MalCertain: Enhancing Deep Neural Network Based Android Malware Detection by Tackling Prediction Uncertainty

Haodong Li
Beijing University of Posts and Telecommunications
China
lihd@bupt.edu.cn

Guosheng Xu*
Beijing University of Posts and Telecommunications
China
guoshengxu@bupt.edu.cn

Liu Wang
Beijing University of Posts and Telecommunications
China
w_liu@bupt.edu.cn

Xusheng Xiao
Arizona State University
USA
xusheng.xiao@asu.edu

Xiapu Luo
The Hong Kong Polytechnic University
China
csxluo@comp.polyu.edu.hk

Guoai Xu
Harbin Institute of Technology, Shenzhen
China
xga@hit.edu.cn

Haoyu Wang*
Huazhong University of Science and Technology
China
haoyuwang@hust.edu.cn

ABSTRACT

The long-lasting Android malware threat has attracted significant research efforts in malware detection. In particular, by modeling malware detection as a classification problem, machine learning based approaches, especially deep neural network (DNN) based approaches, are increasingly being used for Android malware detection and have achieved significant improvements over other detection approaches such as signature-based approaches. However, as Android malware evolve rapidly and the presence of adversarial samples, DNN models trained on early constructed samples often yield poor decisions when used to detect newly emerging samples. Fundamentally, this phenomenon can be summarized as the uncertainly in the data (noise or randomness) and the weakness in the training process (insufficient training data). Overlooking these uncertainties poses risks in the model predictions. In this paper, we take the first step to estimate the prediction uncertainty of DNN models in malware detection and leverage these estimates to enhance Android malware detection techniques. Specifically, besides training a DNN model to predict malware, we employ several uncertainty estimation methods to train a Correction Model that determines whether a sample is correctly or incorrectly predicted by the DNN model. We then leverage the estimated uncertainty output by the Correction Model to correct the prediction results, improving the accuracy of the DNN model. Experimental results show that our proposed MalCertain effectively improves the accuracy of the underlying DNN models for Android malware detection by around 21% and significantly improves the detection effectiveness of adversarial Android malware samples by up to 94.38%. Our research sheds light on the promising direction that leverages prediction uncertainty to improve prediction-based software engineering tasks.

CCS CONCEPTS

• **Security and privacy** → **Software security engineering**; • **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Android Malware Detection, Uncertainty, DNN

ACM Reference Format:
Haodong Li, Guosheng Xu, Liu Wang, Xusheng Xiao, Xiapu Luo, Guoai Xu, and Haoyu Wang. 2024. MalCertain: Enhancing Deep Neural Network Based Android Malware Detection by Tackling Prediction Uncertainty. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639122

1 INTRODUCTION

Malware is a long-lasting threat in the Android app ecosystem. Millions of emerging Android malicious apps were identified from time to time, even in the official Google Play market [2]. The increasing Android malware threats have attracted significant research efforts in our community. According to the statistics [9], a large number of papers (10,000+) were focused on Android malware detection, and a plethora of approaches have been proposed, including signature-based approaches [11, 12, 53], behavior-based approaches [10, 26, 46], and machine learning based approaches [59,

*Corresponding authors: Haoyu Wang (haoyuwang@hust.edu.cn) and Guosheng Xu (guoshengxu@bupt.edu.cn).

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3639122

66, 76]. Among these Android malware detection approaches, Deep Learning (DL) based approaches such as Deep Neural Networks (DNNs) [20, 34, 55, 61] have been adopted by many researchers in recent years [3, 18, 32, 41, 44], as they have shown promising achievements over other machine learning based approaches. For example, a number of different malware detection models including Multilayer perceptron (MLP) [18], CNNs [44] and RNNs [41] are created based on different static features, and all of them report promising results over other existing malware detection approaches.

However, most existing DNN models for Android malware detection are trained under the assumption of a closed-world scenario [3, 38, 39, 62, 69], which assumes that the training and testing datasets are drawn from in-distribution (ID). In reality, due to the rapid evolution of Android malware, DNN models trained on early constructed samples often yield poor decisions when used to detect newly emerging samples [70], which is known as concept-drift [6, 28]. Furthermore, DNN models are highly sensitive to out-of-distribution (OOD) samples and adversarial samples, which makes them less robust [8, 17, 24, 35, 45, 69], especially when dealing with samples from different malware families and domain shifts [28, 71]. To address these issues, most models adopt the approach of periodic retraining [27, 30, 51]. Unfortunately, this approach has significant drawbacks. First, periodic retraining requires a large amount of annotated data, which is costly to obtain. Second, it is challenging to conduct targeted training and anticipate the weaknesses of the model in advance, particularly in light of the wide variety of Android malware. Third, it is difficult to estimate the timing for retraining due to many unpredictable zero-day malware [16, 52, 73], and by the time the model's performance deteriorates, it may have already been subject to numerous attacks [28].

As suggested by recent research [14, 31], the limitations of DNN-based Android malware detection are mainly caused by the uncertainty inherent in the data (*aleatoric uncertainty* [31]) or the weakness of the neural network in the training process (*epistemic uncertainty* [31]). Specifically, aleatoric uncertainty is usually caused by the inherent noise or randomness in the dataset, which cannot be overcome by expanding the training set. Epistemic uncertainty is usually caused by a lack of knowledge and insufficient cognition of the model in the training process, which refers to the confidence level of the model in its prediction. This form of uncertainty occurs when there are insufficient training data and/or the model's parameters are improperly adjusted. When the model is required to predict the samples generated by the shifted version of the training data or the samples outside the data distribution region, there may be a high degree of epistemic uncertainty. Overlooking these uncertainties poses great risks on the model prediction accuracy, and also makes models more vulnerable to adversarial malware samples. To overcome the aforementioned limitations, new techniques of uncertainty estimation are in dire need to identify uncertain predictions, so that these predictions can be passed on to human experts for verification [13], or even fixed automatically.

Recognizing the importance of uncertainty estimates, recent efforts [4, 48] have been put forth to leverage uncertainty in improving the model's prediction accuracy. For example, in image recognition [75] area, many researchers propose approaches to measure the differences in uncertainty metrics between different types of samples, and define some uncertainty metrics to distinguish common benign samples from malicious samples generated by attacks. Therefore, a simple way to enhance DNN-based Android malware detection is to draw on the idea that prediction uncertainty can distinguish different samples (correctly classified samples and incorrectly classified samples). However, these existing works are mostly limited to a single or few uncertainty assessment metrics. Thus, it is not clear if the uncertainty estimations are potentially subject to the bias of their training data, and what are the optimal way to integrate these estimations to improve the model accuracy.

In this paper, we take the first step to explore how we can leverage the prediction uncertainty to improve DNN-based Android malware detection models. Our key insight is *if we can identify uncertainty metrics that differ greatly between correct and incorrect predictions, we can use these metrics to pinpoint the potentially incorrectly-classified samples and correct their classification results accordingly.*

To this end, we first conduct a characteristic study that aims to measure which uncertainty estimation method and metrics can distinguish correct and incorrect predictions of DNN-based Android malware detection models (**see § 3**). Specifically, we train a DNN-based Android malware detection model and apply the model on a test dataset that contains OOD app samples (i.e., data having uncertainty) to obtain the correct and incorrect predictions. We then employ existing uncertainty estimation methods (e.g., Variational Bayesian Inference [21, 65]) to train a set of model ensembles[1], apply these model ensembles on the test dataset to obtain their predictions, and compute uncertainty estimation metrics (e.g., *Entropy* [47, 48] and *Kullback-Leibler (KL) divergence* [37]) based on these prediction results. The results show that all these metrics can effectively differentiate correct and incorrect predictions. In particular, the *Kullback-Leibler (KL) divergence* metric computed using the Variational Bayesian Inference method is among the best to identify incorrect predictions.

Motivated by this study, we propose a general framework, MalCertain, which finds an optimal way of using prediction uncertainty to improve the performance of DNN-based Android malware detection models (**see § 4**).

Specifically, given a DNN-based Android malware detection model (called the *Base Model*), MalCertain first trains a number of model ensembles atop the Base Model, and applies the Base Model and these model ensembles on a *correction training dataset* to obtain their prediction results. MalCertain then computes uncertainty estimation metrics based on the prediction results of the model ensembles, groups the prediction results of the Base Model into two classes, i.e., correct predictions and incorrect predictions, and trains a machine learning model (called the *Correction Model*) that can distinguish whether a sample is correctly or incorrectly predicted by the underlying DNN model. With the Correction Model, when the Base Model is applied to obtain new prediction results, the Correction Model can determine whether the prediction results are reliable or not and correct the unreliable prediction results to improve the accuracy of the Base Model.

In summary, this paper makes the following contributions:

[1] A model ensemble is a group of slightly different models.

- We make the first attempt to leverage the characteristics of prediction uncertainty in DNN-based Android malware detection models to improve their model accuracies. We show the discrepancy in prediction uncertainty between correct and incorrect predictions of these models can be detected using a number of metrics (e.g., Entropy, KL divergence).
- We propose a novel framework, MalCertain, to pinpoint the samples that are likely to be incorrectly classified by a DNN-based Android malware detection model and correct the incorrectly classified predictions to improve the model performance.
- We conduct extensive experiments on 26,748 apps to evaluate the effectiveness of MalCertain. By applying MalCertain to two state-of-the-art DNN-based Android malware detection models, we show that when these models are used to detect OOD samples, MalCertain is capable of improving the model performance (around 21% improvements in accuracy and 49% improvements in the F1 score). Meanwhile, MalCertain can also improve these models' capabilities in identifying the adversarial samples.

To facilitate future research, we have made our tool open-source as a project [42].

2 BACKGROUND AND RELATED WORK

2.1 Uncertainty in Android Malware Detection

DNN-based Android malware detection inevitably suffers from the two types of uncertainties. First, due to the explosive growth and the increasing complexity of Android apps [23], it is almost infeasible to manually label new apps for training new models, and thus the most commonly used labeling method is based on antivirus services. However, such a mechanism often leads to a certain degree of mislabeling, causing noisy labels in the training dataset (referred to as *Aleatoric Uncertainty* [14]). Second, to evade detection or apply new malicious tactics, Android malware developers' ever-evolving techniques have resulted in a constantly shifting data distribution within the realm of Android malware samples, causing detection models to become less effective in identifying malware over time. Particularly, samples that exhibit distributional differences from detection models' training datasets are often referred to as OOD samples, and these OOD samples introduce another type of uncertainty to the trained models (referred to as *Epistemic Uncertainty*).

Estimating the Uncertainty. In general, the methods for estimating the prediction uncertainty can be categorized in four types [14]:

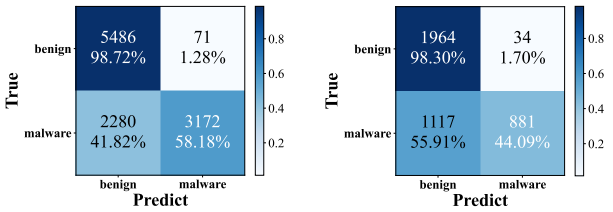
- **Single deterministic methods.** For a single deterministic neural network, the parameters are deterministic. It gets the same predicted value for the same sample. For multi-classification tasks, we can use the output of the softmax layer to represent the probability of each class, and then quantify the uncertainty according to this set of probability values. For regression tasks, Oala et al. [49] introduced an uncertainty score based on the lower and upper bound output of an interval neural network.
- **Variational Bayesian Inference (VBI)** approximates the (in general intractable) posterior distribution by optimizing over a family of tractable distributions [5, 21].
- **Ensemble methods** combines the predictions of several different deterministic networks at inference. There are subtle differences between these determination networks, such as different training epochs, different initialization parameters, different loss functions and optimizers, etc [54, 56].
- **Test-time augmentation methods.** The prediction is given based on the deterministic network, but multiple predictions will be obtained by increasing the number of sample inputs [37].

Metrics. To measure the degree of uncertainty, a number of estimation metrics have been proposed and widely used in our community. For example, *Prediction Confidence Score (PCS)* is defined as the difference in probability between the two classifications with the highest probability [75]. It is often used to estimate the uncertainty of a softmax output (a vector representing the probability distribution for each category) in a multi-class classification task. *Softmax Entropy* is defined as the entropy of the multi-categorization outputs [43] to quantify the level of confusion in the softmax outputs. Besides, given that the models often make overconfident predictions and become unreliable, researchers seek to generate multiple softmax outputs for a sample, and adopt metrics such as *Mutual Information* [33] and *Variance* [77], which quantify the difference between a single softmax output and the mean of all softmax outputs to achieve more precise uncertainty estimation. For binary-classification tasks (such as Android malware detection) that output confidence scores, researchers construct model ensembles based on different uncertainty estimation methods to obtain a set of probability values and quantify the differences among this set of output values through some metrics. The common metrics include *Entropy*, *Kullback-Leibler (KL) divergence*, *Standard deviation*, etc., which are detailed in § 4.2.1.

Related Work on Applying the Uncertainty. Recently, some researchers have made efforts in measuring and understanding uncertainty. For example, Zhang et.al [74] introduced a new definition of label uncertainty, and measured the robustness of the image classification model from the perspective of label uncertainty. Specifically, in the software engineering community, there have been a few research attempts. For example, one of the earliest studies [4] made an initial attempt to enhance software analysis performance by leveraging uncertainty. Nguyen et.al [47] explored the utility of the information contained within the Bayesian neural network in detecting OOD data in PE malware detection. Li et.al [37] conducted an empirical study to evaluate the quality of predictive uncertainties of malware detectors and compared different uncertainty estimation methods and metrics as well as differences before and after calibration. Inspired by previous research, this paper aims to enhance the capabilities of DNN-based Android malware detectors by tackling the prediction uncertainties of the samples.

2.2 DNN Based Android Malware Detection

Deep neural networks have made remarkable achievements in image recognition, Natural Language Processing (NLP) and other fields. In recent years, researchers have applied DNNs to Android malware detection and achieved promising results [18, 32, 41, 44]. For example, Grosse et.al. [18] implemented an Android malware detector based on Multilayer perceptron (MLP), taking advantage of features extracted from Drebin [3] (including used permission and



(a) Results on Drebin dataset (b) Results on AMD dataset

Figure 1: The confusion matrix of the Base Model's prediction results on two datasets.

sensitive APIs, etc.). Kim et.al. [32] implemented MultimodelDNN, a DNN-based malware detector that extracts 5 kinds of features including permissions, sensitive strings, system APIs, Dalvik opcode and the ARM opcodes from native binaries. Besides, there are also studies that convert apps into images/bitmaps and then process them through DNN algorithms using image-based features [22, 60, 72]. For example, Unver et.al. [60] converted some files in the Android app source to gray-scale images, and then some image-based local and global features were extracted from the constructed gray image dataset, and used to train a CNN model for Android malware detection. Huang et al. [22] converted the bytecode of classes.dex into RGB color code, and converted it into a fixed-size color image. They input color images into a convolution neural network for automatic feature extraction and training.

2.3 The Adversarial Examples in DNN

An adversarial example is a sample of input data that has been slightly modified with the aim of making the DNN model misclassify it. It is known that the existence of adversarial examples poses challenges to DNNs' generalization ability. Szegedy et al. [58] found that neural networks are relatively fragile and vulnerable to attacks by adversarial examples. Only a few changes to the samples can make mistakes in the neural network classifier. At present, academia has not yet reached a final conclusion on the cause of adversarial examples. Szegedy et al. [58] believe that the highly nonlinear nature of neural networks leads to the existence of adversarial examples. Goodfellow et al. [15] believe that the linear behavior of neural networks in high-dimensional space is the real reason for the existence of adversarial examples. Based on this point of view, the authors designed a simple and effective method to quickly generate adversarial examples, namely the fast gradient symbol method (FGSM). Zhang et.al. [75] made an empirical study and found the characteristics of the uncertainty metrics of malicious samples and normal samples, and then constructed adversarial examples that did not meet these metrics.

3 CHARACTERISTIC STUDY

To address the problem of OOD samples for DNN-based Android malware detection models, we conduct a characteristic study that aims to understand whether existing uncertainty estimation metrics can be used as a reliable indicator to distinguish correct and incorrect predictions. Specifically, we first train a Base Model based on an open-source DNN-based Android malware detection model, DeepDrebin [18], using a training dataset that contains malware from MalRadar and MalGenome and benign apps from Androzoo. We then test the Base Model using a test dataset that contains malware from two other datasets (Drebin dataset and AMD dataset) and different benign apps from Androzoo.

Prediction on OOD Samples. The detection results are shown in Figure 1. We can see that the vast majority of the false predictions belong to the cases where malicious samples are misclassified as benign samples (41.82% of the malware in the Drebin dataset and 55.91% of the malware in the AMD dataset). This indicates that the model's false negative rate (FNR) is far greater than the false positive rate (FPR). The poor model performance on OOD samples also shows that DNN-based Android malware detection models suffer from concept-drift.

Uncertainty Estimation Method and Metrics. In order to estimate the uncertainty of these predictions, we choose five commonly used methods to construct model ensembles: (i) Epoch Ensemble, (ii) Variational Bayesian Inference, (iii) Monte Carlo Dropout, (iv) Deep Ensemble, and (v) Weighted Deep Ensemble. See § 4.1.2 for details of these methods. We train five uncertainty estimation model ensembles using the same training set. Further, to quantitatively assess the prediction uncertainty, we select three commonly used metrics: entropy, Kullback-Leibler (KL) divergence, and standard deviation, all of which can be used to quantify the degree of dispersion in a set of data. Fundamentally, these metrics are used to describe the extent to which a sample varies in its predictions (i.e., the predicted probabilities belonging to each class) from multiple similar models in each model ensemble. Typically, a larger metric value indicates a greater disagreement between the models in the ensemble, and thus the uncertainty is higher.

Result Analysis. The samples are divided into two categories based on whether they are misclassified by the Base Model or not. After that, we computed uncertainty metrics for each sample in each category. Figure 2 shows the CDF distribution of three uncertainty metrics for the correctly classified samples (solid line) and the incorrectly classified samples (dotted line), with each color marking a type of model ensemble. We can see that all the uncertainty metrics (entropy, KL divergence, and standard deviation) display significantly different distributions in the two types of samples. The uncertainty values of the incorrectly classified samples are generally larger than those of the correctly classified samples. The difference is especially notable for the VBI method, with roughly 75% of correct predictions having uncertainty metrics of 0, compared to roughly 10% of incorrect predictions. Thus, these uncertainty metrics computed based on different uncertainty estimation methods can be used as indicators to differentiate reliable and unreliable predictions. This motivates us to design an automated approach to flag unreliable predictions and correct these prediction results.

4 DESIGN OF MALCERTAIN

Enlightened by the findings of the preliminary study, we propose MalCertain, a general framework to improve the performance of DNN-based Android malware detection models based on prediction uncertainty. Figure 3 illustrates the overall architecture of the framework. Its operation goes through three main phases: (i) DNN

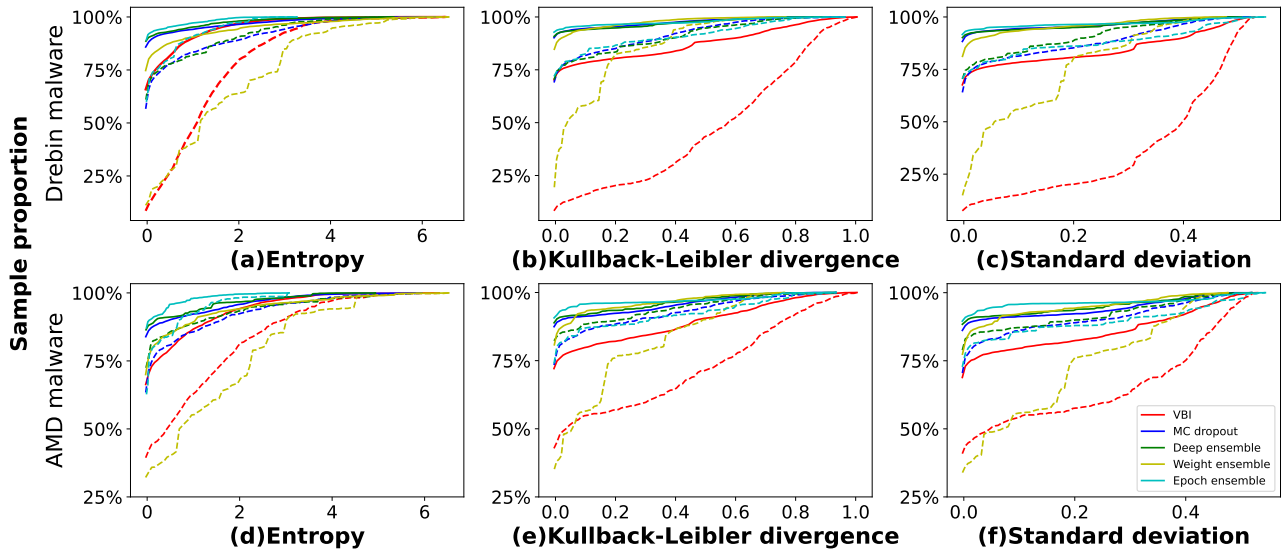


Figure 2: A comparison of Entropy, Kullback-Leibler divergence and Standard deviation of prediction results for correctly classified and incorrectly classified samples. The three subgraphs (a), (b), and (c) in the upper half of the graph are used to compare the uncertainty differences of samples in the Drebin dataset. The three subgraphs (d), (e) and (f) in the lower part of the figure are used to compare the uncertainty differences of the samples in the AMD dataset. The solid line represents the correctly classified sample, and the dotted line represents the incorrectly classified sample.

model training: we select a DNN-based Android Malware Detection Model and generate feature vectors to train a *"Base Model"*. In addition, we identify some proper uncertain estimation methods to train a number of model ensembles by strategically mutating the Base Model. (ii) Uncertainty-based Correction Model training: we design some uncertainty metrics to train a *"Correction Model"* by grouping the samples into two sets - correctly and incorrectly classified; (iii) Result correction: we revise the original predicted results of the Base Model according to the Correction Model's detection.

4.1 DNN Model Training

4.1.1 Base Model. In the first phase, we choose a DNN-based Android malware detection model, such as DeepDrebin. Then the apps from the training set are transformed into feature vectors through a "Feature Extraction" process, and these feature vectors are used to train the chosen malware detection model, i.e., the Base Model. We strictly follow the corresponding malware detection approaches to extract features and train the models.

4.1.2 Uncertainty Estimation Methods. The intuition behind estimating uncertainty is to see if there is a significant variation in the predictions made by the model after making a slight modification to the model. Given a classification model M and a test sample S, imagine that after we modify one parameter of M and get a new model M', the prediction made by M' for S varies a lot compared to M, then the uncertainty of the prediction made by M for S is considered high (i.e., the prediction is likely to be unreliable). Following prior studies [5, 14, 21, 31, 37, 49], we applied 5 types of uncertainty estimation methods in our framework. Based on these methods, we will extract uncertainty metrics that can be used to pinpoint unreliable predictions (see § 4.2).

(i) Epoch Ensemble: It refers to training a set of models, where the models differ only in the training epochs from the Base Model. Different epoch values are employed in the training process for the underlying model, leading to some models might be under-fitting and some might be over-fitting. In our setting, we integrated 5 members with different training epochs.

(ii) Variational Bayesian Inference (VBI): We develop a Bayesian Neural Network (BNN) mirroring the architecture of the Base Model (DNN). Based on the principle of Bayesian inference, the parameters of the DNN are treated as random variables, conforming to the prior distribution $p(w)$. Thus, for each prediction, the parameters are randomly selected from the distribution for each inference to infer the results, and we conduct multiple predictions on the same sample to quantify the uncertainty of that sample. In our experiments, each sample is predicted 10 times (following the prior work [37, 47, 57]).

(iii) Monte Carlo (MC) Dropout: Adding a dropout layer before the input of every layer is a key modification. This dropout operation is performed in both the training and inference phases. Because some neurons are randomly discarded at the dropout layer, there will be slight differences between model parameters in each inference. These differences will eventually lead to different predictions for a certain sample. In our experiments, each sample is predicted 10 times (following the prior work [37, 43]).

(iv) Deep Ensemble: This constitutes an ensemble model that aggregates n members. The difference between these members stems from the different initialization parameters used for training. Following the prior work [37, 50], we integrated 10 members.

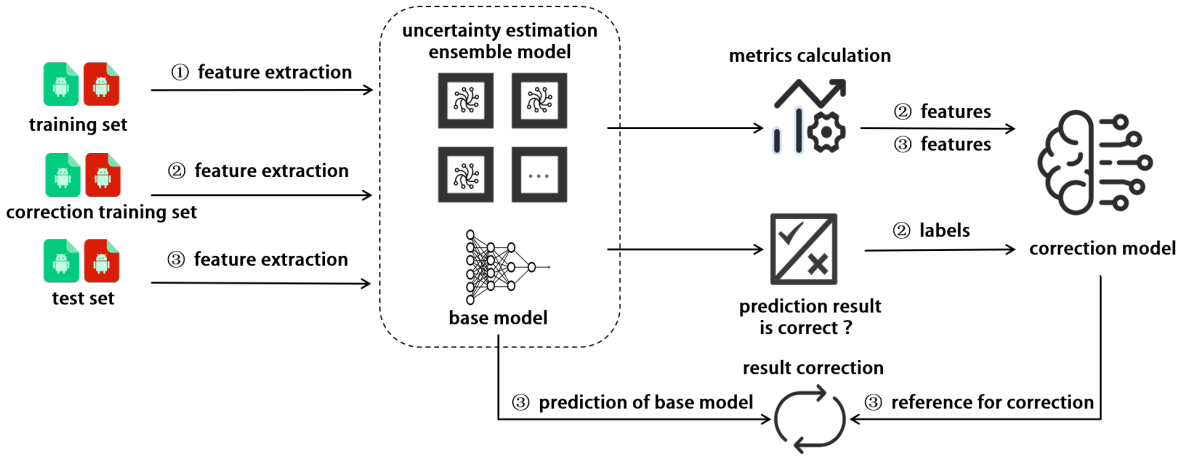


Figure 3: Overview of MalCertain. Our framework consists of 3 phases, namely, 1) DNN Model Training, 2) Uncertainly-based Correction Model and 3) Result Correction. Each step is marked with a number (see ①, ②, ③) indicating its corresponding phase.

(v) Weighted Deep Ensemble (wEnsemble): It integrates n models and employs distinct initialization parameters, similar to Deep Ensemble. Additionally, it dynamically adjusts the weights assigned to each individual model during the training process. Except for weighting, other settings are the same as Deep Ensemble.

4.2 Uncertainty-based Correction Model

In the second phase, we aim to automatically distinguish the reliable and unreliable predictions based on uncertainty metrics. Note that, although we show that there is an obvious contrast between correctly classified and incorrectly classified samples for some metrics (see § 3), it is hard or inaccurate to straightforwardly define a threshold for correcting the prediction results. Thus, we formalize it as a classification task, that we use a set of features (i.e., uncertainty metrics) to train a model to classify whether a prediction result (of the Base Model) is correct.

4.2.1 Feature (Uncertainty Metrics) Extraction. The outputs of the uncertainty estimation methods in § 4.1.2 are a set of predicted probabilities for each sample. The differences in these predicted probabilities can be considered as a manifestation of the underlying model's prediction uncertainty. If the predicted probabilities for a sample vary greatly, the underlying model has a high uncertainty in the prediction of this sample, which means that this prediction may be unreliable. We craft a number of metrics to measure the uncertainty of the underlying model for those samples, which will serve as features for training the Correction Model. We first resort to metrics that are commonly used in existing work. However, since the Android malware detection task is a binary-class classification task, there are many metrics that we cannot directly use (they are mostly used for multi-class classification tasks). As a result, we chose three metrics that we could apply to our task, as detailed below.

- **Entropy:** It is defined as a measure of randomness or disorder of a system. Given a set of predicted probabilities from an uncertainty estimation method, the larger the entropy value, the more chaotic the predicted probabilities, and the greater the uncertainty of the underlying model's prediction for the sample.
- **Kullback-Leibler (KL) divergence:** It is a measure of the dissimilarity between two probability distributions. When applied to a set of predicted probabilities, it is computed as the differences between the distribution of predicted probabilities and the uniform distribution of their means. The greater the KL divergence, the greater the uncertainty of the underlying model's prediction.
- **Standard deviation:** It measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance. We calculate the standard deviation of each set of predicted probabilities. The larger the standard deviation, the greater the uncertainty of the underlying model's prediction.

In addition to the three widely used metrics mentioned above, we also empirically design some statistical metrics that are derived from the statistical properties as follows.

- **sub(maximum,sec_maximum)**[2] It is calculated by subtracting the second-largest value from the largest value in a set of predicted probabilities. A high *sub(maximum,sec_maximum)* indicates a low uncertainty in the prediction of the underlying model for the given sample.
- **sub(maximum,median):** It is not enough to subtract the second-largest value from the maximum value. For example, if two models make incorrect predictions, the indicator of subtracting the second largest value from the maximum value will become invalid and the information will be lost. This indicator describes the difference between the maximum value and the median value. If the value of this indicator is relatively small, it indicates that at least half of the models have made the same prediction; If the value of this indicator is relatively large, it can make up for the defect of *sub(maximum,sec_maximum)*, covering more than two models making incorrect judgments
- **sub(maximum,mean):** This indicates the relationship between the maximum value and the mean value. A smaller value of this indicator indicates smaller uncertainty (all members of the model

[2] sub(a , b) implies a minus b.

group give approximate prediction results); On the contrary, a larger one indicates larger uncertainty (at least one model makes a quite different judgment).

- **sub(median,mean):** We use the threshold to determine the final result of the sample, as long as the predictive value is greater than 0.5, the sample will be classified into the same class, so just taking the mean of the prediction results will lose a lot of information. We can solve this problem by subtracting the mean from the median. If an ensemble model with 5 members has a prediction result of [1.00, 0.41, 0.42, 0.43, 0.44] for a sample, the Base Model is very likely to classify it as benign software because the prediction value is less than 0.5. However, if the average value is greater than 0.5, it should be classified as malicious software. Therefore, the larger this metric, the higher the uncertainty.
- **sub(mean,minimum):** This metrics is symmetrical with metrics *sub(maximum,mean)*.
- **sub(median,minimum):** This metrics is symmetrical with metrics *sub(maximum,median)*.

As described in § 3, we have verified that the first three metrics (Entropy, KL divergence, Standard deviation) can be used to distinguish incorrectly and correctly classified samples. Similarly, for the other six uncertainty metrics adopted in our design, the differences in their values for the correctly and incorrectly classified samples with metrics of 0 are also more than 30%. Thus, we consider that these uncertainty metrics can be used as indicators to differentiate reliable and unreliable predictions.

4.2.2 Correction Model. We calculate the aforementioned uncertainty metrics for each predicted sample. The samples are then grouped into two sets, i.e., reliable or unreliable predictions. Subsequently, the metrics are taken as features to train a machine learning model which is designed to classify reliable and unreliable predictions. This model is considered the "Correction Model", as we can modify the initial prediction results for those samples that are classified as unreliable predictions by the model. Specifically, we select four representative ML algorithms for the Correction Model, which are Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Decision Tree (DT), and Random Forest (RF).

4.3 Result Correction

Given a set of test samples, we first get their prediction labels by the Base Model. In parallel, we can determine whether the prediction of the underlying model (i.e., Base Model) is reliable through the Correction Model. Further, we seek to correct the predicted labels of the Base Model based on the classification of the Correction Model. Specifically, we have two strategies for the results.

- *Correcting Unreliable Results*: MalCertain will correct all unreliable prediction results (i.e., the Correction Model classifies the prediction results as "unreliable").
- *Correcting Unreliable FNs*: In Section 3, we show that most of the incorrect classifications are cases where malicious apps are wrongly classified as benign apps, which indicates that the FNR of the Base Model will be larger than the FPR. Thus, this strategy corrects only potential FNs. Specifically, MalCertain will correct a prediction result only if the Correction Model classifies the result as "unreliable" and the result is benign.

5 EVALUATION

Our evaluations aim to assess the effectiveness of MalCertain in improving the performance of the existing DNN-based Android Malware Detection Models. In particular, we seek to answer the following research questions.

- **RQ1:** How effective is MalCertain in improving the accuracy of the Base Model on out-of-distribution (OOD) data?
- **RQ2:** How effective is MalCertain in improving the accuracy of the Base Model on adversarial examples?
- **RQ3:** How does the size and balance of the data used to train the Correction Model affect the effectiveness of MalCertain?

5.1 Evaluation Setup

We implement MalCertain using the Tensorflow framework and Tensorflow probability libraries, and train the models using GeForce GTX TITAN X driven by CUDA.

5.1.1 Evaluation Methodology. To measure the effectiveness of MalCertain in improving DNN-based Android Malware Detection Models, given a DNN model, we construct 3 models (a *Base Model*, a *Correction Model*, and a *Large Base Model*) as follows:

- **Base Model**: we use a dataset (*training set*) to train a Base Model and obtain some uncertainty estimation model ensembles.
- **Correction Model**: we use another dataset (*correction training set*) and the predictions of the Base Models to train Correction Models based on 4 ML algorithms (SVM, KNN, DT, RF).
- **Large Base Model**: we also merge the training set and the correction training set to train a Large Base Model.

We construct these 3 models to obtain an objective measurement of MalCertain's effectiveness. While we can measure the improvements by comparing the results of the Correction Model with the results of the Base Model, this comparison is not entirely fair as the Correction Model is trained using more data. Thus, based on the recent approaches [27, 30, 51] that aim to address the conceptual drift problem of DNN models using incremental training and retraining, we further train the Large Base Model using the merged dataset. We use the Base model and the Large Base Model as the baseline models and compare their performances. Simply put, if the detection results processed by the Correction Model outperform the two baseline models, it is fair to say that our framework MalCertain effectively improves the performance of DNN models for Android malware detection.

5.1.2 Evaluation Subject. We collect both malicious apps and real-world benign apps to form our evaluation datasets. The malicious apps are from the widely used malware datasets in our research community, and they were collected independently with temporal and distributional differences. The benign apps are collected from Androzoo [1]. To further sanitize the benign datasets, we send these apps to VirusTotal service for inspection and make sure that they are not detected as malicious by any of the engines. In particular, we divide the collected apps into 3 types of datasets:

- **Training Set**: This dataset is used to train the Base Models of each DNN model and obtain a number of model ensembles (specified by different uncertainty estimation methods), which are used to estimate the uncertainty of the predictions of the underlying

Base Model. To the best of our knowledge, MalGenome [78] and MalRadar [63] are the most reliable malware datasets in our community as they were created by carefully examining Android-related security reports and blog contents from established antivirus companies and active experts. We therefore use them to train the Base Model. Specifically, MalGenome contains 1,260 samples and MalRadar contains 4,534 samples, resulting in a total of 5,794 malware samples from 196 unique families.

- **Correction Training Set**: This dataset is used to train a Correction Model. These samples are fed into the Base Models so that they are grouped into two sets, i.e., correctly and incorrectly classified. They are also fed into each model ensemble so that the uncertainty metrics are computed for each sample (as features). Specifically, the malware samples come from Drebin [3] which contains 5,560 apps from 179 different malware families.
- **Test Set**: This dataset is used to check the performance of the detector before and after revising the prediction results, which allows for evaluating the efficacy of the Correction Model. The malware samples come from AMD [64], which contains 24,650 apps from 71 different malware families. We randomly select 2,000 malware samples that do not coincide with the Drebin dataset.

As the three datasets were collected independently across different time span, which leads to an imbalance in the distribution of samples across malware families. For example, for AMD, over half of the samples are adware (e.g., Airpush, Dowgin and KuGuo). However, for MalRadar, only about 0.06% are related to adware. Following the definition of prior work [71], this contributes to the problem of concept-drift among the three datasets.

5.1.3 Configurations. Without loss of generality, we select two different DNN-based malware detectors as Base Models. The first Base Model is DeepDrebin, which is an MLP model consisting of two fully connected hidden layers with 200 neurons trained for 30 epochs. The second one is MultimodelDNN [32], which consists of five headers and an integrated part. Each header is composed of two fully connected layers containing 500 neurons, and the integrated part is composed of a fully connected layer containing 200 neurons. It is also trained for 30 epochs. All hyperparameters related to the base model are consistent with those in the original papers.

For each Base Model, we compute several model ensembles according to each uncertainty estimation method, including *Epoch ensemble*, *VBI*, *MC dropout*, *Deep Ensemble* and *wEnsemble*. Specifically, for Epoch Ensemble, we train five models with the same parameter settings except for different training epochs ranging from 10 to 50 (i.e., 10, 20, 30, 40, and 50 respectively). For MC dropout, we add a dropout layer with a dropout rate of 0.4 into the fully connected layer, convolution layer and LSTM layer. We have integrated 10 member models. For VBI, we sample the parameters of full-connection layer or convolution layer (i.e. weight and deviation) from the Gaussian distribution. The variables in the Gaussian distribution are the mean and standard deviation, which are learned through backpropagation. We finally integrated 10 member models. For Deep Ensemble and Weighted Deep Ensemble, we also integrated 10 members respectively.

5.1.4 Evaluation Metrics. We use the commonly used metrics (accuracy and F1) to measure the effectiveness of the models. Accuracy is a fundamental metric for assessing the overall performance of a classifier. F1 is a robust metric that takes into account both precision and recall, offering a more comprehensive measurement when dealing with imbalanced datasets. Additionally, as we observe that the majority of misclassifications are cases where malicious apps are misclassified as benign apps (i.e., FNR is much larger than FPR), we considered an alternative strategy that modifies only false negatives (FNs) in the process of correcting prediction results. Thus, we also incorporate two metrics, "Acc@FN" and "F1@FN", which measure the accuracy and F1 of this alternative strategy.

5.2 RQ1: Improvement on Malware Detection

We first evaluate how well MalCertain improves the performance of the baseline models when detecting OOD examples. Table 1 shows the results. For DeepDrebin, the Base Model and the Large Base Model have an accuracy of 71.2% and 83.68%, respectively. The F1 scores are 60.49% and 80.81%, respectively. This indicates that increasing the size of the training data can help improve the model performances to some extent. After the result correction of the Correction Model, the model accuracy is improved to 88.09% and the F1 score is improved to 86.78% (using the SVM algorithm).

Moreover, when we adopt another strategy to modify prediction results, i.e., modifying the prediction results only if they are considered as false negatives by the Correction Model (see Column "Acc @ FN"), the accuracies of the models are improved to 84.01% (KNN algorithm) to 89.06% (SVM algorithm). The F1 scores are improved to 81.99% (KNN algorithm) to 88.04% (SVM algorithm). Compared with the Base Models, MalCertain can improve the accuracy by 21.0% (average of 4 algorithms), and improve the accuracy of the Large Base Models by 3.0% (average of 4 algorithms). For the F1 score, the improvements achieved by MalCertain are 49% and 4.79% (average of 4 algorithms), for Base Model and Large Base Model, respectively.

As for MultimodelDNN, the Base Model and the Large Base Model have an accuracy of 75.78% and 84.66%, F1 scores of 68.72% and 81.62%, respectively. The best configuration is to use the DT algorithm and the modification strategy of only modifying false negatives. The accuracy can be improved to 86.85% and the F1 to 85.36%. Overall, the results show that MalCertain works well for the two kinds of Base Models. Their Correction Models can effectively calibrate the unreliable predictions and thus improve the performance of the Base Models in Android malware detection.

5.2.1 Performance of Correction Model. Our Correction Model identifies the misclassified samples from the Base Model. For DeepDrebin, the Base Model incorrectly classified 1,151 samples. The Correction Model flags a total of 793 samples, with 734 being true misclassifications, achieving a precision of 92.56% (SVM algorithm). Similarly, for MultimodelDNN, the Base Model misclassified 963 samples. Here, the Correction Model identifies 536 samples as "misclassified", of which 476 are confirmed misclassifications, achieving a precision of 88.81% (DT algorithm). The results show that our correction model achieves a high precision in identifying misclassified samples, correcting over half of the misclassified samples.

Table 1: Evaluation results of the Base Models, the Large Base Models, and the Correction Models on out-of-distribution (OOD) data. The best results are shown in bold.

DeepDrebin		Acc	F1	Acc @ FN	F1 @ FN
Base Model		71.20%	60.49%	-	-
Large Base Model		83.68%	80.81%	-	-
Correction Model	SVM	**88.09%**	**86.78%**	**89.06%**	**88.04%**
	KNN	83.16%	80.71%	84.01%	81.99%
	DT	84.18%	82.17%	85.66%	84.24%
	RF	85.14%	83.36%	85.91%	84.49%
MultimodelDNN		**Acc**	**F1**	**Acc @ FN**	**F1 @ FN**
Base Model		75.78%	68.72%	-	-
Large Base Model		84.66%	81.62%	-	-
Correction Model	SVM	84.94%	82.61%	86.22%	84.37%
	KNN	84.76%	82.79%	85.24%	83.49%
	DT	**86.24%**	**84.51%**	**86.85%**	**85.36%**
	RF	85.84%	84.09%	86.22%	84.64%

Answer to RQ1: MalCertain is applied on two different malware detectors and achieves performance improvements for both of them (by 21.0% on average). These results indicate that MalCertain can enhance the performance of the existing DNN-based malware detectors by adjusting their prediction results based on the uncertainty.

5.3 RQ2: Improvement on Adversarial Detection

We next evaluate how well MalCertain improves the Base Models in detecting adversarial samples. We collected adversarial Android malware samples based on a new attack method called *mixture of attacks* [36]. This attack method mutates training samples in both the APK file space and the feature space and feeds these samples to the attacked models to search for the samples that have the highest probability to be classified as "benign". Its supported mutation operations include adding or removing features in the manifest files (e.g., request extra permissions, state additional activities, services, Intent-filter, etc.), adding junk code (e.g., null OpCode, debugging information, dead functions or classes) to the *.dex* file, and flipping values in the feature vectors. We collected two kinds of adversarial samples: 'Basic DNN' (800 samples) and 'AT-RFGSM' (800 samples). 'Basic DNN' samples are generated by attacking a basic DNN-based Android malware detection model, such as Drebin [3]. 'AT-RFGSM' samples are generated by attacking an enhanced DNN model that incorporates adversarial training with the inner maximizer solved by iterative FGSM using randomized "rounding".

Table 2 shows the results of detecting adversarial samples. Both DeepDrebin and MultimodelDNN perform poorly in detecting adversarial samples. For the 'AT-RFGSM' samples, the accuracies of the Base Models are only 30.38% and 39.75%, and the F1 scores are 46.60% and 56.89%, respectively. Even the Large Base Models can achieve accuracies only around 63.88% and 60.75%, and the F1 scores around 77.96% and 75.58%. For the 'Basic DNN' samples, the accuracy and the F1 score are almost zero. This is expected as these samples are generated by mutating Drebin's extracted features.

Table 2: Evaluation results on adversarial samples.

DeepDrebin		AT-RFGSM			
		Acc	**F1**	**Acc @ FN**	**F1 @ FN**
Base Model		30.38%	46.60%	-	-
Large Base Model		63.88%	77.96%	-	-
Correction Model	SVM	71.25%	83.21%	77.75%	87.48%
	KNN	71.38%	83.30%	75.75%	86.20%
	DT	71.50%	83.38%	76.63%	86.77%
	RF	72.25%	83.89%	78.63%	88.03%
DeepDrebin		**Basic DNN**			
		Acc	**F1**	**Acc @ FN**	**F1 @ FN**
Base Model		0.00%	0.00%	-	-
Large Base Model		0.00%	0.00%	-	-
Correction Model	SVM	4.75%	9.07%	4.75%	9.07%
	KNN	4.75%	9.07%	4.75%	9.07%
	DT	25.88%	41.11%	25.88%	41.11%
	RF	13.25%	23.40%	13.25%	23.40%
MultimodelDNN		**AT-RFGSM**			
		Acc	**F1**	**Acc @ FN**	**F1 @ FN**
Base Model		39.75%	56.89%	-	-
Large Base Model		60.75%	75.58%	-	-
Correction Model	SVM	75.63%	86.12%	87.13%	93.12%
	KNN	73.75%	84.89%	86.25%	92.62%
	DT	78.75%	88.11%	87.75%	93.48%
	RF	78.88%	88.19%	87.63%	93.40%
MultimodelDNN		**Basic DNN**			
		Acc	**F1**	**Acc @ FN**	**F1 @ FN**
Base Model		0.00%	0.00%	-	-
Large Base Model		0.13%	0.25%	-	-
Correction Model	SVM	94.38%	97.11%	94.38%	97.11%
	KNN	82.88%	90.64%	82.88%	90.64%
	DT	16.13%	27.77%	16.13%	27.77%
	RF	2.13%	4.16%	2.13%	4.16%

By using the Correction Model, MalCertain can greatly improve the detection accuracy against these adversarial samples. For the 'AT-RFGSM' samples, by correcting all unreliable results, MalCertain can improve the F1 scores of the Base Models to at least 83.21% (DeepDrebin, SVM algorithm) and 84.89% (MultimodelDNN, KNN algorithm), achieving over 78.56% and 49.22% improvements, respectively. By correcting only unreliable FNs, MalCertain can improve the F1 scores of the Base Models to at least 86.2% (DeepDrebin, KNN algorithm) and 92.62% (MultimodelDNN, KNN algorithm), achieving over 84.98% and 62.86% improvements, respectively. Correcting only unreliable FNs achieves slightly better improvements than correcting all unreliable results. For the 'Basic-DNN' samples, MalCertain can improve the F1 scores ranging from 9.07% to 41.11% on the DeepDrebin Base Model and ranging from 4.16% to 97.11% on the MultimodelDNN Base Model. Note that the Correction Models trained by four machine learning algorithms have comparable effectiveness in improving the accuracy of detecting adversarial samples.

Answer to RQ2: We use 2 kinds of adversarial samples for detection, and both the Base Model (accuracy being 0 to 39.75%)

Table 3: Impacts of data balance and scales (DeepDrebin).

DeepDrebin		Base Model Acc: 71.20%						
Large Base Model Acc		Data Ratio		Algo.	Acc		Acc@FN	
unbal	bal	unbal	bal	-	unbal	bal	unbal	bal
83.68%	85.91%	100% (11320)	100% (4672)	SVM	88.09%	88.76%	89.06%	90.39%
				KNN	83.16%	83.91%	84.01%	85.59%
				DT	84.18%	80.31%	85.66%	84.64%
				RF	85.14%	84.06%	85.91%	86.09%
83.63%	82.68%	80% (9056)	80% (3738)	SVM	88.41%	88.14%	89.36%	89.94%
				KNN	84.28%	83.91%	84.69%	85.41%
				DT	82.88%	83.68%	84.33%	85.49%
				RF	84.81%	84.18%	85.46%	86.31%
80.78%	81.71%	40% (4528)	40% (1868)	SVM	88.11%	87.89%	89.52%	89.92%
				KNN	84.43%	84.48%	85.04%	86.24%
				DT	83.88%	80.88%	84.64%	83.68%
				RF	84.99%	85.59%	85.44%	87.19%
82.21%	82.31%	20% (2264)	20% (934)	SVM	87.79%	86.44%	89.06%	89.14%
				KNN	84.86%	85.34%	85.44%	86.59%
				DT	87.04%	83.11%	88.81%	85.09%
				RF	84.79%	83.21%	85.29%	86.59%
82.21%	82.26%	10% (1132)	10% (467)	SVM	89.94%	85.51%	88.04%	87.79%
				KNN	84.76%	85.14%	84.66%	86.11%
				DT	80.56%	79.93%	83.66%	83.61%
				RF	83.91%	84.36%	84.99%	86.19%

Table 4: Impacts of data balance and scales (MultimodelDNN).

MultimodelDNN		Base Model Acc: 75.78%						
Large Base Model Acc		Data Ratio		Algo.	Acc		Acc@FN	
unbal	bal	unbal	bal	-	unbal	bal	unbal	bal
84.66%	84.03%	100% (11320)	100% (4672)	SVM	84.94%	86.70%	86.22%	88.31%
				KNN	84.76%	82.77%	85.24%	86.72%
				DT	86.24%	85.31%	86.85%	88.23%
				RF	85.84%	85.19%	86.22%	88.23%
84.14%	80.48%	80% (9056)	80% (3738)	SVM	84.63%	86.14%	85.89%	88.11%
				KNN	84.66%	83.48%	85.04%	86.29%
				DT	85.44%	81.51%	87.10%	85.76%
				RF	86.32%	84.43%	86.47%	86.62%
84.57%	84.41%	40% (4528)	40% (1868)	SVM	84.73%	83.83%	86.24%	86.37%
				KNN	84.73%	85.14%	85.14%	86.37%
				DT	85.46%	82.34%	86.90%	87.40%
				RF	85.51%	83.07%	86.09%	87.98%
83.80%	83.73%	20% (2264)	20% (934)	SVM	85.81%	83.75%	86.72%	87.95%
				KNN	85.19%	83.53%	85.49%	85.99%
				DT	86.75%	80.28%	87.30%	85.14%
				RF	85.54%	82.37%	85.76%	86.77%
82.02%	83.53%	10% (1132)	10% (467)	SVM	84.23%	81.64%	85.87%	88.36%
				KNN	85.51%	83.73%	85.87%	84.58%
				DT	81.66%	80.94%	84.41%	83.05%
				RF	85.61%	82.87%	86.70%	85.59%

and the Large Base Model (accuracy being 0.13% to 63.88%) perform poorly. MalCertain can significantly improve their detection effectiveness for both kinds of adversarial samples (achieve over 85.5% to 158.8% improvements for AT-RFGSM and over 2.13% to 94.38% improvements for Basic DNN).

5.4 RQ3: Impacts of Data Balance and Scale

In general, the number of samples that are correctly and incorrectly classified by the Base Models differs greatly, where the number of correctly classified samples is much higher than that of the incorrectly classified samples. Therefore, in the process of training the Correction Models, we face the problem of *sample imbalance*. In order to understand the impacts of the data (im)balance and the scale of the training dataset on the performance of the Correction Models, we further conducted a series of experiments using balanced and unbalanced datasets with varied data scales.

As aforementioned, we use a correction training set containing 5,560 benign apps and 5,560 malicious apps. We first use all of the samples to train the Correction Model. Since the number of correctly and incorrectly classified samples by the Base Model is unequal (70% versus 30%), these samples used for training the Correction Models form the unbalanced dataset. Next, we construct 4 more unbalanced datasets of smaller scales by randomly sampling 80%, 40%, 20%, and 10% from the unbalanced dataset. To obtain a balanced dataset, as the number of incorrectly predicted samples is much smaller than the number of correctly predicted samples, we use all the incorrectly predicted samples and randomly choose an equal number of correctly predicted samples. We then also construct 4 more balanced datasets of smaller scales by sampling 80%, 40%, 20%, and 10% of the balanced. Finally, we train the Correction Models using these 5 balanced datasets and 5 unbalanced datasets.

Table 3 and Table 4 show the impacts of data balance and scales on the Correction Models based on Deepdrebin and MultimodelDNN, respectively. Overall, we can derive the following observations. First, the performances of the Correction Models are relatively insensitive to the data balance. For example, comparing the results with the same data scales in Table 3, we can see that the Correction Models trained using the balanced datasets (columns "bal") generally have similar performances as those trained using the imbalanced datasets (columns "unbal"). We observed that in the unbalanced setting, fewer "incorrectly classified" samples are flagged compared to balanced settings. This is attributed to the unbalanced setting having more "correctly classified" training data, skewing the model toward this category. While in the balanced setting, the model flags more "incorrectly classified" samples, but finds fewer "correctly classified" samples. Thus, the balanced setting's increased true positives and false positives balance out, resulting in a similar overall accuracy as the unbalanced setting's. Second, the performances of the Correction Models are also insensitive to the data scales. The performances of the Correction Models with the data scales ranging from 10% to 100% are relatively close to each other. For example, the improved accuracy of the SVM Correction Model trained using 20% of the samples (87.79%) is close to that of the Correction Model trained using all samples (88.09%). Although the results in Table 4 show that the accuracies of the Correction Models gradually decrease with the decrease of the data scale, the differences are less than 5%. Thus, MalCertain only needs 10% of the samples to achieve reasonably good improvements.

Answer to RQ3: The scale of the correction training set and the balance of the samples have no significant impacts on the performance of the Correction Model. Thus, MalCertain can be tailored to datasets of small sizes (e.g., with only hundreds of correction training samples).

6 THREATS TO VALIDITY

First, in this paper, we select two widely used malware detectors for our experiments and demonstrate the generality of the framework. However, there are actually other DNN Android malware detector [7, 19, 38] in the research community that could be applied over, Besides, the selected detectors cannot represent all the techniques used in the DNN-based Android malware detection approaches. Other detectors using more complex features and model structures (e.g., CNN or LSTM) have not been used for the experiment. We will try more DNN malware detection approaches in the future.

Second, we make efforts to incorporate different kinds of uncertainty metrics to train the Correction Model. However, some uncertainty metrics we design are relatively straightforward, e.g., sub(maximum, mean). Other uncertainty metrics can be quantified and calculated as features to train the Correction Model. We further need to quantify the importance of each feature (uncertainty metrics), so as to improve our Correction Model.

Third, the current experimental results show that the improvement of MalCertain on the performance of Large Base Models is not quite significant in some cases. To boost MalCertain's capability, more effective uncertainty assessment methods can be explored and more advanced techniques can be tried to train the Correction Model. Nevertheless, we show that it is a promising direction to leverage prediction uncertainty to improve the performance of DNN detectors, especially when handling adversarial samples.

7 DISCUSSION

Application and Generalization of the Approach. While our design and experiments focus on DNN-based Android malware detection, the proposed framework can be generalized to other ML-based malware detection models. For example, for Reinforcement Learning (RL) models, there are a number of methods for estimating uncertainty [25, 29, 40]. In future work, we plan to investigate these methods to enable MalCertain for RL models.In addition, the proposed framework can be extended to other DNN-based classification tasks, e.g., security-sensitive behavior classification [67, 68]. For multi-class classification tasks, although the basic framework is applicable, we cannot directly use the existing strategy of flipping the prediction label for each unreliable prediction to achieve accuracy improvement. To generalize to multi-class classification, some alternative strategies need to be devised (e.g., modifying the predicted label to the second possible label).

ML Algorithms for the Correction Model. For the Correction Model, we have chosen four commonly used machine learning methods. Among them, the SVM algorithm achieves the best performance, but it cannot scale well. The KNN algorithm performs well in large-scale training sets, but it has a lower tolerance for noisy samples. The decision tree algorithm is a lightweight algorithm, but it suffers severely from model overfitting. The random forest algorithm integrates multiple decision tree models, and can effectively address the model overfitting. Our evaluation results demonstrate that these algorithms can work effectively under the MalCertain framework. Nevertheless, there remain other machine learning algorithms (e.g., MLP) that are more heavyweight and may achieve better results. We plan to conduct more experiments to integrate more effective machine learning algorithms into MalCertain.

Overhead of MalCertain. MalCertain employs five uncertainty estimation methods and produces a collection of uncertainty estimation models. Among them, the method based on Epoch Ensemble contains multiple base models with different training epochs, the methods based on Deep Ensemble and Weighted Deep Ensemble contain multiple base models with different initialization parameters, and the VBI and MC Dropout methods require repeated inference for each sample. While they introduce more computational costs and require more storage resources in model training, the training is a one-time effort and it is a worthwhile price to pay for increased performance. The prediction time of our approach for predicting a sample is about 0.0125s, which is still quite efficient even though it is larger than the Base Model (0.00025s).

8 CONCLUSION

In this paper, we propose MalCertain, a general optimization framework to improve the performance of DNN models for Android malware detection. We first conduct a characteristic study to understand how these DNN models are impacted by OOD samples that are often observed in zero-day malware or adversarial samples. We further adopt widely used uncertainty estimation methods and design a number of metrics to distinguish correctly-classified and incorrectly-classified samples based on their prediction uncertainties. Based on these metrics, we train an effective Correction Model to fine-tune the prediction results of DNN detectors. Extensive experiments indicate that MalCertain is capable of improving the performance of existing DNN models for Android malware detection, and can significantly improve the detection effectiveness of adversarial samples.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China (No. 2022YFB3104400), the National Natural Science Foundation of China (grant No.62072046), the Key R&D Program of Hubei Province (2023BAB017, 2023BAB079), the Knowledge Innovation Program of Wuhan-Basic Research, and Hong Kong RGC Project (No. PolyU15224121).

REFERENCES

[1] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th international conference on mining software repositories*. 468–471.

[2] AppBrain. 2022. "Number of Android Applications,". https://www.appbrain.com/stats..

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

[4] Michael Backes and Mohammad Nauman. 2017. LUNA: quantifying and leveraging uncertainty in Android malware analysis through Bayesian machine learning. In *2017 IEEE European symposium on security and privacy (euros&p)*. IEEE, 204–217.

[5] David Barber and Christopher M Bishop. 1998. Ensemble learning in Bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences* 168 (1998), 215–238.

[6] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 805–823.

[7] Jarrett Booz, Josh McGiff, William G Hatcher, Wei Yu, James Nguyen, and Chao Lu. 2018. Tuning Deep Learning Performance for Android Malware Detection. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial*

Intelligence, Networking and Parallel/Distributed Computing (SNPD). IEEE, 140–145.

[8] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *computers & security* 73 (2018), 326–344.

[9] Clarivate. [n. d.]. "Web of Science". https://www.webofscience.com/.

[10] Muneer Ahmad Dar and Javaid Parvez. 2013. Evaluating Smartphone Application Security: A Case Study on Android. *Global Journal of Computer Science and Technology* 13, E12 (2013), 9–15.

[11] Anthony Desnos. 2012. Android: Static analysis using similarity distance. In *2012 45th Hawaii international conference on system sciences*. IEEE, 5394–5403.

[12] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. 576–587.

[13] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. PMLR, 1050–1059.

[14] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. 2021. A Survey of Uncertainty in Deep Neural Networks. *arXiv preprint arXiv:2107.03342* (2021).

[15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[16] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. 281–294.

[17] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2016. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435* (2016).

[18] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *European symposium on research in computer security*. Springer, 62–79.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[20] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. 2019. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 558–567.

[21] G. E. Hinton and D. V. Camp. 1993. Keeping the neural networks simple by minimizing the description length of the weights. (1993).

[22] TonTon Hsien-De Huang and Hung-Yu Kao. 2018. R2-d2: Color-inspired Convolutional Neural Network (CNN)-based Android Malware Detections. In *2018 IEEE international conference on big data (big data)*. IEEE, 2633–2642.

[23] Donghui Hu, Zhongjin Ma, Xiaotian Zhang, Peipei Li, Dengpan Ye, Baohong Ling, et al. 2017. The Concept Drift Problem in Android Malware Detection and Its Solution. *Security and Communication Networks* 2017 (2017).

[24] Weiwei Hu and Ying Tan. 2017. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv preprint arXiv:1702.05983* (2017).

[25] Wenzhen Huang, Junge Zhang, and Kaiqi Huang. 2019. Bootstrap estimated uncertainty of the environment model for model-based reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 3870–3877.

[26] Sumedh Ingale and Sunil Gupta. 2014. SECURITY IN ANDROID BASED SMARTPHONE. *International Journal of Application or Innovation in Engineering Management* 3 (2014).

[27] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. 2020. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *2020 IEEE symposium on security and privacy (SP)*. IEEE, 1190–1206.

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

[29] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. 2017. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182* (2017).

[30] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D Joseph, and JD Tygar. 2013. Approaches to adversarial drift. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. 99–110.

[31] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems* 30 (2017).

[32] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. 2018. A multimodal deep learning method for Android malware detection using various features. *IEEE Transactions on Information Forensics and Security* 14, 3 (2018), 773–788.

[33] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems* 30 (2017).

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

[35] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. 2017. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325* (2017).

[36] Deqiang Li and Qianmu Li. 2020. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3886–3900.

[37] Deqiang Li, Tian Qiu, Shuo Chen, Qianmu Li, and Shouhuai Xu. 2021. Can We Leverage Predictive Uncertainty to Detect Dataset Shift and Adversarial Examples in Android Malware Detection?. In *Annual Computer Security Applications Conference*. 596–608.

[38] Dongfang Li, Zhaoguo Wang, and Yibo Xue. 2018. Fine-grained Android Malware Detection based on Deep Learning. In *2018 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 1–2.

[39] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. 2015. MARVIN: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis. In *2015 IEEE 39th annual computer software and applications conference*, Vol. 2. IEEE, 422–433.

[40] Björn Lütjens, Michael Everett, and Jonathan P How. 2019. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 8662–8668.

[41] Zhuo Ma, Haoran Ge, Zhuzhu Wang, Yang Liu, and Ximeng Liu. 2020. Droidetec: Android malware detection and malicious code localization through deep learning. *arXiv preprint arXiv:2002.03594* (2020).

[42] MalCertain. 2024. "MALCERTAIN:Enhancing Deep Neural Network Based Android Malware Detection by Tackling Prediction Uncertainty". https://github.com/Dirtyboy1029/MALCERTAIN/.

[43] Gilberto Manunza, Matteo Pagliardini, Martin Jaggi, and Tatjana Chavdarova. 2021. Improved Adversarial Robustness via Uncertainty Targeted Attacks. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*.

[44] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. 2017. Deep Android malware detection. In *Proceedings of the seventh ACM on conference on data and application security and privacy*. 301–308.

[45] John Mitros and Brian Mac Namee. 2019. On the validity of Bayesian neural networks for uncertainty estimation. *arXiv preprint arXiv:1912.01530* (2019).

[46] Tiwari Mohini, Srivastava Ashish Kumar, and Gupta Nitesh. 2013. Review on Android and smartphone security. *Research Journal of Computer and Information Technology Sciences* 2320 (2013), 6527.

[47] Andre T Nguyen, Fred Lu, Gary Lopez Munoz, Edward Raff, Charles Nicholas, and James Holt. 2022. Out of Distribution Data Detection Using Dropout Bayesian Neural Networks. *arXiv preprint arXiv:2202.08985* (2022).

[48] Andre T Nguyen, Edward Raff, Charles Nicholas, and James Holt. 2021. Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints. *arXiv preprint arXiv:2108.04081* (2021).

[49] Luis Oala, Cosmas Heiß, Jan Macdonald, Maximilian März, Wojciech Samek, and Gitta Kutyniok. 2020. Interval neural networks: Uncertainty scores. *arXiv preprint arXiv:2003.11566* (2020).

[50] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. 2019. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems* 32 (2019).

[51] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*. 729–746.

[52] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, Yu Wang, and Yang Xiang. 2019. A3CM: Automatic Capability Annotation for Android Malware. *IEEE Access* 7 (2019), 147156–147168.

[53] A-D Schmidt, Rainer Bye, H-G Schmidt, Jan Clausen, Osman Kiraz, Kamer A Yuksel, Seyit Ahmet Camtepe, and Sahin Albayrak. 2009. Static Analysis of Executables for Collaborative Malware Detection on Android. In *2009 IEEE International Conference on Communications*. IEEE, 1–5.

[54] Alexandru Constantin Serban, Erik Poll, and Joost Visser. 2018. Adversarial Examples - A Complete Characterisation of the Phenomenon. *arXiv preprint arXiv:1810.01185* (2018).

[55] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[56] Lewis Smith and Yarin Gal. 2018. Understanding Measures of Uncertainty for Adversarial Example Detection. *arXiv preprint arXiv:1803.08533* (2018).

[57] Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. 2019. Functional variational Bayesian neural networks. *arXiv preprint arXiv:1903.05779* (2019).

[58] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[59] Suman R Tiwari and Ravi U Shukla. 2018. An Android Malware Detection Technique Based on Optimized Permissions and API. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, 258–263.

[60] Halil Murat Ünver and Khaled Bakour. 2020. Android malware detection based on image-based features and machine learning techniques. *SN Applied Sciences* 2 (2020), 1–15.

[61] Sara Vicente, Joao Carreira, Lourdes Agapito, and Jorge Batista. 2014. Reconstructing pascal voc. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 41–48.

[62] R Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. 2019. Robust Intelligent Malware Detection Using Deep Learning. *IEEE access* 7 (2019), 46717–46738.

[63] Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. 2022. MalRadar: Demystifying Android Malware in the New Era. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 2 (2022), 1–27.

[64] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep Ground Truth Analysis of Current Android Malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 252–276.

[65] Andrew G Wilson and Pavel Izmailov. 2020. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems* 33 (2020), 4697–4708.

[66] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia joint conference on information security*. IEEE, 62–69.

[67] Shengqu Xi, Shao Yang, Xusheng Xiao, Yuan Yao, Yayuan Xiong, Fengyuan Xu, Haoyu Wang, Peng Gao, Zhuotao Liu, Feng Xu, , and Jian Lu. 2019. DeepIntent: Deep Icon-Behavior Learning for Detecting Intention-Behavior Discrepancy in Mobile Apps. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[68] Xusheng Xiao, Xiaoyin Wang, Zhihao Cao, Hanlin Wang, and Peng Gao. 2019. IconIntent: Automatic Identification of Sensitive UI Widgets based on Icon Classification for Android Apps. In *Proceedings of the International Conference on Software Engineering (ICSE)*.

[69] Ke Xu, Yingjiu Li, Robert H Deng, and Kai Chen. 2018. DeepRefiner: Multi-layer Android Malware Detection System Applying Deep Neural Networks. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 473–487.

[70] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. 2021. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334* (2021).

[71] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th USENIX Security Symposium (USENIX Security 21)*. 2327–2344.

[72] Peter Zegzhda, Dmitry Zegzhda, Evgeny Pavlenko, and Gleb Ignatev. 2018. Applying deep learning techniques for Android malware detection. In *Proceedings of the 11th International Conference on Security of Information and Networks*. 1–8.

[73] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. 2014. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 1105–1116.

[74] Xiao Zhang and David Evans. 2021. Understanding Intrinsic Robustness Using Label Uncertainty. In *International Conference on Learning Representations*.

[75] Xiyue Zhang, Xiaofei Xie, Lei Ma, Xiaoning Du, Qiang Hu, Yang Liu, Jianjun Zhao, and Meng Sun. 2020. Towards characterizing adversarial defects of deep learning software from the lens of uncertainty. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 739–751.

[76] Kai Zhao, Dafang Zhang, Xin Su, and Wenjia Li. 2015. Fest: A feature extraction and selection tool for Android malware detection. In *2015 IEEE symposium on computers and communication (ISCC)*. IEEE, 714–720.

[77] Xujiang Zhao, Yuzhe Ou, Lance Kaplan, Feng Chen, and Jin-Hee Cho. 2019. Quantifying classification uncertainty using regularized evidential neural networks. *arXiv preprint arXiv:1910.06864* (2019).

[78] Yajin Zhou and Xuxian Jiang. 2012. Dissecting Android Malware: Characterization and Evolution. In *2012 IEEE symposium on security and privacy*. IEEE, 95–109.