Can We Leverage Predictive Uncertainty to Detect Dataset Shift and Adversarial Examples in Android Malware Detection?

Deqiang Li
Nanjing University of Science and Technology
Nanjing, China
lideqiang@njust.edu.cn

Tian Qiu
Nanjing University of Science and Technology
Nanjing, China
qiutian@njust.edu.cn

Shuo Chen
RIKEN
Saitama, Japan
shuo.chen.ya@riken.jp

Qianmu Li*
Nanjing University of Science and Technology
Nanjing, China
qianmu@njust.edu.cn

Shouhuai Xu
University of Colorado Colorado Springs
Colorado Springs, Colorado, USA
sxu@uccs.edu

ABSTRACT

The deep learning approach to detecting *mal*icious soft*ware* (malware) is promising but has yet to tackle the problem of *dataset shift*, namely that the joint distribution of examples and their labels associated with the test set is different from that of the training set. This problem causes the degradation of deep learning models *without* users' notice. In order to alleviate the problem, one approach is to let a classifier not only predict the label on a given example but also present its uncertainty (or confidence) on the predicted label, whereby a defender can decide whether to use the predicted label or not. While intuitive and clearly important, the capabilities and limitations of this approach have not been well understood. In this paper, we conduct an empirical study to evaluate the quality of predictive uncertainties of malware detectors. Specifically, we re-design and build 24 Android malware detectors (by transforming four off-the-shelf detectors with six calibration methods) and quantify their uncertainties with nine metrics, including three metrics dealing with data imbalance. Our main findings are: (i) predictive uncertainty indeed helps achieve reliable malware detection in the presence of dataset shift, but cannot cope with adversarial evasion attacks; (ii) approximate Bayesian methods are promising to calibrate and generalize malware detectors to deal with dataset shift, but cannot cope with adversarial evasion attacks; (iii) adversarial evasion attacks can render calibration methods useless, and it is an open problem to quantify the uncertainty associated with the predicted labels of adversarial examples (i.e., it is not effective to use predictive uncertainty to detect adversarial examples).

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; • **Mathematics of computing → Probabilistic inference problems**.

KEYWORDS

malware detection, predictive uncertainty, deep learning, dataset shift, adversarial malware examples

ACM Reference Format:
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 (ACSAC '21), December 6–10, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3485832.3485916

1 INTRODUCTION

Malware is a big threat to cybersecurity. Despite tremendous efforts, communities still suffer from this problem because the number of malware examples consistently increases. For example, Kaspersky [24] detected 21.6 million unique malware in 2018, 24.6 million in 2019, and 33.4 million in 2020. This highlights the necessity of automating malware detection via machine learning techniques [48]. However, a fundamental assumption made by machine learning is that the training data distribution and the test data distribution are identical. In practice, this assumption is often invalid because of the *dataset shift* problem [37]. Note that dataset shift is related to, but different from, the *concept drift* problem, which emphasizes that the conditional distribution of the labels conditioned on the input associated with the test data is different from its counterpart associated with the training data [46]. While some people use these two terms interchangeably [28, 50], we choose to use dataset shift because it is a broader concept than concept drift.

Several approaches have been proposed for alleviating the dataset shift problem, such as: periodically retraining malware detectors [10, 47], extracting invariant features [49], and detecting example anomalies [20]. One fundamental open problem is to quantify the *uncertainty* that is inherent to the outcomes of malware detectors (or the confidence a detector has in its prediction). A well-calibrated uncertainty indicates the potential risk of the accuracy decrease

*Also with Wuyi University.

This work is licensed under a Creative Commons Attribution International 4.0 License.

ACSAC '21, December 6–10, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8579-4/21/12.
https://doi.org/10.1145/3485832.3485916

and enables malware analysts to conduct informative decisions (i.e., using the predicted label or not). One may argue that a deep learning model associates its label space with a probability distribution. However, this "as is" method is poorly calibrated and causes poor uncertainty estimates [18]. This problem has motivated researchers to propose multiple methods to calibrate the probabilities, such as: variational Bayesian inference [6], Monte Carlo dropout [15], stochastic gradient MCMC [45], and non-Bayesian methods such as ensemble [25]. Quantifying the uncertainty associated with predicted labels (i.e., predictive uncertainty) in the presence of dataset shift has received a due amount of attention in the context of image classification [6, 25], medical diagnoses [26], and natural language processing [39], but not in the context of malware detection. This motivates us to answer the question in the title.

Our contributions. In this paper, we empirically quantify the predictive uncertainty of 24 deep learning-based Android malware detectors. These detectors correspond to combinations of four malware detectors, which include two *multiple layer perceptron* based methods (i.e., DeepDrebin [17] and MultimodalNN [22]) and one *convolutional neural network* based method (DeepDroid [30]), and the *recurrent neural network* based method (Droidetec [29]); and six *calibration* methods, which are vanilla (no effort made for calibration), *temperature scaling* [18], Monte Carlo (MC) dropout [15], Variational Bayesian Inference (VBI) [39], deep ensemble [25] and its weighted version. The vanilla and temperature scaling calibration methods belong to the post-hoc strategy, while the others are ad-hoc and require us to transform the layers of an original neural network in a principled manner (e.g., sampling parameters from a learned distribution). In order to evaluate the quality of predictive uncertainty on *imbalanced* datasets, we propose useful variants of three standard metrics.

By applying the aforementioned 24 malware detectors to three Android malware datasets, we draw the following insights: (i) We can leverage predictive uncertainty to achieve reliable malware detection in the presence of dataset shift to some extent, while noting that a defender should trust the predicted labels with uncertainty below a certain threshold. (ii) Approximate Bayesian methods are promising to calibrate and generalize malware detectors to deal with dataset shift. (iii) Adversarial evasion attacks can render calibration methods useless and thus the predictive uncertainty.

We have made the code of our framework publicly available at https://github.com/deqangss/malware-uncertainty. It is worth mentioning that after the present paper is accepted, we became aware of a preprint [32], which investigates how to leverage predictive uncertainty to deal with false positives of Windows malware detectors, rather than dealing with dataset shift issues.

2 PROBLEM STATEMENT

2.1 Android Apps and Malware Detection

Since Android malware is a major problem and deep learning is a promising technique, our empirical study focuses on Android malware detection. Recall that an Android Package Kit (APK) is a zipped file that mainly contains: (i) *AndroidManifest.xml*, which declares various kinds of attributes about the APK (e.g., *package* name, required *permissions*, *activities*, *services*, *hardware*); (ii) *classes.dex*, which contains the APK's functionalities that can be understood by the Java virtual machines (e.g., Android Runtime); (iii) *res* folder and resources.arsc, which contain the resources used by an APK; (iv) *lib* folder, which contains the native binaries compatible to different Central Processing Unit (CPU) architectures (e.g., ARM and x86); (v) *META-INF* folder, which contains the signatures of an APK. For analysis purposes, one can unzip an APK to obtain its files in the binary format, or disassemble an APK by using an appropriate tool (e.g., Apktool [42] and Androguard [13]), to obtain human-readable codes and manifest data (e.g., *AndroidManifest.xml*).

A malware detector is often modeled as a supervised binary classifier that labels an example as benign ('0') or malicious ('1'). Let $\mathcal{Z}$ denote the example space (i.e., consisting of benign and malicious software) and $\mathcal{Y} = \{0, 1\}$ denote the label space. Let $p^*(z, y) = p^*(y|z)p^*(z)$ denote the underlying joint distribution, where $z \in \mathcal{Z}$ and $y \in \mathcal{Y}$. The task of malware detection deals with the conditional distribution $p^*(y|z)$. Specifically, given a software sample $z \in \mathcal{Z}$, we consider Deep Neural Network (DNN) $p(y = 1|z, \theta)$, which uses the *sigmoid* function in its output layer, to model $p^*(y|z)$, where θ represents the learnable parameters. A detector denoted by $f : \mathcal{Z} \to \mathcal{Y}$, which consists of DNNs and is learned from the training set D_{train}, returns 1 if $p(y = 1|z, \theta) \geq 0.5$ and 0 otherwise. We obtain $p(y = 0|z) = 1 - p(y = 1|z, \theta)$. Moreover, let $p'(x, y)$ denote the underlying distribution of test dataset D_{test}.

2.2 Problem Statement

There are two kinds of uncertainty associated with machine learning: *epistemic* vs. *aleatoric* [39]. The *epistemic* uncertainty tells us about which region of the input space is not perceived by a model [5]. That is, a data sample in the dense region will get a low epistemic uncertainty and get a high epistemic uncertainty in the sparse region. On the other hand, the *aleatoric* uncertainty is triggered by data noises and is not investigated in this paper.

It is widely assumed that $p'(z, y) = p^*(z, y)$ in malware detection and in the broader context of machine learning. However, this assumption does not hold in the presence of *dataset shift*, which reflects non-stationary environments (e.g., data distribution changed over time or the presence of adversarial evasion attacks) [37]. Most dataset shifts possibly incur changes in terms of epistemic uncertainty (excluding label flipping or concept of input z changed thoroughly). We consider three settings that potentially trigger the $p'(z, y) \neq p^*(z, y)$:

- *Out of source*: The D_{test} and D_{train} are drawn from different sources [12].
- *Temporal covariate shift*: The test data or $p'(z)$ evolves over time [35].
- *Adversarial evasion attack*: The test data is manipulated adversarially; i.e., $(z', y = 1) \sim p'(z, y = 1)$ is perturbed from $(z, y = 1) \sim p^*(z, y = 1)$, where z' and z have the same functionality and '$\sim$' denotes "is sampled from" [27].

One approach to coping with these kinds of dataset shifts is to make a malware detector additionally quantify the uncertainty associated with a prediction so that a decision-maker decides whether to use the prediction [25, 39, 44]. This means that a detector $p(y|z, \theta)$ should be able to model $p^*(y|z)$ well while predicting examples z of $(z, y) \sim p'(z, y)$ with high uncertainties when $(z, y) \nsim p^*(z, y)$, where $\nsim$ denotes "does not obey the distribution of". Specifically, the

quality of prediction uncertainty can be assessed by their confidence scores. This can be achieved by leveraging the notion of *calibration* [25]. A malware detector is said to be *well-calibrated* if the detector returns a same confidence score $q \in [0, 1]$ for a set of test examples $Z_q \subseteq \mathcal{Z}$, in which the malware examples are distributed as q [43]. Formally, let $\Pr(y = 1|Z_q)$ denote the proportion of malware examples in the set Z_q that are indeed malicious. We have

Definition 1 (Well-calibrated malware detector, adapted from [43]). *A probabilistic malware detector $p(\cdot, \theta) : \mathcal{Z} \to [0, 1]$ is well-calibrated if for each confidence $q \in [0, 1]$ and $Z_q = \{z : p(y = 1|z, \theta) = q, \forall z \in \mathcal{Z}\}$, it holds that $\Pr(y = 1|Z_q) = q$.*

Note that Definition 1 means that for a well-calibrated detector, the fraction of malware examples in example set Z_q is indeed q. This means that we can use q as a confidence score for quantifying uncertainty when assessing the trustworthiness of a detector's predictions. This also implies that *detection accuracy* and *well-calibration* are two different concepts. This is so because an accurate detector is not necessarily well-calibrated (e.g., when correct predictions with confidence scores near 0.5). Moreover, a well-calibrated malware detector is also not necessarily accurate.

It would be ideal if we can rigorously quantify or prove the uncertainty (or bounds) associated with a malware detector. However, this turns out to be a big challenge that has yet to be tackled. This motivates us to conduct an empirical study to characterize the uncertainty associated with Android malware detectors. Hopefully the empirical findings will contribute to theoretical breakthrough in the near future.

Specifically, our empirical study is centered at the question in title, which is further disintegrated as four Research Questions (RQs) as follows:

- **RQ1**: What is the predictive uncertainty of malware detectors in the absence of dataset shift?
- **RQ2**: What is the predictive uncertainty of malware detectors with respect to out-of-source examples?
- **RQ3**: What is the predictive uncertainty of malware detectors under temporal covariate shift?
- **RQ4**: What is the predictive uncertainty of malware detectors under adversarial evasion attacks?

Towards answering these questions, we need to empirically study the predictive distribution $p(y|z, \theta)$ of malware detectors in a number of scenarios.

3 EMPIRICAL ANALYSIS METHODOLOGY

In order to design a competent methodology, we need to select malware detector that are accurate in detecting malware, select the calibration methods that are appropriate for these detectors, and metrics. This is important because a "blindly" designed methodology would suffer from incompatibility issues (e.g., integrating different feature extractions modularly, or integrating post- and ad-hoc calibration methods together). The methodology will be designed in a modular way so that it can be extended to accommodate other models or calibration methods in a plug-and-play fashion.

3.1 Selecting Candidate Detectors

We focus on deep learning based Android malware detectors and more specifically choose the following four Android malware detectors.

DeepDrebin [17]: It is a Multiple Layer Perceptron (MLP)-based malware detector learned from the Drebin features [4], which are extracted from the *AndroidManifest.xml* file and the *classes.dex* file reviewed above. These features are represented by binary vectors, with each element indicating the presence or absence of a feature.

MultimodalNN [22]: It is a multimodal detector that contains five headers and an integrated part, all of which are realized by MLP. The 5 headers respectively learn from 5 kinds of features: (i) *permission-component-environment* features extracted from the manifest file; (ii) *strings* (e.g., IP address); (iii) system APIs; (iv) *Dalvik* opcode; and (v) ARM opcodes from native binaries. These features are represented by their occurrence frequencies. These 5 headers produce 5 pieces of high-level representations, which are concatenated to pass through the integrated part for classification.

DeepDroid [30]: It is Convolutional Neural Network (CNN)-based and uses the TextCNN architecture [23]. The features are *Dalvik* opcode sequences of *smali* codes.

Droidetec [29] is an RNN-based malware detector with the architecture of Bi-directional Long Short Term Memory (Bi-LSTM) [41]. Droidetec partitions a Function Call Graph (FCG) into sequences according to the caller-callee relation and then concatenates these sequences for passing through the Bi-LSTM model.

These detectors leverage several types of deep learning models. Indeed, we also implement an end-to-end method (*R2-D2* [19]) and find it ineffectiveness due to the over-fitting issue. Therefore, we eliminate it from our study.

3.2 Selecting Calibration Methods

In order to select calibration methods to calibrate the malware detectors, the following two criteria can be used: (i) they are known to be effective and (ii) they are scalable for learning a deep learning model. In particular, the preceding highlights the importance of the computational complexity of a calibration method. Based on a previous study [39], these criteria lead us to select the following six methods.

Vanilla: It serves as a baseline and means that no effort is made to calibrate a model $p(y = 1|z, \theta)$.

Temperature scaling (Temp scaling) [18]: It is a post-processing method that learns extra parameters to scale the *logits* of deep learning models on the validation set, where *logits* are the input of the activation of sigmoid.

Monte Carlo dropout (MC dropout) [15]: It technically adds a *dropout* layer [40] before the input of every layer contained in a model. This dropout operation is performed in both the training and test phases different from the normal usage that turns dropout on in the training phase and off in the test phase. In theory, MC dropout is an approximate Bayesian inference, which takes as input an example z and marginalizes the parameters θ out for returning the predictive probability:

$$p(y = 1|z, D_{train}) = \int p(y = 1|z, \theta)p(\theta|D_{train})d\theta \qquad (1)$$

Due to the intricate neural networks, an analytical solution to obtaining $p(\theta|D_{train})$ is absent. One alternative is to approximate $p(\theta|D_{train})$ via a known functional form distribution $q(\omega)$ with variables ω, leading to $p(y|z, D_{train}) \approx \mathbb{E}_{q(\theta|\omega)}p(y|z, \theta)$. Specifically, let $\theta = \{\mathbf{W}_i\}_{i=1}^{l}$ be the set of l parameters of a neural network, MC dropout defines $q(\omega)$ as:

$$\mathbf{W}_i = \mathbf{M}_i \cdot \mathbf{V}_i \text{ with } \mathbf{V}_i \sim \text{Bernoulli}(r_i), \tag{2}$$

where $\mathbf{M}_i$ is learnable variables, entities of $\mathbf{V}_i$ are sampled from a Bernoulli distribution with the probability r_i (i.e., dropout rate), and $\omega = \{\mathbf{M}_i, r_i\}_{i=1}^{l}$. In the training phase, *variational learning* is leveraged to look for $\{\mathbf{M}_i\}_{i=1}^{l}$ [6, 16], which minimizes the Kullback-Leibler (KL) divergence between $p(\theta|D_{train})$ and $q(\omega)$. In the test phase, Eq.(1) is degraded by averaging $T(T > 0)$ models $\{\theta^j\}_{j=1}^{T}$ (each of which is sampled from $q(\omega)$), namely:

$$p(y = 1|z) = \frac{1}{T}\sum_{j=1}^{T} p(y = 1|z, \theta^i). \tag{3}$$

Eq.3 says $p(y = 1|z)$ is obtained just by keeping the dropout switched and averaging the results of T times predictions.

Variational Bayesian Inference (VBI) [39]: It is also an approximate Bayesian method. Distinguishing from MC dropout, the parameters θ of neural network are directly sampled from a known form distribution $q(\omega)$ with variables ω. That is

$$\mathbf{W} \sim q(\omega) \text{ with } \mathbf{W} \in \theta \tag{4}$$

The training and test manner is the same as MC dropout.

Deep Ensemble [25]: It learns T independent neural networks, which are diversified by randomly initialized parameters. The intuition behind this idea is that *dropout* itself is an ensemble [40].

Weighted Deep Ensemble (wEnsemble): It has the same setting as Deep Ensemble, except for the weighted voting

$$p(y|z) = \sum_{i=1}^{T} w_i p(y = 1|z, \theta^i) \tag{5}$$

with $w_i \geq 0$ and $\sum_{i=1}^{T} w_i = 1$.

3.3 Calibrating Detectors

Figure 1 highlights our methodology in calibrating deep malware detectors for answering the research questions mentioned above (i.e., RQ1-RQ4). Each calibration method is calibrated into each of the select detectors. The methodology has a training phase and a testing phase. Each phase has five modules: *preprocessing*, *layers customization*, *deep neural network*, *ensemble wrapper*, and *model post-processing*, which are described below.

The *preprocessing* module transforms program files into the data formats that can be processed by deep learning models, including feature extraction and low-level feature representation. The *layers customization* module modifies the layers of standard deep learning models to incorporate appropriate calibration methods, such as placing a *dropout* layer before the input of the fully connected layer, the convolutional layer, or the LSTM layer; sampling parameters from learnable distributions. The *deep neural network* module constructs the deep learning models with the customized layers mentioned above to process the preprocessed data for training or testing purposes. The *ensemble wrapper* module uses an ensemble of

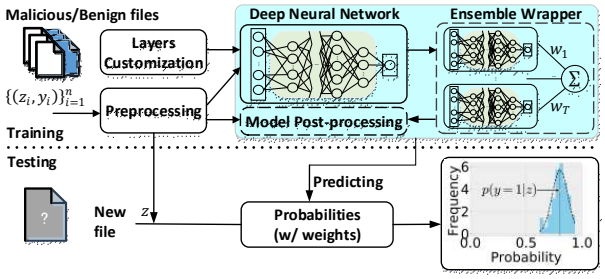


Figure 1: Our methodology for calibrating deep malware detectors to answer the aforementioned RQ1-RQ4, where dashed boxes indicate that the modules may not be necessary for some calibration methods. In the training phase, we learn the calibrated malware detectors. In the test phase, predicted probabilities (with weights if applicable) are obtained.

deep learning models in the training and testing phases according to the incorporated calibration method. In this module, we load basic block models sequentially in the training and test phases for relieving the memory complexity. The *model post-processing* module copes with the requirements of post-hoc calibration methods. The input to this module is the detector's output (i.e., predicted probability that a file belongs to which class) and a validation dataset (which is a fraction of data sampled from the same distribution as the training set). This means that this module does not affect the parameters θ of neural networks.

3.4 Selecting Metrics

In order to quantify the predictive uncertainty of a calibrated detector, we need to use a test dataset, denoted by D_{test}, and some metrics. In order to make the methodology widely applicable, we consider two scenarios: the ground-truth labels of testing dataset are available vs. are not available. It is important to make this distinction is important because ground-truth labels, when available, can be used to validate the trustworthiness of detectors' predictive uncertainties. However, ground-truth labels are often hard to costly to obtain, which motivates us to accommodate this more realistic scenario. This is relevant because even if the detector trainer may have access to ground-truth labels when learning detectors, these ground-truth labels may not be available to those that aim to quantify the detectors' uncertainties in making predictions.

3.4.1 Selecting Metrics When Ground-Truth Labels Are Given. There are standard metrics that can be applied for this purpose. However, these metrics treat each example as equally important and are insensitive to data imbalance, which is often encountered in malware detection. This prompts us to propose variants of standard metrics to deal with the issue of data imbalance. Specifically, we use the following three predictive uncertainty metrics, which are applicable when ground-truth labels are known.

The first standard metric is known as Negative Log-Likelihood (NLL), which are commonly used as a loss function for training a model and intuitively measure the goodness of a model fitting the dataset [9]. A smaller NLL value means a better calibration.

Formally, this metric is defined as:

$$\mathbb{E}_{(z,y)\in D_{test}} - (y \log p(y=1|z,\theta) + (1-y)\log(1-p(y=1|z,\theta))).$$

In order to deal with imbalanced data, we propose using the following variant, dubbed Balanced NLL (bNLL), which is formally defined as: $\frac{1}{|\mathcal{Y}|}\sum_{i=0}^{|\mathcal{Y}|} \text{NLL}(D^i_{test})$, where $\text{NLL}(D^i_{test})$ is the expectation of negative log-likelihood on the test set D^i_{test} of class i, $i \in \{0,1\}$. bNLL treats each class equally important, while NLL treats each sample equally.

The second standard metric is known as Brier Score Error (BSE), which measures the accuracy of probabilistic predictions [7, 39]. A smaller BSE value means a better calibration. This metric is formally defined as: $\mathbb{E}_{(z,y)\in D_{test}}(y - p(y=1|z,\theta))^2$. In order to deal with imbalanced data, we propose using the following variant, dubbed balanced BSE (bBSE), which is formally defined as: $\frac{1}{|\mathcal{Y}|}\sum_{i=0}^{|\mathcal{Y}|} \text{Brier}(D^i_{test})$, where $\text{Brier}(D^i_{test})$ is the expectation of BSE on the test set D^i_{test} of class i, $i \in \{0,1\}$.

The third standard metric is known as Expected Calibration Error (ECE), which also measures accuracy of predicted probabilities yet in a fine-grained manner [31]. A smaller ECE value means a better calibration. Formally, this metric is defined as follows. Given S buckets corresponding to quantiles of the probabilities $\{\rho_i\}_{i=1}^{S}$, the ECE is defined as

$$\text{ECE} = \sum_{s=1}^{S} \frac{B_s}{N} |\Pr(y=1|B_s) - \text{conf}(B_s)|, \tag{6}$$

where with a little abuse of notation, $B_s = \{(z,y) \in D_{test} : p(y=1|z,\theta) \in (\rho_s, \rho_{s+1}]\}$ is the number of examples in bucket s, $\Pr(y=1|B_s) = |B_s|^{-1}\sum_{(z,y)\in B_s}[y=1]$ is the fraction of malicious examples, $\text{conf}(B_s) = |B_s|^{-1}\sum_{(z,y)\in B_s} p(y=1|z,\theta)$, and $N = |D_{test}|$. The ECE metric suffers from imbalanced data because the majority class dominates some bins owing to the weights B_s/N. In order to deal with imbalanced data, we propose using the following variant, dubbed Unweighted ECE (uECE), which is formally defined as follows.

$$\text{uECE} = \sum_{s=1}^{S} \frac{1}{S} |\Pr(y=1|B_s) - \text{conf}(B_s)|.$$

3.4.2 Selecting Metrics When Ground-Truth Labels Are Not Given. There are three metrics [21, 25, 34] that can be applied to quantify predictive uncertainty in the absence of ground-truth labels. These metrics are applied to a point (i.e., expectation is not considered), which indicates these metrics do not suffer from the imbalanced dataset.

The first metric is known as Entropy, which intuitively measure a state of disorder in a physical system [21]. A larger entropy value means a higher uncertainty; Formally, this metric is defined as:

$$-(p(y|z,\theta)\log p(y|z,\theta) + (1-p(y|z,\theta))\log(1-p(y|z,\theta))), \tag{7}$$

where $p(y|z,\theta)$ denotes the model output $p(y=1|z,\theta)$ or Eq.(3).

The second metric is known as Standard Deviation (SD), which intuitively measures the inconsistency between the base classifiers and the ensemble one [34]. A larger SD value means a higher uncertainty. Formally, this metric is defined as:

$$\sqrt{\frac{T}{T-1}\sum_{i=1}^{T} w_i \left(p(y|z,\theta^i) - p(y|z,\theta)\right)^2},$$

where θ^i denotes the i-th DNN model, which has a weight w_i (ref. Eq.(5)) or $w_i = 1/T$.

The third metric is known as the KL divergence, which is an alternative to SD [25]. A larger KL value means a higher uncertainty. Formally, this metric is defined as:

$$\sum_{i=1}^{T} w_i(\text{KL}(p(y|z,\theta^i)||p(y|z,\theta))),$$

where KL denotes the Kullback-Leibler divergence.

3.5 Answering RQs

At this point, one can apply the calibrated detectors to quantify their predictive uncertainties. It should be clear that this methodology can be adopted or adapted by other researchers to conduct empirical studies with more kinds of detectors and more metrics.

4 EXPERIMENTAL RESULTS AND ANALYSIS

4.1 Experimental setup

We implement the framework using TensorFlow [2] and TensorFlow Probability libraries [1] and run experiments on a CUDA-enabled GTX 2080 Ti GPU. We below detail datasets and hyperparameters.

4.1.1 Datasets. We use 3 widely-used Android datasets: Drebin [4], VirusShare [11], and Androzoo [3].

Drebin: The Drebin dataset is built before the year of 2013 and is prepossessed by a recent study [27], which relabels the APKs using the VirusTotal service [38] that contains more than 70 antivirus scanners. An APK is treated as malicious if four or more scanners say it is malicious, and benign if no scanners say it is malicious; theoretical justification of such heuristic but widely-used practice has yet to be made [14]. The resultant Drebin dataset contains 5,560 malicious APKs and 42,333 benign APKs.

VirusShare: VirusShare is a repository of potential malware. We chose the APKs collected in 2013 and treat this dataset as *out-of-source* in regards to the Drebin dataset. These APKs are labeled using the same fashion as the Drebin dataset, producing 12,383 malicious APKs and 340 benign APKs, while throwing away the

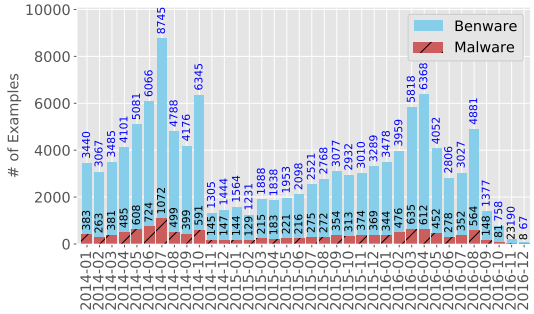


Figure 2: The Androzoo examples of APKs with time [35].

469 APKs in the grey area (i.e., at least one, but at most three, scanners say they are malicious).

Androzoo: Androzoo is an APK repository, including over 10 million examples, along with their VirusTotal reports. These APKs were crawled from the known markets (e.g., Google Play, AppChina). Following a previous study [35], we use a subset of these APKs spanning from January 2014 to December 2016, which includes 12,735 malicious examples and 116,993 benign examples. Figure 2 plots the monthly distribution of these examples with time [35].

4.1.2 Hyper-parameters. We present the hyper-parameters for malware detectors and then for calibration methods.

DeepDrebin [17] is an MLP, consisting of two fully-connected hidden layers, each with 200 neurons.

MultimodalNN [22] has five headers and an integrated part, where each header consists of two fully-connected layers of 500 neurons. The integrated part consists of two fully-connected layers of 200 neurons.

DeepDroid [30] has an embedding layer, followed by a convolutional layer and two fully-connected layers. The vocabulary of the embedding layer has 256 words, which correspond to 256 *Dalvik* opcodes; the embedding dimension is 8; the convolutional layer has the kernels of size 8×8 with 64 kernels; the fully-connected layer has 200 neurons. Limited by the GPU memory, DeepDroid can only deal with a maximum sequence of length 700,000, meaning that APKs with longer opcode sequences are truncated and APKs with shorter opcode sequences are padded with 0's (which correspond to nop).

Droidetec [29] has an embedding layer with vocabulary size 100,000 and embedding dimension 8, the bi-directional LSTM layer with 64 units, and a fully-connected hidden layer with 200 neurons. Droidetec allows the maximum length of API sequence 100,000. We further clip the gradient values into the range of [-100, 100] for Droidetec in case of gradient explosion.

All models mentioned above use the ReLU activation function. We also place a dropout layer with a dropout rate 0.4 before the last fully-connected layer (i.e., the output layer). We implement the four malware detectors by ourselves. The hyperparameters of calibration methods are detailed below.

Vanilla and **Temp scaling**: We have the same settings as the malware detectors mentioned above.

MC dropout: We use a dropout layer with a dropout rate 0.4. We add the dropout layer into the fully-connected layer, convolutional layer, and the LSTM layer, respectively. Following a recent study [39], we neglect the ℓ_2 regularization that could decline the detection accuracy. In the test phase, we sample 10 predictions for each example.

VBI: We sample the parameters of the fully-connected layer or the convolutional layer (i.e., weights and bias) from Gaussian distributions. A Gaussian distribution has the variables (*mean* and *standard deviation*), which are learned via back propagation using the reparameterization technique [6]. We do not implement VBI for Bi-LSTM, due to the effectiveness issue [39, 44]. This means only the last layer of Droidetec is calibrated by VBI. In the test phase, we sample 10 predictions for each example.

Ensemble and **wEnsemble**: We learn 10 base instances for each ensemble-based method.

We learn these models using the Adam optimizer with 30 epochs, batch size 16, and learning rate 0.001. A model is selected for evaluation when it achieves the highest accuracy on the validation set in the training phase. In addition, we calculate the validation accuracy at the end of each epoch.

4.2 Answering RQ1

In order to quantify the predictive uncertainty of malware detectors in the absence of dataset shift, we learn the aforementioned 24 malware detectors on the Drebin dataset, by splitting it into three disjoint sets of 60% for training, 20% for validation, and 20% for testing.

Table 1 summarizes the results, including detection estimation using the metrics False Negative Rate (FNR), False Positive Rate (FPR), Accuracy (Acc) or percentage of detecting benign and malicious samples correctly, balanced Accuracy (bAcc) [8] and F1 score [36], and uncertainty evaluation using the metrics NLL, bNLL, BSE, bBSE, ECE, and uECE. We make four observations. First, *Temp scaling* achieves the same detection accuracy as its vanilla model because it is a post-processing method without changing the learned parameters. On the other hand, MC dropout, Ensemble and wEnsemble improve detection accuracy but VBI degrades detection accuracy somewhat when compared with the vanilla model. In terms of balanced accuracy, the calibration methods do not always improve detection accuracy because the vanilla MultimodalNN actually achieves the highest balanced accuracy 98.61% among all those detectors. The reason may be that MultimodalNN itself is an ensemble model (e.g., 5 headers are equipped).

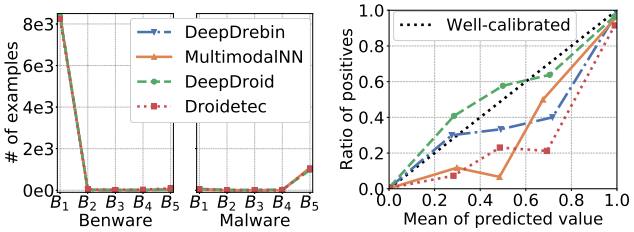
Second, ensemble methods (i.e., Ensemble and wEnsemble) reduce the calibration error when compared with the respective vanilla models. Moreover, the two ensemble methods exceed the other calibration methods in terms of the NLL, BSE and bBSE metrics, except for DeepDrebin (suggesting MC Dropout is best for calibration). Third, the measurements of the balanced metrics are notably larger than their imbalanced counterparts (e.g., bNLL vs. NLL), because benign examples dominate the test set and malware detectors predict benign examples more accurately than predicting malicious ones.

Fourth, uECE shows inconsistent results in terms of bNLL and bBSE. In order to understand the reasons, we plot the *reliability diagram* [33], which demonstrates the difference between the fraction of malicious examples in each bin, namely the difference between the $\Pr(y = 1|B_s)$ in Eq.(6) and the mean of the predicted confidence $conf(B_s)$. Figure 3 plots the results of the vanilla malware detectors, along with the number of examples in the bins. Figure 3a shows that most examples belong to bins B_1 and B_5. Figure 3b says DeepDroid achieves the lowest error (because it is closest to the diagonal than others), and shall be best calibrated, which contracts the ECE values in Table 1 (demonstrating that MultimodalNN is best instead). This is because as shown in Figure 3a, most benign examples belong to bin B_1 and most malicious examples belong to bin B_5. As a comparison, uECE does not suffer from this issue.

Insight 1. *Calibration methods reduce malware detection uncertainty; variational Bayesian inference degrades detection accuracy and F1 score in the absence of dataset shift; balanced metrics (i.e.,*

Table 1: Detection estimation and uncertainty evaluation of calibrated malware detectors in the absence of dataset shift.

Malware detector	Calibration method	Detection estimation (%)					Uncertainty evaluation					
		FNR	FPR	Acc	bAcc	F1	NLL	bNLL	BSE	bBSE	ECE	uECE
DeepDrebin	Vanilla	3.90	0.31	99.28	97.90	96.84	0.100	0.329	0.007	0.020	0.006	0.104
	Temp scaling	3.90	0.31	99.28	97.90	96.84	0.052	0.109	0.006	0.018	0.007	0.062
	MC Dropout	3.45	0.32	99.32	**98.12**	97.04	**0.033**	**0.094**	0.006	**0.015**	**0.002**	**0.056**
	VBI	**3.36**	0.83	98.88	97.91	95.22	0.054	**0.094**	0.009	0.016	0.012	0.102
	Ensemble	3.99	**0.19**	**99.37**	97.91	**97.24**	0.063	0.211	**0.005**	0.018	0.005	0.160
	wEnsemble	3.99	0.20	99.36	97.90	97.20	0.058	0.190	**0.005**	0.018	0.004	0.095
MultimodalNN	Vanilla	**2.16**	0.63	99.20	**98.61**	96.58	0.087	0.207	0.007	0.013	0.007	0.162
	Temp scaling	**2.16**	0.63	99.20	**98.61**	96.58	0.053	**0.086**	0.007	**0.012**	0.010	0.182
	MC Dropout	2.97	0.39	99.31	98.32	97.03	0.077	0.225	**0.005**	0.014	0.003	0.072
	VBI	8.45	0.33	98.73	95.61	94.35	0.073	0.253	0.010	0.036	0.004	0.078
	Ensemble	2.52	0.39	99.36	98.55	97.26	0.063	0.188	**0.005**	**0.012**	0.004	0.144
	wEnsemble	2.88	**0.26**	**99.44**	98.43	**97.56**	**0.046**	0.153	**0.005**	0.013	**0.002**	**0.045**
DeepDroid	Vanilla	8.53	0.69	98.41	95.39	92.99	0.097	0.311	0.013	0.039	0.009	0.059
	Temp scaling	8.53	0.69	98.41	95.39	92.99	0.076	0.220	0.013	0.038	**0.002**	**0.023**
	MC Dropout	7.62	0.66	98.54	95.86	93.57	0.078	0.239	0.012	0.035	0.004	0.055
	VBI	12.7	0.35	98.22	93.47	91.88	0.084	0.305	0.014	0.052	0.010	0.092
	Ensemble	7.44	**0.11**	99.05	96.23	95.73	**0.049**	0.171	0.008	0.029	0.006	0.162
	wEnsemble	**4.99**	0.27	**99.18**	**97.37**	**96.41**	0.050	**0.131**	**0.007**	**0.022**	0.008	0.066
Droidetec	Vanilla	4.14	1.53	98.17	97.17	92.30	0.119	0.208	0.016	0.026	0.014	0.205
	Temp scaling	4.14	1.53	98.17	97.17	92.30	0.101	0.167	0.016	0.026	0.015	0.180
	MC Dropout	**3.03**	1.40	98.41	97.78	93.32	0.088	0.142	0.014	0.020	0.013	0.207
	VBI	3.22	1.67	98.15	97.55	92.29	0.118	0.202	0.015	0.022	0.015	0.262
	Ensemble	3.13	0.75	98.98	**98.06**	95.60	0.055	0.107	0.009	**0.016**	0.010	0.143
	wEnsemble	3.49	**0.59**	**99.07**	97.96	**95.98**	**0.046**	**0.101**	**0.007**	**0.016**	**0.008**	**0.116**



(a) # of examples per bin. (b) Reliability diagram

Figure 3: Reliability diagram of vanilla malware detectors. There are 5 bins $B_1, \ldots, B_5$ and "Benware" denotes the *ben*ign soft*ware*.

bNLL, bBSE and uECE) are more sensitive than their imbalanced counterparts (i.e., NLL, BSE and ECE) when data imbalance is present.

4.3 Answering RQ2

In order to quantify the predictive uncertainty of malware detectors with respect to out-of-source examples, we apply the Drebin malware detectors to the VirusShare dataset. We assess predictive distribution and report the accuracy of malware detectors on the datasets obtained after removing the examples for which the detectors are uncertain (i.e., with an entropy value above a threshold τ); this corresponds to the real-world usefulness of quantifying predictive uncertainty (i.e., discarding prediction results for which detector is uncertain).

Table 2 summarizes the uncertainty evaluation and the corresponding detection accuracy. We make three observations. (i) Malware detectors achieve low accuracy with out-of-source test examples. Nevertheless, DeepDrebin incorporating VBI obtain an accuracy of 82.69%, which notably outperforms other detectors. It is reminding that VBI hinders the detection accuracy in the absence of dataset shift. (ii) Calibration methods (e.g., VBI or Ensemble) reduce the uncertainty in terms of bNLL and bBSE when compared with the vanilla models, except for the MultimodalNN model incorporating MC dropout. (iii) DeepDrebin incorporating VBI also achieves the best calibration results, suggesting that VBI benefits from both regularization and calibration. On the other hand, DeepDroid and Droidetec suffer from the setting of *out of source*. Both models handle the very long sequential data that would be truncated due to the limited GPU memory, leading to the inferior results.

Figure 4 illustrates the density of predictive entropy. Figure 4b further shows the accuracy on the examples after removing the ones for which the detectors are uncertain about their predictions. We make the following observations. (i) The vanilla models return zero

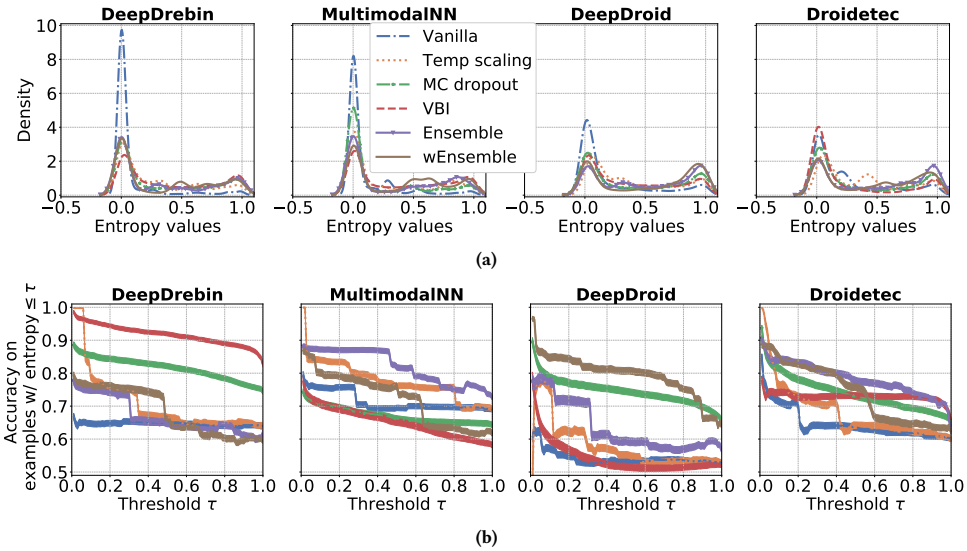


Figure 4: Predictive entropy (see Eq.(7)) of malware detectors trained on the Drebin dataset and tested on the VirusShare: (a) histogram of predictive entropy; (b) accuracy on the dataset after excluding the examples for which the detector has high uncertainties (i.e., the examples for which the predictive entropy is above a pre-determined threshold τ), which corresponds to the real-world use of the quantified predictive uncertainty.

Table 2: Predictive uncertainty of malware detectors that are learned on the Drebin dataset and tested on the VirusShare.

	Calibration method	Detection(%)		Uncertainty evaluation					
		Acc	bAcc	NLL	bNLL	BSE	bBSE	ECE	uECE
DeepDrebin	Vanilla	63.96	77.59	5.52	3.58	0.35	0.21	0.359	0.483
	Temp scaling	63.96	77.59	1.65	1.39	0.31	0.19	0.363	0.468
	MC Dropout	74.39	83.53	1.48	0.95	0.20	0.13	0.274	0.463
	VBI	**82.69**	**87.07**	**0.64**	**0.52**	**0.13**	**0.10**	**0.210**	**0.434**
	Ensemble	61.76	78.04	3.43	2.15	0.32	0.19	0.394	0.466
	wEnsemble	59.49	76.59	3.08	1.79	0.32	0.19	0.397	0.469
MultimodalNN	Vanilla	69.02	80.48	4.05	2.59	0.29	0.18	0.306	0.463
	Temp scaling	69.02	80.48	**1.45**	**1.13**	0.25	0.17	0.303	0.464
	MC Dropout	64.38	78.38	4.61	2.58	0.32	0.18	0.361	0.473
	VBI	58.17	77.21	2.32	1.25	0.33	0.18	0.414	0.478
	Ensemble	**72.66**	**82.20**	2.24	1.34	**0.21**	**0.14**	**0.279**	**0.450**
	wEnsemble	61.73	77.74	2.21	1.26	0.30	0.17	0.379	0.471
DeepDroid	Vanilla	52.85	70.02	3.44	2.11	0.42	0.27	0.463	0.481
	Temp scaling	52.85	70.02	2.09	1.31	0.39	0.25	0.460	0.477
	MC Dropout	**65.50**	**74.22**	1.54	1.10	0.26	0.20	**0.338**	0.465
	VBI	56.57	72.65	2.44	1.46	0.40	0.25	0.476	0.475
	Ensemble	56.57	72.65	1.78	1.12	0.33	0.20	0.435	0.475
	wEnsemble	64.11	**74.22**	**1.22**	**1.00**	**0.24**	**0.19**	0.350	**0.461**
Droidetec	Vanilla	59.98	70.04	2.22	1.59	0.34	0.253	0.389	0.476
	Temp scaling	59.98	70.04	1.66	1.20	0.32	0.235	0.390	0.476
	MC Dropout	64.99	72.61	1.57	1.33	0.27	0.222	0.344	0.472
	VBI	**65.30**	**74.21**	2.14	1.85	0.27	0.210	**0.315**	0.473
	Ensemble	63.83	72.45	**1.33**	**0.94**	**0.24**	**0.185**	0.343	0.470
	wEnsemble	62.26	71.64	1.50	1.23	0.28	0.219	0.375	0.465

entropy value for many examples, but calibration methods relieve

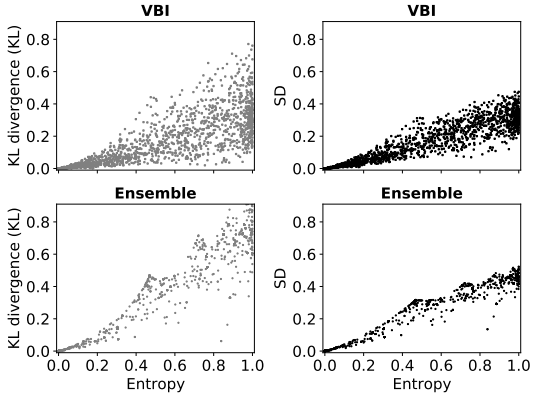


Figure 5: Scatter points of the randomly selected 2,000 test examples from the VirusShare dataset with paired values (Entropy, KL) and (Entropy, SD) that are obtained by applying DeepDrebin incorporating VBI or Ensemble.

this situation notably. Considering that the higher entropy delivers more uncertainty, vanilla model is poorly calibrated regarding the out-of-source examples. This is further confirmed that vanilla models exhibit a limited change along with increasing thresholds in Figure 4b. (ii) Ensemble and wEnsemble increase the robustness of deep learning models in detecting out-of-source examples. For example, the MultimodalNN, DeepDroid, and Droidetec models can be enhanced by Ensemble and wEnsemble to some extent, especially when applied to predicting the examples with entropy values below 0.3 (i.e., the detectors are relatively certain about their predictions). (iii) DeepDrebin incorporating either VBI or *MC dropout* make a significant achievement by comparing with ensemble-based

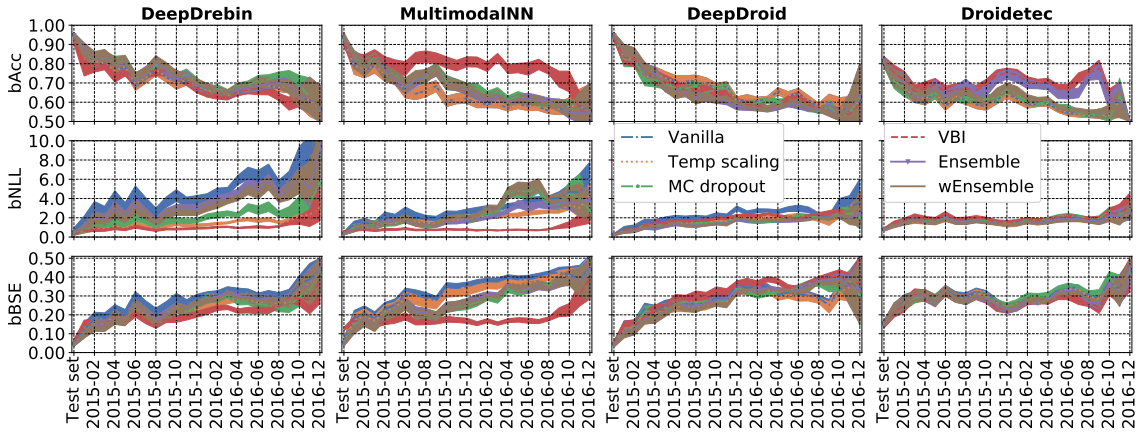


Figure 6: Illustration of balanced accuracy (bAcc), balanced NLL (bNLL), balanced BSE (bBSE) under temporal covariate shift.

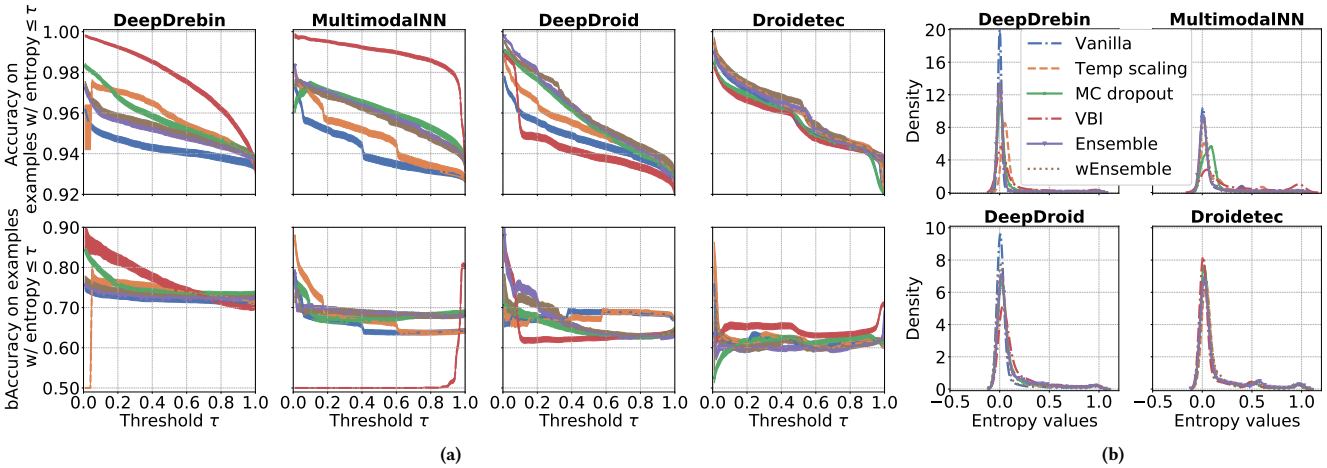


Figure 7: Predictive entropy (see Eq.(7)) of malware detectors on Androzoo dataset: (a) accuracy (upper row) and bAccuracy (bottom row) on the Androzoo test set after excluding the examples for which the detectors have high uncertainties (i.e., the examples for which the predictive entropy is above a pre-determined threshold τ); (b) test sample density of predictive entropy.

calibrations. Because VBI requires suitable prior distributions, it is challenging to generalize this calibration to all models effectively [6]. (iv) *Temp scaling* demonstrates an un-intuitive phenomenon that it achieves almost 100% accuracy at the start of the curves when applied to DeepDrebin, MultimodalNN and Droidetec, but degrades the accuracy of DeepDroid. The is because *Temp scaling* tends to over-approximate the predicted probabilities, resulting in high confidence score for examples, some of which are mis-classified, however. In addition, the balanced accuracy demonstrates the similar experimental results (cf. the appendix materials for details).

Figure 5 plots the relationship between the entropy and the KL divergence (KL), and the relationship between the entropy and the Standard Deviation (SD). A scatter point represents a test example based on the paired value. We observe that these three measurements are closely related, which explains why we use the entropy to characterize the predictive uncertainty solely. We only report the results for the calibrated DeepDrebin with VBI and Ensemble, because similar phenomena are observed for the other models. Note that Vanilla and Temp scaling do not apply to KL divergence and standard deviation.

Insight 2. *Deep ensemble benefits calibration of malware detectors against out-of-source test examples, but a carefully tuned VBI model could achieve a higher quality of uncertainty than ensemble-based methods; Measurements of entropy, KL divergence and standard deviation are closely related.*

4.4 Answering RQ3

In order to quantify the predictive uncertainty of malware detectors under temporal covariate shift, we use Androzoo dataset. Specifically, we train malware detectors upon the APKs collected in the year of 2014, and test these malware detectors upon APKs collected in the year of 2015 and 2016 at the month granularity. We also split the APKs in the year of 2014 into three disjoint datasets, 83.4% for

Table 3: Effectiveness of calibrated malware detectors under adversarial evasion attacks.

Malware detector	Calibration method	No attack				"Max" PGDs+GDKDE attack				Mimicry attack			
		Acc (%)	NLL	BSE	ECE	Acc (%)	NLL	BSE	ECE	Acc (%)	NLL	BSE	ECE
DeepDrebin	Vanilla	96.09	0.629	0.037	0.039	**0.00**	33.22	1.000	1.000	66.09	4.778	0.317	0.334
	Temp scaling	96.09	0.184	0.033	0.042	**0.00**	**7.015**	**0.985**	**0.992**	66.09	1.427	0.266	0.332
	MC Dropout	**96.55**	0.186	0.029	**0.040**	**0.00**	33.22	1.000	1.000	69.18	1.639	0.245	**0.317**
	VBI	96.27	**0.142**	**0.025**	0.051	**0.00**	33.22	1.000	1.000	**69.91**	**1.034**	**0.211**	0.320
	Ensemble	96.00	0.403	0.034	0.042	**0.00**	33.22	1.000	1.000	64.82	3.296	0.295	0.363
	wEnsemble	96.00	0.362	0.034	0.042	**0.00**	33.22	1.000	1.000	64.64	2.944	0.296	0.362
Multimo -dalNN	Vanilla	**97.82**	0.368	0.020	**0.023**	**0.00**	33.22	1.000	1.000	87.64	1.530	0.107	**0.119**
	Temp scaling	**97.82**	**0.129**	**0.019**	0.025	**0.00**	**8.871**	**0.996**	**0.998**	87.64	0.562	0.094	0.122
	MC Dropout	97.18	0.399	0.024	0.030	**0.00**	33.22	1.000	1.000	85.64	1.822	0.121	0.152
	VBI	96.82	0.166	0.026	0.042	**0.00**	33.22	1.000	1.000	**89.64**	**0.506**	**0.080**	**0.119**
	Ensemble	97.45	0.355	0.021	0.026	**0.00**	33.22	1.000	1.000	88.09	1.148	0.091	0.129
	wEnsemble	97.09	0.295	0.025	0.035	**0.00**	33.22	1.000	1.000	84.27	1.124	0.123	0.187
DeepDroid	Vanilla	91.55	0.587	0.073	0.095	85.45	0.773	0.116	0.148	86.09	0.786	0.110	0.142
	Temp scaling	91.55	0.404	0.069	0.113	85.45	0.536	0.105	0.165	86.09	0.538	0.101	0.158
	MC Dropout	92.55	0.451	0.066	0.097	**93.55**	**0.273**	**0.048**	**0.074**	90.18	0.529	0.083	0.126
	VBI	87.27	0.592	0.102	0.151	84.00	0.592	0.117	0.180	82.00	0.705	0.136	0.200
	Ensemble	92.55	0.329	0.058	0.099	90.64	0.366	0.068	0.129	89.55	0.433	0.079	0.142
	wEnsemble	**95.00**	**0.237**	**0.040**	**0.069**	93.00	0.309	0.057	0.111	**92.82**	**0.348**	**0.061**	**0.111**
Droidetec	Vanilla	98.08	0.123	0.016	0.028	66.03	2.331	0.298	0.341	88.80	0.656	0.099	0.126
	Temp scaling	98.08	0.113	0.017	0.039	66.03	1.717	0.285	0.345	88.80	0.514	0.095	0.138
	MC Dropout	98.81	0.063	0.009	0.018	67.30	2.066	0.272	0.330	93.17	0.268	0.052	0.090
	VBI	98.54	0.095	0.012	0.016	71.31	1.996	0.250	0.292	93.08	0.396	0.060	0.078
	Ensemble	**99.18**	0.060	**0.008**	**0.014**	**80.87**	**0.657**	**0.138**	**0.215**	**96.17**	0.175	0.030	0.068
	wEnsemble	99.00	**0.048**	**0.008**	0.015	80.05	0.776	0.151	0.232	95.90	**0.156**	**0.029**	**0.066**

training, 8.33% for validation (8.33% is the average percentage of APKs emerging in each month of 2014), and 8.33% for testing.

Figure 6 plots the balanced accuracy (bAccuracy), balanced NLL (bNLL) and balanced BSE (bBSE) under temporal covariate shift (more results are presented in the appendix materials). We make the following observations. (i) Malware detectors encounter a significantly decreasing of accuracy and increasing of bNLL and bBSE with newer test data. This can be attributed to the natural software evolution that Google gradually updates Android APIs and practitioners upgrade their APKs to support new services. In particular, Droidetec suffer a lot from temporal covariate shift and exhibits a low detection accuracy. As mentioned earlier, this may be that Droidetec is permitted to learn from a limited number of APIs, which inhibits handling the APKs with a broad range of APIs. (ii) Temp scaling has the same effect as the vanilla model in terms of bAccuracy; Ensemble enhances the vanilla models (DeepDrebin, MultimodalNN, and DeepDroid) at the start of several months but then this enhancement diminishes; VBI makes MultimodalNN to achieve the highest robustness under data evolution (but only achieves ~80% bAccuracy). (iii) Ensemble methods benefit calibration in terms of bNLL and bBSE when compared with the vanilla model; VBI incorporating DeepDrebin or MultimodalNN achieves an impressive result.

Figure 7a plots the accuracy (at the upper half) and the balanced accuracy (at the lower half) after excluding the examples for which the detectors have entropy values greater than a threshold τ. Figure 7b plots the sample density of predictive entropy. We observe that (i) either accuracy or balanced accuracy decreases dramatically when the entropy increases, which is particularly true for DeepDroid and Droidetec. (ii) MultimodalNN incorporating VBI seems to work very well in terms of accuracy, but not necessary for balanced Accuracy. This is because the model classifies most benign samples correctly but not malicious ones until the threshold value is approach 0.9. Moreover, it is interesting to see that MultimodalNN incorporating VBI achieves the best bAccuracy without considering uncertainty (cf. Figure 6, which is in sharp contrast to Figure 7a). The reason behind this is that MultimodalNN incorporating VBI correctly classifies a portion of examples with high entropy value, as shown in Figure 7b. (iii) DeepDrebin incorporating VBI outperforms the other Drebin-based models, which resonates the results obtained in our second group of experiments (cf. Figure 4b in Section 4.3). (iv) Figure 7b says that for all of the malware detectors except MultimodalNN incorporating VBI and DeepDrebin incorporating VBI, most examples tend to have a small entropy. This suggests ineffective calibrations of malware detectors under temporal covariate shifts and a lack of good calibration method.

Insight 3. *Calibrated malware detectors cannot cope with temporal datashit shift effectively, but VBI is promising for calibration and generalization under temporal covariate shift.*

4.5 Answering RQ4

In order to quantify the predictive uncertainty of malware detectors under adversarial evasion attacks, we wage *transfer* attacks and generate adversarial APKs via a *surrogate* DeepDrebin model. We do not include adversarial APKs with respect to MultimodalNN, DeepDroid, and Droidetec because we do not find effective solutions. The surrogate DeepDrebin model consists of two fully-connected layers of 160 neurons with the ReLU activation function. We learn the model using the Adam optimizer with learning rate 0.001, batch size 128, and 150 epochs. We then generate adversarial examples against the surrogate model to perturb the 1,112 malicious APKs in the test dataset. Specifically, by following a recent study [27], we first perturb the feature vectors of the APKs using the *"max" PGDs+GDKDE* attack and the *Mimicry* attack, and then obtain adversarial APKs by using obfuscation techniques. In total, we obtain 1100 perturbed APK files for both attacks, respectively.

Table 3 summarizes the results of the malware detectors under the *"max" PGDs+GDKDE* attack and the Mimicry attack. Because the test dataset contains malware samples solely, we consider the Accuracy, NLL, BSE and ECE rather than their balanced versions. We observe that (i) the *"max" PGDs+GDKDE* attack renders DeepDrebin and MultimodalNN models useless, regardless of the calibration methods. Nevertheless, DeepDroid is robust against this attack because *opcode* are not used by the DeepDrebin and thus is unfocused by the attacker. However, DeepDroid still suffers somewhat from this attack because of the *opcode* manipulations is leveraged for preserving the malicious functionality. (ii) Under the Mimicry attack, VBI makes DeepDrebin and MultimodalNN achieve the best accuracy and the lowest calibration error (in terms of NLL and BSE), while weighted Ensemble makes both obtain the worst results. However, this situation is changed in regards to DeepDroid and Droidetec. This might be that DeepDrebin and MultimodalNN are more sensitive to Mimicry attack than DeepDroid and Droidetec, leading to that an ensemble of vulnerable models decreases the robustness against the attack.

Insight 4. *Adversarial evasion attacks can render calibrated malware detectors, and therefore the quantified predictive uncertainty, useless, but heterogeneous feature extraction does improve the robustness of malware detectors against the transfer attacks.*

5 CONCLUSION

We empirically quantified the predictive uncertainty of four deep malware detectors with six calibration strategies (i.e., 24 detectors in total). We found that the predictive uncertainty of calibrated malware detectors is useful except for adversarial examples. We hope this study will motivate and inspire more research in quantifying the uncertainty of malware detectors, which is of paramount importance in practice but it is currently little understood.

ACKNOWLEDGMENTS

Q. Li is supported in part by the National Key R&D Program of China under Grants 2020YFB1804604 and 2020YFB1804600, the 2020 Industrial Internet Innovation and Development Project from Ministry of Industry and Information Technology of China, the Fundamental Research Fund for the Central Universities under Grants 30918012204 and 30920041112, the 2019 Industrial Internet Innovation and Development Project from Ministry of Industry and Information Technology of China. S. Xu is supported in part by NSF Grants #2122631 (#1814825) and #2115134, ARO Grant #W911NF-17-1-0566, and Colorado State Bill 18-086.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, and et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[2] Martín Abadi, Paul Barham, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI' 16*. USENIX Association, Savannah, GA, USA, 265–283.
[3] Kevin Allix, Tegawendé F. Bissyandé, et al. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *International Conference on MSR* (Austin, Texas). ACM, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508
[4] Daniel Arp, Michael Spreitzenbarth, et al. 2014. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, Vol. 14. The Internet Society, San Diego, California, USA, 23–26.
[5] Umang Bhatt, Yunfeng Zhang, and et al. 2020. Uncertainty as a Form of Transparency: Measuring, Communicating, and Using Uncertainty. *CoRR* abs/2011.07586 (2020). https://arxiv.org/abs/2011.07586
[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight Uncertainty in Neural Network. In *Proceedings of the 32nd International Conference on Machine Learning*, Vol. 37. PMLR, Lille, France, 1613–1622. https://proceedings.mlr.press/v37/blundell15.html
[7] Glenn W Brier. 1950. Verification of forecasts expressed in terms of probability. *Monthly weather review* 78, 1 (1950), 1–3.
[8] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann. 2010. The Balanced Accuracy and Its Posterior Distribution. In *2010 20th International Conference on Pattern Recognition*. IEEE Computer Society, Istanbul, Turkey, 3121–3124.
[9] Joaquin Quiñonero Candela, Carl Edward Rasmussen, Fabian H. Sinz, Olivier Bousquet, and Bernhard Schölkopf. 2005. Evaluating Predictive Uncertainty Challenge. In *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop*, Vol. 3944. Springer, Southampton, UK, 1–27.
[10] Lingwei Chen, Yanfang Ye, and Thirimachos Bourlai. 2017. Adversarial Machine Learning in Malware Detection: Arms Race between Evasion Attack and Defense. In *EISIC'2017*. IEEE Computer Society, Athens, Greece, 99–106.
[11] Forensics Corvus. 2020. *VirusShare*. https://virusshare.com/
[12] Ambra Demontis, Marco Melis, et al. 2017. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE TDSC* 16, 4 (2017), 711–724.
[13] Anthony Desnos. 2020. *Androguard*. https://github.com/androguard/androguard
[14] Pang Du, Zheyuan Sun, Huashan Chen, Jin-Hee Cho, and Shouhuai Xu. 2018. Statistical Estimation of Malware Detection Metrics in the Absence of Ground Truth. *IEEE Trans. Inf. Forensics Secur.* 13, 12 (2018), 2965–2980.
[15] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. JMLR.org, NY, USA, 1050–1059.
[16] Alex Graves. 2011. Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011*. Curran Associates Inc., Granada, Spain, 2348–2356.
[17] Kathrin Grosse, Nicolas Papernot, et al. 2017. Adversarial examples for malware detection. In *ESORICS*. Springer, Oslo, Norway, 62–79.
[18] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, Sydney, NSW, Australia, 1321–1330.
[19] T. H. Huang and H. 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, Seattle, WA, USA, 2633–2642.
[20] Roberto Jordaney, Kumar Sharad, et al. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *USENIX Security 17*. USENIX Association, Vancouver, BC, 625–642. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney
[21] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision?. In *NeurIPS*. Curran Associates Inc., Long

Beach, CA, USA, 5574–5584.
[22] T. Kim, B. Kang, et al. 2019. A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. *IEEE Trans. Info. Forensics and Sec.* 14, 3 (2019), 773–788.
[23] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). ACL, Doha, Qatar, 1746–1751.
[24] Kaspersky Lab. 2020. *Kaspersky.* https://www.kaspersky.com
[25] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NeurIPS*. Curran Associates Inc., Long Beach, CA, USA, 6402–6413.
[26] Christian Leibig, Vaneeda Allken, Murat Seçkin Ayhan, Philipp Berens, and Siegfried Wahl. 2017. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports* 7, 1 (2017), 1–14.
[27] Deqiang Li and Qianmu Li. 2020. Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection. *IEEE Trans. Info. Forensics and Sec.* 15 (2020), 3886–3900.
[28] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review. *IEEE Trans. Knowl. Data Eng.* 31, 12 (2019), 2346–2363.
[29] Zhuo Ma, Haoran Ge, Zhuzhu Wang, Yang Liu, and Ximeng Liu. 2020. Droidetec: Android malware detection and malicious code localization through deep learning. *CoRR* abs/2002.03594 (2020). https://arxiv.org/abs/2002.03594
[30] Niall McLaughlin, Jesus Martinez del Rincon, et al. 2017. Deep Android Malware Detection. In *CODASPY '17* (Scottsdale, Arizona, USA). ACM, NY, USA, 301–308. https://doi.org/10.1145/3029806.3029823
[31] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. 2015. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, Austin, Texas, USA, 2901–2907.
[32] André T. Nguyen, Edward Raff, Charles Nicholas, and James Holt. 2021. Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints. *CoRR* abs/2108.04081 (2021). https://arxiv.org/abs/2108.04081
[33] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *ICML*. ACM, Bonn, Germany, 625–632.
[34] Tim Pearce, Felix Leibfried, et al. 2020. Uncertainty in Neural Networks: Approximately Bayesian Ensembling. In *AISTATS*. PMLR, Online [Palermo, Sicily, Italy], 234–244.
[35] Feargus Pendlebury, Fabio Pierazzi, et al. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *USENIX Security 19*. USENIX Association, Santa Clara, CA, 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury
[36] Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. 2016. A Survey on Systems Security Metrics. *ACM Comput. Surv.* 49, 4 (Dec. 2016), 1–35.
[37] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. 2009. *Dataset Shift in Machine Learning*. The MIT Press, Cambridge, MA.
[38] Hispasec Sistemas. 2020. *VirusTotal.* Alphabet, Inc. https://www.virustotal.com
[39] Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D Sculley, Joshua Dillon, Jie Ren, and Zachary Nado. 2019. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*. Curran Associates Inc., Vancouver, BC, Canada, 13969–13980.
[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.
[41] Trias Thireou and Martin Reczko. 2007. Bidirectional long short-term memory networks for predicting the subcellular localization of eukaryotic proteins. *IEEE/ACM transactions on computational biology and bioinformatics* 4, 3 (2007), 441–446.
[42] Connor Tumbleson and Ryszard Wiśniewski. 2020. *Apktool.* https://ibotpeaches.github.io/Apktool
[43] Juozas Vaicenavicius, David Widmann, Carl R. Andersson, Fredrik Lindsten, Jacob Roll, and Thomas B. Schön. 2019. Evaluating model calibration in classification. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS*, Vol. 89. PMLR, Naha, Okinawa, Japan, 3459–3467.
[44] Cheng Wang, Carolin Lawrence, and Mathias Niepert. 2021. Uncertainty Estimation and Calibration with Finite-State Probabilistic RNNs. In *9th International Conference on Learning Representations*. OpenReview.net, Virtual Event, Austria.
[45] Max Welling and Yee W Teh. 2011. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*. Omnipress, Madison, WI, USA, 681–688.
[46] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the Presence of Concept Drift and Hidden Contexts. *Mach. Learn.* 23, 1 (1996), 69–101.
[47] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. 2014. An evasion and counter-evasion study in malicious websites detection. In *IEEE Conference on Communications and Network Security (CNS'2014)*. IEEE, San Francisco, CA, USA, 265–273.
[48] Yanfang Ye, Tao Li, and et al. 2017. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* 50, 3 (2017), 41:1–41:40.
[49] Xiaohan Zhang, Yuan Zhang, and et al. 2020. Enhancing State-of-the-Art Classifiers with API Semantics to Detect Evolved Android Malware. In *CCS 2020* (Virtual Event, USA). Association for Computing Machinery, New York, USA, 757–770.
[50] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. 2016. *An Overview of Concept Drift Applications*. Springer International Publishing, Cham, 91–114.

A EXPERIMENTAL RESULTS ON THE VIRUSSHARE DATASET

Figure 8 plots the balanced accuracy on the VirusShare dataset with decision referral. We observe that Figure 8 exhibit the trends that are similar to what are exhibited by Figure 4b, except for Temp scaling on DeepDrebin and MultimodalNN. This is because the model predicts benign examples accurately, but do not predict malicious examples accurately.

B EXPERIMENTAL RESULTS ON THE ANDROOZOO DATASET

Figure 9 plots the Accuracy, NLL and BSE of malware detectors under temporal covariate shifts. We observe that the Accuracy, NLL, and BSE are smaller than their balanced counterparts plotted in Figure 6, owing to the data imbalance exhibited by the Androzoo dataset, despite that they all exhibit a similar trend.

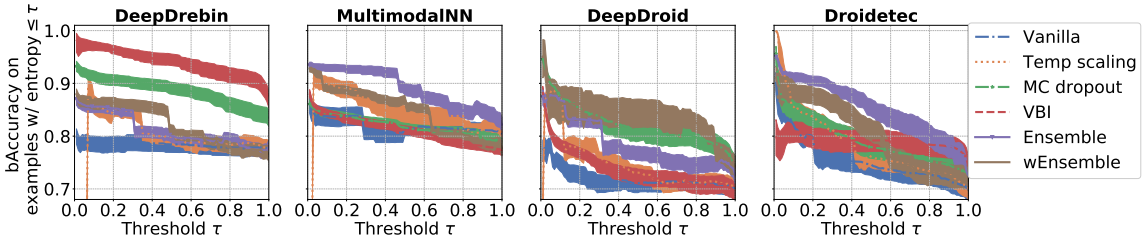


Figure 8: The balanced accuracy on the VirusShare dataset after excluding the examples for which the detector has high uncertainties (i.e., the examples for which the predictive entropy is above a pre-determined threshold τ). For each curve, a shadow region is obtained from the 95% confidence interval using a bootstrapping with 10^3 repetitions and sampling size being the number of examples in the VirusShare dataset.

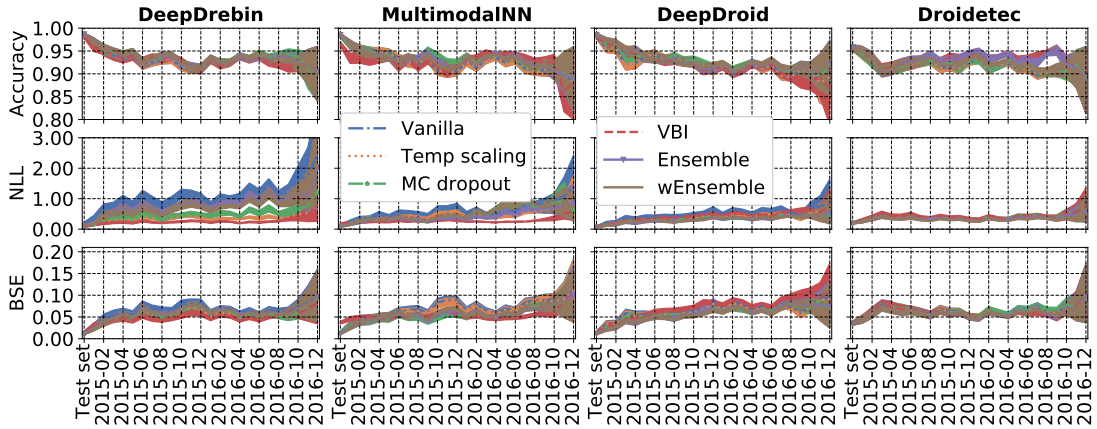


Figure 9: Illustration of accuracy, NLL, BSE under temporal covariate shift on the Androzoo dataset. A shadow region is obtained from the 95% confidence interval using a bootstrapping with 10^3 repetitions and sampling size being the number of APKs in per month (cf. Figure 2).