Transcending TRANSCEND: Revisiting Malware Classification with Conformal Evaluation

Federico Barbero[†], Feargus Pendlebury[†‡§], Fabio Pierazzi[†], Lorenzo Cavallaro[†]

[†] *King's College London*
[‡] *Royal Holloway, University of London*
[§] *The Alan Turing Institute*

Abstract

Machine learning for malware classification shows encouraging results, but real deployments suffer from performance degradation as malware authors adapt their techniques to evade detection. This phenomenon, known as *concept drift*, occurs as new malware examples evolve and become less and less like the original training examples. One promising method to cope with concept drift is *classification with rejection* in which examples that are likely to be misclassified are instead quarantined until they can be expertly analyzed.

We revisit TRANSCEND, a recently proposed framework for performing rejection based on conformal prediction theory. In particular, we provide a formal treatment of TRANSCEND, enabling us to refine *conformal evaluation theory*—its underlying statistical engine—and gain a better understanding of the theoretical reasons for its effectiveness. In the process, we develop two additional conformal evaluators that match or surpass the performance of the original while significantly decreasing the computational overhead. We evaluate our extension on a large dataset that removes sources of experimental bias present in the original evaluation.

Finally, to aid practitioners, we determine the optimal operational settings for a TRANSCEND deployment and show how it can be applied to many popular learning algorithms. These insights support both old and new empirical findings, making TRANSCEND a sound and practical solution for the first time. To this end, we release our implementation of TRANSCEND as open source, to aid the adoption of rejection strategies by the security community.

1 Introduction

Machine learning (ML) algorithms have displayed superhuman performance across a wide range of classification tasks such as computer vision [18] and natural language processing [14]. However, a great deal of this success is conditional on one central assumption: that the training and testing data are drawn identically and independently from the same underlying distribution (i.i.d.) [9].

In a security setting this assumption often does not hold. In particular, malware classifiers are deployed in dynamic, hostile environments where new paradigms of malware evolve to chase profits, new variants arise as novel exploits are discovered, and adversaries switch behavior suddenly and dramatically when faced with strengthened defenses. This causes the incoming test distribution to diverge from the original training distribution, a phenomenon known as *concept drift* [21]. As time goes by, the performance of the classifier begins to degrade as the model fails to classify the new objects correctly.

There appear to be two broad approaches to tackling concept drift: the first is to design systems which are intrinsically more resilient to drift by developing more robust feature spaces. For example, it has recently been suggested that neural networks may be more resilient to concept drift as the latent feature space better generalizes to new variants [28]. However, designing robust feature spaces is an open research question and it is not clear if there exists such a malware representation for which concept drift will not occur.

A second solution is to *adapt* to the drift, for example by updating the model using incremental retraining or online learning [22, 40]. However, in order to be effective, decisions about when and how to take action on aging classifiers must be taken quickly and decisively. To do so, accurate detection and quantification of drift is vital.

This problem is precisely the focus of TRANSCEND [16], a statistical framework that builds on conformal prediction theory [37] to detect aging malware detectors during deployment—before their accuracy deteriorates to unacceptable levels. TRANSCEND [16] proposes a *conformal evaluator* that utilizes the notion of *non-conformity* to identify and reject new examples that differ from the training distribution and are likely to be misclassified; the corresponding apps can then be quarantined for further analysis and labelling. While effective, the original proposal suffers from experimental bias, is extremely resource intensive and thus impractical, fails to provide guidance on how to integrate it into a detection pipeline and, perhaps more importantly, lacks a theoretical analysis that explains its effectiveness.

In this paper we revisit conformal evaluator and TRAN-SCEND [16] to root its internal workings in sound theory and determine its most effective operational settings. We additionally propose a novel extension that matches or surpasses the performance of the original while significantly decreasing the computational overhead.

In summary, we make the following contributions:

- **Formal Treatment.** We investigate the theory underpinning the motivation and intuition of conformal evaluation to explain its effectiveness and support the empirical evaluations presented in both this work and the original (§3). Building on this insight, we formalize TRANSCEND's calibration procedure as an optimization problem (§4).

- **Novel Conformal Evaluators.** We develop two novel conformal evaluators: *inductive conformal evaluator* (ICE) (§5.2) and *cross-conformal evaluator* (CCE) (§5.3), both of which are firmly grounded in conformal prediction theory and able to effectively identify and reject drifting examples while being significantly less computationally demanding than the original. We evaluate our proposals with a dataset spanning 5 years (2014–2018) containing ~10% malware that eliminates sources of bias present in past evaluations (§6).

- **Operational Guidance.** We compare different operational settings of TRANSCEND including the effects of including algorithm *confidence* (§6.3) and of different search strategies (§6.4) during thresholding. To aid practitioners in adopting TRANSCEND as a rejection framework, we include a discussion of how to integrate TRANSCEND into a typical security detection pipeline (§7).

To enable researchers and practitioners to make better use of classification with rejection strategies, we publicly release our implementation of TRANSCEND, along with our improvements and the data used for our empirical evaluations (§9).

2 Background

We focus on classification for security tasks (§2.1) which are affected by concept drift (§2.2). In particular, we are interested in improving the state-of-the-art approaches for classification with rejection (§2.3).

2.1 Machine Learning and Security Detection

Machine learning is a set of statistical methods for automating data analysis and enabling systems to perform tasks on the data without being explicitly programmed for them. In the malware domain, typical tasks include binary classification (detecting malicious examples [4, 40]) and multiclass classification (predicting the malware family [13, 33, 34]) but can also extend to more complex tasks such as predicting the

number of AV engines that would detect an example [17], inferring Android malware app permissions based on their icons [39], or generating functional Windows malware using reinforcement learning [3].

In this paper we focus on classification tasks where a classifier g aims to learn a function mapping $\mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \subseteq \mathbb{R}^n$ is a feature space of vectors capturing the interesting properties of the applications and $\mathcal{Y}$ is a label space containing binary labels in the detection task or malware family names in the multiclass classification task.

2.2 Concept Drift

One of the greatest challenges facing machine learning-based malware classifiers is the presence of *dataset shift* [1, 16, 20] as the distribution of malware at test time begins to diverge from the training distribution. This violates one of the core assumptions of most classification algorithms: that the training and test time are identically and independently drawn from the same joint distribution (i.i.d.). As this assumption weakens over time, the classifier's predictions become less and less reliable and performance degrades.

Dataset shift can be broadly categorised into three types of shift [21]. *Covariate shift* refers to a change in the distribution of $P(\boldsymbol{x} \in \mathcal{X})$, when the frequency of certain features rises or falls (e.g., variations in API call frequencies over time). *Prior probability shift* or *label shift* is a change in the distribution of $P(y \in \mathcal{Y})$, when the base rate of a particular class is altered (e.g., an increase in malware prevalence over time). *Concept drift* is a change in the distribution $P(y \in \mathcal{Y} | \boldsymbol{x} \in \mathcal{X})$. This often occurs when the definition of the ground truth changes, for example, if a new family of malware arises which, given the feature space representation $\mathcal{X}$, is indistinguishable from benign applications. Due to the model's limited knowledge, the model will start misclassifying examples from the new family, even if no covariate or prior probability shift is occurring. In practice, it can be extremely difficult to determine how much error should be attributed to each type of shift [21]. Given this, it is common in the security community to collectively refer to all types of shift as *concept drift*, a custom that we continue in this work.

The impetus for concept drift in malware classification is the adversarial nature of the task. Malware authors are driven by the profit motive to try and evade detection or classification by app store owners, antivirus companies, and other security agents. This incentivizes them to innovate: to obfuscate features of their malware, develop new methods for exploitation and persistence, and explore new avenues of profiteering and abuse. This causes the definition of malware to evolve over time, sometimes in drastic or unexpected ways.
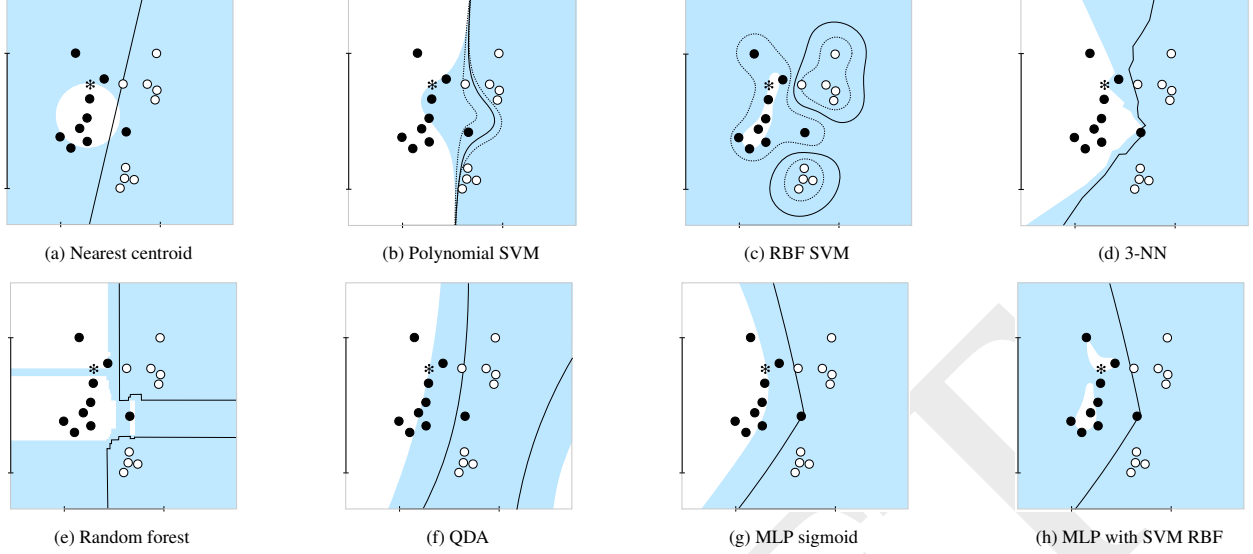
Figure 1: Possible NCMs for different classification algorithms: nearest centroid, support-vector machines (SVMs), nearest neighbors (NN), random forest, quadratic discriminant analysis (QDA), and multilayer perceptron (MLP). The solid line delineates the decision boundary between classes ● and ○ while the dotted lines show SVM margins. The shaded region captures points which are *more nonconform* (i.e., 'less similar') than the new testing point, shown by the asterisk, with respect to class ●. As NCMs, (a) uses the distance from the class centroid; (b) and (c) use the negated absolute distance from the hyperplane; (d) uses the proportion of nearest neighbors belonging to class ○; (e) uses the proportion of decision trees that predict ○; (f) uses the negated probability of belonging to class ●; (g) uses the negated probability output by the final sigmoid activation layer; (h) uses the outputs of the final hidden layer to train an SVM with RBF kernel and uses the negated absolute probabilities output by that SVM—note that the decision boundary still depends on the MLP output alone.

2.3 Rejection

There are multiple routes to dealing with concept drift. The most effective would be to design a feature space X such that it is entirely robust to concept drift, essentially distilling all possible malware behaviour down to a 'Platonic ideal' [29] that captures maliciousness no matter what form it takes. While recent proposals for augmenting feature spaces with robust features are promising (e.g., [35]), the diversity of malware makes it extremely difficult to design such a feature space. Additionally, some behaviour is only considered malicious due to its context, for example, requesting access to device contacts might be considered suspicious for a torch app but not for a social messaging app [41].

An orthogonal approach is to identify, track, and mitigate the drift as it occurs. One promising method is classification with rejection [6], in which low confidence predictions, caused by drifting examples, are *rejected*. Drifting apps can then be quarantined and dealt with separately, either warranting manual inspection or remediation through other means.

TRANSCEND [16] is a state-of-the-art framework for performing classification with rejection in security tasks. It uses a *conformal evaluator* to generate a quality measure to assess whether a new test example is drifting with respect to the training data. If the prediction of an underlying classifier appears to be affected by the drift, the prediction is rejected. The original proposal presented two case studies: Android malware detection—a binary classification task, and Windows malware family classification—a multiclass classification task. The ex-

periments showed that the framework is consistently able to identify drifting examples, providing a significant improvement over thresholding on the classifiers' output probabilities. However, the computational complexity of the framework limited its use in real-world deployments.

3 Conformal Evaluator

The statistical engine that drives TRANSCEND's rejection mechanism is the *conformal evaluator*, a tool for measuring the quality of predictions output by an underlying classifier. Conformal evaluator design is grounded in the theory of *conformal prediction* [37], a method for providing predictions that are correct with some guaranteed confidence. In this section we investigate the relationship between the two to provide novel insights and intuition into why conformal evaluation is effective in the classification with rejection setting.

3.1 Conformal Evaluation vs. Prediction

Here we give an overview of conformal prediction and how it motivates the use of conformal evaluation; for a more formal treatment we refer to Vovk et al. [37]. Conformal prediction allows for predictions to be made with precise levels of confidence by using past experience to account for uncertainty. Given a classifier g, a new example $z = (x, y)$, and a significance level ε, a conformal predictor produces a *prediction region*: a set of labels in the label space $\mathcal{Y}$ that is guaranteed

Figure 2: The nested intervals at which labels ● and ○ are present in the output label set for a testing example with per-class p-values $p_\bullet = 0.32$ and $p_\circ = 0.08$. The shaded areas outline how credibility and confidence relate to the intersection of prediction regions for which the label set contains a single element. The relatively high probability of the empty set containing the correct label (i.e., low credibility) indicates that one of conformal prediction's assumptions may have been violated. In conformal evaluation, this is used as a signal that the new example is likely to be out-of-distribution and is indicative of concept drift.

to contain the correct label y with probability no more than $1 - \varepsilon$. To calculate this label set, the conformal predictor relies on a *nonconformity measure* (NCM) derived from g and uses it to generate scores representing how *dissimilar* each example is from previous examples of each class. To quantify this relative dissimilarity, *p-values* are calculated by comparing the nonconformity scores between examples (§3.2). As well as these p-values, two important metrics are derived from the prediction region, confidence and credibility (§3.3), which can be used to judge the effectiveness of the conformal prediction framework. Conformal predictors are able to make strong guarantees on the correctness of each prediction so long as two assumptions about new test examples hold: the *exchangeability* assumption, that the sequence of examples is exchangeable, a generalization of the i.i.d. property; and the *closed-world* assumption, that new examples belong to one of the classes observed during training.

Rather than making predictions, conformal evaluators [16] borrow the same statistical tools (i.e., nonconformity measures and p-values) but use them to *evaluate* the quality of the prediction made by the underlying classifier g. By detecting instances which appear to violate the aforementioned assumptions they can, with high confidence, *reject* new drifting examples which would otherwise be at risk of being misclassified.

3.2   Nonconformity Measures and P-values

In order to reject a new example that cannot be reliably classified, conformal evaluators rely on a notion of *nonconformity* to quantify how dissimilar the new example is to a history of past examples. In general, a *nonconformity measure* (NCM) [30] is a real-valued function that outputs a score describing how different an example z is from a bag of previous examples $B = \{z_1, z_2, ..., z_n\}$:

$$\alpha_z = A(B, z). \tag{1}$$

The greater the value of α_z, the less similar z is to the elements of the bag B. An NCM is typically formed of two components: a metric $d(z, z')$ to measure the distance between two points, and a *point predictor* $\hat{z}(B)$ to represent B:

$$A(B, z) := d(\hat{z}(B), z). \tag{2}$$

Illustrating this, Figure 1a shows an NCM for a nearest centroid classifier in which the Euclidean distance is used for $d(z, z')$, and the nearest class centroid is used for $\hat{z}(B)$.

For a new example z^*, the conformal evaluator must decide whether or not to approve the null hypothesis asserting that z^* *does not belong* in the prediction region formed by elements of B. To perform such a hypothesis test, *p-values* are calculated using the NCM values for each point. First the nonconformity score of z^* must be computed (Equation 3) along with nonconformity scores of elements in B (Equation 4), then the the p-value p_{z^*} for z^* is given as the proportion of points with greater or equal nonconformity scores (Equation 5):

$$\alpha_{z^*} = A(B, z^*) \tag{3}$$
$$S = \{A(B \setminus \{z\}, z) : z \in B\} \tag{4}$$
$$p_{z^*} = \frac{|\alpha \in S : \alpha >= \alpha_{z^*}|}{|S|} \tag{5}$$

In the classification context, TRANSCEND [16] can calculate p-values in a *label conditional* manner, such that B contains only previous examples of class $\hat{y} \in Y$ where $\hat{y} = g(z^*)$ is the predicted class of the new example. If p_{z^*} falls below a given significance level the null hypothesis is disproved and $\hat{y}$ is accepted as a valid prediction. TRANSCEND [16] computes *per-class thresholds* to use as significance levels (§4).

As p-values are calculated by considering nonconformity scores relative to one another, NCMs can be transformed monotonically without any impact on the resulting p-values at all. Thus, when designing an NCM in the form given by Equation 2, the distance metric $d(z, z')$ is of significantly less important than the point predictor $\hat{z}(B)$. It is important to note that conformal evaluator algorithms are agnostic to the underlying NCM chosen, but the quality of the NCM—and particularly of $\hat{z}(B)$, can have a large effect on the ability of conformal evaluators to discriminate between valid and invalid predictions [30].

An *alpha assessment* [16] can be used to empirically evaluate the appropriateness of an NCM for a given dataset. An alpha assessment plots the distribution of p-values for each class, further split into whether the prediction was correct or incorrect. As incorrect predictions should be rejected, they are expected to fall below the threshold, while correct predictions are expected to fall above the threshold. Well-separated

correct and incorrect predictions suggest a viable threshold exists to separate them at test time. Poorly separated distributions of correct and incorrect prediction p-values indicate an inappropriate NCM. An example of an alpha assessment on a toy dataset is shown in Figure 3d.

Figure 1 illustrates possible nonconformity measures for different algorithms on a toy binary classification task with existing class examples ●/○ and new testing example ✳. The solid line delineates the decision boundary between the two classes, the dotted lines show SVM margins where relevant, and the blue shaded region captures points that are *more nonconform* (i.e., less similar) than ✳ with respect to class ●. Note that the shape of the nonconformal region need not reflect the shape of the regions for the predicted classes (e.g., Figure 1a) and that there may be multiple viable NCMs for the same underlying algorithm (e.g., Figures 1g and 1h).

3.3 Credibility and Confidence

Recall that conformal prediction produces a prediction region given a significance level ε. The possible prediction regions are nested such that the higher the confidence level, the more labels will be present. As a trivial example, a prediction region containing all possible labels may be produced for a significance level of ε = 0 (maximum likelihood) as it will contain the true label *y* with certainty. At the other extreme, an empty set can be produced at a significance level of ε = 1 (minimum likelihood), as this is an impossible result under the closed-world assumption of conformal prediction.

Of particular interest is the prediction region containing a single element which lies between these extremes. Related to this prediction region, a conformal predictor also outputs two metrics: *confidence* and *credibility*.

Confidence is the greatest 1 − ε for which the prediction region contains a single label which can be calculated as the complement to 1 of the second highest computed p-value. Confidence quantifies the likelihood that the new element belongs to the predicted class.

Credibility is the greatest ε for which the prediction region is empty and corresponds to the largest computed p-value. Conformal predictors can be forced to output single predictions (rather than a label set induced by ε), in which case they will output the class with the highest credibility. Credibility quantifies how relevant the training set is to the prediction. A low credibility indicates that conformal prediction might not be a suitable framework to use with the given data. This is because a low credibility means that the probability of the correct label being in the empty set is relatively high, which is an impossible result under the closed-world assumption of conformal prediction.

We propose that conformal evaluation's effectiveness stems from this relationship: that in conformal *evaluation*, this probability is being directly interpreted as the probability that the i.i.d. assumption has been violated. Thus, a low credibility

means that there is a high probability that the corresponding example is *drifting* with respect to the previous history of training examples. Such an example is at risk of being misclassified due to limited knowledge of the classifier.

It should be noted that formally, conformal evaluation defines credibility and confidence slightly differently. In conformal evaluation, the credibility is the p-value corresponding to the predicted class and the confidence is the complement to 1 of the maximum p-value excluding the p-value corresponding to the predicted class (i.e., the credibility p-value). This subtle difference is important to clarify the operational context of a conformal evaluator: whereas conformal predictors output the final classification decision, conformal evaluators output a statistical measure *separate* to the decision of the underlying classifier (hence the nomenclature: one predicts and the other evaluates). In practice, given reasonable NCMs, these definitions can be treated as equivalent.

3.4 Transductive Conformal Evaluator (TCE)

In assessing the quality of a prediction for a new test point, there is still the question of which previously encountered points the new point should be compared to—that is, which elements are included in the bag *B* of Equation 3, and how. Typically, new test points are compared against a set of calibration points. In practice, there are a number of possibilities for deciding on this calibration set, each with their own advantages and disadvantages.

In Jordaney et al. [16], conformal evaluation was realized using a Transductive Conformal Evaluator (TCE). With a TCE, every training point is also used as a calibration point. To generate the p-value of a calibration point, it's first removed from the set of training points and the underlying classifier trained on the remaining points. Given the newly trained classifier, a predicted label is generated for the calibration point. Finally, using a given NCM, its p-value is computed with respect to the points whose ground truth label matches its predicted label. This procedure is repeated for every training point. Following this, TRANSCEND's thresholding mechanism operates on the calculated p-values to determine per-class rejection thresholds (§4). At test time, the underlying classifier is retrained on the entire training set, and, similarly to the calibration points, the p-values are computed with respect to the p-values of the calibration sets.

Algorithm 2 describes the calibration and testing process using TCEs in a general form that also applies to our novel approximate variant (see §5.1)

4 The TRANSCEND Framework

TRANSCEND [16] is a framework that uses similarity scores generated by a conformal evaluator to derive thresholds with which to accept or reject new testing examples.

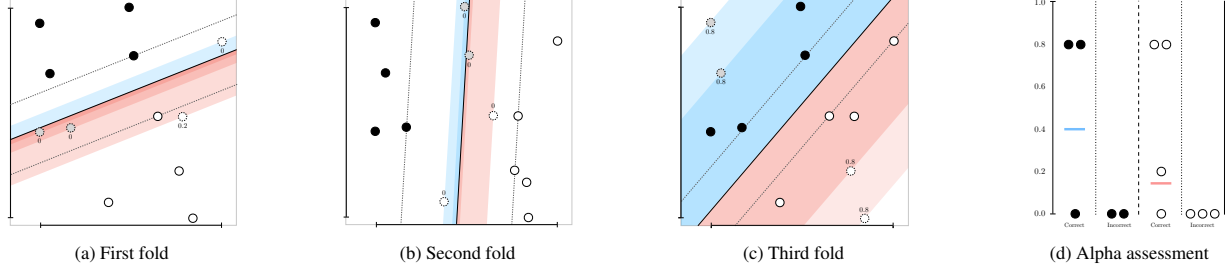(a) First fold   (b) Second fold   (c) Third fold   (d) Alpha assessment

Figure 3: Thresholding procedure applied to a linear SVM with approximate-TCE (3 folds). Four points highlighted with dotted outlines are left out as calibration in each fold, with the decision boundary obtained with the remaining points as training. P-values, shown above or below each calibration point, are calculated using the negated absolute distance from the decision boundary as an NCM. The shaded regions capture points which are *more nonconform* with respect to the predicted class (blue for class ● and red for class ○). The alpha assessment (d) shows the distribution of p-values and per-class thresholds derived from Q1 of the correctly classified points (see §5.4 for a discussion of more complex search strategies for finding thresholds).
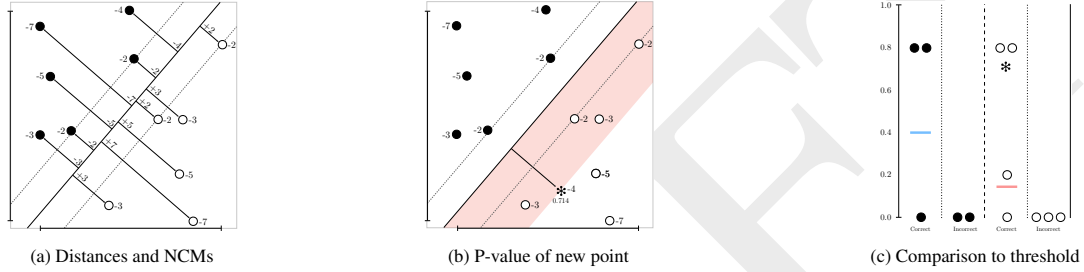

(a) Distances and NCMs   (b) P-value of new point   (c) Comparison to threshold

Figure 4: Test-time procedure applied to a linear SVM and calibrated TRANSCEND with distances from hyperplane and corresponding noncomformity scores shown in (a). In (b) a new test point is classified as class ○. The p-value is calculated as the proportion of points belonging to ○ with equal or greater nonconformity scores (captured by the shaded region) than the new point. In (c), the new point is compared against the threshold for class ○ as derived during the calibration phase (Figure 3). As the p-value of the new point is greater than the threshold for the predicted class, the prediction is accepted.

4.1 Calibration Phase

The first phase of TRANSCEND is the calibration procedure which searches for a set of per-class *credibility* thresholds $\mathcal{T} = \{ \tau_y \in [0,1] : y \in \mathcal{Y} \}$ with which to separate drifting from non-drifting points. Given that low credibility represents a violation of conformal prediction's assumptions, these points are likely to be misclassified by the underlying classifier that similarly relies on the i.i.d. assumption. Note that thresholds can be found with different optimization criteria and it is also possible to threshold on a combination of credibility and *confidence*, which we explore in §6.3.

Calibration aims to answer the question: "how low a credibility is too low?", by analyzing the p-value distribution of points in a representative, preferably stationary, environment such as the training set. Exactly which points are selected as calibration points depends on the underlying conformal evaluator, and this comes with various trade-offs (see §5). Typically, each calibration point (or partition of the calibration set) is held out and the underlying classifier trained on the remaining points. Then a class is predicted for the calibration point(s) with p-values calculated with respect to that predicted class. This process is repeated until all calibration points are assigned a corresponding p-value. Using the ground truth, these p-values can be divided into *correct* and *incorrect* predictions. Finally, $\mathcal{T}$ can be chosen to effectively separate the

correct and incorrect predictions, either using manual methods (e.g., picking a quartile visually using an alpha assessment) or automated search strategies (e.g., grid search).

Figure 3 shows an example of the TRANSCEND [16] thresholding procedure on a toy dataset composed of two classes: ● and ○. A linear SVM is paired with a TCE (§3.4) to generate NCMs and p-values for the binary classification with rejection task. The decision boundary is depicted as a solid line and margins are drawn through support vectors with dotted lines. Due to the use of approximate TCE, the dataset is partitioned into folds, where each fold leaves out four points for calibration and trains on the remainder. The three folds are depicted in Figures 3a, 3b, and 3c. Calibration points are shown with dotted outlines and are faded for class ●.

In each fold, a p-value is calculated for each calibration point as the proportion of other objects that are *at least as dissimilar* to the predicted class as the calibration point itself. In the linear SVM setting shown, less similar objects are those closest to the decision boundary (i.e., those with a higher NCM) residing in the shaded area between the decision boundary and the parallel line intersecting the point (blue for class ● and red for class ○). The calculated p-values are shown aligned above or below each calibration point.

To evaluate how appropriate an NCM is for a given model, the p-values can be analyzed with an *alpha assessment*. Here the distribution of p-values for each class are divided into

groups depending on whether the calibration point was correctly or incorrectly predicted as that class. Given that there may not enough incorrectly classified examples to perform the assessment with, it is standard to perform an alpha assessment in a *non-label-conditional* manner, using p-values computed with respect to all classes, not just each point's predicted class. The greater the margin separating the distributions of correct and incorrect p-values, the better suited an NCM is for a model. The alpha assessment in Figure 3d shows the distribution of p-values for correct and incorrectly predicted calibration points for classes ● and ○. Note that given the size of the example dataset, the assessment is computed in a *label-conditional* manner. A threshold is chosen for each class, in Figure 3d this is simply set at Q1 of the p-values for correctly classified points (more insight into threshold search strategies can be found in §5.4). Test points generating p-values that fall below this threshold will be rejected.

4.2 Test Phase

At test time, there are $|\mathcal{Y}| + 1$ outcomes. When a new testing object z^* arrives, its p-value $p_{z^*}^{\hat{y}}$ is calculated with respect to the predicted class $\hat{y}$ (label conditional). If $p_{z^*}^{\hat{y}} < \tau_{\hat{y}}$, the threshold for the predicted class, then the null hypothesis—that z^* is drifting relative to the training data and does not belong to $\hat{y}$—is approved and the prediction rejected. If $p_{z^*}^{\hat{y}} \geq \tau_{\hat{y}}$, the prediction is accepted and the object classified as $\hat{y}$.

Figure 4 follows on from the calibration example above. Figure 4a illustrates the NCM being used: the negated absolute distance from the hyperplane. In Figure 4b, a new test example ✳ appears and is classified as class ○. The p-value $p_{✳}^{○} = 0.714$ is calculated as the proportion of points belonging to ○ with equal or greater nonconformity scores than ✳. Finally, Figure 4c shows $p_{✳}^{○}$ compared against the threshold $\tau_{○}$ and, as $p_{✳}^{○} \geq \tau_{○}$, the prediction is accepted.

4.3 Rejection Cost

What happens to rejected points depends on the rest of the detection pipeline. In a simple setting, rejected points may be manually inspected and labelled by specialists. Alternatively, they may continue downstream to further automated analyses or to other ML algorithms such as unsupervised systems.

In all cases there will be some cost associated with rejecting classifier decisions. As a result, when choosing rejection thresholds, it's vital to keep this cost in mind and weight it against the potential performance gains. The TESSERACT framework [28] defines three important metrics to use when tuning or evaluating a system for mitigating time decay.

Performance ensures that robustness against concept drift is measured appropriately depending on the end goal (e.g., high F_1 score or high TPR at an acceptable FPR threshold).

Quarantine cost is a measure of the cost incurred by rejecting points. This cost is important for putting the perfor-

mance of kept elements into perspective and there will often be a trade-off between rejecting a large amount of points and higher performance on kept points.

Labelling cost is a measure of the manual effort needed to find ground truth labels for new points. While this is more pertinent to retraining strategies, it's heavily related to the overhead associated with rejection as many rejected points may need to be manually labelled. As an example, Miller et al. [20] estimated that an average company would be able to manually inspect and label 80 samples per day. It may not be feasible for such a company to employ rejection thresholds that reject a greater amount.

4.4 Formalizing the Threshold Search

In order to ensure quality thresholds are found, TRANSCEND's calibration procedure can be modeled as an optimization problem in which the aim is to maximize a given performance metric (e.g., F_1, Precision, or Recall of kept elements). Usually this maximization is subject to some acceptable performance in another metric. For example, it is trivial to attain high F_1 performance in kept elements by accepting only a few high quality predictions. However, this will involve rejecting unacceptably high number of objects, incurring high costs (quarantine, manual inspection, relabelling, etc).

Formally, given n calibration points, we represent this as:

$$\underset{\mathcal{T}}{\arg\max} \quad \mathcal{F}(Y, \hat{Y}, P; \mathcal{T})$$
$$\text{subject to} \quad \mathcal{G}(Y, \hat{Y}, P; \mathcal{T}) \geq \mathcal{C}, \tag{6}$$

where Y and $\hat{Y}$ are n-dimensional vectors of ground truth and predicted labels respectively, P is a $|\mathcal{Y}| \times n$-dimensional matrix of calibration p-values and $\mathcal{T} = \{\tau_y \in [0,1] \mid y \in \mathcal{Y}\}$ is the set of thresholds. The objective function $\mathcal{F}$ maps these inputs to the metric of interest in $\mathbb{R}$, for example F_1 of kept elements, while $\mathcal{G}$ maps to the metric to be constrained, such as the number of per-class rejected elements. $\mathcal{C}$ is the threshold value that bounds the constraint function. In §5 we propose an implementation using random search.

5 Improving Conformal Evaluation

While the Transductive Conformal Evaluator (TCE) used in the original proposal [16] appears to perform well, the complexity of the TCE is such that it becomes increasingly difficult to apply to larger datasets. For instance, it is not computationally feasible to apply vanilla TCE during the experiments in §6 where fitting a single instance of the underlying classifier takes 10 CPU minutes. In this case, we estimate a single run using vanilla TCE would take 1.9 CPU years.

Here we propose a number of novel conformal evaluators that overcome this limitation and present their advantages and disadvantages. A comparison of their runtime complexities is presented in Table 1.

5.1 Approximate Transductive Conformal Evaluator (Approx-TCE)

Our first attempt at reducing the computational overhead induced by the Transductive Conformal Evaluator is the *approximate Transductive Conformal Evaluator* (approx-TCE). In the original TCE, p-values are generated for each calibration point by removing them from the training set, retraining the underlying classifier on the remaining points, and repeating until a p-value is computed for every training point.

In approx-TCE, calibration points are left out in batches, rather than individually. The training set is randomly partitioned into k folds of equal size. From the k folds, one fold is used as the target of the calibration and the remaining $k-1$ folds are used as the bag to which those points are compared to. This process repeats k times, until each fold has been used as the calibration set exactly once. Note that all of the k calibration sets are mutually exclusive; the corresponding batches of p-values are then concatenated in the same manner as in TCE. Algorithm 2 describes this process formally, while Figure 5 depicts the graphical intuition.

The statistical soundness of the approx-TCE relies on the assumption that the decision boundary obtained from leaving out calibration points in batches approximates each of the decision boundaries that would have been obtained per calibration point in the batch if the point had been left out individually. If this assumption holds, the generated p-values will be the same as, or similar to, the p-values generated with a TCE. The approximation grows more accurate as k increases until k equals the cardinality of the training set at which point the approx-TCE and the TCE are equivalent. In this sense, the approx-TCE can be viewed as a generalization of the TCE.

While this assumption is more likely to hold with algorithms with lower variance (e.g., linear models), it becomes less and less likely to hold as the variance increases unless k is increased also—sacrificing the computation saved to mitigate the statistical instability.

5.2 Inductive Conformal Evaluator (ICE)

The second conformal evaluator we propose is the *Inductive Conformal Evaluator* (ICE) which, unlike the approx-TCE, is based on a corresponding approach from conformal prediction theory [25, 36, 37]. The ICE directly splits the training set into two non-empty partitions: the *proper training set* and the *calibration set*. The underlying algorithm is trained on the proper training set, and p-values are computed for each example in the calibration set. In contrast to the TCE, p-values are not calculated for every training point, but only for examples in the calibration set, with the proper training set having no role in the calibration at all. The ICE aims to inductively learn a *general rule* on a single fold of the training set. Algorithm 3 describes this process formally, while Figure 5 depicts the graphical intuition.

Table 1: Runtime complexities for conformal evaluators calibration where n is the number of training examples and p is the proportion of examples included in the *proper training set* each split/fold.

CONFORMAL EVALUATOR	COMPLEXITY
TCE	$O(n^2)$
Approx-TCE, $1/(1-p)$ folds	$O(n/(1-p))$
ICE	$O(pn)$
CCE, $1/(1-p)$ folds	$O(pn/(1-p))$

This induces significantly less computational overhead than the TCE and approx-TCE (see Table 1) and in practice is extremely fast, but it is also very *informationally inefficient*. Only a small proportion of the training data is used to calibrate the conformal evaluator, when ideally we would use all of it. Additionally, the performance of the evaluator depends heavily on the quality of the split and the calibration set's ability to generalize to the remainder of the dataset. This results in some uncertainty: an ICE may perform worse than its TCE counterpart due to a lack of information, or better due to a lucky split.

5.3 Cross-Conformal Evaluator (CCE)

The *Cross-Conformal Evaluator* (CCE) draws on inspiration from k-fold cross validation and aims to reduce both the computational and informational inefficiencies of the TCE and ICE. Like the ICE, the CCE has a counterpart rooted in conformal prediction theory [38].

The training set is partitioned into k folds of equal size. In order to make use of the entire training set, each fold is treated as the calibration set in turn, with p-values calculated as with an ICE, using the union of the $k-1$ remaining folds as the proper training set to fit the underlying classifier.

The question is how to concatenate the p-values in a way which preserves their statistical integrity when evaluating decision quality. The k fit underlying models and corresponding calibration sets should be set aside for test time. When a new point arrives, the prediction from each classifier should be evaluated against the corresponding calibration set. The final result is the majority vote over the k folds, i.e., the prediction of a particular class is accepted if the number of accepted classifications is greater than $\frac{k}{2}$, and rejected otherwise. For this reason it is beneficial to choose an odd k, to guarantee that there will always be a majority. Algorithm 4 describes the calibration process formally, while Figure 5 depicts the graphical intuition.

The CCE can be viewed as k ICEs, one per fold, and these separate ICEs can be operated in parallel to reduce computation time—if the resources are available. However, there is an additional memory cost with storing the separate models.
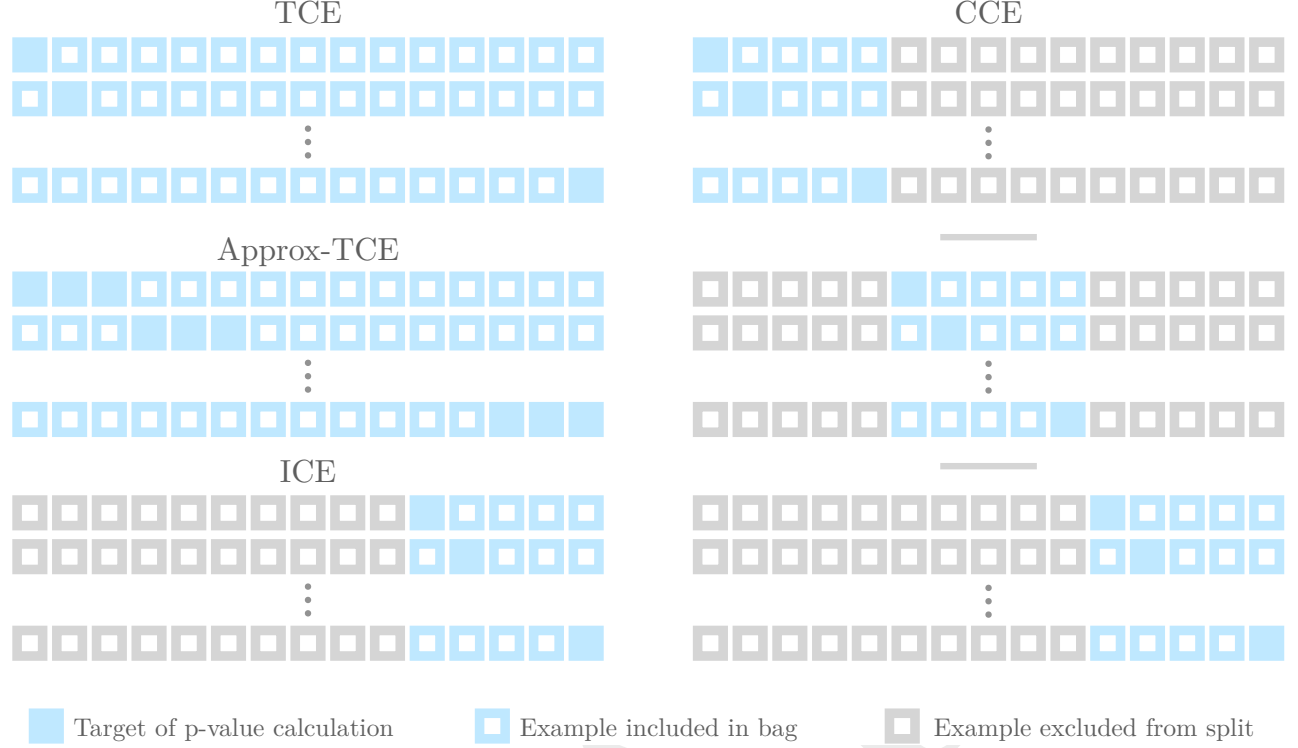
Figure 5: Illustration of the different calibration splits employed by each of the conformal evaluators.

5.4 Threshold Search

To support the calibration of our new conformal evaluators, we propose an alternative random search strategy to replace the exhaustive grid search used in the original paper [16]. In the exhaustive grid search, each possible combination of thresholds over all classes is tested systematically, typically considering some fixed range of variables $V = \{v : v \in [0, 1]\}$. However, this suffers from the curse of dimensionality [7], resulting in $|V|^{|\mathcal{Y}|}$ total trials, growing exponentially with the number of classes. Additionally, reducing the granularity of the range considered in V increases the risk of 'skipping' over an optimal threshold combination. Similarly, the it is difficult to avoid iterating over many useless threshold combinations (where one is either too high or too low). This failure to evenly cover subspaces of interest worsens as the dimensionality increases [8], making it especially problematic for multiclass classification. The granularity at values are chosen for V can be chosen manually based on intuition and past experience, however this results in experiment parameters which are difficult to reproduce and transfer to other settings.

It has been shown for hyperparameter optimization that random search is able to find combinations of variables at least as optimal as those found with full grid search over the same domain, at a fraction of the computational cost [8]. We apply these findings to the threshold calibration and replace the exhaustive grid search with a random search (Algorithm 1). We choose random combinations of thresholds in the inter-

val $[0, 1]$, keeping track of the thresholds that maximize our chosen metric given the constraints (see §4.4). The search continues until either of two conditions are met. A limit is set on the number of iterations, mainly determined by the time and resources that are available to be invested in the calibration. Intuitively a higher number will increase the likelihood of finding better thresholds and so this acts as the upper bound of the optimization. Secondly, the calibration will end when a stop condition is met. In this work we consider a *no-update* approach in which the search will stop once a fixed point is found, i.e., if there is no improvement to performance after a number of consecutive iterations.

We note that this procedure can be easily parallelised. We perform an empirical comparison between the previous grid search and our random search strategy in §6.4.

6 Experimental Evaluation

We evaluate the effectiveness of our novel conformal evaluators when faced with gradual concept drift caused by the evolution of real malicious Android apps over time (§6.2). Additionally, we compare the performance gained when introducing confidence to the decision evaluation against using credibility alone (§6.3), and compare our random search implementation against an exhaustive search (§6.4). From this we determine which settings are optimal for a deployment of TRANSCEND and conformal evaluation in the wild.

6.1 Experimental Settings

Prototype. We develop a prototype of TRANSCEND that encompasses the functionality from the original work [16] as well as our new proposals. The prototype is implemented as a Python library that aims to be familiar to users of popular ML frameworks such as scikit-learn [27]. We release the code as open source and make it available to other researchers (see §9). Note that this is the first publicly available implementation of TRANSCEND in any form.

Dataset As a case study, we focus on malware detection in the Android domain. The original evaluation of TRAN-SCEND [16] artificially simulated concept drift by fusing two datasets: DREBIN [4] and MARVIN [19], a process which may have induced experimental bias [5]. We instead sample 232,848 benign and 26,387 malicious apps (~10% prevalence) from ANDROZOO [2]. This allows us to demonstrate efficacy when faced with a more natural, surreptitious concept drift. The apps span 5 years, from Jan 2014 through to Dec 2018.

Dataset Split We use the TESSERACT [28] framework to carry out temporal evaluations, ensuring that TESSERACT's constraints are accounted for to remove sources of spatial and temporal experimental bias. Training and calibration are performed using apps from 2014 and testing is evaluated over the concept drift that occurs over the remaining period on a month-by-month basis.

Classifier For the underlying classifier, we use DREBIN [4] which has been shown to achieve state-of-the-art performance if a retraining strategy is used to remediate concept drift [28]. Due to this, we hypothesize that if TRANSCEND is used to reject drifting points, DREBIN will be able to classify the remaining points with high accuracy. DREBIN uses a linear SVM as the learning algorithm and a binary feature space in which Android components (activities, permissions, URLs, services, etc) are represented as present or absent.

Calibration To optimize the thresholding, we aim to maximize the F_1 of all kept elements while maintaining a rejection rate of <15%. These metrics are computed in aggregate for each time period of the temporal evaluation. For the random search we use $100,000$ random iterations with early stopping after $3,000$ consecutive events without improvement.

6.2 Novel Conformal Evaluators

Here we compare our novel conformal evaluators: ICE and CCE. Since vanilla TCE is not feasible for this experiment setting due to the size of the dataset, we do not evaluate it and use approx-TCE as a proxy for the transductive setting.

Performance Metrics Figure 6 shows the the F_1, Precision, and Recall (rows) for each of the novel evaluators (columns). The middle dashed line of each plot shows the baseline performance when no rejection mechanism is enforced. This depicts the performance degradation caused by the concept drift present in the dataset resulting from an evolving malicious class. Note that classifiers degrade rapidly, becoming worse than random guessing in under one year.

The upper solid line with square markers shows the performance of kept elements, elements with test p-values that fall above the threshold of their predicted classes. While degradation is still present, approx-TCE and ICE are able to maintain $F_1 > 0.8$ for two years, extending the effectiveness of the model by at least another year—doubling its lifespan.

The lower solid line with circle markers shows the performance of rejected elements. This is the performance on elements that were rejected. High metrics represent erroneously rejected elements whereas low metrics mean that the rejected elements would have been incorrectly classified by the underlying classifier and so were rightfully rejected. Approx-TCE and ICE both have F_1, Precision, and Recall of 0 for rejected elements at every test month meaning that every element that was rejected would have been misclassified.

The result of CCE differs in that it is less conservative in its rejections. The performance of kept elements is much higher, but the performance of rejected elements is higher as well, indicating that a small proportion of the rejected elements would have actually been correctly classified. We observe that this conservatism can be increased or decreased by modifying the conditions of the majority vote. If more folds are required to agree before a decision is accepted, the CCE will be more conservative, rejecting more elements. If less folds are required, more elements will be accepted. In this respect, CCE offers an alternative dimension of tuning in addition to the threshold optimization. Additionally, this is a parameter that can be altered during a deployment, rather than being set at calibration. This allows for some adaptibility depending on the deployment scenario: for example when the cost of False Negatives is very high (e.g., not alerting security teams to attacks in network intrusion detection), or when the cost of False Positives if very high (e.g., witholding benign emails in spam detection, or disabling legitimate user accounts in fake account detection).

Rejection Rates The rejection rate as a proportion of testing elements for each month is shown by the grey columns in each plot of Figure 6. For each of the classifiers the rejection rate begins close to the limit enforced at calibration before slowly rising over each year. This is indicative of the increasing concept drift as the malicious apps diverge further and further from the training distribution. We note that while the rejection rates may appear high, these are symptomatic of poor performance in the underlying classifier and are often preferable to taking incorrect actions on False Positives and
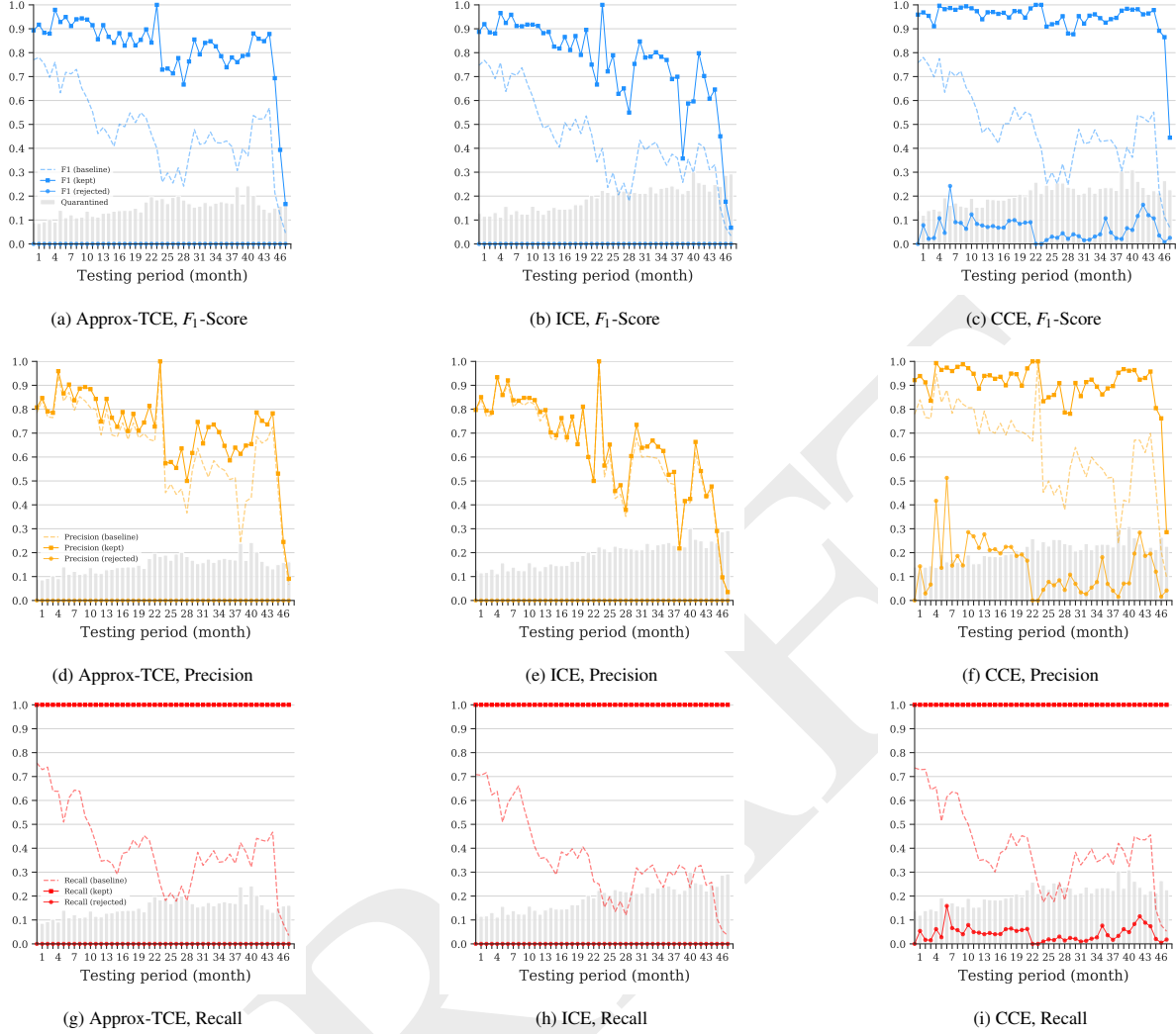
Figure 6: F_1-Score, Precision, and Recall for the three proposed conformal evaluators using credibility p-values. The dashed line shows the performance with no rejection mechanism. The upper line (□ marker) shows the performance on kept examples whose classifications were accepted. The lower line (○ marker) shows the performance on rejected examples. These are the mistakes that would have been made if the predictions were accepted by the degrading model. The bars show the proportion of rejected elements in each period.

False Negatives. In an extreme pathological case where a classifier predicts the opposite of the true label every time, rejection rates could reach 100% but the F1-score of the rejected elements would be 0.

Runtime The runtime of the conformal evaluators during this experiment match what would be expected from their relative complexities (cf. Table 1). The ICE is the quickest, taking 11.5 CPU hours. The CCE took 35.6 CPU hours, but our implementation is parallelized resulting in a wall-clock time identical to the ICE. Finally, the TCE took 46.1 CPU hours. As mentioned previously, vanilla TCE was computationally infeasible, but we estimate a runtime of 1.9 CPU years given the time to fit the underlying classifier (~10 minutes).

We conclude that the ICE is the most useful for settings where resources are limited or models with a rapid iteration cycle (e.g., daily), while the CCE offers greater confidence and flexibility at a slightly higher computational cost.

6.3 Credibility, Confidence, and Probabilities

Credibility with Confidence Intuitively, including confidence thresholds when evaluating a classifier prediction would be beneficial as confidence represents how certain the classifier was in its own prediction. However, as credibility is the main indicator that i.i.d. has been violated, and thus that concept drift is occurring, it is unclear how much further gain confidence provides. Here we test this by evaluating the conformal evaluators under the same conditions as §6.2, using a per-class threshold for both credibility and confidence.
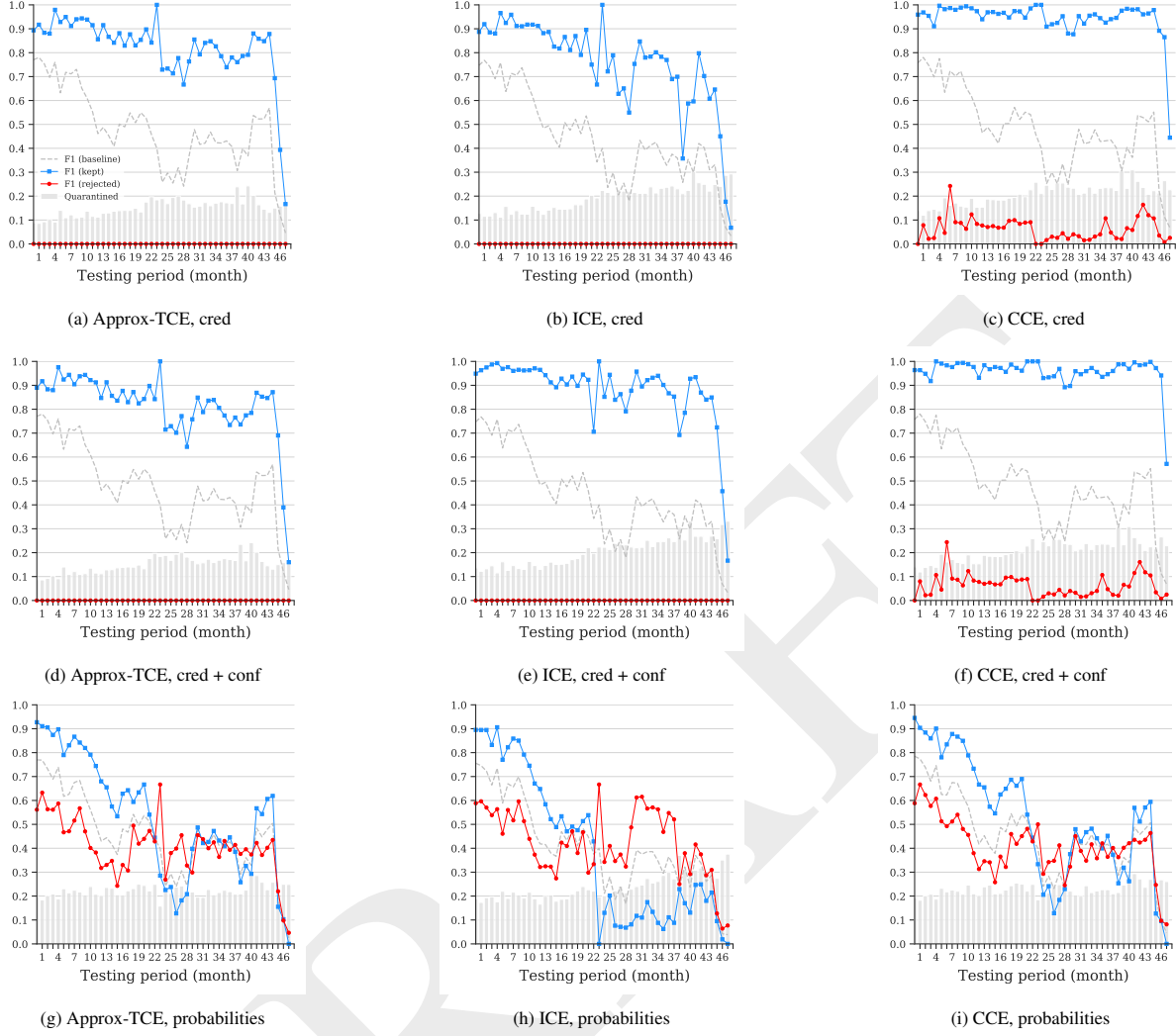
11

Figure 7: F_1-Score for the three proposed conformal evaluators using quality metrics based on credibility p-values, a combination of credibility and confidence p-values, and probabilities. The dashed line shows the performance with no rejection mechanism. The upper line (□ marker) shows the performance on kept examples whose classifications were accepted. The lower line (○ marker) shows the performance on rejected examples. The bars show the proportion of rejected elements in each period. Note the first row (a, b, c) is identical to the first row of Figure 6, replicated here for ease of comparison.

Figure 7 compares the F_1 performance of each conformal evaluator (columns) using different thresholding metrics (rows). Here the upper blue line shows the performance of kept elements while the lower red line shows the performance of rejected elements. The grey dashed middle line depicts the baseline performance when no rejection mechanism is used. Note that the first row of plots if repeated from Figure 6, included to ease comparison.

The second row shows the F_1 performance when confidence is introduced. The performance for the approx-TCE and CCE is relatively unchanged, while the performance of the ICE is markedly improved with degradation being postponed much longer than before. The confidence appears to restore some of the statistical information lost by using only a small amount of the training data for calibration.

However, the computational cost required to find thresholds is much higher than with credibility only. Introducing confidence is equivalent to doubling the number of classes, vastly increasing the length of time required to find acceptable thresholds. From this we conclude that the performance gain obtained from introducing confidence likely does not outweigh the computational cost. Although it might improve the accuracy of an ICE, using a CCE will often provide the same accuracy with comparable calibration time.

Probabilities The lowest row of Figure 7 shows the F_1 performance when the classifier's output probabilities are used for thresholding, rather than generating per-class p-values for each calibration and testing point. For each conformal evaluator, the same training and calibration split is used as was

with p-values, to ensure a fair comparison. The plot shows that probabilities alone offer a very small improvement for kept elements over the baseline in the first year, becoming increasingly volatile and less reliable as the concept drift becomes more severe. In all cases, probabilities result in an unacceptably high level of erroneous rejections compared to p-values. From this we conclude that the statistical support offered by the conformal evaluator's p-value computation is significant and justifies the additional computational overhead that it induces.

6.4 Full Grid Search vs Random search

Here we evaluate the performance of our random search implementation (§5.4) compared to the full grid search used in the original proposal [16]. We aim to show that the random search is able to find high quality calibration thresholds in a more efficient manner than the full search.

As the grid search is extremely expensive, in this experiment we train and calibrate on 1 month of data and test on the remaining 59 months using an approx-TCE with 10 folds. We maximize F_1 while maintaining an acceptable rejection rate of <15%. To ensure our baseline discovers high quality thresholds we use a fine granularity grid and cover 1,317,520 combinations of class thresholds. For the random search we set an upper limit of 10,000 trials.

Table 3 shows the confusion matrices when no rejection is applied (a), when rejection is applied using the thresholds from the full grid search (b), and when rejection is applied using the thresholds from random search (c). Note that there is no significant performance difference between the two search strategies, but the random search was able to cover the same search space with two orders of magnitude fewer trials. We observe that the full grid search makes erroneous assumptions on the distribution of quality thresholds which the random search does not. Additionally, while the random search allows for a variety of stopping conditions, the only mechanism to control the length of the full grid search is the size of the interval to search and the granularity of the search steps—the suitability of which is difficult to determine beforehand.

7 Operational Guidance

Here we discuss some actionable points regarding the use of conformal evaluation and Transcend.

TRANSCEND in a Detection Pipeline TRANSCEND has particular applications in detection tasks where there is a high cost of False Positives, (e.g., spam [24], malware [4, 12, 32], fake accounts [10, 11]). In these cases, it may be preferable to avoid taking a decision on low-confidence predictions or, where a gradated response is possible, diverting rejected examples towards alternative remediation actions. Consider an

example in the fake accounts setting: owners of accounts in the set of rejected positive predictions can be asked to solve a CAPTCHA on their next login (a relatively easy check to pass) while the owners of accounts in the set of kept positive predictions can be asked to submit proof of their identity. Increasing rejection rates signal a performance degradation of the underlying classifier without immediately submitting to the errors it produces, giving engineers more time to iterate and remediate.

Operational Recommendations Based on our empirical evaluation (§6), we make the following recommendations for TRANSCEND deployments:

- TRANSCEND is agnostic to the underlying learning algorithm, but the quality of the rejection relies on the suitability of the NCM. Some examples of possible NCMs for different types of classifiers are described in Figure 1.

- Using an ICE or CCE is preferred over TCE due to their computational efficiency, and are preferred over approx-TCE due to approx-TCE's reliance on assumptions that may not universally hold.

- ICEs are relatively fast and lightweight and excel when resources are limited. CCEs make rejections with higher confidence but at a higher computational cost.

- Thresholding with credibility alone is sufficient to achieve high quality prediction across all conformal evaluators. While confidence can improve the stability of an ICE, it is associated with an increase in calibration time.

- Random search is preferred over exhaustive grid search as it can find more effective thresholds given less calibration time.

- Rising rejection rates should be monitored and used to signal when the underlying model is degrading. This signal can be used to trigger model retraining or other remediation strategies.

8 Related Work

Conformal evaluation is based on conformal prediction theory, a mechanism for producing predictions that are correct with some guaranteed confidence [30]. Additionally, the ICE and CCE are inspired by inductive [25, 36, 37] and cross-conformal predictors [38], respectively. However, conformal prediction is intended to be used exclusively in settings where the exchangeability assumption holds which makes it unsuitable for adversarial contexts such as malware classification. In this regard, we are the first to 'join the dots' between the conformal prediction of Vovk et al. [37] and the conformal

Table 2: Area Under Time (AUT) of F_1 performance with respect to concept drift over the 48 month testing period for different quality metrics: credibility, credibility with confidence, and probabilities (cf. Figure 7).

		Approx-TCE	ICE	CCE
Credibility	AUT(F_1 baseline, 48m)	.480	.440	.483
	AUT(F_1 kept, 48m)	.829	.762	.950
	AUT(F_1 rejected, 48m)	.000	.000	.064
Credibility + Confidence	AUT(F_1 baseline, 48m)	.480	.440	.483
	AUT(F_1 kept, 48m)	.822	.887	.962
	AUT(F_1 rejected, 48m)	.000	.000	.063
Probabilities	AUT(F_1 baseline, 48m)	.456	.405	.455
	AUT(F_1 kept, 48m)	.531	.388	.532
	AUT(F_1 rejected, 48m)	.410	.426	.410

Table 3: Confusion matrices showing the performance of optimal thresholds discovered using a full grid search vs random search. The random search is able to discover thresholds equivalent to the full grid search but with two orders of magnitude fewer trials.

(a) Baseline

Example	Benign	Malicious
Benign	6,518	3,529
Malicious	19,486	225,874

(b) Full grid search

Example	Benign	Malicious
Benign	6,518	2,187
Malicious	0	198,060
Number of trials		1,317,520

(c) Random search

Example	Benign	Malicious
Benign	6,518	3,259
Malicious	0	197,983
Number of trials		10,000

evaluation of Jordaney et al. [16] and show how the violation of conformal prediction's assumptions is detected and exploited by TRANSCEND to detect concept drift.

TRANSCEND was originally proposed at USENIX 2017 [16]. This work introduced the concept of *conformal evaluation* based on conformal prediction theory and the use of p-values for calibrating and enforcing a rejection strategy for malware classification. However the evaluation artificially simulated concept drift by merging malware datasets which could have introduced experimental bias [5, 28]. In our experiments we sample from a single repository of applications and perform a temporal evaluation to simulate natural concept drift caused by the real evolution of malicious Android apps. Additionally, the role of confidence in thresholding was unclear, and the use of exhaustive grid search to find thresholds was suboptimal compared to our random search. Most significantly, the TCE employed by the original work was not practical for real-world deployments, which we rectify by proposing the ICE and CCE.

Other works have explored alternative solutions to tackling concept drift. Xu et al. [40] propose DROIDEVOLVER, a malware detection system motivated by TRANSCEND that identifies drifting examples based on disagreements between models in an ensemble. As models degrade, the examples identified as drifting are used to update the models in an online fashion. While this solution requires less resources than TRANSCEND, the system retrains using predicted labels rather than ground truth labels which can result in a negative feedback loop. Other solutions solely adapt to concept drift without using rejection: DROIDOL [23] and CASANDRA [22]

use online learning to continually retrain the models, using API call graphs as features. Like all online-trained neural networks, these approaches are susceptible to catastrophic forgetting [15], where the performance begins to degrade on older examples as the model attempts to adapt to the new distribution. Pendlebury et al. [28] present a comparison of different strategies for combatting concept drift, including rejection, incremental retraining, and online learning, illustrating the advantages and disadvantages of each.

The related problem of detecting *adversarial examples* is addressed by Sotgiu et al. [31], who propose a rejection strategy for neural network-based classifiers that identifies anomalies in the latent feature representation of an example at different layers of the neural network. Additionally, Papernot and McDaniel [26] combine a conformal predictor with a k-Nearest Neighbor algorithm to identify low quality predictions that are indicative of adversarial inputs. However, both of these methods are restricted to deep learning-based image classification.

9 Availability

We make TRANSCEND's code available to researchers and practitioners alike, to aid with the scalable identification and remediation of concept drift. We also release the data used in the empirical evaluations. The TRANSCEND project website can be found at `https://s2lab.kcl.ac.uk/transcend`.

10 Conclusion

Following the proposal of TRANSCEND [16], rejection strategies have seemed like a promising solution to the issue of concept drift. However, TRANSCEND left a number of open questions around its theoretical soundness, optimal configuration, and real-world practicality.

In this work we provided a formal treatment of TRANSCEND which acts as the *missing link* between conformal prediction and conformal evaluation. We have empirically determined which configurations perform best and how they can be tuned depending on the operational settings. Finally, we have proposed novel conformal evaluators which match or surpass the performance of the original while significantly decreasing the computational overhead.

We envision these improvements will enable researchers and practitioners alike to make use of conformal evaluation and TRANSCEND to build rejection strategies to improve the quality of security detection pipelines. To accommodate this, we also release the first publicly available TRANSCEND implementation to the security community.

Acknowledgements

References

[1] K. Allix, T. F. Bissyandé, J. Klein, and Y. L. Traon. Are your training datasets yet relevant? - an investigation into the importance of timeline in machine learning-based malware detection. In F. Piessens, J. Caballero, and N. Bielova, editors, *ESSoS 2015*, 2015. doi: 10.1007/978-3-319-15618-7_5. URL https://doi.org/10.1007/978-3-319-15618-7_5.

[2] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16. ACM, 2016.

[3] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth. Learning to evade static PE machine learning malware models via reinforcement learning. *CoRR*, abs/1801.08917, 2018.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: effective and explainable detection of android malware in your pocket. In *NDSS*. The Internet Society, 2014.

[5] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and don'ts of machine learning in computer security. 2020.

[6] P. L. Bartlett and M. H. Wegkamp. Classification with a reject option using a hinge loss. *J. Mach. Learn. Res.*, 9:1823–1840, 2008.

[7] R. Bellman. *Adaptive Control Processes - A Guided Tour (Reprint from 1961)*, volume 2045 of *Princeton Legacy Library*. Princeton University Press, 2015. ISBN 978-1-4008-7466-8. doi: 10.1515/9781400874668. URL https://doi.org/10.1515/9781400874668.

[8] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

[9] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[10] Y. Boshmaf, D. Logothetis, G. Siganos, J. Lería, J. Lorenzo, M. Ripeanu, and K. Beznosov. Integro: Leveraging victim prediction for robust fake account detection in osns. In *NDSS*. The Internet Society, 2015.

[11] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *NSDI*, pages 197–210. USENIX Association, 2012.

[12] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert. ZOZZLE: fast and precise in-browser javascript malware detection. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011. URL http://static.usenix.org/events/sec11/tech/full_papers/Curtsinger.pdf.

[13] S. K. Dash, G. Suarez-Tangil, S. J. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro. Droidscribe: Classifying android malware based on runtime behavior. In *IEEE Symposium on Security and Privacy Workshops*, pages 252–261. IEEE Computer Society, 2016.

[14] S. Edunov, M. Ott, M. Auli, and D. Grangier. Understanding back-translation at scale. In *EMNLP*, pages 489–500. Association for Computational Linguistics, 2018.

[15] R. M. French and N. Chater. Using noise to compute error surfaces in connectionist networks: A novel means of reducing catastrophic forgetting. *Neural Comput.*, 14(7):1755–1769, 2002. doi: 10.1162/08997660260028700. URL https://doi.org/10.1162/08997660260028700.

[16] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro. Transcend: Detecting concept drift in malware classification models. In *USENIX Security Symposium*, pages 625–642. USENIX Association, 2017.

[17] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *AISec@CCS*, pages 45–56. ACM, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[19] M. Lindorfer, M. Neugschwandtner, and C. Platzer. MARVIN: efficient and comprehensive mobile app classification through static and dynamic analysis. In *COMPSAC*, pages 422–433. IEEE Computer Society, 2015.

[20] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu, A. D. Joseph, and J. D. Tygar. Reviewer integration and performance measurement for malware detection. In *DIMVA*, volume 9721 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 2016.

[21] J. G. Moreno-Torres, T. Raeder, R. Alaíz-Rodríguez, N. V. Chawla, and F. Herrera. A unifying view on dataset shift in classification. *Pattern Recognit.*, 45(1):521–530, 2012.

[22] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Trans. Emerging Topics in Comput. Intellig.*, 1(3):157–175, 2017.

[23] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Trans. Emerg. Top. Comput. Intell.*, 1(3):157–175, 2017. doi: 10.1109/TETCI.2017.2699220. URL https://doi.org/10.1109/TETCI.2017.2699220.

[24] S. Nilizadeh, F. Labreche, A. Sedighian, A. Zand, J. M. Fernandez, C. Kruegel, G. Stringhini, and G. Vigna. POISED: spotting twitter spam off the beaten paths. In *ACM Conference on Computer and Communications Security*, pages 1159–1174. ACM, 2017.

[25] H. Papadopoulos. Inductive conformal prediction: Theory and application to neural networks. In P. Fritzsche, editor, *Tools in Artificial Intelligence*, chapter 18. IntechOpen, Rijeka, 2008. doi: 10.5772/6078. URL https://doi.org/10.5772/6078.

[26] N. Papernot and P. D. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *CoRR*, abs/1803.04765, 2018. URL http://arxiv.org/abs/1803.04765.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[28] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSER-ACT: eliminating experimental bias in malware classification across space and time. In *USENIX Security Symposium*, pages 729–746. USENIX Association, 2019.

[29] Plato. *The Symposium*. c. 385–370 BC. Penguin Classics edition published 1999, translated by Christopher Gill.

[30] G. Shafer and V. Vovk. A tutorial on conformal prediction. *J. Mach. Learn. Res.*, 9:371–421, 2008.

[31] A. Sotgiu, A. Demontis, M. Melis, B. Biggio, G. Fumera, X. Feng, and F. Roli. Deep neural rejection against adversarial examples. *EURASIP J. Inf. Secur.*, 2020:5, 2020. doi: 10.1186/s13635-020-00105-y. URL https://doi.org/10.1186/s13635-020-00105-y.

[32] N. Srndic and P. Laskov. Detection of malicious PDF files based on hierarchical document structure. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013. URL https://www.ndss-symposium.org/ndss2013/detection-malicious-pdf-files-based-hierarchical-document-structure.

[33] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. B. Alís. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Syst. Appl.*, 41(4):1104–1117, 2014.

[34] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro. Droidsieve: Fast and accurate classification of obfuscated android malware. In *CODASPY*, pages 309–320. ACM, 2017.

[35] L. Tong, B. Li, C. Hajaj, C. Xiao, N. Zhang, and Y. Vorobeychik. Improving robustness of ML classifiers against realizable evasion attacks using conserved features. pages 285–302. USENIX Association, 2019.

[36] V. Vovk. Conditional validity of inductive conformal predictors. *Journal of Machine Learning Research (JMLR)*, 92(2-3):349–376, 2013.

[37] V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic learning in a random world*. Springer-verlag New York Inc., 2010. ISBN 9781441934710.

[38] V. Vovk, I. Nouretdinov, V. Manokhin, and A. Gammerman. Cross-conformal predictive distributions. In *Workshop on Conformal Prediction and its Applications (COPA)*, volume 91 of *Proceedings of Machine Learning Research*, pages 37–51. PMLR, 2018.

[39] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, and J. Lu. Deepint: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps. In *CCS*, pages 2421–2436. ACM, 2019.

[40] K. Xu, Y. Li, R. H. Deng, K. Chen, and J. Xu. Droidevolver: Self-evolving android malware detection system. In *EuroS&P*, pages 47–62. IEEE, 2019.

[41] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *ICSE (1)*, pages 303–313. IEEE Computer Society, 2015.

A Symbol Table

Table 4 reports the major symbols and abbreviations used throughout the paper.

B Random Search Calibration Algorithm

We present the algorithm for our random search calibration.

C Conformal Evaluator Algorithms

We present algorithms for calibrating TCEs, ICEs, and CCEs.

Table 4: Table of symbols and abbreviations.

SYMBOL	DESCRIPTION
X	Feature space $X \subseteq \mathbb{R}^n$.
$\mathcal{Y}$	Label space.
z	Example pair $(\boldsymbol{x}, y) \in X \times \mathcal{Y}$.
z^*	Previously unseen testing example.
$\hat{y}$	Predicted class $g(z^*)$.
a_z	Nonconformity score output by an NCM for a given point z.
p_z	Statistical p-value for a given point z.
p_z^y	Statistical p-value for a given point z, calculated with respect to class $y \in \mathcal{Y}$ (used in *label conditional* calculations).
τ_y	A TRANSCEND rejection threshold $\tau_y \in [0,1]$ for class $y \in \mathcal{Y}$.
$\mathcal{T}$	The set of all per-class rejection thresholds $\{\tau_y \in [0,1] \mid y \in \mathcal{Y}\}$.
B	Bag of examples $\langle z_1, z_2, ..., z_n \rangle$.
d	Distance function $d(z, z')$.
$\hat{z}$	Point predictor $\hat{z}(B)$.
A	Noncomformity measure (NCM) usually composed of a distance function and point predictor.
S	Collection of noncomformity scores computed in elements of B, relative to other elements in B, $S = \langle A(B \setminus \langle z \rangle, z) : z \in B \rangle$.
g	Classifier $g : X \longrightarrow \mathcal{Y}$ that assigns object $\boldsymbol{x} \in X$ to class $y \in \mathcal{Y}$. Also known as the *decision function*.
ε	Significance level used in conformal prediction to define prediction region with certain confidence guarantees.
TCE	Transductive Conformal Evaluator.
ICE	Inductive Conformal Evaluator.
CCE	Cross-Conformal Evaluator.
NCM	Nonconformity measure.

Algorithm 1: TRANSCEND threshold calibration using random search

Input : y, $\hat{y}$, $pval_c$
Input: $\boldsymbol{Y} \in \mathcal{Y}^n$, ground truth labels for n examples
    $\hat{\boldsymbol{Y}} \in \mathcal{Y}^n$, predicted labels for n examples
    $P \in \mathbb{R}^{n \times |\mathcal{Y}|}$, per-class p-values for n examples
Parameters : $m \in \mathbb{R}$, maximum number of iterations
    $\mathcal{F} : \boldsymbol{Y} \times \hat{\boldsymbol{Y}} \times P \longrightarrow \mathbb{R}$, performance measure to optimize (e.g., F_1)
    $\mathcal{G} : \boldsymbol{Y} \times \hat{\boldsymbol{Y}} \times P \longrightarrow \mathbb{R}$, performance measure to constrain (e.g., kept examples)
    $\mathcal{C} \in \mathbb{R}$, lower bound for constrained measure $\mathcal{G}$
Output: $\boldsymbol{t}^*$, a vector of per-class thresholds
Output: $\boldsymbol{t}^* \in [0,1]^{|\mathcal{Y}|}$, a vector of per-class thresholds

1   $\boldsymbol{t}^* \leftarrow \boldsymbol{0}$
2   counter $\leftarrow 0$
3   **while** counter $< m$ **do**
4      $\boldsymbol{t} \xleftarrow{\$} [0,1]^{|\mathcal{Y}|}$ ▷ Pick random thresholds
5   **if** $\mathcal{F}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, P; \boldsymbol{t}) > \mathcal{F}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, P; \boldsymbol{t}^*)$ *and* $\mathcal{G}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, P; \boldsymbol{t}) \geq \mathcal{C}$ **then**
6          $\boldsymbol{t}^* \leftarrow \boldsymbol{t}$
7   **else if** $\mathcal{F}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, P; \boldsymbol{t}) = \mathcal{F}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, P; \boldsymbol{t}^*)$ *and* $\mathcal{G}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, P; \boldsymbol{t}) > \mathcal{G}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}, P; \boldsymbol{t}^*)$ **then**
8          $\boldsymbol{t}^* \leftarrow \boldsymbol{t}$
9   counter $\leftarrow$ counter $+ 1$
10   **end**
11   **return** $\boldsymbol{t}^*$

Algorithm 2: Transductive Conformal Evaluator (TCE and *approximate* TCE)

Input: $Z = \{z_0, z_1, \ldots, z_{n-1}\}$, n training examples
$\quad\quad Z^* = \{z_0^*, z_1^*, \ldots\}$, stream of testing examples
$\quad\quad A$, NCM for producing nonconformity scores
$\quad\quad k \in \mathbb{N}$, number of folds—TCE is *approximate* when $k < n$
Output: Stream of boolean decisions $0 = reject, 1 = accept$

Calibration Phase

---

1   $\boldsymbol{P} \leftarrow \boldsymbol{0}$
2   $i \leftarrow 0$
3   **partition** Z equally into $Z^{part} \leftarrow \{Z_0', Z_1', \ldots, Z_{k-1}'\}$
4   **foreach** *partition* Z' *of* Z^{part} **do**
5      $Z'' \leftarrow Z \setminus Z'$
6      $g \leftarrow$ Fit (Z'')
7   **foreach** z' *of* Z' **do**
8         $\hat{y} \leftarrow g(z')$ ▷ Predicted label
9         $Z_{\hat{y}}' \leftarrow \{z \in Z' : z.y = \hat{y}\}$ ▷ Bag of examples with same label
10      $\alpha_{z'} \leftarrow A(Z_{\hat{y}}', z')$ ▷ Nonconformity score
11      $S \leftarrow \{A(Z_{\hat{y}}' \setminus \{z\}) : z \in Z_{\hat{y}}'\}$ ▷ Nonconformity scores for bag elements
12      $p_{z'} \leftarrow \frac{|\alpha \in S: \alpha >= \alpha_{z'}|}{|S|}$ ▷ Credibility p-value
13      $\boldsymbol{P}_i \leftarrow p_{z'}$
14      $i \leftarrow i + 1$
15   **end**
16   **end**
17   $\boldsymbol{t}^* \leftarrow$ Transcend.FindThresholds $(Z, \hat{\boldsymbol{Y}}, \boldsymbol{P})$

Test Phase

---

18   $g \leftarrow$ Fit (Z)
19   **foreach** z^* *of* Z^* **do**
20      $\hat{y} \leftarrow g(z^*)$ ▷ Predicted label for test example
21      $Z_{\hat{y}} \leftarrow \{z \in Z : z.y = \hat{y}\}$ ▷ Bag of training examples with same label
22      $\alpha_{z^*} \leftarrow A(Z_{\hat{y}}, z^*)$ ▷ Nonconformity score
23      $S \leftarrow \{A(Z_{\hat{y}} \setminus \{z\}) : z \in Z_{\hat{y}}\}$ ▷ Nonconformity scores for bag elements
24      $p_{z^*} \leftarrow \frac{|\alpha \in S: \alpha >= \alpha_{z^*}|}{|S|}$ ▷ Credibility p-value
25   **if** $P_{z^*} < \boldsymbol{t}_{\hat{y}}^*$ **then emit** 0 **else emit** 1
26   **end**

Algorithm 3: Inductive Conformal Evaluator (ICE)

Input: $Z = \{z_0, z_1, \ldots, z_{n-1}\}$, n training examples
$Z^* = \{z_0^*, z_1^*, \ldots\}$, stream of testing examples
A, NCM for producing nonconformity scores
m, number of examples to use for calibration
Output: Stream of boolean decisions $0 = reject, 1 = accept$

Calibration Phase

1   $P \leftarrow \mathbf{0}$
2   $\hat{Y} \leftarrow \mathbf{0}$
3   $i \leftarrow 0$
4   $Z^{tr} \leftarrow \{z_0, z_1, \ldots, z_{n-m-1}\}$
5   $Z^{cal} \leftarrow \{z_{n-m}, z_{n-m+1}, \ldots, z_{n-1}\}$
6   **foreach** z' *of* Z^{cal} **do**
7      $g \leftarrow$ Fit $(Z^{cal} \setminus \{z'\})$
8      $\hat{y} \leftarrow \hat{Y}_i \leftarrow g(z')$ ▷ Predicted label
9      $Z_{\hat{y}}^{cal} \leftarrow \{z \in Z^{cal} : z.y = \hat{y}\}$ ▷ Bag of examples with same label
10     $\alpha_{z'} \leftarrow A(Z_{\hat{y}}^{cal}, z')$ ▷ Nonconformity score
11     $S \leftarrow \{A(Z_{\hat{y}}^{cal} \setminus \{z\}) : z \in Z_{\hat{y}}^{cal}\}$ ▷ Nonconformity scores for bag elements
12     $p_{z'} \leftarrow \frac{|\alpha \in S : \alpha >= \alpha_{z'}|}{|S|}$ ▷ Credibility p-value
13     $P_i \leftarrow p_{z'}$
14     $i \leftarrow i + 1$
15   **end**
16   $t^* \leftarrow$ Transcend.FindThresholds $(Z, \hat{Y}, P)$

Test Phase

17   $g \leftarrow$ Fit (Z^{tr})
18   **foreach** z^* *of* Z^* **do**
19     $\hat{y} \leftarrow g(z^*)$ ▷ Predicted label for test example
20     $Z_{\hat{y}}^{cal} \leftarrow \{z \in Z^{cal} : z.y = \hat{y}\}$ ▷ Bag of training examples with same label
21     $\alpha_{z^*} \leftarrow A(Z_{\hat{y}}^{cal}, z^*)$ ▷ Nonconformity score
22     $S \leftarrow \{A(Z_{\hat{y}}^{cal} \setminus \{z\}) : z \in Z_{\hat{y}}^{cal}\}$ ▷ Nonconformity scores for bag elements
23     $p_{z^*} \leftarrow \frac{|\alpha \in S : \alpha >= \alpha_{z^*}|}{|S|}$ ▷ Credibility p-value
24   **if** $P_{z^*} < t_{\hat{y}}^*$ **then emit** 0 **else emit** 1
25   **end**

Algorithm 4: Cross-Conformal Evaluator (CCE)

Input: $Z = \{z_0, z_1, \ldots, z_{n-1}\}$, n training examples

       $Z^* = \{z_0^*, z_1^*, \ldots\}$, stream of testing examples

       A, NCM for producing nonconformity scores

       $k \in \{2t+1 : t \in \mathbb{N}\}$, number of folds

Output: Stream of boolean decisions $0 = reject, 1 = accept$

Calibration Phase

---

1   $\boldsymbol{P} \leftarrow \hat{\boldsymbol{Y}} \leftarrow \boldsymbol{G} \leftarrow \boldsymbol{t}^* \leftarrow \boldsymbol{0}$

2   $i \leftarrow j \leftarrow 0$

3   **partition** Z equally into $\{Z_0', Z_1', \ldots, Z_{k-1}'\}$

4   **foreach** j *of* $\{0, 1, \ldots, k-1\}$ **do**

5   **foreach** z' *of* Z_j' **do**

6          $g \leftarrow$ Fit $(Z_j' \setminus \{z'\})$

7          $\hat{y} \leftarrow \hat{\boldsymbol{Y}}_{j,i} \leftarrow g(z')$ ▷ Predicted label

8          $Z_{j\hat{y}}' \leftarrow \{z \in Z_j' : z.y = \hat{y}\}$ ▷ Bag of examples with same label

9          $\alpha_{z'} \leftarrow A(Z_{j\hat{y}}', z')$ ▷ Nonconformity score

10         $S \leftarrow \{A(Z_{j\hat{y}}' \setminus \{z\}) : z \in Z_{j\hat{y}}'\}$ ▷ Nonconformity scores for bag elements

11         $\boldsymbol{P}_{j,i} \leftarrow \frac{|\alpha \in S : \alpha >= \alpha_{z'}|}{|S|}$ ▷ Credibility p-value

12         $i \leftarrow i+1$

13   **end**

14      $\boldsymbol{G}_j \leftarrow$ Fit $(Z \setminus Z_j')$

15      $\boldsymbol{T}_j^* \leftarrow$ Transcend.FindThresholds $(Z_j', \hat{\boldsymbol{Y}}_{\boldsymbol{j}}, \boldsymbol{P}_j)$

16   **end**

Test Phase

---

17   $s \leftarrow 0$

18   **foreach** z^* *of* Z^* **do**

19   **foreach** j *of* $\{0, 1, \ldots, k-1\}$ **do**

20          $\hat{y} \leftarrow G_j(z^*)$ ▷ Predicted label for test example

21          $Z_{j\hat{y}}' \leftarrow \{z \in Z_j' : z.y = \hat{y}\}$ ▷ Bag of training examples with same label

22          $\alpha_{z^*} \leftarrow A(Z_{j\hat{y}}', z^*)$ ▷ Nonconformity score

23          $S \leftarrow \{A(Z_{j\hat{y}}' \setminus \{z\}) : z \in Z_{j\hat{y}}'\}$ ▷ Nonconformity scores for bag elements

24          $p_{z^*} \leftarrow \frac{|\alpha \in S : \alpha >= \alpha_{z^*}|}{|S|}$ ▷ Credibility p-value

25   **if** $P_{z^*} \geq \boldsymbol{T}_{j\hat{y}}^*$ **then** $s \leftarrow s+1$ ▷ Track positive evaluations

26   **end**

27   **if** $s < k/2$ **then emit** 0 **else emit** 1   ▷ Majority vote for final decision

28   **end**