantial noise in the resulting clean set.

To address these challenges, we propose a method that automatically adapts the size of the clean set based on the model's evolving

predictions during the warm-up period. Rather than relying on a single epoch's loss values, we collect and aggregate each sample's hard-label predictions across warm-up epochs. If the model consistently predicts the same label as the observed label, we classify the sample as clean; otherwise, we mark it as noisy. In this way, we reduce the need for strict global thresholds or per-class adjustments, and therefore, this method is more robust to different noise levels and distributions. Algorithm 1 details the step-by-step process.

Epoch-wise Training and Prediction Consistency. To detect label noise, we initially train a single DNN model, f_θ^{DS}, for a predetermined warm-up period consisting of e epochs. At each epoch t during this period, we obtain and record the model's prediction $\hat{y}_{i,t}$ for every sample x_i. These predictions are aggregated into a prediction set:

$$\mathbf{P}_i \;=\; \{\hat{y}_{i,t} \;\mid\; t = 1, \ldots, e\}. \tag{2}$$

Our method builds on the intuition that predictions for clean samples generally stabilize quickly during early epochs, whereas noisy samples exhibit noticeable fluctuations. This is distinct from the existing ensemble-based approach [46], which relies on multiple models and use early training snapshots to improve inference-time prediction quality. Instead, we utilize the prediction consistency from a single model's multiple epochs explicitly for data cleansing purposes prior to final training.

To measure the consistency of predictions, we define a consistency ratio r_i as the fraction of epochs in which the model's predicted label matches the observed (potentially noisy) label $\tilde{y}_i$:

$$r_i = \frac{|\{\, \hat{y} \in \mathbf{P}_i : \hat{y} = \tilde{y}_i \,\}|}{e}. \tag{3}$$

Partitioning into Clean and Noisy Sets. After gathering predictions from the epoch-wise warm-up period, we partition the dataset into clean and noisy subsets based on the consistency of predictions. Using the consistency ratio r_i, computed as defined previously, we apply a strict threshold to determine set membership:

$$\begin{aligned} \mathcal{D}^c &= \big\{(x_i, \tilde{y}_i) \,\big|\, r_i = 1\big\}, \\ \mathcal{D}^n &= \big\{(x_i, \tilde{y}_i, y_i^{\text{maj}}) \,\big|\, r_i < 1\big\}. \end{aligned} \tag{4}$$

Samples with complete prediction consistency across all epochs ($r_i = 1$) are confidently labeled as clean and form the clean set $\mathcal{D}^c$. These represent highly reliable examples that the model consistently classifies correctly throughout the brief training phase.

Conversely, samples with any inconsistency ($r_i < 1$) are considered noisy $\mathcal{D}^n$. For these samples, we further compute a pseudo-label, y_i^{maj}, representing the most frequently predicted label across the warm-up epochs:

$$y_i^{\text{maj}} = \text{majority}\big(\mathbf{P}_i\big), \tag{5}$$

where $\mathbf{P}_i$ is the aggregated prediction set from epoch-wise training.

The clean set thus collects examples the epochs consistently agrees upon, while the noisy set retains samples that exhibit any disagreement. Crucially, storing both the observed label and the majority-voted pseudo-label for noisy samples preserves opportunities for future correction. By structuring the data in this way, we ensure that the initial training can focus on high-confidence labels, laying a strong foundation for the *Continuous Revision* phase that will further interrogate $\mathcal{D}^n$ and resolve lingering ambiguities.

Class-Balanced Loss. Although the above procedure helps distinguish clean vs. noisy samples, class imbalance can severely impede its effectiveness. In many practical malware datasets, one class (*i.e.*, the goodware class) can account for as much as 90% of the data, leaving only 10% for malware samples [34]. If we then further split those malware samples into multiple families, each family may represent only a small fraction of the overall dataset. Under such skewed conditions, empirical risk minimization drives the model to prioritize the majority class, as this leads to the largest reduction in training error. As a result, the model may show high consistency on the majority class and, in some cases, even overfit to its noisy labels. Meanwhile, truly correct samples in underrepresented classes may never achieve perfect consistency, because the model lacks sufficient incentive to learn their characteristics thoroughly.

To address these challenges, we adopt the Class-Balanced (CB) loss [8] during model training. For each class y_b with n_{y_b} samples, we compute its Effective Number (EN):

$$\text{EN}_{y_b} = \frac{1 - \beta^{n_{y_b}}}{1 - \beta}, \tag{6}$$

where $\beta \in [0, 1)$ is a hyperparameter that controls the degree of penalization based on class size. Each sample's loss term is weighted by $1/\text{EN}_{\tilde{y}_i}$, thereby giving minority classes greater influence in the training objective. Concretely, we modify the standard cross-entropy loss in Eq. (1) to

$$L_{\text{CB}}^{\text{noisy}} \;=\; -\frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{EN}_{\tilde{y}_i}} \log\big(p_i(\tilde{y}_i)\big), \tag{7}$$

where $p_i(\tilde{y}_i)$ denotes the model's softmax probability assigned to the observed label $\tilde{y}_i$. By re-weighting samples in this way, the model learns to pay more attention to underrepresented classes, which helps it better identify mislabeled examples within those classes. This becomes especially valuable when the majority class harbors significant noise, since otherwise the network might memorize those incorrect labels and neglect the rare classes, ultimately preventing truly correct minority-class samples from being recognized as clean.

3.4 Continuous Revision

After partitioning the dataset into a clean subset $\mathcal{D}^c$ and a noisy subset $\mathcal{D}^n$ during the *Data Split* stage, SLB enters an iterative *Continuous Revision* phase, as described in Algorithm 2. In contrast to prior approaches that discard or ignore noisy samples entirely [12, 30], our method accounts for the possibility that some noisy samples may be correctly labeled or pseudo-labeled, and that some samples in the clean set may be mislabeled. To handle this uncertainty, we leverage both $\mathcal{D}^c$ and $\mathcal{D}^n$ to iteratively revise label quality and expand the set of reliably labeled samples.

EMA Initialization from Epoch-wise Predictions. Before training begins, we initialize an EMA of soft predictions for each sample x_i. Formally, we define:

$$\bar{p}_i^{(0)} = \frac{1}{e} \sum_{t=1}^{e} p_{i,t}, \tag{8}$$

Algorithm 2 Continuous Revision

Require: Noisy dataset $\tilde{\mathcal{D}}$, observed labels $\tilde{y}_i$, origin flags $o_i \in \{\mathsf{c}, \mathsf{n}\}$, pseudo-labels y_i^{maj}, initial soft predictions $\bar{p}_i^{(0)}$, total epochs T, warm-up $m < T$, smoothing $\alpha \in (0, 1)$

Initialize model $\hat{f}_\theta$
for $r = 1$ to m **do** ▷ Warm-up phase
  Train f_θ for one epoch on $\mathcal{D}_0^c$
  for $i = 1$ to N **do**
    $p_i^{(r)} = \text{softmax}(\hat{f}_\theta(x_i))$; $\bar{p}_i^{(r)} = \alpha p_i^{(r)} + (1-\alpha)\bar{p}_i^{(r-1)}$
  end for
end for
for $i = 1$ to N **do** ▷ Derive initial hard labels
  $\hat{y}_i^{(m)} = \arg\max \bar{p}_i^{(m)}$
end for
$\mathcal{D}_{m+1}^c \leftarrow \emptyset$, $\mathcal{D}_{m+1}^n \leftarrow \emptyset$
for $i = 1$ to N **do** ▷ Rebuild sets after warm-up
  if $\hat{y}_i^{(m)} = \tilde{y}_i$ **then**
    $\mathcal{D}_{m+1}^c \leftarrow \mathcal{D}_{m+1}^c \cup \{(x_i, \tilde{y}_i)\}$
  else if $o_i = \mathsf{n} \wedge \hat{y}_i^{(m)} = y_i^{\text{maj}}$ **then**
    $\mathcal{D}_{m+1}^c \leftarrow \mathcal{D}_{m+1}^c \cup \{(x_i, y_i^{\text{maj}})\}$
  else
    $\mathcal{D}_{m+1}^n \leftarrow \mathcal{D}_{m+1}^n \cup \{(x_i, \tilde{y}_i, \mathtt{maj})\}$, where $\mathtt{maj} \leftarrow \tilde{y}_i$ if $o_i = \mathsf{c}$, else y_i^{maj}
  end if
end for
for $t = m + 1$ to T **do** ▷ Continuous refinement
  Train $\hat{f}_\theta$ for one epoch on $\mathcal{D}_t^c$
  for $i = 1$ to N **do**
    $p_i^{(t)} = \text{softmax}(f_\theta(x_i))$; $\bar{p}_i^{(t)} = \alpha p_i^{(t)} + (1-\alpha)\bar{p}_i^{(t-1)}$
    $\hat{y}_i^{(t)} = \arg\max \bar{p}_i^{(t)}$
  end for
  $\mathcal{D}_{t+1}^c \leftarrow \emptyset$, $\mathcal{D}_{t+1}^n \leftarrow \emptyset$
  for $i = 1$ to N **do** ▷ Update label sets
    if $\hat{y}_i^{(t)} = \tilde{y}_i$ **then**
      $\mathcal{D}_{t+1}^c \leftarrow \mathcal{D}_{t+1}^c \cup \{(x_i, \tilde{y}_i)\}$
    else if $o_i = \mathsf{n} \wedge \hat{y}_i^{(t)} = y_i^{\text{maj}}$ **then**
      $\mathcal{D}_{t+1}^c \leftarrow \mathcal{D}_{t+1}^c \cup \{(x_i, y_i^{\text{maj}})\}$
    else
      $\mathcal{D}_{t+1}^n \leftarrow \mathcal{D}_{t+1}^n \cup \{(x_i, \tilde{y}_i, \mathtt{maj})\}$, where $\mathtt{maj} \leftarrow \tilde{y}_i$ if $o_i = \mathsf{c}$, else y_i^{maj}
    end if
  end for
end for
return $\mathcal{D}_{T+1}^c$, $\mathcal{D}_{T+1}^n$, $p_i^{(T+1)}$, and final model $\hat{f}_\theta$

where $p_{i,t}$ denotes the softmax output of the model f_θ^{DS} at epoch t during the *Data Split* stage. This aggregated prediction acts as an ensemble-based "teacher" signal, smoothing out noisy fluctuations from individual epochs and providing a stable initialization for subsequent revision.

Warm-up Training on $\mathcal{D}^c$. We then perform a warm-up phase by training a new model $\hat{f}_\theta$ solely on the clean set D^c for m epochs. At the end of each epoch, the model produces soft predictions $p_i^{(r)}$ for each sample x_i from $\tilde{\mathcal{D}}$, which are incorporated into the EMA:

$$\bar{p}_i^{(r)} = \alpha\, p_i^{(r)} + (1 - \alpha)\, \bar{p}_i^{(r-1)}, \tag{9}$$

where $\alpha \in (0, 1)$ is a smoothing parameter and $\bar{p}_i^{(0)}$ is initialized from the *Data Split* stage. After the final warm-up epoch, a hard label is assigned to each sample based on the EMA:

$$\hat{y}_i^{(m)} = \arg\max \bar{p}_i^{(m)}. \tag{10}$$

These revised labels serve as the starting point for the iterative revision process that follows.

Rebuilding the Datasets. Immediately after obtaining the updated hard labels $\hat{y}_i^{(m)}$, we reassemble the dataset by revising the assignments to $\mathcal{D}^c$ and $\mathcal{D}^n$. Specifically, a sample $x_i \in \mathcal{D}^c$ is demoted to $\mathcal{D}^n$ if its revised label $\hat{y}_i^{(m)}$ no longer matches its observed label $\tilde{y}_i$. Conversely, a sample $x_i \in \mathcal{D}^n$ is promoted to $\mathcal{D}^c$ if $\hat{y}_i^{(m)}$ matches either its observed label $\tilde{y}_i$ or its pseudo-label y_i^{maj}.

This mechanism recognizes that the initial separation could be imperfect, allowing incorrectly excluded samples to become part of the clean set once the model's predictions align. It also expels from $\mathcal{D}^c$ any samples that appear to have been mislabeled or were too complex to retain their observed label.

Iterative Revision over T Epochs. After the first reassembly, SLB continues in iterative fashion for T revision epochs. At the start of each epoch t, we train the model $\hat{f}_\theta$ on the current revised clean set $\mathcal{D}_t^c$ for one pass. Next, the newly obtained soft predictions $p_i^{(t)} = \text{softmax}(\hat{f}_\theta(x_i))$ for each sample x_i are used to update the EMA:

$$\bar{p}_i^{(t)} \;=\; \alpha\, p_i^{(t)} \;+\; (1 - \alpha)\, \bar{p}_i^{(t-1)}, \tag{11}$$

A new hard label $\hat{y}_i^{(t)} = \arg\max \bar{p}_i^{(t)}$ is then derived. Finally, the sets $\mathcal{D}_t^c$ and $\mathcal{D}_t^n$ are rebuilt by comparing $\hat{y}_i^{(t)}$ with each sample's observed label $\tilde{y}_i$ and pseudo-label y_i^{maj}.

Repeating these steps over T epochs systematically adjusts which samples reside in the clean or noisy subsets. If the EMA consistently favors either $\tilde{y}_i$ or y_i^{maj} for a sample in $\mathcal{D}_n$, that sample migrates into $\mathcal{D}_c$. Conversely, if a sample in $\mathcal{D}_c$ conflicts with the model's revised predictions, it is moved to $\mathcal{D}_n$. This dynamic approach ensures that the clean set is not fixed to its initial composition and can gradually incorporate valid (but previously misjudged) samples while filtering out entrenched mislabels. As a result, the classifier benefits from a growing reservoir of high-quality labels and avoids being constrained by early-stage partitioning errors.

Flipping Labels Between $\tilde{y}_i$ and y_i^{maj}. A critical facet of *Continuous Revision* is the ability to flip a sample's training label between the observed label $\tilde{y}_i$ and the pseudo-label y_i^{maj}. Specifically, a noisy sample might initially join $\mathcal{D}^c$ with its observed label, yet the EMA predictions could steadily favor y_i^{maj}. In that case, the label is flipped to y_i^{maj} in subsequent epochs to align with the model's increasingly confident predictions. Conversely, if the sample was integrated with y_i^{maj} but later stabilizes around $\tilde{y}_i$, we revert the label to $\tilde{y}_i$. This dynamic relabeling mechanism is vital for correcting errors that emerge during both the *Data Split* and early revision epochs.

SLB *Outputs.* At the end of this iterative training, EMA updates, and data reassembly, SLB yields three primary outcomes: (1) a robust classifier $\hat{f}_\theta$ that is both more robust to label noise and better generalizes to unseen samples, ultimately providing a more reliable foundation for downstream malware detection or family classification tasks, (2) a refined subset $\mathcal{D}^c_{T+1}$ containing samples confidently identified as correctly labeled; and (3) a revised dataset $\hat{\mathcal{D}}$, in which labels are updated based on the final EMA probabilities $p_i^{(T+1)}$, capturing the full dataset while allowing for minor residual label inaccuracies.

4 Evaluation

We evaluated SLB using five security-focused datasets that feature both real-world and synthesized label noise. In the following subsections, we detail our experimental setup (§4.1), outline our experiment design (§4.2), and present the results of our experiments (§4.3, §4.4, and §4.5).

4.1 Experimental Setup

4.1.1 Hardware, Software, and Architecture. All experiments were run on a Linux system (Ubuntu 22.04.5 LTS) with the following hardware: an AMD EPYC 7502P 32-core processor, 128 GB of RAM, and four NVIDIA A40 GPUs (each with 48 GB of memory). The software environment comprised Python 3.8.2, PyTorch 2.1.2, and CUDA 11.8. We used a MultiLayer Perceptron (MLP) with three hidden layers containing 512, 256, and 128 neurons, respectively. Each layer was initialized using Xavier initialization. The MLP network was trained for 140 epochs (*i.e.*, $T = 140$) with the Adam optimizer, a learning rate of 0.001, and a batch size of 128. All experiments and baselines were executed with this configuration unless otherwise stated.

4.1.2 Datasets. We used five security datasets: Malware PE (PE) [61], APIGraph 2017–2018 (AG) [7, 67], VirusShare 2018 (VS18) [27], CICIDS 2017 (IDS17) [22, 43], and Virus-MNIST (VM) [31]. In addition, we included a variant of PE and VS18 with real-world label noise, denoted as PE-Real and VS18-Real. Below, we briefly describe each dataset; additional details are provided in Appendix A.3.

Malware PE (PE). This dataset comprises 6,674 Windows executable samples, comprising 500 benign files and 6,174 malicious binaries from 11 families [61]. Each sample was labeled by at least three security experts, each having over five years of professional experience, making the dataset highly reliable and likely to be free from label noise. To create a test partition, the authors randomly selected 100 samples per class, leaving the remainder for training. They also introduced a variant called PE-Real by injecting label noise derived from VT reports. Specifically, if any AV engine assigned a label different from the given ground-truth, its label was changed accordingly. This resulted in approximately 13.88% of the labels being incorrect.

APIGraph 2017–2018 (AG). This Android dataset originates from [67], with the version we used provided by [7]. It comprises 69,241 samples collected from 2017 to 2018, comprising 62,993 benign and 6,248 malicious samples. The authors collected VT reports between 2022 and 2023[6] and applied a threshold-based heuristic for labeling: a sample is labeled as malicious if at least *sixteen* AV engines flag it, as benign if no detections are reported, and samples with intermediate results (*i.e.*, grayware) are discarded. We randomly split the remaining samples into an 80% training set (55,394 samples) and a 20% test set (13,847 samples).

VirusShare 2018 (VS18). This dataset originally comprised 28,543 Android samples [27], which we labeled by applying the threshold-based heuristic described in [7] to VT reports that we scanned and collected in 2025. Specifically, a sample was labeled as malicious if at least *sixteen* AV engines flagged it, and as benign if no engines reported detections. Samples with intermediate detection counts were considered ambiguous (*i.e.*, grayware) and excluded. Family labels were assigned using AVClass2 [42]. However, we identified some inconsistencies in the labeling. In particular, certain family names were reported by only a single AV engine, and some samples had conflicting or similarly weighted family assignments. For example, the sample with MD5 hash `9c1349f4569637323b8aa5ff8688afd1` was assigned both the `Triada` and `SMSReg` families, each by two engines. To address this, we applied an additional thresholding step: a family label was only accepted if it was supported by more than *five* AV engines. Otherwise, the sample was treated as grayware and excluded from the dataset. After this procedure, 1,417 benign samples and 8,529 malicious samples remained, distributed across 7 malware families. We randomly allocated 20% of the data for testing (1,989 samples) and 80% for training (7,955 samples). To simulate real-world label noise, we created VS18-Real by labeling the training subset exclusively with a single AV engine, following the methodology in [6, 61]. We selected the `Lionic` engine for its comprehensive family coverage, which ultimately produced an incorrect label rate of 31.44%.

CICIDS 2017 (IDS17). Although not malware-focused, this NIDS dataset is included for its highly reliable labels. It was generated in a controlled lab setting with prior knowledge of the attacker's identity and actions, which enabled precise label assignments and minimized the risk of unrecognized label noise. The version used here, obtained from [22], contains about 1.91 million samples representing 15 classes, with roughly 78.28% deemed benign. Owing to its large size, we sampled 12,000 benign instances and 1,920 samples from eight primary attack classes. We allocated 80% of this subset for training (11,600 samples) and 20% for testing (2,320 samples).

Virus-MNIST (VM). This dataset is an image-based representation of Windows portable executables [33]. It comprises 51,880 images, including 2,516 benign samples and 49,364 malicious samples. We used the version extended by [31]. The authors subdivided the malicious class into nine distinct categories using K-means clustering. The dataset comes with a pre-defined split, allocating 3,458 samples to testing and 48,422 samples to training.

4.1.3 Baselines. We evaluate SLB against three baselines: `Vanilla`, Differential Training (`DT`), and `MORSE`. `Vanilla` serves as a simple baseline that trains directly on the noisy dataset $\tilde{\mathcal{D}}$ without any noise-handling mechanism. `DT` is a state-of-the-art approach for noise mitigation in malware detection. It trains two DNNs over n rounds; at the end of each round, it identifies suspicious samples and flips their labels. For instance, if a sample is labeled malicious and flagged as noisy, its label is flipped to benign. However, this binary label-flipping strategy is not applicable to multi-class settings. To

[6] We thank Yizheng Chen for confirming the timeframe of VT report collection.

address this limitation, we re-engineered DT to support multi-class datasets by replacing the binary flip rule with a pseudo-labeling strategy inspired by SLB. Specifically, we aggregate per-epoch predictions during each round and assign the majority-voted label to samples identified as noisy. We refer to this variant as DT modified (DTm). This adjustment also improves DT's performance on the binary dataset (see §4.3.1). MORSE is another state-of-the-art approach for noise mitigation in malware classification. At each epoch, it selects low-loss samples for supervised training and applies consistency regularization to the remaining samples using the FixMatch framework. For all methods (DT, MORSE, and SLB), we performed a dedicated hyperparameter search, with full details provided in Appendix A.2.

4.1.4 Metrics. We repeated each experiment ten times to reduce randomness and obtain reliable performance estimates. For each run, we computed two metrics. The first is accuracy (Acc), a standard measure of the proportion of correctly classified samples. Acc is widely used to evaluate both label correction and model classification performance, especially under label noise [12, 23, 37, 38, 44, 65, 68]. However, as noted in prior work [61], label noise disproportionately affects minority classes. In such cases, Acc alone can be misleading because it may obscure poor performance on underrepresented classes. To address this, we also report the macro F1-score (mF1). This metric computes the F1-score, defined as the harmonic mean of precision and recall, for each class and then averages them. As a result, mF1 gives equal importance to all classes, making it more robust to class imbalance and better suited for noisy, imbalanced datasets. Final results are reported as the average across all ten runs.

4.2 Experiment Design

We designed our experiments to comprehensively evaluate the performance and robustness of SLB under diverse conditions. Our evaluation covers three key dimensions:

Classification Performance (§4.3). We evaluate the effectiveness of the final model $\hat{f}_{\hat{\theta}}$ produced by the *Continuous Revision* phase across three settings: random label noise (§4.3.1), real-world noise (§4.3.2), and clean datasets (§4.3.3). For synthetic noise, we inject random label noise into clean datasets (PE, VS18, AG, IDS17) at rates ranging from 10% to 70% to assess how performance deteriorates as noise increases (§4.3.1). For real-world noise, we simulate label inconsistencies based on VT reports to evaluate the robustness of SLB and baseline methods under realistic conditions (§4.3.2). Finally, we test SLB on noise-free datasets to verify that it does not degrade performance when the labels are already accurate (§4.3.3).

Label Correction Performance (§4.4). We assess the quality of the revised dataset $\hat{\mathcal{D}}$ produced by SLB through two experiments: first, by comparing the corrected labels $\hat{y}_i$ to the ground-truth clean labels y_i to quantify the accuracy of the correction process (§4.4.1); and second, by training a variety of ML models on $\hat{\mathcal{D}}$ to evaluate its downstream utility and generalization beyond our own training framework (§4.4.2).

Sensitivity Analysis (§4.5). We examine SLB's robustness to architecture changes, dataset and model scaling, component ablation, and hyperparameter variation. Specifically, we evaluate SLB on advanced deep learning models with image-based malware representations (§4.5.1); scale model capacity with larger datasets (§4.5.2); ablate components such as continuous revision to measure their contribution under real and synthetic noise (§4.5.3); and assess key hyperparameters, namely the number of epochs e for data split and the warm-up period m for continuous revision (§4.5.4).

Table 1: Performance on completely clean data (mean ± std)

Method	PE		VS18	
	Acc	mF1	Acc	mF1
Vanilla	94.12 ± 0.28	94.04 ± 0.27	82.01 ± 0.20	**82.59** ± 0.94
SLB	**94.31** ± 0.32	**94.33** ± 0.31	**82.21** ± 0.32	82.45 ± 0.60

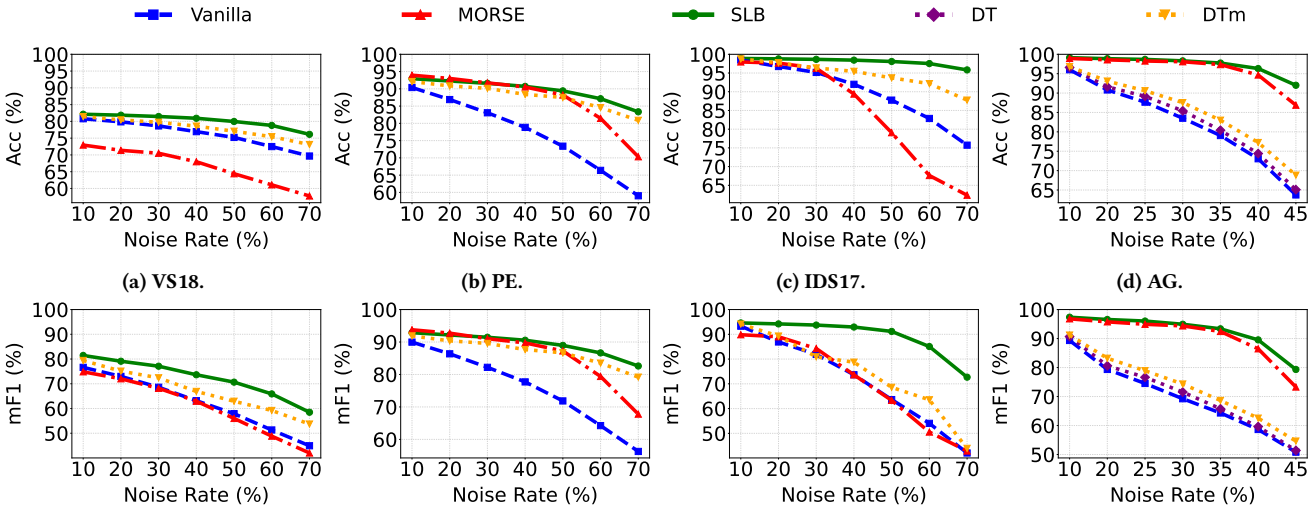
Method	IDS17		AG	
	Acc	mF1	Acc	mF1
Vanilla	**99.54** ± 0.04	**98.08** ± 0.19	**99.36** ± 0.03	**98.06** ± 0.09
SLB	99.06 ± 0.06	95.53 ± 0.48	99.29 ± 0.03	97.85 ± 0.09

4.3 Classification Performance

4.3.1 Performance on Random Noise. We generate random label noise by selecting a fixed percentage of samples from each class and flipping their labels to a different class. This method, which is widely used in both security [58, 61, 62] and non-security [12, 30] research, allows us to control the noise rate and examine its effect on model performance. In our experiments, we varied the noise rate from 10% to 70% across four datasets (VS18, PE, IDS17, and AG), as illustrated in Figure 2. The experimental results reveal three key findings. First, SLB consistently mitigates the impact of label noise across all tested noise rates. For instance, at a 10% noise rate, SLB improves the mF1 score over the Vanilla baseline by 1.3% on IDS17 and by 8.02% on AG. At higher noise levels, this advantage grows even more pronounced: SLB surpasses the Vanilla baseline by 30.62% on IDS17 (70% noise rate) and by 28.51% on AG (45% noise rate). Similar gains are observed in terms of Acc, indicating that SLB improves classification performance for both major and minor classes.

Second, existing State-of-the-Art (SOTA) noise-handling techniques such as MORSE and DT are more sensitive to specific datasets and noise rates. Although MORSE performs well on AG and PE, it often underperforms relative to the Vanilla baseline on VS18 and IDS17. Meanwhile, DT and its re-engineered variant, DTm, show noticeable improvements over Vanilla in most cases, yet their effectiveness still varies by dataset. DTm achieves large performance gains on VS18 and PE but less so on the other two datasets. Notably, DTm consistently outperforms DT on the binary AG dataset, confirming the positive impact of our modifications.

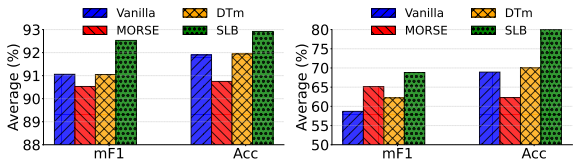
Third, SLB generally surpasses these SOTA methods. For example, under 70% noise on IDS17, SLB exceeds the best SOTA method (DTm) by 28.76% (mF1) and 8.07% (Acc). At lower noise levels, SLB also outperforms SOTA baselines across most datasets, with one minor exception on the PE dataset, where MORSE achieves slightly higher results: gains of 0.9% in mF1 and 1.01% in Acc at 10% noise, and 0.56% in mF1 and 0.73% in Acc at 20% noise.



(a) VS18. (b) PE. (c) IDS17. (d) AG.

(e) VS18. (f) PE. (g) IDS17. (h) AG.

Figure 2: Performance on synthetic noise datasets.



(a) PE-Real. (b) VS18-Real.

Figure 3: Performance on real noise datasets.

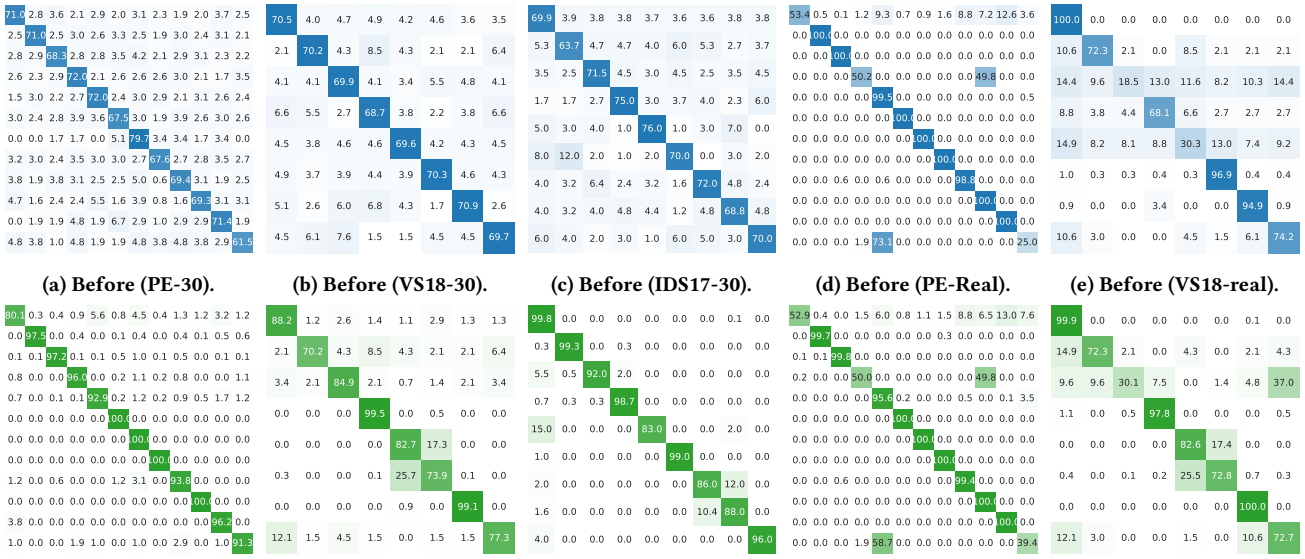
4.3.2 Performance on Real Noise. So far, we have shown that SLB can effectively mitigate random label noise. However, real-world noise tends to be more complex and follows a different distribution. To examine whether our framework can handle such complexities, we evaluated SLB on two real-world noisy malware datasets: PE-Real and VS18-Real. In these datasets, label noise was introduced using threshold-based heuristics on VT reports, mirroring current automatic labeling practices. The results, shown in Figure 3, reveal three key points. First, SLB improves performance under real-world noise, providing measurable gains over the `Vanilla` baseline. For example, SLB increases the mF1 and Acc scores on PE-Real by 1.47% and 1.02%, respectively, and on VS18-Real by 10.12% (mF1) and 11.1% (Acc). Second, these improvements are particularly meaningful because SLB's performance approaches that of a model trained on a noise-free dataset (Table 1). For instance, on PE-Real, SLB achieves a 92.92% Acc, compared to 91.90% for the noisy `Vanilla` baseline and 94.12% for a fully noise-free model. A similar pattern emerges on VS18-Real, where SLB attains an 80.03% Acc, approaching the noise-free model's 82.01% while surpassing the `Vanilla` baseline's 68.93%. Third, SLB consistently surpasses the SOTA techniques: on VS18-Real, it outperforms the best baseline method in terms of mF1 (`MORSE`) by 3.68% and Acc (`DTm`) by 10.01%.

4.3.3 Performance on Clean Datasets. It is often challenging to ascertain whether a malware dataset is entirely free of label noise. Consequently, any noise handling technique should ideally have minimal adverse impact when applied to clean data. To evaluate this, we investigated the effect of SLB on completely clean datasets. Table 1 summarizes the results of this experiment. Out of 8 measurements, in 3 cases SLB outperformed `Vanilla` by 0.23% on average, and in 4 cases it underperformed by the same amount (0.23%, on average). In the worst-case scenario, the mF1 on the IDS17 dataset decreased by 2.55%. Note that already at 10% noise SLB outperforms `Vanilla` on the same dataset by 1.3%, and at 70% it outperforms by 30.62%. This indicates that our approach is robust enough to be worth applying when the noise level is low or uncertain, as it does not significantly harm performance even on perfectly labeled data.

4.4 Label Correction Performance

4.4.1 Labels Accuracy. To evaluate the effectiveness of our label correction process, we compute the accuracy of the final labels $\hat{y}_i$ obtained from $p_i^{(T+1)}$ relative to the true labels y_i. As shown in Figures 4 and 5, SLB substantially reduces the noise rate across different datasets. For instance, in the VS18-Real dataset, the proportion of correctly labeled samples increases from 68.56% to 80.72%. Under random noise conditions, SLB also achieves notable improvements: at a 70% noise rate, it lowers noise to under 15% in PE, 27% in VS18, and 4% in IDS17. On the binary AG dataset with 45% noise rate, SLB reduces it to below 9%.

4.4.2 Performance of ML Models. We show that SLB not only increases the resilience of deep learning models to incorrect labels but also produces corrected labels with high accuracy. Although our initial focus was on deep learning, conventional ML algorithms can also be effective for malware detection and classification (*e.g.*,



(a) Before (PE-30). (b) Before (VS18-30). (c) Before (IDS17-30). (d) Before (PE-Real). (e) Before (VS18-real).

(f) After (PE-30). (g) After (VS18-30). (h) After (IDS17-30). (i) After (PE-Real). (j) After (VS18-Real).

Figure 4: Confusion matrices before and after label revision. Each subfigure is annotated with dataset name and noise rate (*e.g.*, VS18-30 for VS18 with 30% noise).

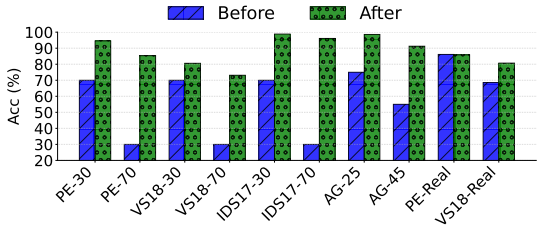


Figure 5: Label accuracy before and after correction.

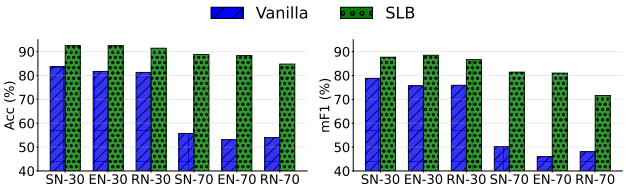
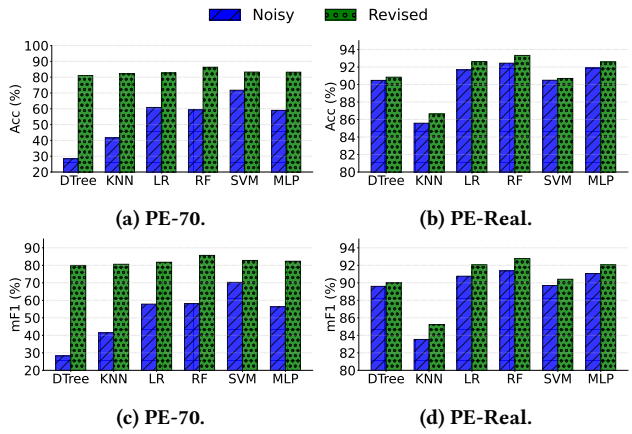


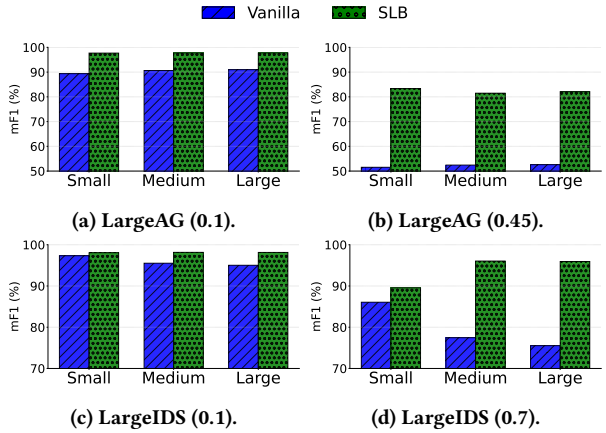
Figure 7: Performance across complex architectures.



(a) PE-70. (b) PE-Real.

(c) PE-70. (d) PE-Real.

Figure 6: Performance of ML models with SLB-revised labels.



(a) LargeAG (0.1). (b) LargeAG (0.45).

(c) LargeIDS (0.1). (d) LargeIDS (0.7).

Figure 8: SLB performance on large datasets and models.

[3, 15]). Using SLB-revised labels, we trained five ML models (Support Vector Machine (SVM), RF, Decision Tree (DTree), Logistic Regression (LR), and K-Nearest Neighbors (KNN)) as well as an MLP classifier. As shown in Figure 6, the revised labels substantially improve their performance.

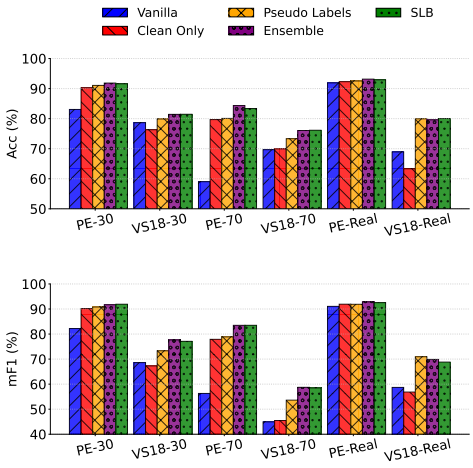
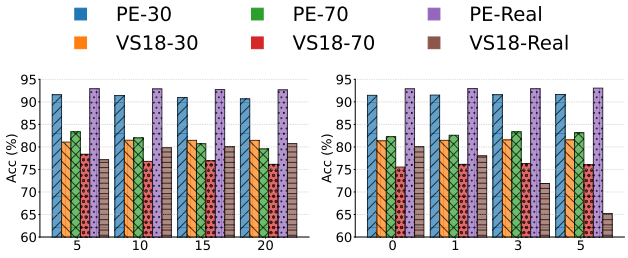


Figure 9: Ablation study.



(a) Data Split Epochs (e). **(b) Revision Warm-up (m).**

Figure 10: Hyperparameter sensitivity.

4.5 Sensitivity Analysis

4.5.1 Performance on Complex Architectures. While SLB demonstrated strong performance on a simple deep learning model with three hidden layers, real-world applications often rely on more complex architectures. To assess its impact on such models, we integrated SLB with three established image-based networks: ResNet18 (RN) [13], EfficientNet B0 (EN) [50], and ShuffleNet v2 (SN) [25]. Experiments were conducted on the Virus-MNIST dataset [31], which provides image-based representations of malware. Owing to the high computational cost (*e.g.*, ResNet18 comprises approximately 11.7 million parameters), we limited experiments to five runs. As shown in Figure 7, SLB substantially improves performance on the noisy dataset, suggesting that the framework generalizes well across diverse deep learning architectures.

4.5.2 Large Datasets vs. Large Models. So far, we have shown that SLB consistently improves performance across both simple and complex models on relatively small datasets. An open question is whether these benefits extend to large-scale settings, where both dataset size and model capacity are substantially higher. To explore this, we evaluate SLB under random label noise on two large benchmarks: LargeIDS (the full IDS17 dataset) and LargeAG (the complete APIGraph dataset from 2012–2018). For each dataset, we train three networks of increasing capacity: a small model ([512, 256, 128]), a medium model ([1024, 1024, 512]), and a large model ([1024, 2048, 1024]). To keep experiments computationally feasible, we fix the batch size to 1024. As shown in Figure 8, SLB consistently produces noise-resistant classifiers across different dataset sizes and model capacities, suggesting that its robustness extends to larger and more demanding scenarios.

4.5.3 Ablation Study. We conduct an ablation study with three alternative configurations to systematically assess our design choices. First, Clean Only omits the continuous revision step and trains a new model solely on the initially identified clean set. Second, Pseudo Labels also removes continuous revision but trains a new model on the entire dataset, relying on pseudo labels generated by majority vote. Third, we explore a more complex partitioning approach by training an ensemble of five models for e epochs each, generating $5 \times e$ predictions per sample during the data split. Figure 9 summarizes the results of these three configurations. Overall, the findings show that preserving all SLB features yields the best results. While Clean Only performs reasonably in some cases, it is inadequate for datasets such as VS18-30 and VS18-Real. Pseudo Labels outperforms the `Vanilla` baseline but usually falls short of full SLB, aside from a slight mF1 advantage on VS18-Real. Finally, Ensemble provides a small performance gain (up to 1.06% mF1 on VS18-Real) but at the cost of higher computation, making it an optional extension of SLB when resources permit.

4.5.4 Hyperparameters Sensitivity. Figure 10 shows the sensitivity of SLB to its hyperparameters. As illustrated in Figure 10a, the hyperparameter e has a measurable but modest effect on performance. Notably, the PE-70 dataset exhibits the largest variation, with performance decreasing by 3.81 when e increases from 5 to 20. Based on these observations, we recommend setting e in the range of 5 to 15, which balances sufficient training iterations against the risk of overfitting to noise. Figure 10b analyzes the impact of the hyperparameter m. For most datasets, the difference between the minimum and maximum performance values is modest (averaging around 0.384). However, VS18-Real is an exception; increasing m in this dataset significantly degrades performance, with scores dropping from 80.03 at $m = 0$ to 78.02 at $m = 1$, 71.88 at $m = 3$, and 65.23 at $m = 5$. This is because initial clean set contain approximately 20% noise. Consequently, we recommend keeping m low (between 0 and 3) to ensure prompt label corrections, enabling the SLB to quickly eliminate any noise present in the initial clean set.

5 Discussion

Application and Generalization of SLB. Our results show that SLB is effective across a wide range of settings, but several challenges remain before it can be confidently deployed in operational environments. First, our real-label noise experiments have focused primarily on VT reports, the most common labeling approach, even though alternative methods (*e.g.*, [16, 39, 63]) may produce different noise patterns. Future work should systematically evaluate SLB under these diverse labeling strategies. Second, while our real-label noise experiments were conducted on two datasets (consistent with current practices [58, 61]), a broader empirical study is needed to build more comprehensive and representative benchmarks. Third,

although we have demonstrated SLB's potential beyond malware classification, our exploration in other security domains has so far been limited to a single NIDS dataset (IDS17). Future research should extend our approach to a wider range of security problems. Finally, although SLB has been validated on real datasets, its long-term performance and robustness in operational settings remain to be studied. Further research is therefore required to confirm SLB's effectiveness over time in real-world environments.

Overhead of SLB. SLB introduces two additional components compared to the Vanilla baseline: a preliminary training phase for e epochs to perform the data split and a continuous label revision process during the main T-epoch training phase. These steps incur a modest overhead relative to the baseline. For example, on the VS18-30 dataset, the Vanilla model trains in approximately *44* seconds, while SLB takes about *50* seconds. In contrast, SOTA methods impose substantially higher costs, with MORSE taking roughly *133* seconds and DTm about *4284* seconds. Thus, we argue that the slight extra cost of SLB is justified by its significant performance gains.

Label-flipping Attacks. SLB is designed to mitigate label noise resulting from legitimate user labeling methods. However, a related challenge is label flipping (or label poisoning) attacks, where an attacker deliberately injects label noise into the training set. Such attacks aim to degrade the classifier's overall performance or induce targeted misclassification [54]. We argue that most existing label noise mitigation techniques may be susceptible to these adversarial manipulations. Consequently, further research is needed to understand how poisoned samples can be crafted to remain undetected as clean, as well as to develop strategies to counter these attacks. We consider this a critical avenue for future work.

6 Related Work

Deep Learning-based Malware Systems. Deep learning has been widely applied to malware classification and detection, tackling various challenges such as detection and classification [17, 24, 47], resilience against evolving malware [9, 10, 21, 67], and detection/adaptation for both new malware variants and evolving malware families [7, 9, 64]. For example, MalDozer [17] trains a convolutional neural network on API calls for Android malware classification. Meanwhile, APIGraph [67] leverages Android documentation to group related APIs, thereby enhancing resilience against diverse malware implementations. CADE [64] adopts a contrastive learning approach to detect novel malware variants. Our approach is related to these previous methods in that it addresses a fundamental challenge in learning-based malware detection systems: label noise. Specifically, SLB leverages deep learning to improve the quality of the training set labels, resulting in both a more resilient model and a dataset with significantly reduced noise.

Learning from Noisy Malware Datasets. Malware datasets are often prone to label noise, yet little research has addressed this issue during training. To the best of our knowledge, only three works explicitly tackle this challenge [58, 61, 62]. Wang *et al.* [58] adapt Confident Learning [32] to extract a clean subset from an imperfectly labeled Android dataset. They relabel noisy samples using app developer information (*e.g.*, ID and name) by clustering samples and assigning labels based on cluster majority. This approach is Android-specific and assumes noisy samples have correctly labeled counterparts from the same developer. Xu *et al.* [62] propose DT, a framework for Android malware detection in low-noise, binary settings. They train two DNN models: one on the full dataset and another on a down-sampled subset. These models generate loss vectors, which are processed by an ensemble of 13 outlier detection algorithms (*e.g.*, One-Class SVM). Detection thresholds are set using a known containment rate (the actual noise rate). Noisy samples are identified, and their labels flipped to the opposite class with a random acceptance probability. Wu *et al.* [61] propose MORSE, a framework applicable to any malware classifier for both binary and multi-class tasks. In each epoch, MORSE splits data into two groups: samples with lowest loss values are used for supervised learning, and the rest for consistency regularization. For regularization, FixMatch is applied, where each sample is augmented twice—once with weak and once with strong augmentation. Both are done using random replacement, where a subset of features is replaced with features from two randomly selected samples, generating two distinct augmented versions. In our experiments, we show that SLB outperforms both DT and MORSE in classification performance while incurring substantially lower training costs.

Other Related Work. Several studies [57, 69] examined VT report dynamics and proposed threshold-based heuristics to smooth label fluctuations over time. Although they provide valuable insights into VT label evolution, their focus is on understanding dynamics rather than directly reducing label noise. As a result, it remains unclear whether these heuristics can lower noise levels in malware datasets. Other works [41, 46] used ensemble learning to address overfitting and enhance prediction reliability in DNN. For instance, Salman and Liu [41] propose a classification-with-rejection strategy that uses multiple models to filter out low-confidence predictions. Stern *et al.* [46] train several models and collect snapshots from each to build a large ensemble, which is then used at inference to identify and correct mispredicted samples. In contrast, our approach achieves effective noise filtration and label correction with a single model during training. This eliminates the need to maintain large ensembles at inference, resulting in a more efficient and practical solution for mitigating label noise in malware datasets.

7 Conclusion

In this paper, we introduced SLB, a framework designed to mitigate the adverse effects of label noise in imperfectly labeled malware datasets. SLB leverages an ensemble of training epochs for noise detection and employs a continuous revision strategy to iteratively refine noisy labels during training. We evaluated SLB across various architectures, noise types, and noise levels with two goals: producing a cleaner training dataset and developing a robust classifier. Our experiments show that SLB effectively reduces label noise and yields classifiers that are resilient to noise. However, our evaluation was limited to VT reports and five datasets. Future work will investigate additional labeling methods, more diverse datasets, and long-term operational performance. Overall, SLB lays a strong foundation for advancing label-noise mitigation in malware datasets.

Acknowledgments

The authors gratefully acknowledge VirusTotal for granting access to their academic API for research use. Alotaibi's research is funded

by a scholarship from the Deanship of Scientific Research at Najran University, Kingdom of Saudi Arabia.

References

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*.

[2] Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron C. Courville, Yoshua Bengio, and Simon Lacoste-Julien. 2017. A Closer Look at Memorization in Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Vol. 70. PMLR, 233–242.

[3] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending TRANSCEND: Revisiting Malware Classification in the Presence of Concept Drift. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 805–823. doi:10.1109/SP46214.2022.9833659

[4] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[5] Zhenquan Cai and Roland H. C. Yap. 2016. Inferring the Detection Logic and Evaluating the Effectiveness of Android Anti-Virus Apps. In *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*. ACM, 172–182. doi:10.1145/2857705.2857719

[6] Mahinthan Chandramohan, Hee Beng Kuan Tan, and Lwin Khin Shar. 2012. Scalable malware clustering through coarse-grained behavior modeling. In *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*. ACM, 27. doi:10.1145/2393596.2393627

[7] Yizheng Chen, Zhoujie Ding, and David A. Wagner. 2023. Continuous Learning for Android Malware Detection. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 1127–1144.

[8] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J. Belongie. 2019. Class-Balanced Loss Based on Effective Number of Samples. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 9268–9277. doi:10.1109/CVPR.2019.00949

[9] Mirabelle Dib, Sadegh Torabi, Elias Bou-Harb, Nizar Bouguila, and Chadi Assi. 2022. EVOLIoT: A Self-Supervised Contrastive Learning Framework for Detecting and Characterizing Evolving IoT Malware Variants. In *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*. ACM, 452–466. doi:10.1145/3488932.3517393

[10] Yujie Fan, Mingxuan Ju, Shifu Hou, Yanfang Ye, Wenqiang Wan, Kui Wang, Yinming Mei, and Qi Xiong. 2021. Heterogeneous Temporal Graph Transformer: An Intelligent System for Evolving Android Malware Detection. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*. ACM, 2831–2839. doi:10.1145/3447548.3467168

[11] 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, Atlanta, GA, USA, October 17-21, 2022*. ACM, 3948–3952. doi:10.1145/3511808.3557533

[12] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 8536–8546.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778. doi:10.1109/CVPR.2016.90

[14] Zeyu He, Chieh-Yang Huang, Chien-Kuang Cornelia Ding, Shaurya Rohatgi, and Ting-Hao Kenneth Huang. 2024. If in a Crowdsourced Data Annotation Pipeline, a GPT-4. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*. ACM, 1040:1–1040:25. doi:10.1145/3613904.3642834

[15] Roberto Jordaney, Kumar Sharad, Santanu Kumar 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 2017, Vancouver, BC, Canada, August 16-18, 2017*. USENIX Association, 625–642.

[16] Robert J Joyce, Dev Amlani, Charles Nicholas, and Edward Raff. 2023. Motif: A malware reference dataset with ground truth family labels. *Computers & Security* 124 (2023), 102921.

[17] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. 2018. MalDozer: Automatic framework for android malware detection using deep learning. *Digital Investigation* 24 Supplement (2018), S48–S59. doi:10.1016/J.DIIN.2018.01.007

[18] Jinsung Kim, Younghoon Ban, Eunbyeol Ko, Haehyun Cho, and Jeong Hyun Yi. 2022. MAPAS: a practical deep learning-based android malware detection system. *International Journal of Information Security* 21, 4 (2022), 725–738. doi:10.1007/S10207-022-00579-6

[19] David Korczynski and Heng Yin. 2017. Capturing Malware Propagations with Code Injections and Code-Reuse Attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 1691–1708. doi:10.1145/3133956.3134099

[20] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy, ICISSP 2017, Porto, Portugal, February 19-21, 2017*. SciTePress, 253–262. doi:10.5220/0006105602530262

[21] 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 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 150:1–150:13. doi:10.1145/3597503.3639122

[22] Lisa Liu, Gints Engelen, Timothy M. Lynar, Daryl Essam, and Wouter Joosen. 2022. Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In *10th IEEE Conference on Communications and Network Security, CNS 2022, Austin, TX, USA, October 3-5, 2022*. IEEE, 254–262. doi:10.1109/CNS56114.2022.9947235

[23] Yangdi Lu and Wenbo He. 2022. SELC: Self-Ensemble Label Correction Improves Learning with Noisy Labels. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*. ijcai.org, 3278–3284. doi:10.24963/IJCAI.2022/455

[24] Jhu-Sin Luo and Dan Chia-Tien Lo. 2017. Binary malware image classification using machine learning with local binary pattern. In *2017 IEEE International Conference on Big Data (IEEE BigData 2017), Boston, MA, USA, December 11-14, 2017*. IEEE Computer Society, 4664–4667. doi:10.1109/BIGDATA.2017.8258512

[25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV*, Vol. 11218. Springer, 122–138. doi:10.1007/978-3-030-01264-9_8

[26] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, Anthony D. Joseph, and J. D. Tygar. 2016. Reviewer Integration and Performance Measurement for Malware Detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, Vol. 9721. Springer, 122–141. doi:10.1007/978-3-319-40667-1_7

[27] Tomás Concepción Miranda, Pierre-François Gimenez, Jean-François Lalande, Valérie Viet Triem Tong, and Pierre Wilke. 2022. Debiasing Android Malware Datasets: How Can I Trust Your Results If Your Dataset Is Biased? *IEEE Transactions on Information Forensics and Security (TIFS)* 17 (2022), 2182–2197. doi:10.1109/TIFS.2022.3180184

[28] Abedelaziz Mohaisen and Omar Alrawi. 2013. Unveiling Zeus: automated classification of malware samples. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*. International World Wide Web Conferences Steering Committee / ACM, 829–832. doi:10.1145/2487788.2488056

[29] Aziz Mohaisen and Omar Alrawi. 2014. AV-Meter: An Evaluation of Antivirus Scans and Labels. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA 2014, Egham, UK, July 10-11, 2014. Proceedings*, Vol. 8550. Springer, 112–131. doi:10.1007/978-3-319-08509-8_7

[30] Duc Tam Nguyen, Chaithanya Kumar Mummadi, Thi-Phuong-Nhung Ngo, Thi Hoai Phuong Nguyen, Laura Beggel, and Thomas Brox. 2020. SELF: Learning to Filter Noisy Labels with Self-Ensembling. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

[31] David A. Noever and Samantha E. Miller Noever. 2021. Virus-MNIST: A Benchmark Malware Dataset. *CoRR* abs/2103.00602 (2021). arXiv:2103.00602

[32] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. 2021. Confident Learning: Estimating Uncertainty in Dataset Labels. *Journal of Artificial Intelligence Research (JAIR)* 70 (2021), 1373–1411. doi:10.1613/JAIR.1.12125

[33] Angelo Oliveira. 2019. Malware Analysis Datasets: Raw PE as Image. doi:10.21227/8brp-j220

[34] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. USENIX Association, 729–746.

[35] Miguel Quebrado, Edoardo Serra, and Alfredo Cuzzocrea. 2021. Android Malware Identification and Polymorphic Evolution Via Graph Representation Learning. In *2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, December 15-18, 2021*. IEEE, 5441–5449. doi:10.1109/BIGDATA52589.2021.9671437

[36] Moheeb Abu Rajab, Lucas Ballard, Noe Lutz, Panayiotis Mavrommatis, and Niels Provos. 2013. CAMP: Content-Agnostic Malware Protection. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*.

[37] Scott E. Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. 2015. Training Deep Neural Networks on Noisy Labels with Bootstrapping. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

[38] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to Reweight Examples for Robust Deep Learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Vol. 80. PMLR, 4331–4340.

[39] Aleieldin Salem. 2021. Towards Accurate Labeling of Android Apps for Reliable Malware Detection. In *CODASPY '21: Eleventh ACM Conference on Data and Application Security and Privacy, Virtual Event, USA, April 26-28, 2021*. ACM, 269–280. doi:10.1145/3422337.3447849

[40] Aleieldin Salem, Sebastian Banescu, and Alexander Pretschner. 2021. Maat: Automatically Analyzing VirusTotal for Accurate Labeling and Effective Malware Detection. *ACM Transactions on Privacy and Security (TOPS)* 24, 4 (2021), 25:1–25:35. doi:10.1145/3465361

[41] Shaeke Salman and Xiuwen Liu. 2019. Overfitting Mechanism and Avoidance in Deep Neural Networks. *CoRR* abs/1901.06566 (2019). arXiv:1901.06566

[42] Silvia Sebastián and Juan Caballero. 2020. AVclass2: Massive Malware Tag Extraction from AV Labels. In *ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*. ACM, 42–53. doi:10.1145/3427228.3427261

[43] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*. SciTePress, 108–116. doi:10.5220/0006639801080116

[44] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. 2019. Meta-Weight-Net: Learning an Explicit Mapping For Sample Weighting. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 1917–1928.

[45] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. 2020. FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[46] Uri Stern, Daniel Shwartz, and Daphna Weinshall. 2024. United We Stand: Using Epoch-Wise Agreement of Ensembles to Combat Overfit. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*. AAAI Press, 15075–15082. doi:10.1609/AAAI.V38I13.29429

[47] Jiawei Su, Danilo Vasconcellos Vargas, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi Sakurai. 2018. Lightweight Classification of IoT Malware Based on Image Recognition. In *2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC 2018, Tokyo, Japan, 23-27 July 2018, Volume 2*. IEEE Computer Society, 664–669. doi:10.1109/COMPSAC.2018.10315

[48] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 843–852. doi:10.1109/ICCV.2017.97

[49] Tiezhu Sun, Nadia Daoudi, Weiguo Pian, Kisub Kim, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2024. Temporal-incremental learning for Android malware detection. *ACM Transactions on Software Engineering and Methodology* (2024).

[50] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Vol. 97. PMLR, 6105–6114.

[51] Antti Tarvainen and Harri Valpola. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 1195–1204.

[52] Saravanan Thirumuruganathan, Mohamed Nabeel, Euijin Choo, Issa Khalil, and Ting Yu. 2022. SIRAJ: A Unified Framework for Aggregation of Malicious Entity Detectors. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 507–521. doi:10.1109/SP46214.2022.9833725

[53] Kurt Thomas, Juan A. Elices Crespo, Ryan Rasti, Jean-Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, Lucas Ballard, Robert Shield, Nav Jagpal, Moheeb Abu Rajab, Panayiotis Mavrommatis, Niels Provos, Elie Bursztein, and Damon McCoy. 2016. Investigating Commercial Pay-Per-Install and the Distribution of Unwanted Software. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. USENIX Association, 721–739.

[54] Zhiyi Tian, Lei Cui, Jie Liang, and Shui Yu. 2023. A Comprehensive Survey on Poisoning Attacks and Countermeasures in Machine Learning. *Comput. Surveys* 55, 8 (2023), 166:1–166:35. doi:10.1145/3551636

[55] Danish Vasan, Mamoun Alazab, Sobia Wassan, Babak Safaei, and Zheng Qin. 2020. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Computers & Security* 92 (2020), 101748. doi:10.1016/J.COSE.2020.101748

[56] Haoyu Wang, Junjun Si, Hao Li, and Yao Guo. 2019. RmvDroid: towards a reliable Android malware dataset with app metadata. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*. IEEE / ACM, 404–408. doi:10.1109/MSR.2019.00067

[57] Jingjing Wang, Liu Wang, Feng Dong, and Haoyu Wang. 2023. Re-measuring the Label Dynamics of Online Anti-Malware Engines from Millions of Samples. In *Proceedings of the 2023 ACM on Internet Measurement Conference, IMC 2023, Montreal, QC, Canada, October 24-26, 2023*. ACM, 253–267. doi:10.1145/3618257.3624800

[58] Liu Wang, Haoyu Wang, Xiapu Luo, and Yulei Sui. 2022. MalWhiteout: Reducing Label Errors in Android Malware Detection. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 69:1–69:13. doi:10.1145/3551349.3560418

[59] Jiaheng Wei, Zhaowei Zhu, Hao Cheng, Tongliang Liu, Gang Niu, and Yang Liu. 2022. Learning with Noisy Labels Revisited: A Study Using Real-World Human Annotations. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[60] Jennifer Williams and Charlie Dagli. 2017. Twitter Language Identification Of Similar Languages And Dialects Without Ground Truth. In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects, VarDial 2017, Valencia, Spain, April 3, 2017*. Association for Computational Linguistics, 73–83. doi:10.18653/V1/W17-1209

[61] Xian Wu, Wenbo Guo, Jia Yan, Baris Coskun, and Xinyu Xing. 2023. From Grim Reality to Practical Solution: Malware Classification in Real-World Noise. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2602–2619. doi:10.1109/SP46215.2023.10179453

[62] Jiayun Xu, Yingjiu Li, and Robert H. Deng. 2021. Differential Training: A Generic Framework to Reduce Label Noises for Android Malware Detection. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*.

[63] Ke Xu, Yingjiu Li, Robert H. Deng, Kai Chen, and Jiayun Xu. 2019. DroidEvolver: Self-Evolving Android Malware Detection System. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 47–62. doi:10.1109/EUROSP.2019.00014

[64] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. USENIX Association, 2327–2344.

[65] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor W. Tsang, and Masashi Sugiyama. 2019. How does Disagreement Help Generalization against Label Corruption?. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Vol. 97. PMLR, 7164–7173.

[66] Xiaobo Zhang, Yutao Liu, Hao Wang, Wei Wang, Panpan Ni, and Ji Zhang. 2023. CoSaR: Combating Label Noise Using Collaborative Sample Selection and Adversarial Regularization. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*. ACM, 3184–3194. doi:10.1145/3583780.3614826

[67] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. ACM, 757–770. doi:10.1145/3372297.3417291

[68] Songzhu Zheng, Pengxiang Wu, Aman Goswami, Mayank Goswami, Dimitris N. Metaxas, and Chao Chen. 2020. Error-Bounded Correction of Noisy Labels. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Vol. 119. PMLR, 11447–11457.

[69] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. USENIX Association, 2361–2378.

[70] Binghui Zou, Chunjie Cao, Longjuan Wang, Sizheng Fu, Tonghua Qiao, and Jingzhang Sun. 2024. FACILE: A capsule network with fewer capsules and richer hierarchical information for malware image classification. *Computers & Security* 137 (2024), 103606. doi:10.1016/J.COSE.2023.103606

Table 2: PE Malware Dataset Statistics

Class	Training Samples		Noise Rate (%)	Testing Samples
	Clean	Noisy		
Benign	749	400	0.00	100
VirLock	796	800	0.50	100
WannaCry	818	820	0.24	100
Upatre	656	340	3.23	100
Cerber	801	944	15.57	100
Urelas	467	472	1.05	100
WinActivator	59	66	10.60	100
Pykspa	632	644	1.86	100
Ramnit	160	224	29.46	100
Gamarue	127	508	75.00	100
InstallMonster	105	199	47.23	100
Locky	59	57	54.38	100

Table 3: Dataset Statistics

(a) IDS17

Class	Train	Test
Benign	10 000	2000
DDoS	300	60
DoS Hulk	300	60
Portscan	250	50
Infiltration	250	50
DoS GoldenEye	200	40
DoS Slowloris	100	20
FTP-Patator	100	20
SSH-Patator	100	20

(b) Virus-MNIST

Class Examples	Train	Test
Benign[a]	2341	175
Adware	7188	496
Trojan 1	2832	205
Trojan 2	2228	176
Installer	738	58
Backdoor 1	6206	456
Crypto	14 374	1003
Backdoor 2	7003	491
Downloader	2398	173
Heuristic	3114	225

[a]This class is completely benign

A Appendices

A.1 Implementation

We used the open-source implementation of MORSE provided by its authors (https://github.com/nuwuxian/morse/) and the implementation of DT from [58] (https://github.com/MalTools/MalWhiteout/tree/master/dt). To ensure fair comparison, all frameworks, including SLB and the Vanilla, were trained with the same architecture, hyperparameters (*e.g.*, batch size), and initial weights. For each experiment, we fixed the seeds (1–10) and initialized model weights with Xavier initialization.

A.2 Hyperparameters Space

We performed a grid search to determine the optimal hyperparameters for each method. Our search criterion was the highest mF1 score obtained using seed 1; once the best configuration was identified, we ran experiments with seeds 1 through 10. For random noise experiments, we selected the hyperparameters that performed best at a 10% noise rate and then applied this configuration consistently across all noise levels. Although ideally we would perform a separate grid search for each seed and noise rate, this approach is computationally prohibitive; hence, we adopt this strategy as a practical compromise between cost and tuning rigor.

For SLB, we tuned three parameters: epochs for the data split e (Eq. 2), epochs before label flipping m (Eq. 10), and the CB loss weight β (Eq. 6). The EMA parameter α was fixed at 0.95 (Eq. 9) due to minimal impact. Search spaces were $e \in \{2, 5, 10, 15, 20\}$, $m \in \{0, 1, 3, 5\}$, and $\beta \in \{0.0, 0.9999\}$, with 0.0 indicating no weighting. For MORSE, which also uses CB loss, we tuned reweighting start $\{20, 30, 40\}$. Other hyperparameters matched SLB: warm-up epochs shared the same e range, β the same space, and $\epsilon \in \{0.7, 0.9\}$ per the authors' recommendations. For DT, we tuned rounds $n \in \{3, 5, 7, 10\}$ and epochs per round $e \in \{2, 5, 10, 15, 20\}$.

A.3 Datasets

PE Malware. We used the multi-class Windows PE malware dataset from [61]. The dataset comprises 1,024 features derived from four main sources: byte entropy histograms, PE imports, string histograms, and PE metadata. Each sample has two labels: a ground-truth label and a noisy label. Table 2 outlines the dataset's details. The noise rate indicates the fraction of samples incorrectly assigned to a label; for example, among the 944 samples labeled as Cerber in the noisy dataset, 15.57% are actually not Cerber.

Table 4: VS18 Dataset Statistics

Class	Training Samples		Noise Rate (%)	Testing Samples
	Clean	Noisy		
Benign	1132	1680	32.61	283
Shedun	3146	1000	4.60	786
SMSReg	3119	3452	12.42	780
Hiddad	182	438	71.68	46
Gappusin	146	300	91.00	37
Wapron	117	382	70.94	29
Youmi	66	379	87.07	16
Dowgin	47	324	89.50	12

CIC-IDS2017. We used the enhanced multi-class CIC-IDS2017 dataset from [22], which contains packet captures and network flows generated by CICFlowMeter [20]. Our experiments focused on the network flows. Labels were derived from attacker/victim IPs, ports, and timestamps, which we removed to prevent shortcut learning. From the full dataset, we sampled 12,000 benign and 1,920 attack instances spanning eight primary attack classes. Table 3a summarizes the distribution.

VirusShare 2018. We used the multi-class Android malware dataset of [27], which includes 215 features per app, covering characteristics such as file size, embedded URLs (has_url), and requested permissions (*e.g.*, READ_SMS). The class distribution is shown in Table 4.

APIGraph 2017–2018. We used the 2017 and 2018 portions of the Android application dataset from [7], originally introduced in [67]. This dataset was selected because (i) reports were collected at multiple time points (before 2020 in [67] and 2022–2023 in [7]), helping mitigate label noise; (ii) a strict threshold was applied, set at *sixteen* AV engines; and (iii) it reflects real-world distributions, with only 9% malicious samples. The dataset includes 1,159 features (excluding the label), extracted with DREBIN [1].

Virus-MNIST. We used this dataset to evaluate SLB on advanced architectures *e.g.*, ResNet18; see Section 4.5.1), which are common in image-based malware analysis [11, 35, 49, 70]. Virus-MNIST converts PE malware files into 32x32 grayscale images by sampling the first 1024 bytes of each PE [31]. Unlike standard classification tasks, the dataset includes a benign class, while malicious samples are grouped into nine clusters via k-means to emulate an MNIST-like distribution. Thus, the task shifts from labeling malware families to identifying structurally similar clusters. The dataset distribution is shown in Table 3b.