From Grim Reality to Practical Solution: Malware Classification in Real-World Noise

Xian Wu* Wenbo Guo+ Jia Yan† Baris Coskun‡ Xinyu Xing*
*Northwestern University +UC Berkeley † Penn State ‡ AWS

***Abstract*—Malware datasets inevitably contain incorrect labels due to the shortage of expertise and experience needed for sample labeling. Previous research demonstrated that a training dataset with incorrectly labeled samples would result in inaccurate model learning. To address this problem, researchers have proposed various noise learning methods to offset the impact of incorrectly labeled samples, and in image recognition and text mining applications, these methods demonstrated great success. In this work, we apply both representative and state-of-the-art noise learning methods to real-world malware classification tasks. We surprisingly observe that none of the existing methods could minimize incorrect labels' impact. Through a carefully designed experiment, we discover that the inefficacy mainly results from extreme data imbalance and the high percentage of incorrectly labeled data samples. As such, we further propose a new noise learning method and name it after `MORSE`. Unlike existing methods, `MORSE` customizes and extends a state-of-the-art semi-supervised learning technique. It takes possibly incorrectly labeled data as unlabeled data and thus avoids their potential negative impact on model learning. In `MORSE`, we also integrate a sample re-weighting method that balances the training data usage in the model learning and thus handles the data imbalance challenge. We evaluate `MORSE` on both our synthesized and real-world datasets. We show that `MORSE` could significantly outperform existing noise learning methods and minimize the impact of incorrectly labeled data.**

1. Introduction

The fight against malware has been bolstered not by the increase in the cybersecurity community workforce but by the significant advances in artificial intelligence. Recently, many supervised deep learning (DL) techniques have been introduced to improve malware classification (*e.g.,* [1], [2], [3]). Today, they have become a critical technique to help the fight against malware threats. Under the facilitation of these techniques, it becomes more efficient and effective to categorize malware into corresponding families.

However, one key assumption behind these supervised DL techniques is that the training data must have sufficient, correct data labels. However, such a common assumption does not always hold. In practice, labeling a malware sample is a time-consuming, labor-intensive process. The malware label assignment heavily relies on an analyst's professional experience and expertise. As such, it is prevalent that a malware sample could be assigned with an incorrect label.

Previous research indicated that incorrectly labeled data could cause a supervised algorithm to learn an inaccurate model. To address such a problem, researchers, therefore, have proposed many so-called noise learning solutions to offset the incorrectly labeled data's impact on model learning (*e.g.,* [4], [5], [6], [7], [8], [9], [10], [11]). In some typical ML applications (*e.g.,* image recognition [12], [13], text classification [14], [15], and speech recognition [16], [17]) – where some samples are mistakenly labeled – existing works have demonstrated outstanding potential to minimize incorrect labels' impact. Some works even demonstrated that the models learned from noisy training data could be as accurate as those from immaculate training data.

In this work, we apply the successful noise learning methods to real-world malware categorization tasks where training data contained noisy labels. We surprisingly discover that the existing methods could not even learn a multi-class malware classifier outperforming the classifier learned directly on the noisy training dataset. To understand the reason behind this discovery, we carefully design an experiment and discovered that the unexpectedly low classifier learning ability mainly comes from two aspects.

First, the malware dataset usually has an extreme data imbalance issue compared with training data in other tasks, like image recognization. For some malware families, the available data samples are much fewer than those in other families [18]. For example, the data imbalance ratio between major and minor families is 70x in a Microsoft PE malware dataset [19]. As we will elaborate in Section 3, this data imbalance issue forces the learning process to be biased toward the majority class. Second, the real-world malware dataset could contain a higher ratio of incorrectly labeled data for some malware families. Recent research [20], [21] shows that datasets that use AntiVirus vendors to give labels could introduce a high noisy label rate due to the errors of AV or inappropriate aggregation mechanisms (*e.g.,* MOTIF dataset [21] has a noise rate of 40%). This error rate, or in other words, the noise rate, is significantly higher than we have witnessed on the image recognition task, which amplifies the inefficacy of existing noise learning methods.

In this work, we propose a new noise learning method to address the challenges imposed by the high noise rate and extreme data imbalance. We name it after `MORSE`,

‡ *This work does not relate to Baris Coskun's position at Amazon.*

standing for **M**alware classificati**O**n f**R**om noi**S**y lab**E**ls. Technically speaking, our method first identifies the training data possibly correctly labeled. Then, it treats these possibly correct data as labeled data and the rest as unlabeled ones. Further, it utilizes a customized semi-supervised learning method to learn a classifier. Since the data samples less likely to be correct are treated as unlabeled data, even if the noise rate is relatively high, incorrect labels have a minimal impact on training, which resolves the high noise challenge.

To handle the data imbalance issue, we further integrate a sample re-weighting mechanism into our semi-supervised learning approach. The re-weighting mechanism assigns a higher weight to the samples associated with minor classes. This ensures that our customized semi-supervised learning method does not bias toward the majority classes. As we will show in Section 6, our customized semi-supervised learning method combined with sample re-weighting could significantly minimize the impact of incorrect labels.

To the best of our knowledge, this research is the first work that systematically evaluates existing noise learning methods and pinpoints their limitations. Besides, we also argue that MORSE is one of the pioneering works exploring classification on noisy malware training datasets.[1] Using MORSE, we show that security professionals could learn a relatively accurate malware classifier even if the training data is highly skewed and some malware classes contain more than 50% of incorrect labels. We demonstrate that MORSE could outperform all existing noise learning approaches in synthetic and real-world datasets. Last but not least, we also show that MORSE does not introduce significant learning overhead, which indicates our proposed method could be used as an efficient, practical technical solution for real-world malware classification tasks. We release the source code, learning models, and datasets at the repository https://github.com/nuwuxian/morse. In summary, this paper makes the following contributions.

- We summarize existing noise learning methods, systematically evaluate their effectiveness in offsetting noisy labels' impact, and point out their limitations in the malware classification context.
- We propose a customized, extended semi-supervised learning method to train a malware classifier from an extremely imbalanced dataset with a large amount of incorrectly labeled data samples.
- We implement our semi-supervised learning method as a noise learning algorithm and evaluate its effectiveness and efficiency in real-world and synthetic datasets where the noise rate is high, and class imbalance is extreme.

2. Summary of Existing Methods

From the perspective of training data usage, existing noise learning approaches could be classified into two kinds: using all the noisy training data for training or employing only the training data with possibly correct labels. From the technical perspective, existing works primarily follow four methods – ❶ sample selection, ❷ label sanitization, ❸ loss robustification, and ❹ noise matrix estimation.

Sample selection minimizes the impact of noisy labels in two steps (see Figure 1a). First, this method identifies the incorrectly labeled samples in a training dataset. Second, it eliminates or downplays these noisy samples from the model training procedure. Take the research works [4], [12], [23] for example. These methods first utilize the sample loss as an indicator to determine label correctness. Then, they train the corresponding model using only the identified clean samples but not the entire noisy training data.

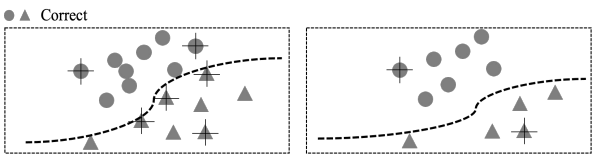
As is mentioned above, besides incorrect sample elimination, downplaying the incorrectly labeled sample is also a popular method of reducing the noisy labels' impact. For example, the works [5], [24] learn a weight for each training sample. The weight indicates how likely the corresponding sample's label is correct. When training models, they, therefore, use this weight to adjust the impact of training samples. The samples with lower weights play a less important role in model training. Therefore, while using all data for training, the proposed methods downplay the impact of noisy data.

Following these two works [5], [24], Jang *et al.* extend this idea [6] and demonstrated a more vital ability to learn a model from incorrectly labeled data. Technically, they first employ MentorNet [25] to identify training data with correct labels and assign them with higher weights. Second, they utilize a data augmentation method, mixup [26], to blend other training samples with relatively clean training samples. The authors show that this data augmentation could further offset the impact of noisy data upon the model learning.

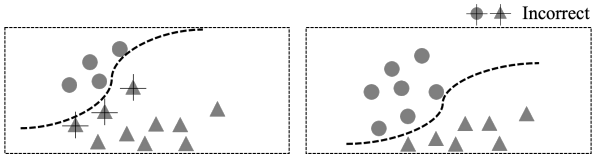
Label sanitization uses the entire dataset for training. Different from sample selection, this method corrects label errors and thus offsets their negative impact (see Figure 1b). For example, the works [27], [7] both fuse the given labels with model predicted labels. This idea ensures that data samples' incorrect labels could be potentially reset back to their correct ones. Following these two works, a recent work [28] introduces a new method to infer the correctness of given labels and the predicted ones. Using this new method, one could determine which labels (*i.e.,* predicted or given labels) should be used for parameter updates. Zhang *et al.* show that this proposed method could theoretically guarantee the improvement of model learning even if the training dataset contains incorrect labels.

Loss robustification designs new loss functions and then uses the entire noisy dataset for model learning (see Figure 1c). Research shows that incorrectly labeled data has a different impact on loss functions. Inspired by this discovery, the works [8], [29] proposed new loss functions robust against noisy labels. For example, Zhang *et al.* [8] find that the Mean Absolute Error (MAE) loss is more robust against noisy labels. However, when using it to learn a model, the learning process inevitably suffers from the under-fitting issue. To benefit from MAE's robustness and at the same time avoid the under-fitting issue, Zhang *et al.* propose to

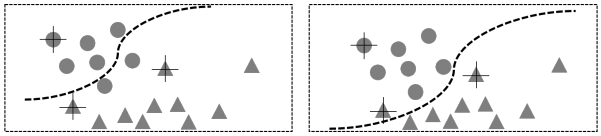
1. There is a recent work [22] exploring noise learning techniques in malware identification. As we will elaborate in Section 7, this method, however, is designed only for binary categorization and cannot be extended to the tasks that assign malware to different families.



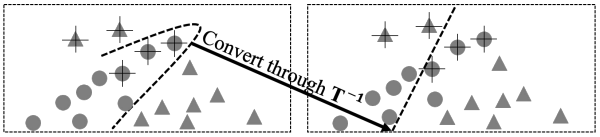
(a) Sample selection selects samples possibly correctly labeled and then learns the decision boundary.



(b) Label sanitization corrects mistakenly labeled samples and then learns the decision boundary.



(c) Loss robustification uses a new, robust loss function to learn the decision boundary.



(d) Noise matrix estimation learns a transition matrix ("**T**") and then uses it to correct the inaccurate decision boundary.

Figure 1: The demonstration of existing noise learning methods in binary classification. Each geometry pattern represents a sample. The circle and triangle pattern indicates the different label of the sample. The geometry with a cross indicates the sample is incorrectly labeled. For example, circle with a cross denotes the sample is mistakenly labeled as "circle" and its true label should be "triangle". In each subfigure, the left shows the decision boundary learned directly from the noisy training dataset, whereas the right depicts the decision boundary learned by the corresponding noise learning method.

combine the Cross-Entropy (CE) loss with the MAE loss. They demonstrate that the combined loss functions provide a better ability to learn a model under noisy datasets.

Loss robustification methods also introduce proper regularization terms to the loss functions. Recent research [30], [31] discovers that a learning algorithm usually attempts to fit the training data with clean labels at the early training phase. The data with incorrect labels kick in their effects usually at the later training phase. Motivated by this discovery, a recent work [9] adds a regularization term to the learning loss. The regularization term prevents the later-stage learning process from significantly varying the model (learned at the early stage). With this design, the noisy data could impose less impact on model learning.

Noise matrix estimation learns a transition matrix from the entire noisy dataset. The matrix indicates the probability that a correct label was mistakenly flipped to a corresponding incorrect label. As such, using this transition matrix, one could flip the incorrect prediction result made by the model trained on the noisy training dataset (see Figure 1d). To learn a transition matrix accurately, past research proposes various methods under different assumptions [10], [14], [32], [15], [13], [33]. For example, Goldberger *et al.* [10] model the noise transition matrix as part of the classifier. Using the method commonly adopted for training a classifier, they jointly estimate the classifier's parameters and noise matrix.

3. Evaluation of Existing Methods

Existing research has demonstrated the effectiveness of the above noise learning techniques in image and text datasets. However, a security dataset containing incorrect labels usually has its own characteristics, which imposes challenges on these methods. In the following, we introduce two real-world malware datasets with incorrect labels - a Windows PE malware dataset and an Android malware dataset. We discuss the uniqueness of these datasets and use them to evaluate existing noise learning methods.

TABLE 1: Statistics of the Windows PE malware dataset.

ID	Family	# of Samples	# of Training samples	Noise rate in training set
0	Benign	500	400	0.000
1	VirLock	900	800	0.005
2	WannaCry	920	820	0.002
3	Upatre	440	340	0.032
4	Cerber	1044	944	0.156
5	Urelas	572	472	0.011
6	WinActivator	166	66	0.106
7	Pykspa	744	644	0.019
8	Ramnit	324	224	0.295
9	Gamarue	608	508	0.750
10	InstallMonster	299	199	0.472
11	Locky	157	57	0.544

TABLE 2: The statistics of the Android malware dataset.

ID	Family	# of Samples	# of Training samples	Noise rate in training set
0	Smserg	2687	2587	0.040
1	Benign	4683	4583	0.699
2	Autoins	200	100	0.150
3	Jiagu	678	578	0.235
4	Shedun	10867	10767	0.548
5	Wapron	857	757	0.136
6	Dnotua	136	36	0.583
7	Hiddad	185	85	0.000
8	Secneo	203	103	0.223
9	Triada	299	199	0.005
10	Secapk	155	55	0.055
11	Smspay	281	181	0.028
12	Qappusin	158	58	0.000

3.1. Dataset & Characteristics

Windows PE dataset. This dataset is obtained from a security lab. It contains 500 benign executables and 6,174 malicious PE files. As shown in Table 1, the malicious executables come from 11 unique malware families. Each malware sample has been successfully reverse-engineered and carefully analyzed by at least three security analysts

with 5+ years of malware analysis experience. As a result, the labels provided in this dataset are 100% accurate. We then generate noisy labels via a widely used labeling method (*i.e.,* labeling data using AntiVirus vendors). Specifically, we first upload all executable files to VirusTotal [34] and gather the labels provided by different vendors. We discover that 11.38% PE files are provided with at least one wrong label by a vendor. For example, our dataset assigns the malware sample 2749fcf947b4cbc6a5551b2ecdbb8a06 into the Locky family. On VirusTotal, 35.71% of vendors mark it as Cerber or Upatre, with different malware families. We then randomly change a PE sample's label to one that other vendors provided if a different label exists. In this way, we construct a noisy dataset containing 13.88% incorrect labels.

Android dataset. We also use another real-world Android malware dataset – VirusShare2018 [35], which is of a much larger scale than the first dataset. This dataset contains 4,683 benign ware and 16,706 android malware, coming from 12 families (see Table 2). It is extremely hard to manually check the label correctness for such a large-scale dataset. As such, we automatically obtain clean/noisy labels via VirusTotal. To obtain noisy labels, we follow a real-world method [36] that uses the result of one select vendor to label data (*e.g.,* we adopted Ikarus [34]). To get clean labels, we conduct a majority vote across all vendors. With the noisy and clean labels, we construct a noisy training dataset with a noise rate of 47.50%. This dataset simulates a real-world noisy distribution on a large-scale dataset.

Compared with the image and text datasets, these two malware datasets have the following key characteristics.

Data imbalance. The class distribution of our datasets is extremely imbalanced. For example, in the Android dataset, the sample number in the Shedun family is 300x more than that in the Dnotua family. Different from some existing malware datasets, where the imbalance comes from the dominant number of benign samples and a small number of malicious samples, our imbalance comes from the difference in sample sizes between malware families. Since our goal is to correctly classify malware families rather than conducting anomaly detection where datasets have many more benign samples than malicious ones, we do not consider the imbalance between benign and malicious samples.

High noise ratio. The label quality of our malware data is relatively lower than that of image datasets. To label image samples, one could use cost-efficient crowdsourcing and reduce the label noise rate. Unlike image labeling, labeling security data requires extensive security expertise. Besides, security analysts need to spend hundreds and even thousands of hours zooming in on a suspicious program to determine its actual maliciousness. As such, malware data are more likely to be mistakenly labeled than images. For example, malware family Dnotua in the Android dataset and Gamarue in the PE dataset have a noise ratio of 58.3% and 75.0%, respectively. This is way higher than noisy image datasets.

Non-uniform noise distribution. To bypass the detection and counteract defense analysis, malware writers developed and applied various techniques to manipulate their malicious code. Malware writers usually have different expertise and experience in misleading analysts' reverse-engineering efforts. Therefore, various malware may contain the footprints of different anti-debugging, packing, and obfuscation techniques. These techniques impose different difficulty levels in determining a suspicious program's maliciousness. As such, the noise rates across malware families vary significantly. For example, the PE dataset contains a noise rate as high as 75.0% (Gamarue) and as low as 0.2% (WannaCry).

TABLE 3: Summary of our selected noise learning methods.

Category	Representative Method	State-of-the-art Method
Sample selection	Coteaching+[12]	Mentormix[6]
Label sanitization	Bootstrap[27]	LRT[28]
Loss robustification	GCE[8]	ELR[9]
Noise matrix estimation	Noise-adaption[10]	LIO[13]

TABLE 4: The test accuracy (recall), precision, and F_1 score of existing noise learning methods on the PE dataset. Each experiment runs six times under different parameter initializations. The result in the cell contains the average and standard deviation. "Vanilla DNN" indicates the performance of the model learned from the noisy training dataset.

Methods	Average (%)			Class-11 Locky (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	93.08/0.25	93.65/0.51	92.57/0.36	44.33/3.78	96.34/4.22	60.48/3.22
Coteaching+	87.39/0.65 p = 0.999	92.24/0.39 p = 0.998	87.90/1.13 p = 0.999	2.00/2.89 p = 0.999	99.44/1.84 p = 0.055	3.92/3.16 p = 0.996
Mentormix	92.34/0.10 p = 0.997	93.40/0.14 p = 0.812	92.32/0.28 p = 0.322	41.17/4.13 p = 0.767	95.24/1.34 p = 0.505	57.82/2.14 p = 0.434
Bootstrap	92.92/0.30 p = 0.876	93.33/0.23 p = 0.829	92.27/1.91 p = 0.503	46.33/5.15 p = 0.260	92.71/2.69 p = 0.858	61.03/2.49 p = 0.191
LRT	92.52/0.21 p = 0.995	93.38/0.24 p = 0.815	92.15/0.18 p = 0.817	42.50/1.50 p = 0.887	93.85/4.67 p = 0.753	59.34/2.47 p = 0.557
Noise-adaption	92.65/0.24 p = 0.978	93.25/0.30 p = 0.861	92.18/0.22 p = 0.488	40.83/1.46 p = 0.968	90.04/3.68 p = 0.968	56.18/2.47 p = 0.687
LIO	92.30/0.35 p = 0.990	93.21/0.25 p = 0.879	92.01/0.15 p = 0.435	38.00/4.04 p = 0.910	90.12/4.23 p = 0.958	53.42/1.45 p = 0.956
GCE	92.15/0.27 p = 0.998	93.40/0.32 p = 0.809	91.74/0.65 p = 0.962	36.33/7.40 p = 0.926	94.85/3.11 p = 0.705	52.38/5.95 p = 0.991
ELR	91.84/0.18 p = 0.999	93.27/0.24 p = 0.882	91.20/0.27 p = 0.998	34.50/2.75 p = 0.999	96.39/3.37 p = 0.493	50.01/2.70 p = 0.997

3.2. Evaluation

Experiment Setup. To evaluate the effectiveness of existing noisy label learning methods, we choose two works from each category of existing methods discussed in Section 2. One represents the state-of-the-art noise label learning technique in that category. The other indicates the most representative work in the corresponding category.[2] Table 3 shows all the works of our selection, and we argue that the selection of these methods well represents existing research efforts.

For each malware dataset, we randomly reserve 100 samples from each class as our testing dataset and the rest as our training dataset (see Table 1 and 2). For the PE dataset, we extract features from each PE sample. Our feature extraction follows a widely adopted method proposed in [37] (see Appendix 10.1 for more details). After the feature extraction, each sample turns into a 1,024 dimension feature vector. For the Android dataset, we directly use the feature vectors provided by the original dataset [35]. Note that the deep neural network (DNN) structure used in the selected works are designed for image data. We customized the network structures for malware classification. To ensure our model performance has a minimal impact from model

2. Note that the most representative work in the category usually is the work with the highest number of citations.

TABLE 5: Noise learning methods on the Android dataset.

Methods	Average (%)			Class-6 Dnotua (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	73.00/0.35	78.65/2.21	69.96/0.54	2.67/5.53	19.79/34.31	4.63/9.52
Coteaching+	72.73/0.61 p = 0.895	75.83/1.76 p = 0.949	69.44/0.41 p = 0.963	0.00/0.00 p = 0.835	0.00/0.00 p = 0.873	0.00/0.00 p = 0.837
Mentormix	75.52/0.24 p = 0.001	78.20/3.12 p = 0.645	72.15/0.88 p = 0.003	3.75/6.49 p = 0.767	20.24/31.24 p = 0.563	6.30/10.18 p = 0.620
Bootstrap	72.24/0.42 p = 0.993	77.41/3.94 p = 0.738	68.97/0.70 p = 0.999	0.33/0.47 p = 0.817	16.67/23.57 p = 0.591	0.65/0.92 p = 0.816
LRT	72.07/0.49 p = 0.991	79.02/3.15 p = 0.416	69.15/0.87 p = 0.933	2.50/5.59 p = 0.517	15.62/34.94 p = 0.574	4.31/9.64 p = 0.519
Noise-adaption	74.73/0.80 p = 0.003	78.33/3.01 p = 0.605	71.62/0.82 p = 0.004	5.50/7.46 p = 0.132	39.17/41.87 p = 0.114	9.40/12.63 p = 0.138
LIO	72.56/0.41 p = 0.838	80.02/2.83 p = 0.233	72.78/1.00 p = 0.001	3.00/5.86 p = 0.468	25.00/38.19 p = 0.417	5.24/10.09 p = 0.466
GCE	72.67/0.95 p = 0.710	78.90/3.98 p = 0.459	69.90/0.87 p = 0.539	5.00/7.00 p = 0.314	32.29/45.70 p = 0.333	8.66/12.24 p = 0.314
ELR	70.99/0.12 p = 0.999	76.46/0.77 p = 0.967	67.62/0.25 p = 0.999	0.00/0.00 p = 0.835	0.00/0.00 p = 0.873	0.00/0.00 p = 0.837

parameter initialization and data partition, we re-initialize model parameters and divide the malware dataset based on the aforementioned criteria whenever training a malware classifier. In this way, we can compute the average model accuracy(recall)/precision/F_1 scores and their standard deviations. Since our testing datasets are balanced, the reported metrics are also balanced. Besides, we compute a p-value to examine if the performance gain is statistically significant. If p-value is smaller than 0.05 [38], the result is statistically significant (see Appendix 10.2 for more details). Note that we carefully tune the hyper-parameters and report the best performance for each method (see Appendix 10.2).

Experiment Results. Table 4 and 5 shows the performance of selected noise learning methods on the two datasets, respectively. For the PE dataset, none of the existing methods perform well in handling noisy labels. To our surprise, the accuracy of the classifiers is even worse than that of the classifiers trained on a noisy training set (vanilla method). On the Android dataset, only MentorMix, Noise-adaption, and LIO outperform the vanilla method by a small margin.

Looking at each class closely, we find the performance on the smallest class in each dataset is the worst (the Locky class in the PE dataset and the Dnotua class in the Android dataset). As shown in Table 4 and 5, none of the existing methods outperform the vanilla method. For some methods, the performance drops significantly compared to the vanilla method. We hypothesize that the reasons are as follows. First, both class is the smallest class, approximately 16x and 300x less than the largest class of the corresponding dataset. Many prior research works indicate that using a highly skewed dataset, even without the impact of noisy labels, the trained classifier is not likely to perform well on the small class. Second, both class has a high noise rate (*i.e.,* 54% for Locky and 58.3% for Dnotua). These high noise rates further reduce the number of clean samples useful for classifier training. Since none of the noise learning methods guarantees the selected samples are clean, when accidentally picking up several incorrectly labeled samples from the small class, the damage to the corresponding classification performance would be inevitably amplified.

4. Hypothesis Test

To validate the hypothesis above, we design a controlled experiment. First, we synthesize four datasets using a public dataset – BODMAS [39], each of which has a different noise rate or data imbalance setup. Second, we learn multi-class classification models on each dataset using the eight noise learning methods. Finally, we validate our hypothesis by investigating whether the aforementioned factors impact the classification performance of the learned classifiers.

Synthesizing datasets. To study the noise rate and data imbalance's impact on noisy label learning, we need a dataset. Its data imbalance and noise rate should be under control. However, existing datasets have their own data imbalance and contain no incorrect labels. As such, we synthesize datasets from the public malware dataset BODMAS [39]. It contains 57,293 samples from 581 families, and each malware sample is represented as a vector of 2,381 features. We choose 10 malware families with the largest sample size. For each family, we randomly select 1,000 samples for training and 500 samples for testing. We treat our entire reserved training data corpus as the base dataset.

With the base dataset in hand, we construct 4 synthesized datasets. First, we remove a certain proportion of samples from 5 families and thus obtain two datasets with different data imbalances (20x and 100x). We select 20x to simulate the imbalance setting in our real-world dataset, which has an imbalance ratio of roughly 17x) and 100x to simulate an extreme imbalance setting in a Microsoft PE Malware dataset [19]. We then randomly change (*i.e.,* label a sample as a family other than its true family) 30% and 60% labels for each dataset. 30% simulates setups where the labelers use multiple AntiVirus vendors to give labels (*e.g.,* MOTIF with a noise rate of 40%), and 60% simulates setups labeling data with manual analysis or single AntiVirus vendor [40]. In this way, we obtain 20x-imbalanced and 100x-imbalanced datasets with 30% and 60% noise rates, respectively. In this work, we use the aforementioned existing noise learning methods to train models on these four synthesized datasets and then evaluate the performance of the models on the reserved testing data corpus mentioned above. Appendix 10.2 presents more implementation details. To ensure the performance of these baseline methods is not compromised by hyper-parameters choices, we carefully tuned the critical hyper-parameters and selected the best hyper-parameters for our experiment (see Appendix 10.2 for the tuning process).

Hypothesis validation. Table 6 shows the performance of existing methods in the four settings above. As we can observe, when the imbalance ratio is 20x or 100x, and the noise rate is as high as 60%, the models learned through existing methods cannot even perform better than that learned directly from the noisy data. It means a high noise rate and the extreme (or ultra-extreme) data imbalance impose significant challenges to existing noise learning methods. This observation well validates our hypothesis in Section 3.

From Table 6, we discover that when the noise rate is relatively lower (30%), most of the existing methods remain ineffective under the 20x and 100x data imbalance. However, some methods, such as MentorMix and Noise-adaption learning, start to function, demonstrating some abilities to offset the impact of noisy training data. We believe the reason behind the functioning roots in the design

TABLE 6: The testing performance of existing methods on the synthetic datasets. "overall cls" represents the model's overall performance. "rare cls" indicates the model's performance on minority classes.

Methods	Noise rate 0.3 and imbalance ratio 20x						Noise rate 0.3 and imbalance ratio 100x					
	Overall cls (%)			Rare cls (%)			Overall cls (%)			Rare cls (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	75.55/1.38	82.94/3.78	73.46/0.90	57.89/3.69	85.74/6.65	63.03/1.20	70.83/0.87	76.29/0.85	67.80/1.19	48.45/1.44	79.47/0.62	56.77/0.24
Coteaching+	74.49/2.56 p = 0.738	78.23/1.45 p = 0.951	72.79/1.25 p = 0.683	56.71/3.08 p = 0.715	77.92/2.50 p = 0.952	62.09/1.03 p = 0.871	69.25/0.92 p = 0.977	74.63/3.38 p = 0.863	67.02/1.95 p = 0.817	45.57/3.68 p = 0.891	75.96/7.39 p = 0.834	53.85/4.53 p = 0.898
MentorMix	77.58/0.70 p = 0.043	80.80/1.05 p = 0.715	74.15/1.13 p = 0.052	62.77/1.14 p = 0.058	79.52/1.02 p = 0.893	63.85/1.02 p = 0.082	72.58/0.56 p = 0.020	75.25/0.24 p = 0.998	67.51/0.10 p = 0.738	48.54/1.06 p = 0.430	78.86/0.83 p = 0.975	56.73/1.02 p = 0.481
Bootstrap	75.17/1.43 p = 0.624	78.80/1.05 p = 0.934	72.78/1.13 p = 0.826	58.58/3.00 p = 0.398	77.55/1.79 p = 0.940	62.11/2.10 p = 0.992	70.30/0.66 p = 0.970	76.45/0.74 p = 0.362	67.55/1.24 p = 0.637	49.24/1.18 p = 0.947	79.65/0.56 p = 0.082	56.51/1.46 p = 0.662
LRT	73.17/2.16 p = 0.927	78.41/0.61 p = 0.951	72.30/1.11 p = 0.928	58.30/3.66 p = 0.398	78.25/1.10 p = 0.934	62.33/0.88 p = 0.776	65.17/5.24 p = 0.992	74.18/3.94 p = 0.871	64.42/4.93 p = 0.923	40.01/8.61 p = 0.994	77.63/4.04 p = 0.809	51.71/6.40 p = 0.935
Noise-adaption	77.75/0.66 p = 0.016	80.44/1.06 p = 0.846	74.13/1.33 p = 0.066	62.50/0.98 p = 0.028	79.02/2.68 p = 0.943	63.73/1.56 p = 0.096	72.56/1.41 p = 0.006	75.59/0.81 p = 0.846	67.30/1.51 p = 0.691	50.81/2.90 p = 0.075	79.21/0.57 p = 0.975	56.76/0.79 p = 0.504
LIO	75.72/1.78 p = 0.825	81.19/2.74 p = 0.642	72.73/0.75 p = 0.883	60.19/1.04 p = 0.114	84.40/6.52 p = 0.479	63.61/2.44 p = 0.238	70.06/2.34 p = 1.000	75.12/2.35 p = 0.852	65.80/1.12 p = 0.992	46.25/5.68 p = 0.798	79.65/0.30 p = 0.306	52.72/0.28 p = 0.987
GCE	76.98/1.49 p = 0.113	81.52/3.26 p = 0.594	73.68/1.74 p = 0.287	59.08/4.09 p = 0.350	82.89/5.96 p = 0.614	62.67/2.38 p = 0.540	70.83/2.21 p = 0.504	73.25/7.41 p = 0.793	66.94/3.21 p = 0.704	47.58/3.32 p = 0.728	72.31/14.53 p = 0.836	53.49/6.24 p = 0.848
ELR	74.28/0.69 p = 0.953	84.60/1.20 p = 0.077	72.05/1.38 p = 0.899	53.58/2.10 p = 0.982	92.02/1.57 p = 0.027	60.82/1.97 p = 0.944	66.84/1.66 p = 0.998	65.70/0.13 p = 1.000	61.00/0.40 p = 0.999	42.33/3.37 p = 0.987	59.46/0.57 p = 1.000	44.90/0.77 p = 1.000
Methods	**Noise rate 0.6 and imbalance ratio 20x**						**Noise rate 0.6 and imbalance ratio 100x**					
	Overall cls (%)			Rare cls (%)			Overall cls (%)			Rare cls (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	68.04/1.77	70.38/6.31	63.68/4.12	48.93/4.03	73.16/9.31	53.39/7.05	65.57/0.87	71.67/3.30	61.11/0.89	44.39/1.94	76.67/7.45	51.02/1.56
Coteaching+	61.20/4.81 p = 0.993	58.92/5.03 p = 0.997	54.73/4.60 p = 0.998	33.64/6.49 p = 0.998	55.50/9.97 p = 0.997	39.84/8.42 p = 0.997	50.85/7.03 p = 0.997	37.47/4.82 p = 1.000	41.93/4.07 p = 0.999	10.37/14.67 p = 0.998	16.38/7.34 p = 1.000	15.33/6.87 p = 1.000
MentorMix	69.11/0.74 p = 0.129	75.05/1.05 p = 0.934	63.28/1.03 p = 0.826	51.40/2.28 p = 0.376	77.55/3.79 p = 0.151	58.11/2.10 p = 0.124	55.28/1.50 p = 0.998	76.45/0.74 p = 0.362	57.55/1.24 p = 0.999	19.96/2.34 p = 0.999	77.58/4.01 p = 0.082	30.51/3.46 p = 0.999
Bootstrap	68.94/1.40 p = 0.174	71.12/4.69 p = 0.431	64.81/2.92 p = 0.328	50.03/5.04 p = 0.370	71.35/8.21 p = 0.602	55.56/7.42 p = 0.346	65.83/1.50 p = 0.316	72.73/0.67 p = 0.253	60.89/1.10 p = 0.669	45.66/2.09 p = 0.182	79.95/0.07 p = 0.184	50.30/1.21 p = 0.784
LRT	64.97/2.70 p = 0.982	70.83/2.85 p = 0.453	61.18/4.68 p = 0.759	48.40/3.06 p = 0.643	78.15/1.91 p = 0.168	54.61/2.64 p = 0.380	55.51/5.83 p = 0.992	57.23/6.02 p = 0.993	47.04/5.76 p = 0.998	26.78/11.11 p = 0.994	59.15/10.55 p = 0.962	29.60/10.03 p = 0.997
Noise-adaption	70.62/3.54 p = 0.130	77.15/2.84 p = 0.021	66.75/2.11 p = 0.141	53.65/7.79 p = 0.155	83.70/4.08 p = 0.010	58.07/3.34 p = 0.146	66.20/1.79 p = 0.196	72.04/3.48 p = 0.443	62.01/1.18 p = 0.120	44.82/2.55 p = 0.380	75.80/7.18 p = 0.563	53.60/2.67 p = 0.101
LIO	68.54/2.97 p = 0.390	73.43/2.67 p = 0.167	64.63/2.40 p = 0.311	48.73/6.64 p = 0.517	77.28/3.51 p = 0.171	57.39/3.04 p = 0.133	52.54/5.26 p = 1.000	56.90/7.90 p = 0.997	45.92/6.74 p = 0.998	24.53/9.29 p = 0.997	57.12/15.47 p = 0.980	27.23/14.41 p = 0.993
GCE	57.38/4.32 p = 0.996	53.55/6.63 p = 0.993	50.76/6.05 p = 0.992	29.01/7.37 p = 0.997	45.59/10.18 p = 0.994	32.45/7.67 p = 0.995	47.15/0.35 p = 1.000	27.53/1.75 p = 1.000	33.92/1.02 p = 1.000	0/0 p = 1.000	0/0 p = 1.000	0/0 p = 1.000
ELR	68.19/1.93 p = 0.430	68.62/5.42 p = 0.727	62.58/3.58 p = 0.719	43.55/4.02 p = 0.985	66.42/9.61 p = 0.914	48.51/5.95 p = 0.917	63.65/0.53 p = 0.994	60.18/5.11 p = 0.998	58.69/1.41 p = 0.999	36.13/1.07 p = 1.000	50.00/10.00 p = 0.999	40.00/2.46 p = 1.000

of both techniques.

MentorMix utilizes a data augmentation mechanism to fuse some incorrectly labeled training data with relatively clean training samples and thus improve classification capability. When the noise rate is low, the data fusing introduces fewer errors and thus demonstrates better performance even in the context of data imbalance. However, the augmentation inevitably amplifies the errors when the noise is high, making MentorMix futile. For Noise-adaption, it first learns a model f on the noisy data. The learned model could predict given samples' labels (regardless if they are correct or incorrect). Then, noise-adaption estimates a matrix $\mathbf{T}$ indicating the likelihood of a sample's correct label being flipped to an incorrect one. By multiplying f with $\mathbf{T}^{-1}$, one could obtain the corrected label for data samples. In a low noise setup, f could be learned accurately. However, in a high noise setting, the high-noise training dataset is similar to a dataset with most of its data randomly labeled. In this situation, a previous study [30] indicates that it is challenging to learn a highly accurate f, making the noise-adaption method ineffective.

5. Proposed Method

To offset the impact of the high noise rate and the data imbalance issues, we propose a new technical method and name it after MORSE. Technically, MORSE customizes and extends a state-of-the-art semi-supervised learning technique. In the following, we first describe the high-level idea of MORSE. We then detail how we customize and extend the semi-supervised learning method.

5.1. Design Principle & Technical Overview

As is mentioned in Section 2, the existing noise learning techniques follow two different methods to handle noise learning problems from the training data usage perspective. One is to utilize all the noisy training data to learn models (*e.g.,* L2RW [5], ELR [9], and MentorMix [6]). The other is learning models by using only the training data that could be correctly labeled and discarding the rest. The representative works include Coteaching [4] and INCV [23].

For the first method, while researchers introduce various methods to make incorrectly labeled data carry fewer weights on training, incorrectly labeled data inevitably damage model updates. With the increase in noise rate, the model's performance downgrade would be amplified. For the second method, model training rules out many noisy labels. Compared with the first method, noisy labels imposes minimal impact on learned models. However, when the noise rate is high and the data imbalance issue occurs, the data available for training becomes scarce, and the learned model is challenging to perform well.

In this work, we propose MORSE to address this problem. MORSE treats possibly correctly labeled data as labeled data and the rest as unlabeled data. It then utilizes a semi-supervised learning method to facilitate model learning. We further augment the semi-supervised learning method

with a sample re-weighting mechanism to make the semi-supervised learning handle the extreme data skewed issue.

Semi-supervised learning is an ML technique falling between unsupervised and supervised learning. Given a dataset with the mixture of labeled and unlabeled data, it combines both labeled and unlabeled data during training. As a result, it could learn a model with higher model accuracy than supervised and unsupervised learning approaches.

Using it to learn a model on our designated, labeled data (*i.e.,* possibly correctly labeled data) and unlabeled data (*i.e.,* possibly incorrectly labeled data), we can avoid the drawback of the existing technique. First, some incorrect labels can be discarded, reducing their impact on model learning. Second, by treating incorrectly labeled data as unlabeled data and then introducing them as part of the data for semi-supervised learning, we avoid ruling out the majority of incorrectly labeled data for model learning or, in other words, avoiding having too little available training data to learn a model. As we will show in Section 6, this design significantly outperforms the existing methods.

As is mentioned above, in addition to using a semi-supervised learning method, we also extend it with a sample re-weighting mechanism. Sample re-weighting [41], [42] is the state-of-the-art technique recently proposed for dealing with data imbalance. Technically, it weighs the loss computed for different samples differently based on whether they belong to the majority of the minority classes. The re-weighting technique assigns a higher weight to the loss encountered by the samples associated with minor classes. In this way, a learning method could handle a class imbalance in the training dataset. As we will show in Section 6, by integrating this mechanism into a semi-supervised learning framework, we could further improve the ability to counteract incorrect labels. While our proposed method could be viewed as the combination of semi-supervised learning and sample re-weighting, the integration of both is not trivial. There are still many detailed designs needed to be addressed. For example, how to determine labeled and unlabeled data in a noisy training set? How to customize a semi-supervised learning method for malware categorization? How to integrate sample re-weighting into the semi-supervised learning framework? In the following, we discuss our design in detail and provide the background knowledge needed for understanding our proposed method.

5.2. Background of Semi-supervised Learning

As is mentioned above, semi-supervised learning takes both labeled and unlabeled data for parameter update during model training. Over the past years, researchers have proposed various semi-supervised learning techniques. In this work, we choose the state-of-the-art method FixMatch [43], customizing and extending it for our problem. Compared with other semi-supervised learning methods, FixMatch demonstrates better empirical performance and has more simplified learning framework. In the following, we briefly introduce how FixMatch performs model training.

Algorithm 1: FixMatch – the state-of-the-art semi-supervised learning algorithm.

Input: labeled dataset $\mathcal{X}$, unlabeled dataset $\mathcal{U}$, number of total training epochs K, confidence threshold τ, unsupervised loss weight λ.
Initialization: Initialize the weights $\mathbf{\Theta}$ for the model $f(\cdot)$ randomly.
for $k = 0, 1, 2, ..., K$ **do**
  for $iter = 1, 2, ..., num\_batches$ **do**
    From $\mathcal{X}$, draw a mini-batch $\{(\mathbf{x}_b, y_b) : b \in (1, ..., B)\}$
    From $\mathcal{U}$, draw a mini-batch $\{\mathbf{u}_b : b \in (1, ..., B_\mu)\}$
    for $b = 1, 2, ..., B_\mu$ **do**
      $\mathbf{q}_b = f(g(\mathbf{u}_b); \mathbf{\Theta})$
      $\widehat{q}_b = \arg\max(\mathbf{q}_b)$
    end
    $\mathcal{L}_s = \frac{1}{B} \sum_{b=1}^{B} H(y_b, f(g(\mathbf{x}_b)))$
    $\mathcal{L}_u = \frac{1}{B_\mu} \sum_{b=1}^{B_\mu} \mathbf{I}(\max(\mathbf{q}_b) > \tau) H(\widehat{q}_b, f(h(\mathbf{u}_b)))$
    $\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_u$
    Update the model's weights $\mathbf{\Theta}$ by minimizing the loss function $\mathcal{L}$
  end
end
Output: the well trained model $f(\cdot; \mathbf{\Theta})$.

FixMatch is designed to perform semi-supervised learning in the context of image recognition. Technically, FixMatch works as follows. First, it evenly partitions both the labeled and unlabeled datasets into M mini-batches. Second, it utilizes an augmentation function $g(\cdot)$ to convert a batch of the labeled data $\mathcal{X}_\downarrow = \{(\mathbf{x}_b, y_b) : b \in (1, ..., B)\}$ into their weakly augmented form. As is mentioned above, FixMatch is designed for the image recognition task. Therefore, FixMatch defines the weakly augmented sample as the image sample $\mathbf{x}_b$ ($\mathbf{x}_b \in \mathcal{X}_\downarrow$) flipped or shifted horizontally or vertically. With the augmented form of labeled samples in hand, third, FixMatch utilizes an updated deep learning model $f(\cdot)$ – or a randomly initialized mode if the algorithm is in its first iteration – to predict labels for a batch of unlabeled data $\{\mathbf{u}_b : b \in (1, ..., B_\mu)\}$ and thus turns the batch of the unlabeled data into a batch of data with pseudo labels, *i.e.,* $\mathcal{U}_\downarrow = \{(\mathbf{u}_b, \widehat{q}_b) : b \in (1, ..., B_\mu)\}$. Here, $\widehat{q}_b$ indicates the pseudo label of the sample $\mathbf{u}_b$. Mathematically, it is equal to $\arg\max(\mathbf{q}_b)$ representing the index of the maximum value of the model output $\mathbf{q}_b = f(g(\mathbf{u}_b))$.

With both $\mathcal{X}_\downarrow$ and $\mathcal{U}_\downarrow$ in hand, FixMatch further introduces a loss, $\mathcal{L}_s + \lambda \mathcal{L}_u$, which is the summation of two loss terms $\mathcal{L}_s$ and $\mathcal{L}_u$ weighed by hyper-parameter λ. The first loss term also called the supervised loss, is established on the batch of the labeled data $\mathcal{X}_\downarrow$. It is defined as

$$\mathcal{L}_s = \frac{1}{B} \sum_{b=1}^{B} H(y_b, f(g(\mathbf{x}_b))), \tag{1}$$

where H is the cross-entropy loss. By minimizing this loss, FixMatch ensures that the image flip and shift should not influence the model's prediction or, in other words, the model could be robust against standard image flip-and-shift manipulation.

Unlike the supervised loss, Sohn *et al.* name the second loss term $\mathcal{L}_u$ after the unsupervised loss because it is established on the batch of the unlabeled data $\mathcal{U}_\rfloor$. FixMatch defines the unsupervised loss in the following form

$$\mathcal{L}_u = \frac{1}{B_\mu} \sum_{b=1}^{B_\mu} \mathbf{I}(\max(\mathbf{q}_b) > \tau) H(\widehat{q_b}, f(h(\mathbf{u}_b))). \quad (2)$$

Here, $h(\mathbf{u}_b)$ indicates a strongly augmented sample – an image sample transformed from $\mathbf{u}_b$ ($\mathbf{u}_b \in \mathcal{U}_\rfloor$) through brightness adjustment, image rotation etc. By minimizing this loss, FixMatch could ensure that the model does not change its prediction even after someone varies an unlabeled image's brightness, contrast, scale, etc. It should be noted that the unsupervised loss also contains a term $\mathbf{I}(\max(\mathbf{q}_b) > \tau)$ where $\mathbf{I}(\cdot)$ is an indicator function that outputs 1 if the condition in the parentheses holds; otherwise 0. In the equation, $\max(\mathbf{q}_b)$ takes the maximum value $\mathbf{q}_b$ and compares it with a pre-defined threshold τ. It indicates that not all the unlabeled data in the batch $\mathcal{U}_\rfloor$ are used for loss computation. The unlabeled data are selected if their predicted values in one class are above a pre-defined threshold τ. With this term, FixMatch ensures that it takes an unlabeled data sample $\mathbf{u}_b$ into the internal model parameter update process only if the model's output to that data sample ($\mathbf{q}_b = f(g(\mathbf{u}_b))$) is in high confidence.

Algorithm 1 shows the training process that FixMatch follows. As we can observe, FixMatch repeats the procedure above iteratively. In each iteration, it selects a batch of labeled and unlabeled data. Using the model updated in the previous iteration, FixMatch assigns pseudo labels to the batch of the unlabeled data, computes the loss, and updates the model's internal parameters. We discuss how we customize and extend this learning algorithm for our problem in the following.

5.3. Our Proposed Method

To use FixMatch for malware classification, we must address three vital technical problems. First, we need to define labeled and unlabeled data. In our dataset, all the data are labeled. However, the provided labels are not 100% correct. As is mentioned above, we need to identify those correctly labeled data and treat them as our labeled dataset and the rest as unlabeled data. Second, we need to redefine weak and strong augmentation. As is described above, FixMatch mutates image scale, brightness, contrast, etc. The malware semantic is different from that of images. Therefore, the augmentation defined in FixMatch cannot be applied to malware. Third, we need to augment FixMatch with the ability to handle extreme class imbalance. As is discussed above, malware data is highly skewed, which plagues existing noisy label learning methods. While techniques are designed to

Algorithm 2: Proposed learning algorithm. The customized and extended parts are highlighted.

Input: imbalanced noisy training dataset D, number of total training epochs K, labeled data's proportion d, starting re-weighting epoch T_d, confidence threshold τ, unsupervised loss weight λ, learning rate α.
Initialization: Initialize the weights $\mathbf{\Theta}$ for the model $f(\cdot)$ by using entire dataset D with only a few epochs.
for $k = 0, 1, 2, ..., K$ **do**
  Partitioning the training dataset D into labeled dataset $\mathcal{X}$ and unlabeled dataset $\mathcal{U}$: select the top $d\%$ examples with the least loss values from each given class and treat them as labeled data and the rest as unlabeled data.
  for $iter = 1, 2, ..., num\_batches$ **do**
    From $\mathcal{X}$, draw a mini-batch $\{(\mathbf{x}_b, y_b) : b \in (1, ..., B)\}$
    From $\mathcal{U}$, draw a mini-batch $\{\mathbf{u}_b : b \in (1, ..., B_\mu)\}$
    for $b = 1, 2, ..., B_\mu$ **do**
      // Use Eqn. (3) to perform weak augmentation against $\mathbf{u}_b$
      $\mathbf{q}_b = f(g(\mathbf{u}_b); \mathbf{\Theta})$
      $\widehat{q_b} = \arg\max(\mathbf{q}_b)$
    end
    if $k <$ T_d **then**
      // Use Eqn. (3) to perform weak augmentation against $\mathbf{x}_b$
      $\mathcal{L}_s = \frac{1}{B} \sum_{b=1}^{B} H(y_b, f(g(\mathbf{x}_b)))$
      // Use Eqn. (3) to perform strong augmentation against $\mathbf{u}_b$
      $\mathcal{L}_u = \frac{1}{B_\mu} \sum_{b=1}^{B_\mu} \mathbf{I}(\max(\mathbf{q}_b) > \tau) H(\widehat{q_b}, f(h(\mathbf{u}_b)))$
    else
      // Calculate the weight of each training sample using Eqn. (4)
      // Use Eqn. (3) to perform weak augmentation against $\mathbf{x}_b$
      $\mathcal{L}_s = \frac{1}{B} \sum_{b=1}^{B} w_b H(y_b, f(g(\mathbf{x}_b)))$
      // Use Eqn. (3) to perform strong augmentation against $\mathbf{u}_b$
      $\mathcal{L}_{u1} = \frac{1}{B_\mu} \sum_{b=1}^{B_\mu} \mathbf{I}(\max(\mathbf{q}_b) > \tau)$
      $\mathcal{L}_u = \mathcal{L}_{u1} + w_b H(\widehat{q_b}, f(h(\mathbf{u}_b)))$
    end
    $\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_u$
    Update the model's weights $\mathbf{\Theta}$ by minimizing the loss function $\mathcal{L}$
    Optional: decay the learning rate α
  end
end
Output: the well trained model $f(\cdot; \mathbf{\Theta})$.

address data imbalance, it is unclear how to integrate it into FixMatch's model learning procedure. Here, we discuss how we tackle these three technical challenges.

Labeled data selection. We first provide some key notations. Let $D = \{(\mathbf{x}_i, y_i) : i \in (1, ..., N)\}$ be our training dataset, where $\mathbf{x}_i \in \mathbb{R}^d$ is the input feature and $y_n \in \{1, ..., C\}$ is the given label. In each training epoch, we divide the training dataset into labeled data $\mathcal{X} = \{(\mathbf{x}_i, y_i) : i \in (1, ..., N_l)\}$ and unlabeled data $\mathcal{U} = \{\mathbf{u}_i : i \in (1, ..., N_u)\}$, where $N_l + N_u = N$.

As is discussed in Section 2, the sample selection approach utilizes the loss value as an indicator to distinguish correctly labeled data from the other. The samples with lower loss values are usually the ones with correct labels. Therefore, in this work, we follow the sample selection method to determine which samples are more likely to be correctly labeled and thus should be retained as labeled data.

Specifically, at each epoch, we use the updated model to compute the cross-entropy loss value for each data sample in the training set. For each class, we then select the top $d\%$ samples with the least loss values and treat them as the labeled dataset and the remaining as the unlabeled dataset. As is shown in Algorithm 2, the labeled and unlabeled data partition occurs at each training epoch. Note that a decent model is not available at the first epoch. To perform labeled and unlabeled data partition, we follow the sample selection method that employs the entire dataset to pre-train a model with only a few epochs. We then use the pre-trained model as the initial model to complete data separation.

Weak & strong augmentation. FixMatch's augmentation mechanism cannot be applied. Thus, we redefine the weak and strong augmentation used in FixMatch. Given a sample $\mathbf{x} \in \mathbb{R}^d$, we first generate a mask vector $\mathbf{m} = [m_1, ..., m_d]^T \in \mathbb{R}^d$ where m_j is either 0 or 1 randomly sampled from a Bernoulli distribution with probability p_a. Second, we augment the malware sample $\mathbf{x}$ and obtain its augmented version $\tilde{\mathbf{x}}$ by using the following equation

$$\tilde{\mathbf{x}} = \mathbf{x} \odot \mathbf{m} + (1 - \mathbf{m}) \odot \bar{\mathbf{x}}. \tag{3}$$

Here, the i-th feature of $\bar{\mathbf{x}}$ is sampled from that feature's empirical marginal distribution, which is defined as the uniform distribution over the values that feature takes on across the training dataset. For example, given a dataset with three samples {<1,32,5>, <2,66,9>, <9,99,8>}, the value of the 2-nd feature of $\bar{\mathbf{x}}$ is equal to a value randomly sampled from 32, 66, and 99. As we can observe from the equation above, our newly defined augmentation is to replace some feature values with those in other samples. Our data augmentation requires only simple operations in the feature space without making any assumptions about how those features are extracted. As such, our method could generalize well to malware features extracted by different methods or even data in other domains. Since our augmentation method only changes features, the augmented samples may be wrongly labeled. Our learning algorithm is designed to be resilient to such wrong labels because most augmented samples are used as unlabeled data and are relabeled with pseudo labels during the training. To verify this, we demonstrate in Appendix 10.7 that our method achieves a similar result as another one that better preserves the label correctness.

We use this equation for both weak and strong augmentation. The only difference is that the weak augment takes a relatively large value for p_a whereas the strong augmentation takes a small value. As such, our weak augmentation brings about minimal variation to malware samples, but the strong augmentation causes significant changes.

Sample re-weighting. Sample weighting is commonly adopted for handling class imbalance in a training set. The basic idea is to weigh the loss computed for different samples differently based on whether they belong to the majority or minority classes. Under this idea, the samples associated with minor classes could be assigned with a higher weight to the loss. In this work, we select the state-of-the-art sample weighting technique – sample re-weighting [41] – and integrate it into our customized FixMatch method.

The sample re-weighting scheme was introduced by Google in 2019. Instead of weighting the samples as the inverse of the class frequency [44] or the square root of class frequency [45], [46] for the class they belong to, it introduces a novel framework to measure the effective number of samples and treat the weight as the inverse of the effective number of samples.

Given a class y_b, the sample re-weighting scheme defines the effective number of samples as

$$E_{n_{y_b}} = \frac{1 - \beta^{n_{y_b}}}{1 - \beta}. \tag{4}$$

n_{y_b} is the number of samples in class y_b, and $\beta \in [0, 1)$ is a hyperparameter. Using its inverse (*i.e.,* $w_b = \frac{1}{E_{n_{y_b}}}$), we can extend the loss shown in Equation (1) and (2) as

$$\begin{aligned} \mathcal{L}_s &= \frac{1}{B} \sum_{b=1}^{B} w_b H(y_b, f(g(\mathbf{x}_b))); \\ \mathcal{L}_u &= \frac{1}{B_\mu} \sum_{b=1}^{B_\mu} \mathbf{I}(\max(\mathbf{q}_b) > \tau) w_b H(\widehat{q_b}, f(h(\mathbf{u}_b))). \end{aligned} \tag{5}$$

The sample re-weighting scheme is designed for supervised learning. To adapt it to our semi-supervised learning approach, we further customize the way to compute the effective sample number. For labeled data, we deem that n_{y_b} is the number of labeled samples belonging to the class y_b. For unlabeled data, we deem that n_{y_b} indicates the number of samples, the pseudo label of which belongs to the class y_b. Recall that, in Equation (2), we compute the loss using an unlabeled sample if its prediction q_b is above a threshold τ. As such, when counting n_{y_b} for unlabeled data, we consider only the unlabeled samples with the predictions above τ.

Intuition suggests that after replacing the loss of FixMatch with the ones above and then customizing FixMatch as discussed above, we could expect to learn a model that addresses the issues of data imbalance and high noise rate. However, recent research [42] discovers that deferring re-weighting until after the initial stage allows the model to learn an initial representation while avoiding some of the

complications associated with re-weighting. As a result, we further customize FixMatch as follows (See Algorithm 2). First, before annealing the learning rate, we train a model guided by Equation (5) without introducing the re-weighting strategy, *i.e.,* setting the $w_b = 1$ for both labeled and unlabeled samples. Second, we apply the re-weighting into model training with a lower learning rate.

6. Evaluation of Proposed Method

We evaluate the effectiveness and efficiency of our proposed noise learning method on multi-class malware classification tasks by using the datasets in Section 3 and 4.

6.1. Experiment Setup

Since we use the datasets introduced in Section 3 and 4 to assess our proposed methods, we can compare the performance of our proposed method with that of existing noise learning methods. To do it, we ensure the network structure, optimizer, and batch size used for both our approach and the existing methods are consistent. Specifically, we use a 3-layer Multi-layer Perceptron (MLP) network. Both our proposed method and existing techniques can be applied to DNN with different structures. To adapt them for malware data, we use a MLP rather than CNN which is mostly used for images. We train it using Adam optimizer with a learning rate of 0.001, a weight decay of 0.0002, and a batch size of 128. Again, to ensure model parameter initialization and data partition has minimal impact on a malware classifier's performance, we vary these factors and compute the average model performance. As for the hyper-parameters in our method, we set their values to the values demonstrating the optimal overall performance (see Appendix 10.2). Finally, we design an experiment to study how the percentage of labeled data affects our method's overall performance. Based on the experiment result, we set the d (*i.e.,* the percentage of labeled data) as 15 across all the datasets. It should be noted that we train the corresponding malware classifiers on the same machine – a server with 4 NVIDIA RTX A6000 GPUs for all the learning algorithms. All the experiment results are observed from the same computation resource.

6.2. Experiment Design

We design our experiments from two perspectives. First, we quantify the overall performance of the proposed method. Second, we investigate how the critical individual factors (*e.g.,* sample re-weighting, selected labeled data) play a role in offsetting noisy data's impact on overall malware categorization. Below, we describe the detailed experiment designs.

Experiment I: Effectiveness & Efficiency. Here, we want to examine if MORSE outperforms existing methods and offset the incorrectly labeled data's impact on malware categorization. As a result, we treat the aforementioned eight representative noise learning methods as our baseline approaches and compare their performance with MORSE on the above datasets. Note that we also compare MORSE with some widely used traditional ML methods and show the results in Appendix 10.3.

Experiment II: Sample re-weighting. Recall that we introduce a sample re-weighting scheme to deal with the extreme data imbalance problem. To understand the contribution of the sample re-weighting to MORSE, we design an experiment in which we remove the sample re-weighting from our approach, run the revised learning method, and compare the performance of our modified method with our original approach. In Appendix 10.4, we further add the sample re-weighting to baseline and evaluate their performance.

Experiment III: Labeled data's proportion. Recall that our proposed method divides the entire dataset into labeled and unlabeled subsets. When determining labeled data samples, we select the top $d\%$ of samples with the minimal loss values from each given class. To study d's impact on the performance of our proposed method, we design an experiment that varies d's value range from 5 to 95 by increasing 10 each time. In our evaluation, we use the synthetic datasets for this study.

Experiment IV: Different network structures. Recall that we use a MLP rather than CNN to process malware feature vectors. To demonstrate our design is orthogonal to network structures, we replace the MLP with CNN and record the performance on two real-world datasets in Section 3.

6.3. Experiment Results

Effectiveness on synthetic dataset. Table 7 shows the performance of the model learned from our proposed method. As we can observe, our proposed approach demonstrates the capability in offsetting the impact of the incorrectly labeled data. Comparing with the results shown in Table 6, we can also discover that our method demonstrates a better ability to improve the model's classification performance than any existing methods. For example, when synthetic noise reaches 60% and data imbalance is 100x, the model learned from our method shows a 72.05% malware classification accuracy, which is a 6.48% improvement over vanilla (*i.e.,* directly learning a model from the noisy dataset). On the contrary, the models learned from all other approaches demonstrate the performance even lower than that from the noisy dataset.

Effectiveness on real-world datasets. Besides the performance improvement on the synthetic datasets, MORSE also demonstrates the ability to improve model accuracy on the real-world malware datasets. Comparing the results shown in Table 9 with those in Table 5, our technique outperforms the vanilla method and other noisy learning methods on the Android dataset. Under the confidence interval of 0.05, it achieves a 6.76% improvement in classification accuracy. On the PE dataset, MORSE is the only approach that outperforms the vanilla method (with a 1.14% improvement). We argue this improvement is significant because even if we train our network directly on the clean version of this real-world dataset, we could merely obtain about 2% accuracy improvement over the vanilla method. It indicates that MORSE could

TABLE 7: The testing performance of the models learned from MORSE and Vanilla method on the synthetic dataset.

Methods	Noise rate 0.3 and imbalance ratio 20x						Noise rate 0.3 and imbalance ratio 100x					
	Overall cls (%)			Rare cls (%)			Overall cls (%)			Rare cls (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	75.55/1.38	82.94/3.78	73.46/0.90	57.89/3.69	85.74/6.65	63.03/1.20	70.83/0.87	76.29/0.85	67.80/1.19	48.45/1.44	79.47/0.62	56.77/0.24
Vanilla DNN w re-weighting	78.87/0.30 p = 0.000	82.64/0.88 p = 0.335	77.71/0.44 p = 0.000	68.59/2.29 p = 0.000	80.75/2.44 p = 0.878	69.63/1.03 p = 0.000	72.85/1.79 p = 0.083	75.97/0.98 p = 0.656	69.86/1.74 p = 0.049	56.81/4.63 p = 0.009	76.44/1.89 p = 0.988	60.88/2.64 p = 0.008
Our method w/o re-weighting	79.64/0.83 p = 0.000	87.81/1.68 p = 0.011	77.61/1.01 p = 0.000	64.85/2.62 p = 0.001	94.46/2.31 p = 0.009	70.00/2.70 p = 0.000	71.73/2.24 p = 0.138	73.38/4.94 p = 0.900	65.87/3.16 p = 0.891	46.84/3.98 p = 0.752	72.99/9.19 p = 0.917	50.54/5.68 p = 0.972
Our method	79.73/1.85 p = 0.008	87.11/1.82 p = 0.039	79.11/1.30 p = 0.000	67.47/3.17 p = 0.006	88.45/4.63 p = 0.182	70.76/3.42 p = 0.002	73.47/0.91 p = 0.001	78.97/2.11 p = 0.027	70.42/1.02 p = 0.007	54.40/7.97 p = 0.092	82.59/3.85 p = 0.071	62.30/5.55 p = 0.040
Methods	**Noise rate 0.6 and imbalance ratio 20x**						**Noise rate 0.6 and imbalance ratio 100x**					
	Overall cls (%)			Rare cls (%)			Overall cls (%)			Rare cls (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	68.04/1.77	70.38/6.31	63.68/4.12	48.93/4.03	73.16/9.31	53.39/7.05	65.57/0.87	71.67/3.30	61.11/0.89	44.39/1.94	76.67/7.45	51.02/1.56
Vanilla DNN w re-weighting	68.01/1.52 p = 0.464	70.65/2.95 p = 0.448	65.52/2.68 p = 0.184	58.50/2.59 p = 0.011	67.70/5.18 p = 0.960	57.18/3.31 p = 0.106	66.84/2.36 p = 0.137	67.22/2.28 p = 0.979	64.34/2.33 p = 0.021	55.85/4.36 p = 0.001	61.15/4.13 p = 0.996	56.36/4.29 p = 0.037
Our method w/o re-weighting	72.82/2.86 p = 0.012	77.24/1.69 p = 0.050	69.42/2.61 p = 0.038	54.09/3.96 p = 0.051	81.00/1.34 p = 0.061	60.26/2.85 p = 0.041	64.85/4.83 p = 0.624	65.85/5.03 p = 0.927	58.93/2.81 p = 0.988	38.65/9.12 p = 0.895	66.67/8.43 p = 0.898	42.38/6.12 p = 0.987
Our method	76.20/0.59 p = 0.000	79.45/1.69 p = 0.016	72.90/1.17 p = 0.003	61.21/1.00 p = 0.000	80.45/0.99 p = 0.066	64.25/0.20 p = 0.009	72.05/1.94 p = 0.000	74.97/3.41 p = 0.005	68.03/1.01 p = 0.000	53.82/4.98 p = 0.005	76.60/7.48 p = 0.891	58.00/3.80 p = 0.003

TABLE 8: MORSE vs. Vanilla method on the PE dataset.

Methods	Average (%)			Class-11 (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	93.08/0.25	93.65/0.51	92.57/0.36	44.33/3.78	96.34/4.22	60.48/3.22
Vanilla DNN w re-weighting	93.82/0.29 p = 0.003	94.01/0.30 p = 0.102	93.50/0.34 p = 0.001	55.67/2.98 p = 0.001	89.71/2.30 p = 0.974	68.61/1.70 p = 0.001
Our method w/o re-weighting	92.50/0.18 p = 0.980	93.35/0.28 p = 0.924	91.33/0.58 p = 0.992	36.50/2.75 p = 0.995	99.51/1.10 p = 0.089	51.07/5.05 p = 0.979
Our method	94.15/0.43 p = 0.003	94.14/0.57 p = 0.061	93.91/0.67 p = 0.000	65.33/4.20 p = 0.000	90.18/2.08 p = 0.984	76.74/1.22 p = 0.000

TABLE 9: MORSE vs. Vanilla on the Android dataset.

Methods	Average (%)			Class-6 (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	73.00/0.35	78.65/2.21	69.96/0.54	2.67/5.53	19.79/34.31	4.63/9.52
Vanilla DNN w re-weighting	77.32/1.72 p = 0.002	82.20/3.73 p = 0.049	75.68/2.58 p = 0.003	20.83/7.99 p = 0.007	80.39/9.40 p = 0.003	32.03/8.99 p = 0.005
Our method w/o re-weighting	74.58/0.86 p = 0.004	77.33/4.47 p = 0.691	71.00/0.96 p = 0.015	5.17/7.31 p = 0.309	32.35/45.79 p = 0.347	8.91/12.60 p = 0.311
Our method	79.76/1.02 p = 0.000	80.37/1.99 p = 0.088	78.72/1.22 p = 0.000	27.67/8.01 p = 0.000	58.54/18.90 p = 0.039	35.80/7.88 p = 0.000

learn a model on a noisy dataset with accuracy, almost the same as learning on a clean dataset. Recall that existing methods cannot perform well in the smallest class where the high noise rate and the data imbalance are extreme. Here, we, therefore, closely look at the performance of MORSE on the smallest class in both datasets. We discover that MORSE significantly improves the model's accuracy on the smallest class (Table 9 class-6 and Table 8 class-11). This result further validates the effectiveness of MORSE in handling high noise and extreme data skewness.

Effectiveness of MORSE on clean datasets. Table 10 shows the performance of MORSE and the vanilla method on the clean version of our real-world datasets and synthetic datasets. As shown in the table, MORSE also outperforms the vanilla method on these clean datasets, verifying the effectiveness of our designs on clean and imbalance datasets.

Efficiency comparison. From Table 11, we can observe that the average runtime of our method is about 3.7 times slower than the vanilla method (2.45 sec vs. 0.65 sec). Compared to existing noise learning methods, our method is also relatively less computationally efficient. The reason behind this low efficiency comes from two aspects. First, as we mentioned in Section 5, MORSE needs to divide the entire training dataset into labeled and unlabeled subsets at each training epoch. This dataset partition requires computing training samples' loss values, which introduces more computations. Second, our method introduces a more sophisticated loss function than other approaches. Optimizing this loss requires more computation resources. However, we believe that the overhead that MORSE introduces is acceptable not only because it brings significant improvement in model accuracy but, more importantly, the model training is an offline process. Once a malware classification model is trained and deployed, security professionals do not need to update their model constantly.

With & without sample re-weighting. Table 7, 8 and 9 show the performance of the vanilla method and MORSE with/without the sample re-weighting mechanism. As we can first observe from the tables, while removing the sample re-weighting from MORSE, our customization to FixMatch still outperforms the vanilla method, when the data skewness is extreme but not ultra-extreme (*e.g.,* 72.82% vs. 68.04%, when the noise rate is 60% and data imbalance is 20x). This effect is because our customization allows the learning algorithm to treat most of the incorrectly labeled data as unlabeled data, which minimizes their negative impact on model learning. However, This result also shows that that our method sometime fails to offset noisy data's impact on model learning without sample re-weighting, especially when the data imbalance is high (*e.g.,* synthesize datasets with imbalance ratio 100x). It indicates that the sample re-weighting scheme is necessary for our problem.

From Table 7, 8 and 9, we can also observe that even with the sample re-weighting method, the vanilla method still cannot outperform MORSE, verifying the necessity of our choice of semi-supervised learning framework and other customized designs. In Appendix 10.4, we apply the sample re-weighting method to our selected noise learning methods and demonstrate that even augmented with the sample re-weighting, these methods still perform worse than MORSE, further confirming the necessity of our other designs.

Hyper-parameter sensitivity analysis. Figure 2 shows how the percentage of labeled data d impacts the malware classifier's performance. As we can observe, the performance is relatively robust to d to some extent. When the value of d is below a threshold (*i.e.,* 35 in the noise rate 0.6) but not too small (*i.e.,* larger than 5 in this setting), the classification accuracy remains relatively consistent. We note that the classification accuracy significantly decreases after we set the value of d over the threshold. It indicates that when the noise rate is high (60%) and we select more than 35% data as labeled data, more incorrectly labeled data would be acci-

TABLE 10: The performance of MORSE and vanilla method on clean datasets. "Real-PE," "Real-Android," "Syn-imb-20x" and "Syn-imb-100x" refers to PE dataset, Android dataset, synthetic datasets with the imbalance ratio of 20x and 100x, respectively.

Methods	Real-PE (%)			Real-Android (%)			Syn-imb-20x (%)			Syn-imb-100x (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	95.01/0.16	96.06/0.19	94.93/0.19	80.04/0.25	87.66/0.37	79.39/0.20	77.49/0.93	79.86/0.77	74.49/0.94	73.69/0.36	78.09/0.24	70.32/0.31
Our method	95.22/0.12 p = 0.018	95.84/0.13 p = 0.954	95.12/0.13 p = 0.032	82.63/2.23 p = 0.026	85.25/0.84 p = 0.999	82.36/2.05 p = 0.012	80.52/1.12 p = 0.000	87.81/1.72 p = 0.000	78.20/0.86 p = 0.000	77.62/2.06 p = 0.006	83.07/1.38 p = 0.000	75.51/2.45 p = 0.003

TABLE 11: The running time comparison on the synthetic dataset with noise-ratio 0.6 and 20x data imbalance.

Methods	Run time per epoch(s)	Methods	Run time per epoch(s)
Vanilla DNN	0.65	LRT	2.13
Coteaching+	2.11	LIO	1.00
MentorMix	0.94	GCE	0.67
Bootstrap	0.71	ELR	1.12
Noise-adaption	1.30	Our method	2.45

TABLE 12: The testing performance of MORSE and vanilla method using a CNN on the real-world datasets.

Methods	Real-PE (%)			Real-Android (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla CNN	92.38/0.24	93.28/0.48	91.58/0.31	71.23/1.22	82.07/4.24	68.66/1.70
Our method CNN	93.10/0.23 p = 0.002	93.08/0.25 p = 0.774	92.80/0.30 p = 0.000	75.14/0.94 p = 0.000	75.85/1.06 p = 0.991	74.21/1.19 p = 0.000

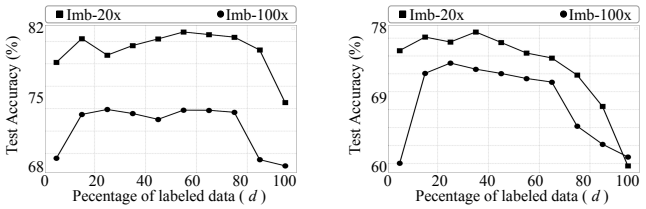
dentally treated as clean data and thus negatively influence the model learning. As a result, considering the malware dataset might be in a high noise rate, we conservatively set a relatively low value for d (*i.e.,* 15 for all tasks).

Generalizability across different network structure. Table 12 shows the results of our method with CNN. First, compared with the results in Table 8 and 9, replacing MLP with CNN causes a performance drop, verifying the necessity of customizing the network structure for malware data. Second, Table 12 shows that MORSE outperforms the vanilla method with this CNN network, showing the effectiveness of our designs on different DNN structures.

7. Other Related Work

Semi-supervised learning. As is mentioned in Section 5, we customize and extend the state-of-art semi-supervised learning method – FixMatch – to handle our problem. In addition to FixMatch, there are many other semi-supervised learning approaches. For example, pseudo-labeling methods [47], [48], [49] use labeled data to predict the unlabeled data's labels and then train the model in a supervised fashion with the combination of labeled and selected pseudo-labeled data. Consistency regularization methods [50], [51], [52] encourage the model to produce similar predictions for the perturbed version of the unlabeled data. Current state-of-the-art semi-supervised learning methods [43], [53] combine these two techniques and produce improved pseudo labels. However, none of the above semi-supervised learning methods are designed to handle the class imbalance problem. Under the extreme data skewness setup, the quality of pseudo labels would be largely impacted by the majority class. In this work, we augment FixMatch with the ability to counteract the impact of data imbalance.

Class-imbalanced supervised learning. There is a large body of research on imbalanced data learning. Classical methods focus on designing re-sampling strategies [54], [55], [56], which over-sample the minority classes and



(a) Dataset with noise-rate 0.3. (b) Dataset with noise-rate 0.6.

Figure 2: The test accuracy of the model learned from our method with d being set up with different values. Note that hyper-parameter d indicates percentage of data samples that MORSE selects as labeled data. "Imb-20x" and "Imb-100x" indicate the 20x and 100x data imbalance, respectively.

under-sample the majority classes. In addition, re-weighting schemes [57], [44], [58] are proposed to adjust the weights during training for different classes or even different samples. Apart from classical methods, another line of research [42], [41], [59] develops a new form of loss functions to handle the imbalance problem. Besides loss function re-design, some works [60], [61] aim to transfer knowledge from majority classes to minority classes. A recent work [62] decouples the learning of representation and classifier. However, all of the methods above assume the training data is all correctly labeled, and their performance is largely unknown in the noisy settings.

Malware detection and classification. Machine learning has been widely used in malware detection [63], [64], [65], [66], [67], [68], [69], [70], [18], [71] and malware family attribution [1], [2], [72] in both academic and industry environments. Among these research works, most assume that all sample labels are correct. To the best of our knowledge, the works [22], [73] are the only two works exploring malware identification against incorrect labels. [3]

[22] cannot be used to handle malware classification in the real world. First, this work is designed to handle only binary but not multi-class malware classification. Specifically, this method corrects wrong labels via flipping, which is applicable only to binary classification. This work does not provide a mechanism for label correction in multi-class setups and thus cannot be generalized to those settings. Second, as is mentioned in Section 3, real-world malware dataset contains high noise and suffers from data imbalance issues. However, the work [22] is designed for synthetic malware datasets in which the noise rate is relatively low, and the data imbalance issues do not exist. In this work, we cannot compare our work with [22], particularly in the experiment settings mentioned above. Therefore, we set up

3. [71] proposed a malware detection method to combat concept drifts. This work studied the impact of noise labels on its method but did not provide a method to handle noisy labels.

a binary classification task for this comparison (see Appendix 10.5). [73] handles malware categorization under inaccurately assigned labels. Using the combination of clustering algorithms, this method clusters malware samples rather than predicting their labels. As such, it needs post-clustering manual efforts to align these clusters with "real" labels. MORSE well complements existing works. It augments semi-supervised learning with the sample re-weighting scheme, allowing security analysts to solve multi-class classification problems under high-noise and imbalanced settings.

Besides designing better ML models, another way to alleviate the impact of noisy labels upon classifiers is to correct label noises, or in other words, assign malware with more accurate labels [74], [75]. Relabeling and MORSE work in different stages of the learning pipeline (*i.e.,* labeling and modeling). They are orthogonal and can be combined. Recall that we show that MORSE has a better performance on clean datasets than noisy datasets in Section 6 This result shows that high-quality labels could indeed improve MORSE. Since the optimal performance for relabeling is to recover clean labels, this result also verifies that relabeling methods could be combined with MORSE and give a better result.

8. Discussion & Future Work

Adversarial attacks. Recent research shown that an attacker could manipulate a malware sample and thus generate an adversarial sample capable of deceiving the malware classifier (*e.g.,* [76], [77]). Like all other ML algorithms, MORSE could also be vulnerable to adversarial attacks. However, we believe building adversary resistance for malware classifiers is orthogonal to our work. Existing approaches to robustifying a learning model could potentially be applied to our proposed method. For example, we could combine the adversarial training technique [78] with our method to improve a malware classifier's robustness against adversarial examples. This paper leaves the evaluation and implementation of MORSE's adversarial robustness as future work.

Label poisoning attacks. Going beyond handling noisy labels, we argue that MORSE could naturally counteract label poisoning attacks [79]. In a label poisoning attack, an adversary flips the training labels and thus influences the performance of the learned classifier. Recall that our method could treat incorrectly labeled samples as unlabeled samples. Using a customized, extended semi-supervised learning method, we could naturally minimize the impact of manipulated training labels. As a result, label poisoning attacks are not likely effective to MORSE. To verify this argument, we show the robustness of MORSE against an existing attack [79] in Appendix 10.6.

Combating concept drifts. MORSE is not designed to handle concept drift. As shown in Appendix 10.8, concept drifts have a negative impact on MORSE. Note that MORSE is still useful even with concept drifts. To train a malware classifier, one needs to first collect an initial training dataset. This training set contains the samples prior to the current time stamps and thus typically does not contain too many concept drifts. With the training set by hand, MORSE provides a useful and practical solution for training a decent classifier. Once the classifier is deployed and new testing samples are continuously being received, these samples can contain concept drift. To address concept drift, we could combine the existing methods [18], [80] to detect drifts, relabel them, and update the MORSE's classifier periodically.

Data augmentation and sample re-weighting. As a defender, we mainly consider the feature space rather than problem space [81] because every input needs to be transformed into the feature space before feeding into the model. As long as our augmentation covers the feature space, it also covers the valid malware in the problem space and thus will maintain the performance of trained classifiers on real malware data. As such, we do not require the augmented sample to be valid malware. It is more of the attacker's concern to maintain the functionality of perturbed malware. Regarding the sample re-weighting method, we selected the state-of-art method [41] that performs the best in our evaluation. Our future work will investigate the efficacy of other re-weighting methods (*e.g.,* [82], [24]).

Other future works. First, we will evaluate larger datasets (*e.g.,* AndroZoo [83]) and more complicated DNN architectures (*e.g.,* Transformer [84]). Second, to handle benign samples, we could build a two-stage/hierarchical model with two MORSE classifiers, where the first classifier detects malicious samples, and the second one identifies the malware families. Finally, to generalize MORSE to other applications, we need two additional effort. (1) Some technical customization might be necessary. For example, when performing data augmentation using our proposed technique, we may need to re-design the weak and strong data augmentation methods based on the semantics of the specific applications. (2) We need to design a decent experiment to gather datasets to reflect the real-world noise on the corresponding applications.

9. Conclusion

A real-world malware dataset can be highly imbalanced and contains many incorrect labels. As such, it is challenging for security analysts to learn an accurate malware classifier using a typical supervised learning method on such datasets. Using and extending semi-supervised learning techniques, we can minimize and even offset the impact of incorrect labels. While previous research also proposed many techniques to offset the impact of incorrect labels. However, when the noise rate is high, and data imbalance is extreme, existing methods become futile. On the contrary, our method consistently shows higher malware classification accuracy. With this discovery, we safely conclude that a semi-supervised learning framework could be a practical, effective approach to handling noise learning problems.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This project was supported in part by NSF grant 2225234, 2225225, by the Amazon Research Award.

References

[1] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *CODASPY*, 2016.

[2] T. Chakraborty, F. Pierazzi, and V. Subrahmanian, "Ec2: Ensemble clustering and classification for predicting android malware families," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[3] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *CCS*, 2020.

[4] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, "Co-teaching: Robust training of deep neural networks with extremely noisy labels," in *NeurIPS*, 2018.

[5] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *ICML*, 2018.

[6] L. Jiang, D. Huang, M. Liu, and W. Yang, "Beyond synthetic noise: Deep learning on controlled noisy labels," in *ICML*, 2020.

[7] E. Arazo, D. Ortego, P. Albert *et al.*, "Unsupervised label noise modeling and loss correction," in *ICML*, 2019.

[8] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training neural networks with noisy labels," in *NeurIPS*, 2018.

[9] S. Liu, J. Niles-Weed, N. Razavian, and C. Fernandez-Granda, "Early-learning regularization prevents memorization of noisy labels," in *NeurIPS*, 2020.

[10] J. Goldberger and E. Ben-Reuven, "Training deep neural-networks using a noise adaptation layer," in *ICLR*, 2016.

[11] Y. Yao, T. Liu, B. Han, M. Gong, J. Deng, G. Niu, and M. Sugiyama, "Dual t: Reducing estimation error for transition matrix in label-noise learning," in *NeurIPS*, 2020.

[12] X. Yu, B. Han, J. Yao *et al.*, "How does disagreement help generalization against label corruption?" in *ICML*, 2019.

[13] Y. Zhang, G. Niu, and M. Sugiyama, "Learning noise transition matrix from only noisy labels via total variation regularization," in *ICML*, 2021.

[14] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, "Using trusted data to train deep networks on labels corrupted by severe noise," in *NeurIPS*, 2018.

[15] X. Xia, T. Liu, B. Han, N. Wang, M. Gong, H. Liu, G. Niu, D. Tao, and M. Sugiyama, "Part-dependent label noise: Towards instance-dependent label noise," in *NeurIPS*, 2020.

[16] A. J. Bekker and J. Goldberger, "Training deep neural-networks based on unreliable labels," in *ICASSP*, 2016.

[17] E. Fonseca, M. Plakal, D. P. Ellis *et al.*, "Learning sound event classifiers from web audio with noisy labels," in *ICASSP*, 2019.

[18] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "{TESSERACT}: Eliminating experimental bias in malware classification across space and time," in *USENIX Security Symposium*, 2019.

[19] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.

[20] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online {Anti-Malware} engines," in *USENIX Security Symposium*, 2020.

[21] R. J. Joyce, D. Amlani, C. Nicholas, and E. Raff, "Motif: A large malware reference dataset with ground truth family labels," *arXiv preprint arXiv:2111.15031*, 2021.

[22] J. Xu, Y. Li, and R. H. Deng, "Differential training: A generic framework to reduce label noises for android malware detection," in *NDSS*, 2021.

[23] Y. Shen and S. Sanghavi, "Learning with bad training data via iterative trimmed loss minimization," in *ICML*, 2019.

[24] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, "Meta-weight-net: Learning an explicit mapping for sample weighting," in *NeurIPS*, 2019.

[25] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *ICML*, 2018.

[26] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[27] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," *arXiv preprint arXiv:1412.6596*, 2014.

[28] S. Zheng, P. Wu, A. Goswami, M. Goswami, D. Metaxas, and C. Chen, "Error-bounded correction of noisy labels," in *ICML*, 2020.

[29] X. Ma, H. Huang, Y. Wang, S. Romano, S. Erfani, and J. Bailey, "Normalized loss functions for learning with noisy labels," in *ICML*, 2020.

[30] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding learning (still) requires rethinking generalization," *Communications of the ACM*, 2021.

[31] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio *et al.*, "A closer look at memorization in deep networks," in *ICML*, 2017.

[32] X. Xia, T. Liu, N. Wang, B. Han, C. Gong, G. Niu, and M. Sugiyama, "Are anchor points really indispensable in label-noise learning?" in *NeurIPS*, 2019.

[33] S. Wu, X. Xia, T. Liu, B. Han, M. Gong, N. Wang, H. Liu, and G. Niu, "Class2simi: A noise reduction perspective on learning with noisy labels," in *ICML*, 2021.

[34] VirusTotal, "Virustotal," https://www.virustotal.com/, 2022.

[35] T. C. Miranda, P.-F. Gimenez, J.-F. Lalande, V. V. T. Tong, and P. Wilke, "Debiasing android malware datasets: How can i trust your results if your dataset is biased?" *IEEE Transactions on Information Forensics and Security*, 2022.

[36] M. Chandramohan, H. B. K. Tan, and L. K. Shar, "Scalable malware clustering through coarse-grained behavior modeling," in *FSE*, 2012.

[37] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *MALWARE*, 2015.

[38] R. V. Hogg, E. A. Tanis, and D. L. Zimmerman, *Probability and statistical inference*. Pearson/Prentice Hall Upper Saddle River, NJ, USA:, 2010.

[39] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "Bodmas: An open dataset for learning based temporal analysis of pe malware," in *S&P Workshop*, 2021.

[40] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, 2018.

[41] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *CVPR*, 2019.

[42] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," in *NeurIPS*, 2019.

[43] K. Sohn, D. Berthelot, N. Carlini, Z. Zhang, H. Zhang, C. A. Raffel, E. D. Cubuk, A. Kurakin, and C.-L. Li, "Fixmatch: Simplifying semi-supervised learning with consistency and confidence," in *NeurIPS*, 2020.

[44] C. Huang, Y. Li, C. C. Loy, and X. Tang, "Learning deep representation for imbalanced classification," in *CVPR*, 2016.

[45] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. Van Der Maaten, "Exploring the limits of weakly supervised pretraining," in *ECCV*, 2018.

[46] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NeurIPS*, 2013.

[47] D.-H. Lee *et al.*, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *ICML Workshop*, 2013.

[48] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *CVPR*, 2020.

[49] M. N. Rizve, K. Duarte, Y. S. Rawat, and M. Shah, "In defense of pseudo-labeling: An uncertainty-aware pseudo-label selection framework for semi-supervised learning," *arXiv preprint arXiv:2101.06329*, 2021.

[50] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning," in *NeurIPS*, 2016.

[51] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *arXiv preprint arXiv:1610.02242*, 2016.

[52] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: a regularization method for supervised and semi-supervised learning," *IEEE TPARMI*, 2018.

[53] D. Berthelot, N. Carlini, E. D. Cubuk, A. Kurakin, K. Sohn, H. Zhang, and C. Raffel, "Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring," *arXiv preprint arXiv:1911.09785*, 2019.

[54] N. Japkowicz, "The class imbalance problem: Significance and strategies," in *IJCAI*, 2000.

[55] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, 2002.

[56] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, 2009.

[57] J. Byrd and Z. Lipton, "What is the effect of importance weighting in deep learning?" in *ICML*, 2019.

[58] C. Huang, Y. Li, C. C. Loy, and X. Tang, "Deep imbalanced learning for face recognition and attribute prediction," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[59] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, "Long-tail learning via logit adjustment," *arXiv preprint arXiv:2007.07314*, 2020.

[60] X. Yin, X. Yu, K. Sohn, X. Liu, and M. Chandraker, "Feature transfer learning for deep face recognition with under-represented data," *arXiv preprint arXiv:1803.09014*, 2018.

[61] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, "Large-scale long-tailed recognition in an open world," in *CVPR*, 2019.

[62] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, "Decoupling representation and classifier for long-tailed recognition," *arXiv preprint arXiv:1910.09217*, 2019.

[63] H. S. Anderson and P. Roth, "Ember: an open dataset for training static pe malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.

[64] Y. Chen, S. Wang, D. She, and S. Jana, "On training robust {PDF} malware classifiers," in *USENIX Security Symposium*, 2020.

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

[66] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *COMPSAC*, 2015.

[67] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.

[68] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "{ZOZZLE}: Fast and precise {In-Browser}{JavaScript} malware detection," in *USENIX Security Symposium*, 2011.

[69] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu *et al.*, "Deepintent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps," in *CCS*, 2019.

[70] J. DeLoach, D. Caragea, and X. Ou, "Android malware detection with weak ground truth data," in *IEEE BigData*, 2016.

[71] Z. Kan, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Investigating labelless drift adaptation for malware detection," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, 2021.

[72] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *MALWARE*, 2017.

[73] J. Liang, W. Guo, T. Luo, V. Honavar, G. Wang, and X. Xing, "Fare: Enabling fine-grained attack categorization under low-quality labeled data," in *NDSS*, 2021.

[74] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar, "Better malware ground truth: Techniques for weighting anti-virus vendor labels," in *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, 2015.

[75] M. Hurier, G. Suarez-Tangil, S. K. Dash, T. F. Bissyandé, Y. Le Traon, J. Klein, and L. Cavallaro, "Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware," in *MSR*, 2017.

[76] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[77] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *KDD*, 2017.

[78] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[79] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri, and M. Conti, "On defending against label flipping attacks on malware detection systems," *Neural Computing and Applications*, 2020.

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

[81] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *S&P*, 2020.

[82] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, 2017.

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

[84] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," in *NeurIPS*, 2017.

[85] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, 2009.

[86] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in computer Virology*, 2011.

[87] T. Wüchner, M. Ochoa, and A. Pretschner, "Robust and effective malware detection through quantitative data flow graph metrics," in *DIMVA*, 2015.

[88] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of computer security*, 2011.

[89] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *NTMS)*. IEEE, 2018.

[90] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *VizSec*, 2011.

[91] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[92] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[93] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[94] A. Paudice, L. Muñoz-González, and E. C. Lupu, "Label sanitization against label flipping poisoning attacks," in *ECML/PAKDD*, 2018.

10. Appendix

10.1. Feature Engineering in the PE Dataset

We adopt a common static feature extraction method [37] to process the PE malware files in our real-world dataset. For each PE, this method extracts four sets of features, each of which is a 256-dimensional vector, and concatenates these feature vectors into a final vector representation with a length of 1,024.

- **Byte entropy histogram features** characterize the frequency of each hex in the binary sequence. Specifically, it first computes the entropy for each hex within a fixed-length sliding window and marks each entropy value and hex value as a pair. Then, it counts the frequency of each hex-entropy pair as the sliding window passes through the binaries. From these pairs, it identifies the maximum and minimum value of the entropy and cuts this value range into 16 intervals with the same length. Similarly, it also splits the [0, 255] value range of hex into 16 even intervals. With these two intervals, it then constructs a 16×16 zero matrix $\mathbf{X}$, where $\mathbf{X}[i, j]$ represents the i-th interval of the hex value and the j-th interval of the entropy value. If the values of a hex-entropy pair fall into the intervals represented by $\mathbf{X}[i, j]$, it adds 1 to $\mathbf{X}[i, j]$. Here, the value of $\mathbf{X}[i, j]$ stands for the number of pairs falling into the corresponding intervals. Finally, it transforms the 16×16 matrix into a 256-dimensional vector.
- **PE import features** capture the function calls within the given malware. To do so, this method first extracts the import address table of the given file and leverages a hash function to project the function name ad its corresponding DDL name of each function call within the table into a number between [0, 255]. Finally, it then counts the frequency of these numbers as the feature values.
- **String histogram features** represent the printable strings in the given file. For each printable string, this method first transforms it into a value between [0, 15] using a hash function and records the length of this string. Using the method in byte entropy histogram features, it then constructs a frequency matrix for hash value and string length. Finally, the feature vector can be generated by flattening the matrix.
- **PE metadata features** capture the frequency of numerical PE fields in the given file. Similar to PE import features, here, the textual name of each PE field is hashed into a number between [0, 255]. Then, the numbers' frequencies are used as the feature values.

Recent research also explores extracting features via dynamic analysis [85], [86], [87], [88] or transforming raw binaries into grey-scale images [89], [90]. Recent work [39] shows that the computational cost of extracting dynamic features is much higher than the cost of extracting static features. Converting malware into images would break the semantic information in the original binaries [91].

10.2. Implementation Details

Implementations. We implement `MORSE` using the `Pytorch` [92] package. As for the existing noise learning methods, we use the code released by the authors or the popular implementations publicly available if the authors didn't release codes. Below are the links of the implementations: Coteaching+: https://github.com/xingruiyu/coteaching_plus; MentorMix: https://github.com/google-research/google-research/tree/master/mentormix; LRT: https://github.com/pingqingsheng/LRT; Noise-adaption: https://github.com/uidbr/noisy_labels; LIO: https://github.com/YivanZhang/lio; ELR: https://github.com/shengliu66/ELR; Bootstrap and GCE: https://github.com/YisenWang/symmetric_cross_entropy_for_noisy_labels.

Hyper-parameters of `MORSE`. Our method has three sets of hyper-parameters: model structure and training hyper-parameters, hyper-parameters inherent from FixMatch and selected sample re-weighting method, hyperparameters introduced by our methods. Regarding the first set of hyper-parameters, we set the structure the same across all the methods: MLP with ReLU activation-2381-1024-1024-10 for the synthetic dataset, MLP with ReLU activation-1024-512-512-12 for the PE dataset, and MLP with ReLU activation-217-512-512-13 for the Android dataset. The CNN structure is as follows. We use two one-dimensional convolutional layers with the kernel size of 1 and 3 and two max-pooling layers with the kernel size of 4. After the second pooling layer, we use a MLP with the structure of 256-256.

We also use the same training hyper-parameters for all the methods: adam [93] with a learning rate of 0.001, a batch size of 128. The epoch number is 100 for synthetic datasets, 140 for the PE dataset, and 100 for the Android dataset. For hyper-parameters inherent from FixMatch and selected sample re-weighting method (unsupervised loss weight: λ, threshold of pseudo labeling: γ, and hyper-parameter in sample re-weighting: β), we tune them and find our method

TABLE 13: Accuracy by varying the unique hyper-parameter of baselines on the PE real-world dataset. The bold one in column-3 is our hyper-parameter choice. We run each experiment six times and report the mean and standard errors.

Methods	Method unique hyper-parameter	Vary range	Mean/std of classification accuracy (%)
Coteaching+	Forget-rate	0.1, **0.2**, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	87.30/0.64, **87.39/0.65**, 87.10/0.20, 87.02/0.74, 87.20/0.69, 86.92/0.21, 87.05/0.70, 86.95/0.53, 86.80/0.60
MentorMix	p-percentile	90%, **80%**, 70%	92.30/0.20, **92.34/0.10**, 92.12/0.13
Bootstrap	Weights of learned label	0.15, **0.25**, 0,35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95	92.88/0.17, **92.92/0.30**, 92.88/0.17, 92.91/0.17, 92.90/0.34, 92.90/0.39, 92.65/0.26, 67.82/8.38, 19.14/5.50
LRT	Epoch that starts to perform label correct	**10**, 15, 30, 35	**92.52/0.21**, 92.50/0.23, 92.30/0.16, 92.35/0.35
Noise-adaption	No extra hyper-parameters	None	None
LIO	Estimation methods of transition matrix	Gradient-based, **Dirichlet posterior update**	92.05/0.33, **92.30/0.35**
GCE	q	**0.1**, 0.2, 0,3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	**92.15/0.27**, 92.10/0.33, 92.08/0.22, 92.01/0.18, 92.03/0.36, 91.98/0.26, 91.93/0.24, 91.88/0.25, 91.82/0.18
ELR	λ	**1**, 3, 5, 7, 10	**91.84/0.18**, 91.70/0.30, 91.56/0.16, 91.40/0.34, 91.29/0.30

TABLE 14: MORSE vs DT on the Drebin dataset.

Methods	Average Accuracy (%)	Average Precision (%)	Average F_1 (%)
Vanilla DNN	84.41/1.02	84.65/1.03	84.39/1.04
DT	78.68/2.97 p = 0.993	77.29/3.52 p = 0.996	77.97/2.38 p = 0.994
Our method w/o re-weighting	86.40/0.18 p = 0.030	86.32/0.15 p = 0.032	86.35/0.17 p = 0.010
Our method	86.60/0.96 p = 0.008	86.45/0.95 p = 0.004	86.39/0.94 p = 0.008

is not that sensitive to these parameters. As such, we use the default choices: λ = 1, γ = 0.95, β = 0.999.

For the hyper-parameters introduced by our method (*i.e.,* labeled data's proportion: d, warm-up epoch, p_a in data augmentation), the sensitivity test in Section 6 shows the influence of labeled data's proportion d. For the other two hyperparameters, our experiment shows that subtly varying them imposes minor influence on the results and we use the values selected via a grid search (*i.e.,* 5 for the warm-up epoch, p_a=0.9/0.8 for weak/strong augmentation).

Hyper-parameters of existing noise learning methods. The network structure and training hyper-parameters are the same across all the methods. Besides the common hyper-parameters, some baselines have their unique hyper-parameters. As mentioned in Section 3, to ensure a fair comparison, we carefully tune these unique hyper-parameters and select the ones with the best performance. Table 13 shows the intermediate results of all the choices we have tried for the unique hyper-parameter of each method.

Paired t-test. We use a paired t-test to to measure the statistical significance of performance comparison between two methods. Specifically, given two sets of classification accuracy obtained by method1 and method2, we first compute their difference, *i.e.,* $diff = acc_{\text{method1}} - acc_{\text{method2}}$. With the $diff$, we set the null hypothesis as $H_0 : \mathbb{E}[diff] \leq 0$ and compute the p-value. If p-value is smaller than a threshold (*i.e.,* 0.05), we can reject H_0 and conclude that method1 is better than method2. Otherwise, we cannot reject H_0 and thus cannot draw the above conclusion. Throughout our evaluation, we treat the vanilla method as method2 and existing noise learning methods and our method as method1. That means if p-value is lower than 0.05, we can conclude the corresponding noise learning method outperforms the vanilla method with statistical significance.

TABLE 15: Traditional ML methods before and after augmenting with our designs on the PE dataset.

Methods	Average (%)			Class-11 (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	93.08/0.25	93.65/0.51	92.57/0.36	44.33/3.78	96.34/4.22	60.48/3.22
SVM	87.74/0.01 p = 1.000	91.92/0.01 p = 0.999	85.43/0.01 p = 1.000	4.00/0.01 p = 1.000	100.00/0.00 p = 0.055	7.69/0.01 p = 1.000
Random forest	92.38/0.17 p = 0.995	94.31/0.18 p = 0.016	91.36/0.32 p = 0.999	28.50/1.50 p = 0.999	100.00/0.00 p = 0.056	44.32/1.52 p = 0.998
Random forest w/ our designs	93.64/0.45 p = 0.021	94.55/0.31 p = 0.006	92.87/0.55 p = 0.073	53.50/5.15 p = 0.006	99.12/1.96 p = 0.107	69.45/4.05 p = 0.010
GBDT	91.83/0.23 p = 0.995	93.09/0.24 p = 0.970	91.25/0.12 p = 0.999	24.30/0.52 p = 0.999	88.89/1.20 p = 0.994	38.06/2.05 p = 1.000
GBDT w/ our designs	93.26/0.28 p = 0.222	94.13/0.41 p = 0.067	92.17/0.65 p = 0.667	40.33/1.97 p = 0.959	97.80/2.88 p = 0.225	54.39/5.39 p = 0.970
Logistic regression	91.58/0.01 p = 1.000	92.29/0.01 p = 0.999	90.58/0.01 p = 0.999	34.00/0.01 p = 0.999	94.44/0.01 p = 0.819	50.00/0.01 p = 0.999
Logistic regression w/ our designs	92.41/0.01 p = 0.999	93.25/0.01 p = 0.924	91.54/0.01 p = 0.997	40.50/0.50 p = 0.964	100.00/0.00 p = 0.055	57.60/0.48 p = 0.939
Our method	94.15/0.43 p = 0.003	94.14/0.57 p = 0.061	93.91/0.67 p = 0.000	65.33/4.20 p = 0.000	90.18/2.08 p = 0.984	76.74/1.22 p = 0.000

TABLE 16: Traditional ML methods before and after augmenting with our designs on the synthetic dataset with a noise rate of 0.6 and an imbalance ratio of 20x.

Methods	Overall cls (%)			Rare cls (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	68.04/1.77	70.38/6.31	63.68/4.12	48.93/4.03	73.16/9.31	53.39/7.05
SVM	58.80/0.01 p = 1.000	63.81/0.01 p = 0.924	54.55/0.01 p = 0.992	27.48/0.01 p = 1.000	60.00/0.01 p = 0.089	35.74/0.01 p = 0.979
Random forest	47.08/0.86 p = 0.995	58.86/1.12 p = 0.996	45.92/1.10 p = 1.000	28.07/1.11 p = 0.999	72.51/1.47 p = 0.562	38.88/1.47 p = 0.998
Random forest w/ our designs	54.00/0.52 p = 1.000	64.74/2.65 p = 0.891	53.64/1.58 p = 0.978	43.76/1.25 p = 1.000	77.24/3.98 p = 0.256	52.86/3.23 p = 0.618
GBDT	47.71/1.24 p = 1.000	51.95/0.28 p = 0.999	40.33/0.12 p = 1.000	14.32/1.41 p = 0.980	57.51/1.10 p = 0.994	20.57/1.05 p = 1.000
GBDT w/ our designs	59.40/0.23 p = 1.000	60.47/0.11 p = 0.991	54.19/0.21 p = 0.998	46.08/0.03 p = 0.980	56.33/1.10 p = 0.995	47.95/1.01 p = 0.968
Logistic regression	63.86/0.01 p = 1.000	65.06/0.01 p = 0.941	60.04/0.00 p = 0.947	53.44/0.01 p = 0.027	76.85/0.01 p = 0.208	58.55/0.01 p = 0.081
Logistic regression w/ our designs	65.20/0.01 p = 0.999	72.79/0.01 p = 0.216	62.56/0.01 p = 0.713	48.76/0.01 p = 0.150	80.60/0.01 p = 0.067	56.51/0.01 p = 0.183
Our method	76.20/0.59 p = 0.000	79.45/1.69 p = 0.016	72.90/1.17 p = 0.003	61.21/1.00 p = 0.000	80.45/0.99 p = 0.066	64.25/0.20 p = 0.009

10.3. MORSE vs. Traditional ML Algorithms

Besides the DL-based methods considered in our experiment, we also compare our method with four representative traditional ML methods – SVM, Random forest, GBDT, and Logistic regression. Specifically, we first run these methods and MORSE on the PE dataset and a synthetic dataset with a noise rate of 60% and an imbalance ratio of 20x. We run each method six times and report the mean accuracy and the p-value comparison with Vanilla DNN in Table 15 and 16. The results show that all four methods have worse performance than the Vanilla DNN on both datasets, indicating these methods also suffer from noisy labels and imbalance.

We further integrate our designs into those traditional ML models except SVM, which is not applicable. The re-

TABLE 17: MORSE vs. baselines augmented with re-weighting on the PE dataset.

Methods	Average (%)			Class-11 (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	93.08/0.25	93.65/0.51	92.57/0.36	44.33/3.78	96.34/4.22	60.48/3.22
Vanilla w/ re-weighting	93.82/0.29 p = 0.003	94.01/0.30 p = 0.102	93.50/0.34 p = 0.001	55.67/2.98 p = 0.001	89.71/2.30 p = 0.974	68.61/1.70 p = 0.001
Coteaching+ w/ re-weighting	93.79/0.36 p = 0.001	93.92/0.72 p = 0.286	93.24/0.79 p = 0.034	58.67/4.38 p = 0.001	93.52/2.63 p = 0.868	69.29/3.56 p = 0.000
MentorMix w/ re-weighting	92.00/0.37 p = 0.752	93.42/0.14 p = 0.520	92.70/0.23 p = 0.048	56.50/1.15 p = 0.001	90.24/3.35 p = 0.992	67.82/2.14 p = 0.001
Bootstrap w/ re-weighting	93.64/0.45 p = 0.021	94.03/0.67 p = 0.117	93.29/0.50 p =0.004	53.50/5.15 p = 0.006	92.39/3.53 p = 0.035	67.70/2.11 p = 0.001
LRT w/ re-weighting	93.25/0.31 p = 0.194	93.67/0.16 p = 0.475	92.99/0.13 p = 0.006	53.00/4.55 p = 0.031	91.07/1.96 p = 0.948	67.13/2.83 p = 0.010
Noise-adaption w/ re-weighting	93.82/0.28 p = 0.007	93.85/0.25 p = 0.160	93.44/0.30 p = 0.000	54.50/2.36 p = 0.000	87.93/1.12 p = 0.996	67.26/1.70 p = 0.002
LIO w/ re-weighting	93.02/0.35 p = 0.320	93.31/0.25 p = 0.621	92.91/0.15 p = 0.012	52.20/3.10 p = 0.020	89.12/4.23 p = 0.991	65.93/2.45 p = 0.001
GCE w/ re-weighting	93.89/0.19 p = 0.000	93.92/0.12 p = 0.129	93.39/0.17 p = 0.000	58.33/2.05 p = 0.000	89.92/4.80 p = 0.985	68.66/0.92 p = 0.001
ELR w/ re-weighting	93.30/0.15 p = 0.124	93.81/0.20 p = 0.274	93.17/0.25 p = 0.002	51.00/3.27 p = 0.031	90.89/4.56 p = 0.927	65.98/3.15 p = 0.001
Our method	94.15/0.43 p = 0.003	94.14/0.57 p = 0.061	93.91/0.67 p = 0.000	65.33/4.20 p = 0.000	90.18/2.08 p = 0.984	76.74/1.22 p = 0.000

TABLE 18: MORSE vs. LS on the poisoned PE malware dataset.

Methods	Average (%)			Class-11 (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	89.14/0.76	90.41/1.23	88.80/0.81	45.83/18.20	73.15/9.00	53.15/12.51
LS	84.00/0.33 p = 1.000	82.87/3.23 p = 0.997	81.98/0.54 p = 1.000	1.33/2.98 p = 0.998	16.67/37.27 p = 0.990	2.47/5.52 p = 0.997
Our method	91.46/1.16 p = 0.003	92.05/0.79 p = 0.020	91.41/1.10 p = 0.001	57.33/2.36 p = 0.111	77.12/3.47 p = 0.234	65.71/1.99 p = 0.032

sults in Table 15 and 16 show that our designs could improve those methods under noisy labels and class imbalance. We also observe that even with our design, these methods still cannot outperform our method with DNN, which verifies the necessity of using DNN-based models.

10.4. Baselines with Sample Re-weighting

To verify (1) the effectiveness of our sample re-weighting method; (2) the necessity of our other designs, we also augment our sample re-weighting method to our comparison baselines on the PE dataset. Similar to the previous experiments, all the methods are run six times, and the results are shown in Table 17. Note that we tune the re-weighting parameter β in each method and report the optimal result. First, we can observe that compared to the results in Table 4, the augmented baselines show better performance than its original version, verifying the efficacy of our sample re-weighting. Table 17 also shows that MORSE still outperforms the augmented baselines, verifying the necessity of our other designs. The augmented noisy learning methods still perform worse than MORSE because: (1) These methods are fully supervised and they cannot extract useful information from unlabeled samples. (2) The design flaws of these methods still exist even with re-weighting.

10.5. MORSE vs. DT

Following [22], we use a binary classification dataset Drebin [65]. To construct a noisy training set, we first randomly select 4,000 benign ware and 4,000 malware and then flipped the labels of 45% randomly selected training samples. For a binary classification dataset, 45% is a very high rate, which is almost near a randomly labeled dataset (with a noise rate of 50%). We then randomly select 1,000 benign ware and 1,000 malware from the rest samples as our testing set. We evaluate our method and DT on the above dataset and show the results in Table 14. The table demonstrates that our method significantly outperforms DT, validating the superiority of our method in dealing with a dataset with a high noise rate. Note that we use the same network structure for both methods and used the default choices for the hyperparameters in [22].

TABLE 19: MORSE with different data augmentation methods.

Methods	Average (%)			Class-11 (%)		
	Accuracy	Precision	F_1	Accuracy	Precision	F_1
Vanilla DNN	93.08/0.25	93.65/0.51	92.57/0.36	44.33/3.78	96.34/4.22	60.48/3.22
Our method w class-level augmentation	94.04/0.28 p = 0.001	94.02/0.34 p = 0.109	93.87/0.32 p = 0.000	66.00/2.83 p = 0.000	86.47/1.09 p = 0.999	74.83/1.93 p = 0.000
Our method	94.15/0.43 p = 0.003	94.14/0.57 p = 0.061	93.91/0.67 p = 0.000	65.33/4.20 p = 0.000	90.18/2.08 p = 0.984	76.74/1.22 p = 0.000

10.6. MORSE against Label Poisoning Attacks

We evaluate the robustness of MORSE against a state-of-the-art label poisoning attack [79] and compare it with a widely used defense Label Sanitization (LS) [94]. Using the PE malware dataset, we construct a poisoned training set with a poison rate as 20% using the attack in [79]. Then, we run vanilla method, our method, and LS on this poisoned dataset. The results are shown in Table 18. As we can observe from the table, MORSE performs the best among all three methods, verifying our argument in Section 8.

10.7. MORSE with class-level Data Augmentation

We replace our augmentation method with another method that better preserves the labels of the augmented samples, *i.e.,* selecting samples from the same class as $\bar{\mathbf{x}}$ and using Equation (3) for augmentation. We run MORSE with this augmentation method on the PE dataset and show the result in Table 19. As we can observe from the table, using this new augmentation method does not improve the model performance. As discussed in Section 5.3, the reason is that rather than using their given labels, most augmented samples are treated as unlabeled data and are relabeled with pseudo labels in the learning process.

10.8. MORSE against Concept Drifts

We run a temporal evaluation for MORSE by following the setup in [39]. Specifically, we split the BODMAS dataset into a training and two testing sets based on time. The training set contains samples in 10 families collected from 08/29/2019 to 10/01/2019. The first testing set is collected from 10/01/2019 to 11/01/2019. and the second is collected from 09/01/2020 to 10/01/2020. The testing set could have unknown malware families. We train six models using MORSE with different initial parameters and test their performance on two testing sets. The average test precision, recall, and F_1 score (%) on the first and second testing set are 51.44/1.14, 84.30/0.96, 62.61/0.97 and 31.99/2.91, 70.86/0.61, 39.76/2.35, respectively. The results show that drifting samples indeed have a negative impact on MORSE.