Citation for published version (APA):
Barbero, F., Pendlebury, F., Pierazzi, F., & Cavallaro, L. (2022). Transcending TRANSCEND: Revisiting Malware Classification in the Presence of Concept Drift. In *IEEE Symposium on Security and Privacy (S&P)* (43rd ed.)

Transcending TRANSCEND: Revisiting Malware Classification in the Presence of Concept Drift

Federico Barbero*[†][||], Feargus Pendlebury*[‡][§][¶], Fabio Pierazzi[†], Lorenzo Cavallaro[¶]
[†] King's College London, [‡] Royal Holloway, University of London, [§] The Alan Turing Institute,
[||] University of Cambridge, [¶] University College London

***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 propose TRANSCENDENT, a rejection framework built on Transcend, a recently proposed strategy 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 TRANSCENDENT on a malware dataset spanning 5 years that removes sources of experimental bias present in the original evaluation. TRANSCENDENT outperforms state-of-the-art approaches while generalizing across different malware domains and classifiers.

To further assist practitioners, we showcase optimal operational settings for a TRANSCENDENT 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 TRANSCENDENT as open source, to aid the adoption of rejection strategies by the security community.

***Index Terms*—security, machine learning, malware detection**

I. INTRODUCTION

Machine learning (ML) algorithms have displayed superhuman performance across a wide range of classification tasks such as computer vision [23] and natural language processing [17]. However, a great deal of this success is conditional on one central assumption: that the training and test data are drawn identically and independently from the same underlying distribution (i.i.d.) [12].

In a security setting this assumption often does not hold. In particular, malware classifiers are deployed in dynamic, hostile environments [34]. New paradigms of malware evolve to pursue 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* [28]. Over time, classifier performance 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 [35]. 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 [30, 50], or rejecting drifting points. However, 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 [20], a statistical framework that builds on conformal prediction theory [47] to detect aging malware detectors during deployment—before their accuracy deteriorates to unacceptable levels. Transcend [20] proposes a *conformal evaluator* that utilizes the notion of *nonconformity* 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 labeling. While effective, the original proposal suffers from experimental bias, is extremely resource intensive and thus impractical, lacks experiments to support generalization claims, fails to provide guidance on how to integrate it into a detection pipeline and, perhaps more importantly, lacks a theoretical analysis to explain its effectiveness.

In this paper, we revisit conformal evaluator and Transcend to root its internal workings in sound theory and determine its most effective operational settings. We additionally propose TRANSCENDENT, a framework that surpasses the performance of the original in terms of drift detection and computational overhead, making it a sound and practical solution.

In summary, we make the following contributions:

- **Formal Treatment.** We investigate the theory underpinning the motivation and intuition of conformal evaluation to provide a missing link between conformal evaluation and conformal prediction theory that explains its effectiveness and supports the empirical evaluations presented in both this work and the original (§III).

*Equal contribution.

- **Novel Conformal Evaluators.** Building on this insight, we develop two novel conformal evaluators: *inductive conformal evaluator* (ICE) (§IV-B) and *cross-conformal evaluator* (CCE) (§IV-C), 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 formalize Transcend's calibration procedure as an optimization problem and propose an improved search strategy for finding thresholds (§V).
- **Operational Guidance.** We evaluate our proposals on a dataset spanning 5 years (2014–2019) that eliminates sources of bias present in past evaluations (§VI). We compare different operational settings, including the effects of including algorithm *confidence* (§VI-C) and of using different search strategies (§VI-D) during thresholding. Our methods outperform existing state-of-the-art approaches (§VI-E), and generalize well across different malware domains and underlying classifiers (§VI-F). To aid practitioners in adopting rejection strategies, we include a discussion of how to integrate TRANSCENDENT into a typical security detection pipeline (§VII).

To enable researchers and practitioners to make better use of classification with rejection strategies, we publicly release our data and implementation of TRANSCENDENT.[1]

II. BACKGROUND

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

A. 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 [6, 50]) and multiclass classification (predicting the malware family [16, 42, 43]) but can also extend to more complex tasks such as predicting how many AV engines would detect an example [22], inferring Android malware app permissions based on their icons [49], or generating Windows malware using reinforcement learning [4].

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 interesting properties of the apps and $\mathcal{Y}$ is a label space containing binary labels for the detection task or the names of malware families for the multiclass classification task.

B. Concept Drift

One of the greatest challenges facing machine learning-based malware classifiers is the presence of *dataset shift* [1, 20, 27] 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 examples 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 [28]. *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, e.g.,, if a new family of malware arises which, given the feature space representation $\mathcal{X}$, is indistinguishable from benign applications. Due to limited knowledge, the model will start misclassifying examples from the new family, even if no covariate or prior probability shift has occurred. In practice, it can be extremely difficult to determine how much error should be attributed to each type of shift [28]. 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 users. 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.

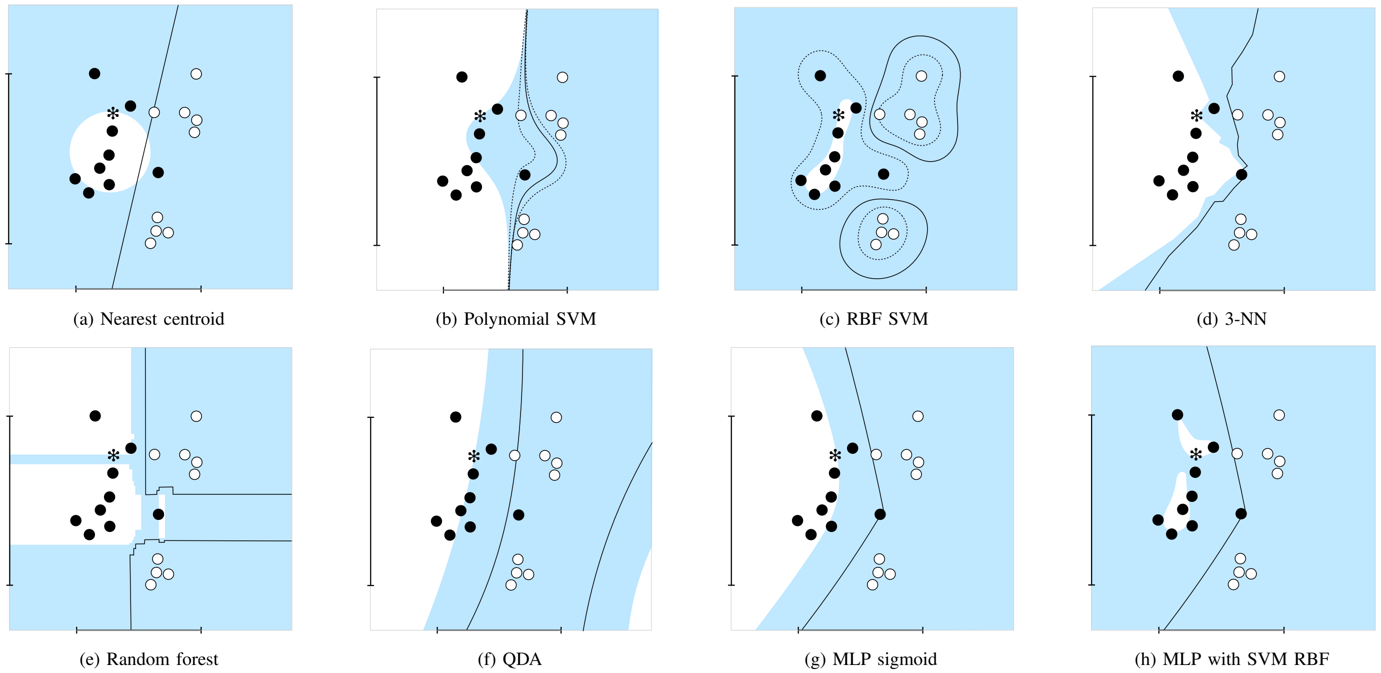
C. Rejection

There are multiple routes to dealing with concept drift. The most effective would be to design a feature space $\mathcal{X}$ such that it is entirely robust to concept drift, essentially distilling all possible malware behaviour down to a 'Platonic ideal' [37] that captures maliciousness no matter what form it takes. While recent proposals for augmenting feature spaces with robust features are promising [e.g., 45, 53], 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 the device contacts might be considered suspicious for a torch app but not for a social messaging app [52].

An orthogonal approach is to identify, track, and mitigate the drift as it occurs. One promising method is classification with rejection [8], 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 [20] 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

[1] https://s2lab.cs.ucl.ac.uk/projects/transcend/



(a) Nearest centroid

(b) Polynomial SVM

(c) RBF SVM

(d) 3-NN

(e) Random forest

(f) QDA

(g) MLP sigmoid

(h) MLP with SVM RBF

Fig. 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 test 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 the decision boundary still depends on the MLP output alone.

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

III. Towards Sound Conformal Evaluation

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* [47], 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.

A. 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 of conformal prediction we refer to Vovk et al. [47]. 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 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 (§III-B). As well as these p-values, two important metrics are derived from the prediction region, confidence and credibility (§III-C), 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 [20] 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.

B. 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) [38] is a real-valued function that outputs a score describing how different an example z is from a bag of previous examples $B = \Lbag z_1, z_2, ..., z_n \Rbag$:

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

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:

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

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):

$$\alpha_{z^*} = A(B, z^*) \tag{3}$$

$$S = \Lbag A(B \setminus \Lbag z \Rbag, z) : z \in B \Rbag \tag{4}$$

$$p_{z^*} = \frac{|\alpha \in S : \alpha >= \alpha_{z^*}|}{|S|} \tag{5}$$

In the classification context, we 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 above a given significance level the null hypothesis is disproved and $\hat{y}$ is accepted as a valid prediction. Transcend [20] computes *per-class thresholds* to use as significance levels (§V).

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. Thus, when designing an NCM in the form given by Equation 2, the distance metric $d(z, z')$ is 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 $\hat{z}(B)$, will impact the ability of conformal evaluators to discriminate between valid and invalid predictions [38].

An *alpha assessment* [20] can be used to empirically evaluate how appropriate an NCM is for a given dataset by plotting 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 distributions of correct and incorrect predictions suggest a viable threshold exists to separate them at test time. Poorly separated prediction p-values indicate an inappropriate NCM. An example of an alpha assessment on a toy dataset is shown in Figure 4d.

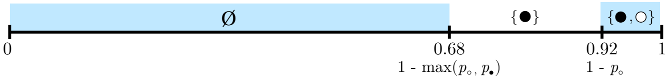


Fig. 2: The nested intervals at which labels ● and ○ are present in the output label set for a test example with per-class p-values $p_\bullet = 0.32$ and $p_\circ = 0.08$. Shaded areas outline how credibility and confidence relate to the intersection of prediction regions for which the label set contains 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 out-of-distribution and is indicative of concept drift.

Figure 1 illustrates possible NCMs for different algorithms on a toy binary classification task with existing class examples ●/○ and new test example $\ast$. The solid line delineates the decision boundary between the two classes, the dotted lines show SVM margins where applicable, and the blue shaded region captures points that are *more nonconform* (i.e., less similar) than $\ast$ 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).

C. Successfully Identifying Drift

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 $\varepsilon = 0$ (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 $\varepsilon = 1$ (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* (Figure 2).

Confidence is the greatest $1 - \varepsilon$ 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), for which they will output the class with the highest credibility. Credibility quantifies how relevant the training set is to the prediction. A low credibility indicates conformal prediction might not be a suitable framework to use with the given data because a low credibility means 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.

IV. Towards Practical Conformal Evaluation

In assessing the quality of a prediction for a new test point, there is 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 Jordaney et al. [20], 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 is 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 (§V). 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.

While the Transductive Conformal Evaluator (TCE) used in the original proposal [20] appears to perform well, it does not scale to larger datasets as a newly trained classifier is required for every training point. Consider the experiments in §VI where fitting a single instance of the underlying classifier takes 10 CPU minutes. In this case, we estimate a single run using vanilla TCE to take 1.9 CPU years.

We propose a number of novel conformal evaluators that overcome this limitation and present their advantages and disadvantages. A comparison of their runtime complexities and operational considerations are presented in Table I and §VII, respectively. Formal algorithms for their calibration and test procedures are included in Appendix F while Figure 3 provides a graphical intuition to their different calibration splits.

Note that while our illustrative examples and evaluation are given for the binary detection task, Transcendent and conformal evaluation are agnostic to the total number of classes and this is captured in the formal definitions. If multiclass NCMs cannot be derived, per-class conformal evaluators may be arranged as a *one-vs-all* ensemble.

A. Approximate TCE (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.

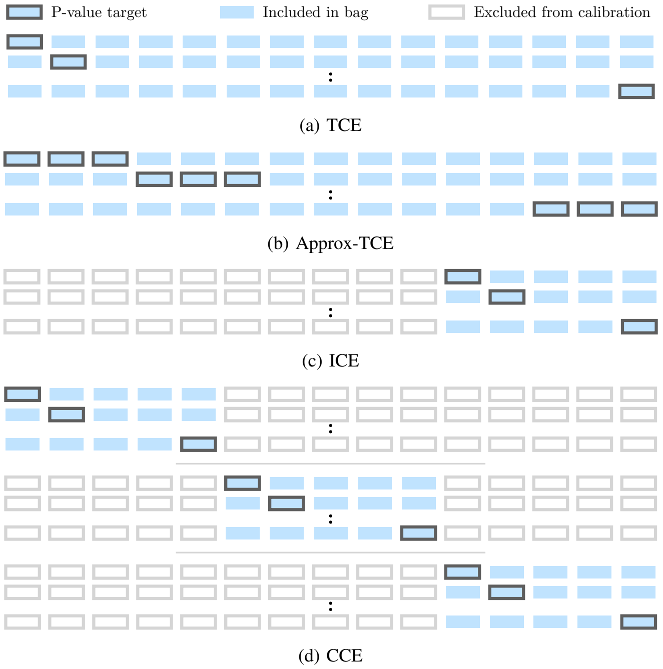
In approx-TCE, calibration points are left out in batches, rather than individually. The training set is randomly partitioned into k folds of equal size. From the k folds, one is used as the target of the calibration and the remaining $k-1$ folds are used as the bag to which those points are compared 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.

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.

This assumption is more likely to hold with algorithms with lower variance (e.g., linear models), but becomes more tenuous as the variance increases unless k increases also—sacrificing the saved computation to mitigate the statistical instability.

B. 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 [32, 46, 47]. 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



(a) TCE

(b) Approx-TCE

(c) ICE

(d) CCE

Fig. 3: Illustration of the different calibration splits employed by each of the conformal evaluators showing the target of the p-value calculation, relative points included in the bag, and points excluded from the calibration.

the proper training set, and p-values are computed for each example in the calibration set. Unlike 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.

This induces significantly less computational overhead than the TCE and approx-TCE (see Table I) and in practice is extremely fast, but 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 a TCE due to a lack of information, or better due to a lucky split.

C. 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 [48].

The training set is partitioned into k folds of equal size. So that a p-value is obtained for every training example, 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.

Finally we concatenate the p-values in a way which preserves their statistical integrity when decision quality is evaluated. We set aside the k fit underlying models and corresponding calibration sets for test time. When a new point arrives, the prediction from each classifier is 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.

TABLE I: Runtime complexities and empirical runtime for conformal evaluator 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	Runtime in §VI-B
TCE	$\mathcal{O}(n^2)$	est. 1.9 CPU yrs
Approx-TCE, $1/(1-p)$ folds	$\mathcal{O}(n/(1-p))$	46.1 CPU hrs
ICE	$\mathcal{O}(pn)$	11.5 CPU hrs
CCE, $1/(1-p)$ folds	$\mathcal{O}(pn/(1-p))$	36.6 CPU hrs

The CCE can be viewed as k ICEs, one per fold, and these ICEs can operate in parallel to reduce computation time—if the resources are available. However, there is an additional memory cost with storing the separate models.

V. Sound and Practical Transcendent

Once p-values are calculated, thresholds are derived to decide when to accept or reject new test examples. Here we revise and formalize the strategy used in Transcend [20] and propose a more efficient search strategy.

A. Calibration Phase

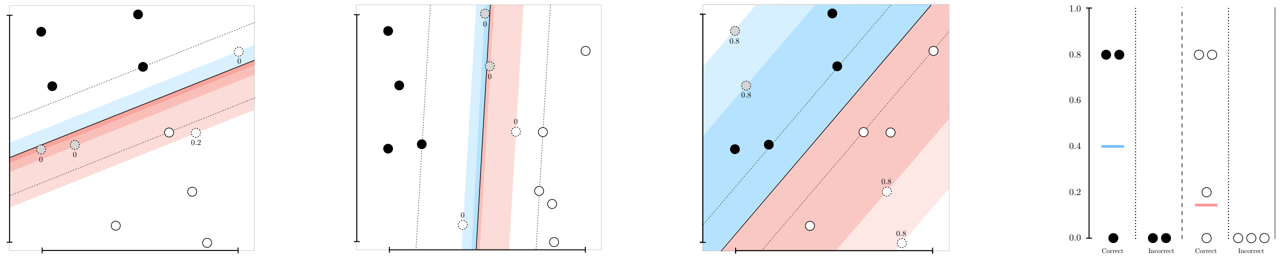
The first phase of Transcend [20] is the calibration procedure which searches for a set of per-class *credibility* thresholds

$$\mathcal{T} = \{\, \tau_y \in [0,1] : y \in \mathcal{Y} \,\}$$

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* (see §VI-C).

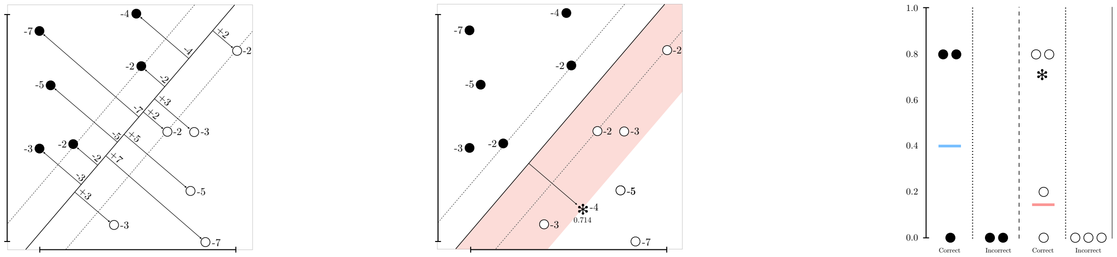
Calibration aims to answer the question: "how low a credibility is too low?", by analyzing the p-value distribution of points in a representative, preferably stationary, environment such as the training set. Which points are selected as calibration points depends on the underlying conformal evaluator, and this comes with various trade-offs (see §IV). 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 partitioned into *correct* and *incorrect* predictions that should be separated by $\mathcal{T}$. Methods to find an effective $\mathcal{T}$ can be manual (e.g., picking a quartile visually using an alpha assessment) or automated (e.g., grid search).

Figure 4 shows an example of the Transcend [20] thresholding procedure on a toy dataset composed of two classes:



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

Fig. 4: 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 §V-D 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

Fig. 5: Test-time procedure applied to a linear SVM and calibrated Transcend [20] with distances from hyperplane and corresponding nonconformity 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 4). As the p-value of the new point is greater than the threshold for the predicted class, the prediction is accepted.

● and ○. A linear SVM is paired with a TCE (§IV) 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 4a, 4b, and 4c. 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 be 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 4d shows the distribution of p-values for correctly and incorrectly predicted calibration points for classes ● and ○. Given the size of the example dataset, the assessment is computed in a *label-conditional* manner and the threshold is set at Q1 of the p-values for correctly classified points (more insight into threshold search strategies can be found in §V-D). Test points generating p-values below this threshold will be rejected.

B. Test Phase

At test time, there are $|\mathcal{Y}|+1$ outcomes. When a new test object z^* arrives, its p-value $p^{\hat{y}}_{z^*}$ is calculated with respect to the predicted class $\hat{y}$ (label conditional). If $p^{\hat{y}}_{z^*} < \tau_{\hat{y}}$, 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 $p^{\hat{y}}_{z^*} \geq \tau_{\hat{y}}$, the prediction is accepted and the object classified as $\hat{y}$.

Figure 5 follows on from the calibration example above. Figure 5a illustrates the NCM being used: the negated absolute distance from the hyperplane. In Figure 5b, a new test example $\ast$ appears and is classified as class ○. The p-value $p^{\circ}_{\ast} = 0.714$ is calculated as the proportion of points belonging to ○ with equal or greater nonconformity scores than $\ast$. Finally, Figure 5c shows $p^{\circ}_{\ast}$ compared against the threshold $\tau_{\circ}$ and, as $p^{\circ}_{\ast} \geq \tau_{\circ}$, the prediction is accepted.

C. 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 labeled 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 predictions. When choosing rejection thresholds, it is vital to keep this cost in mind and weigh it against the potential performance gains. The Tesseract framework [35] 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 measures the cost incurred by rejections. This is important for putting the performance of kept elements in perspective—there will often be a trade-off between the amount of rejections and higher performance on kept points.

Labeling cost measures the manual effort needed to find ground truth labels for new points. While this is more pertinent to retraining strategies, it is related to the overhead associated with rejection as many may need to be manually labeled. As an example, Miller et al. [27] estimate that the labeling capacity for an average company is 80 samples per day.

D. Improving the Threshold Search

Here we model the calibration procedure as an optimization problem for maximizing a given performance metric (e.g., F_1, Precision, or Recall of kept elements). Usually this maximization is subject to some constraint on another metric, for example, it is trivial to attain high F_1 performance in kept elements by accepting very few high quality predictions, but this will cause many correct predictions to be rejected.

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

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

where $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ 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.

Given this formalization, we propose an alternative random search strategy to replace the exhaustive grid search used in the original paper [20]. In the exhaustive grid search, each possible combination of thresholds over all classes is tested systematically, considering some fixed range of variables $V = \{v : v \in [0,1]\}$. However, this suffers from the curse of dimensionality [9], 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, often many useless threshold combinations are considered (where one is either too high or too low). This failure to evenly cover subspaces of interest worsens as the dimensionality increases [10], making it especially problematic for multiclass classification. The granularity for V can be chosen manually based on intuition, however this results in 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 [10]. 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 interval $[0, 1]$, keeping track of the thresholds that maximize our chosen metric given the constraints (see §V-D). The search continues until either of two conditions are met. A limit is set on the number of iterations, determined by the time and resources that are available for the calibration. Intuitively a higher limit will increase the likelihood of finding better thresholds and so acts as the upper bound of the optimization. Secondly, a stop condition can be set. 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 certain number of consecutive iterations. Note that this search procedure can be easily parallelized.

We empirically compare the two search strategies in §VI-D.

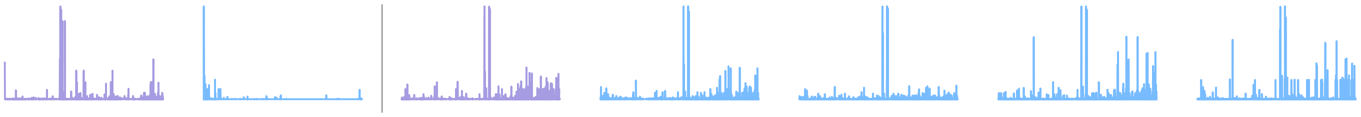
VI. Experimental Evaluation

We evaluate our novel evaluators when faced with gradual concept drift caused by the evolution of real malicious Android apps over time (§VI-B), the performance gained by including confidence scores (§VI-C), how our random search implementation fares against exhaustive search (§VI-D), how the evaluators compare to alternative methods (§VI-E), and perform on PE and PDF malware domains (§VI-F).

A. Experimental Settings

Prototype. We implement Transcendent as a Python library encompassing out new proposals as well as the functionality of the original Transcend [20]. We release the code as open source—note that this is the first publicly available implementation of Transcend in any form.

Dataset. We first focus on malware detection in the Android domain. We sample 232,848 benign and 26,387 malicious apps from AndroZoo [2]. This allows us to demonstrate efficacy when faced with a natural, surreptitious concept drift. The apps span 5 years, from Jan 2014 through to Dec 2018. We use the Tesseract [35] framework to temporally split the dataset, 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



(a) Training [20] (b) Test [20] (c) Training (d) Test at 1 year (e) Test at 2 years (f) Test at 3 years (g) Test at 4 years

Fig. 6: Frequency distributions of features depicting covariate shift between training and test malware examples. The data from Jordaney et al. [20], displayed in (a) and (b), shows a sudden and significant shift, while the data used in §VI, displayed in (c–g), shows a more subtle, natural drift occurring over time.

and testing is evaluated over the concept drift that occurs over the remaining period on a month-by-month basis.

Eliminating Sampling Bias. The original evaluation of Transcend [20] artificially simulated concept drift by fusing two datasets: Drebin [6] and Marvin [25], a process which may have induced experimental bias [7] and made it easier to detect drifting examples. Figure 6 shows a visibly significant covariate shift in the distribution of features for training and test malware examples from Jordaney et al. [20], with a Kullback-Leibler (KL) divergence [24]—an unbounded measure of distribution difference—of 696.66. The covariate shift in our dataset is much more subtle and natural over time, with an average KL divergence of 189.55 between each training and test partition. From this we conclude that the distributions were significantly more different in the original evaluation than would be expected in naturally occurring concept drift, which would have made it easier to detect drifting examples.

Classifier. For the underlying classifier, we use Drebin [6] which has been shown to achieve state-of-the-art performance if a retraining strategy is used to remediate concept drift [35]. Due to this, we hypothesize that if Transcend [20] is used to reject drifting points, Drebin will be able to classify the remaining points with high accuracy. Drebin uses a linear SVM and a binary feature space where components (activities, permissions, URLs, etc) are represented as present or absent.

Calibration. To optimize the thresholding, we maximize the F_1 of all kept elements for a rejection rate less than 15%. These metrics are computed in aggregate for each time period of the temporal evaluation. On our dataset, this would amount to an average rejection of ~20 samples a day, well below the estimated labeling capacity of 80 a day suggested by Miller et al. [27]. However, we note that these constraints may need to be adjusted according to specific operational requirements, for example, it may be more appropriate to minimize the rejection rate while sacrificing F_1 for kept elements. For the random search we use 100,000 random iterations with early stopping after 3,000 consecutive events without improvement. For approx-TCE and CCE we calibrate using $k = 10$ folds.

B. Novel Conformal Evaluators

Here we compare the novel conformal evaluators of TRANSCENDENT. As vanilla TCE is not feasible for this experiment setting due to the size of the training set (§IV), we use approx-TCE as a stand-in, however we provide a minimal experiment in Appendix C to show that the expected performance difference between vanilla TCE and our evaluators is negligible.

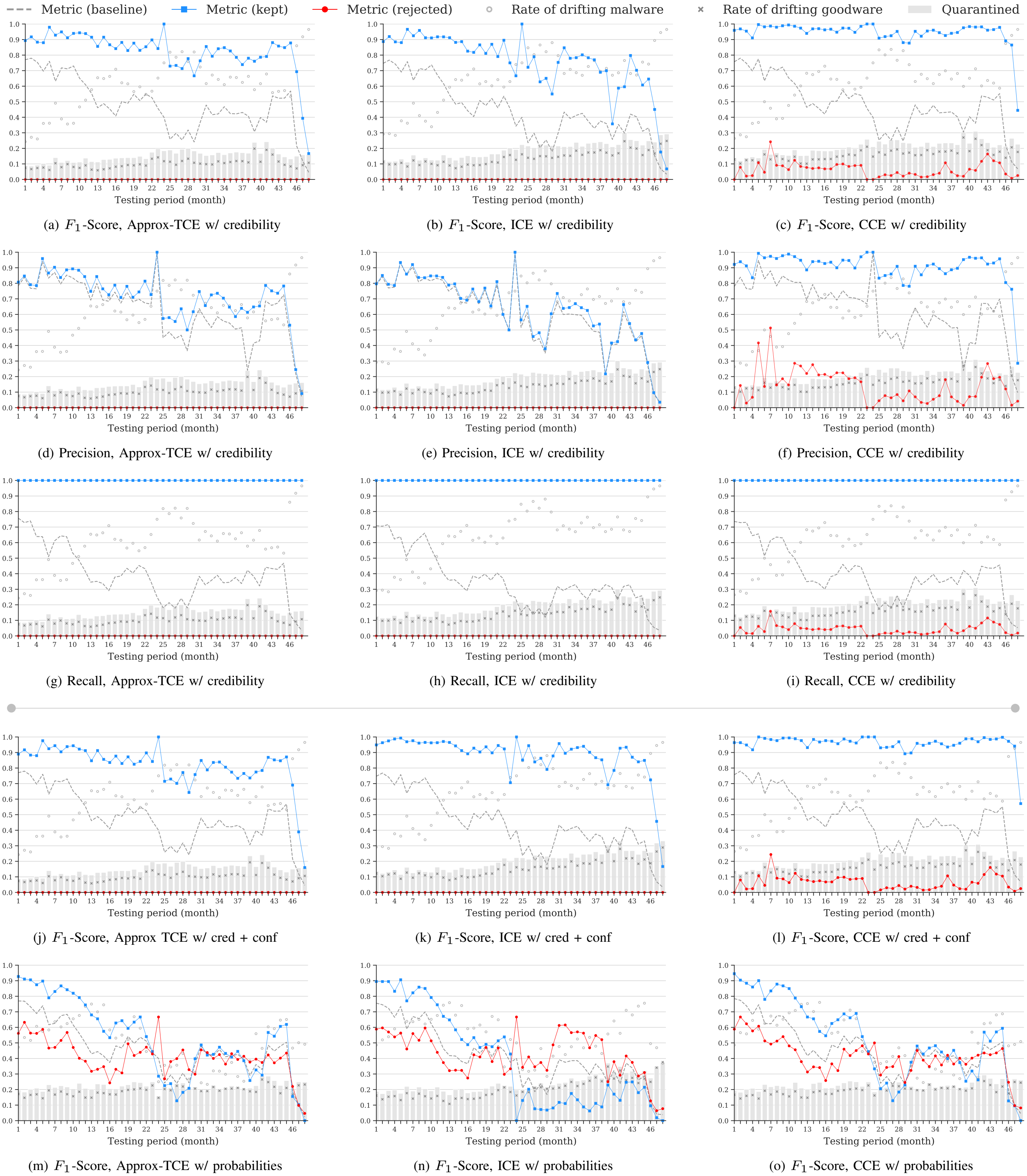
Performance Metrics. Figure 7 shows the the F_1, Precision, and Recall (rows 1–3) for each of the novel evaluators (columns). The middle dashed line shows the baseline performance when no rejection is enforced. This is the performance decay caused by concept drift present in the dataset that results from an evolving malicious class. Note that classifiers degrade rapidly, becoming worse than random in under one year.

The upper solid line shows the performance of kept elements, those with test p-values that fall above the threshold of their predicted classes. While decay is still present, approx-TCE and ICE are able to maintain $F_1 > 0.8$ for two years, doubling the lifespan of the model. Note that the sudden drop in performance of the last three months is likely an artifact of the fewer examples crawled by AndroZoo in those months.

The lower solid line shows the performance of rejected elements. Low metrics mean the rejected elements would have been incorrectly classified by the underlying classifier and were rightfully rejected, while high metrics means rejections were erroneous. Approx-TCE and ICE have F_1, Precision, and Recall of 0 for rejected elements for every test month meaning that all rejected elements 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 also slightly higher for rejected elements, indicating that a small proportion of 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 parameter can be altered during a deployment, rather than being set at calibration. This allows for some adaptability, such as 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 is very high (e.g., withholding benign emails in spam detection, or disabling legitimate user accounts in fake account detection). A further empirical analysis of the effect of the majority vote conditions is included in Appendix D.

Rejection Rates. Gray bars show the proportion of rejected test elements. In each case, rejections begin close to the rate used for calibration before slowly rising each year, averaging 26.45 samples per day. While rejection rates may appear high, these are symptomatic of rising concept drift and deteriorating performance in the underlying classifier and are



(a) F_1-Score, Approx-TCE w/ credibility

(b) F_1-Score, ICE w/ credibility

(c) F_1-Score, CCE w/ credibility

(d) Precision, Approx-TCE w/ credibility

(e) Precision, ICE w/ credibility

(f) Precision, CCE w/ credibility

(g) Recall, Approx-TCE w/ credibility

(h) Recall, ICE w/ credibility

(i) Recall, CCE w/ credibility

(j) F_1-Score, Approx TCE w/ cred + conf

(k) F_1-Score, ICE w/ cred + conf

(l) F_1-Score, CCE w/ cred + conf

(m) F_1-Score, Approx-TCE w/ probabilities

(n) F_1-Score, ICE w/ probabilities

(o) F_1-Score, CCE w/ probabilities

Fig. 7: Performance for the three proposed conformal evaluators (columns) using different quality metrics. The first three rows show F_1-Score, Precision, and Recall, respectively, of the different evaluators using credibility p-values. The lower two rows show F_1-Score using a combination of credibility and confidence p-vaules, and probabilities, respectively. 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, while the x and o markers show the proportion of *ground truth* malware and goodware that was identified as drifting and quarantined, respectively.

often preferable to taking incorrect actions on False Positives and False Negatives. In an extreme case where a classifier always predicts the incorrect label, rejection rates could reach 100% but the F_1 of rejected elements would be 0%. The gray markers show the proportion of ground truth malware and goodware that are rejected each period, illustrating the evaluators' perception of drift in that class. Strikingly, for our evaluators the drift rate of the malicious class is *inversely correlated* to the performance loss in the baseline, while the drift rate for goodware is relatively stable. This supports our hypothesis that performance decay is mostly driven by evolution in the malicious class. We reiterate that in the case of Approx-TCE and ICE, the low F_1 of rejected elements indicates that *all* of the rejected malware would have evaded the classifier if they had not been identified as drifting.

Runtime. The runtime of the conformal evaluators during this experiment match what would be expected from their relative complexities (cf. Table I). The ICE is the quickest at 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. The Approx-TCE took 46.1 CPU hours. As discussed, vanilla TCE was computationally infeasible, but we estimate a runtime of 1.9 CPU years, considering that the time required to fit the underlying classifier once is $\sim$10 minutes and the classifier must be trained once *for each* training example.

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.

C. Credibility, Confidence, and Probabilities

Here we compare the performance under different quality metrics. The exact performance over time for all settings discussed in this subsection is reported in Table II.

Credibility with Confidence. Intuitively, including confidence thresholds when evaluating a classifier prediction would be beneficial as confidence represents how certain the classifier is 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 what further gain confidence could provide. Here we test this by evaluating the conformal evaluators under the same conditions as §VI-B, using per-class thresholds for both credibility and confidence.

Figure 7 compares the F_1 for each conformal evaluator (columns) using different thresholding metrics (rows 1, 4–5). The upper blue line shows the performance of kept elements while the lower red line shows the performance of rejected elements. The gray dashed line depicts the baseline performance when no rejection mechanism is used.

The penultimate row shows the F_1 when confidence is included. Performance for the approx-TCE and CCE is relatively unchanged, but is markedly improved for the ICE with degradation 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 computation required to find thresholds is higher than with credibility only—equivalent to doubling the number of classes. We conclude that the performance gain from including confidence is situationally dependent; although it will improve the accuracy of an ICE, a CCE will often provide the same accuracy with comparable calibration time.

Probabilities. The final row of Figure 7 shows the F_1 when the classifiers' output probabilities are used for thresholding, rather than generating per-class p-values for each calibration and test point. For each evaluator, the same training and calibration split is used as with p-values, to ensure a fair comparison. The plot shows probabilities alone offer a very small improvement for kept elements over the baseline in the first year, becoming increasingly volatile as the concept drift becomes more severe. Additionally, the perceived drift rate for each class has no relation to the baseline performance loss, indicating that the root cause of the drift is not identified. This shows the statistical support offered by the conformal evaluator's p-value computation is significant and justifies the additional computational overhead that it induces.

D. Full Grid Search vs Random Search

Here we evaluate our random search implementation (§V-D) compared to the full grid search used in the original proposal [20]. We show the random search can find high quality calibration thresholds more efficiently than the full search.

Due to the full grid search cost, here 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 for an acceptable rejection rate of less than 15%. To ensure the baseline discovers high quality thresholds we use a fine granularity grid covering 1,317,520 combinations of thresholds. For random search we set an upper limit of 10,000 trials.

Table III compares the performance without rejection, with rejection thresholds from the full grid search, and with rejection thresholds from random search. Note there is no significant performance difference between the two strategies, but the random search is 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—which are difficult to choose beforehand.

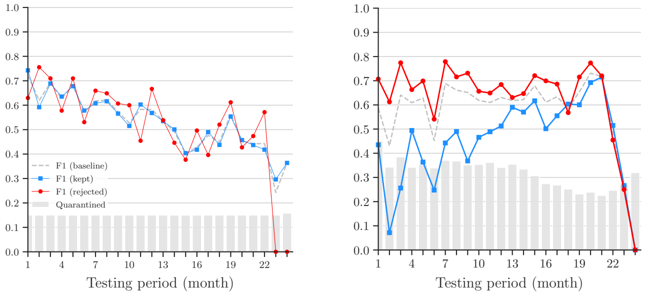
E. Comparison to Prior Approaches

To provide further context on the performance of TRANSCENDENT, we compare against two closely related state-of-the-art approaches: Linusson et al. [26] (which we denote CP-Reject) and DroidEvolver [50].

CP-Reject [26]. The first approach is a recent method for performing rejection using conformal prediction. For a given test set and hyperparameter k, CP-Reject aims to output the

TABLE II: Area Under Time (AUT) of F_1 performance with respect to concept drift over the 48 month test period for different quality metrics: credibility, credibility with confidence, and probabilities (cf. Figure 7). We aim to *maximize* the metrics of kept elements and *minimize* the metrics for rejected elements.

		Approx-TCE	**ICE**	**CCE**
Baseline	AUT(F_1 w/ credibility, 48m)	.480	.440	.483
	AUT(F_1 w/ cred + conf, 48m)	.480	.440	.483
	AUT(F_1 w/ probability, 48m)	.456	.405	.455
Kept Elements	AUT(F_1 w/ credibility, 48m)	.829	.762	.950
	AUT(F_1 w/ cred + conf, 48m)	.822	.887	.962
	AUT(F_1 w/ probability, 48m)	.531	.388	.532
Rejected Elements	AUT(F_1 w/ credibility, 48m)	.000	.000	.064
	AUT(F_1 w/ cred + conf, 48m)	.000	.000	.063
	AUT(F_1 w/ probability, 48m)	.410	.426	.410



(a) CP-Reject [26]

(b) DroidEvolver [50]

Fig. 8: F_1-Score over time for two prior approaches with mechanisms similar to TRANSCENDENT (cf. Figures 7b and 7c).

largest possible set of predictions containing on average no more than k errors, while rejecting test objects for which it is too uncertain. The training and calibration dataset splits are the same as we use for the ICE setting; however while TRANSCENDENT makes decisions on individual test objects as they appear, CP-Reject operates *a posteriori* on a batch of test inputs and predictions. Given this advantage, to ensure a fair comparison we test on each month with k set to the 85th percentile which ensures a rejection rate of 15%—the same rejection rate TRANSCENDENT is calibrated for. The underlying classifier is a random forest classifier with 100 trees and the conformal prediction NCM is the maximum margin between the output probability for the predicted class and the output probabilities for all other classes.

DroidEvolver [50]. The second approach is a state-of-the-art Android malware detector designed for drift *adaptation*, but that includes a rejection component, in which the drift identification mechanism is inspired by the original Transcend [20]. DroidEvolver is built on an ensemble of five linear online learners, with a weighted sum as the ensemble decision function. For each new test object a *juvenilization indicator* (JI) score is computed per model as the proportion of apps in a fixed-size buffer of previously encountered apps, of the same class, that have decision scores greater than the new object. An object is marked as drifting when the JI score falls outside of a precalibrated range and the corresponding decisions are rejected, i.e., excluded from the weighted sum which is used to pseudo-label and update with the drifting point. The ongoing performance of the system relies on the quality of the pseudo-labels and thus indirectly on the quality of the drift identification. The JI scores are very similar to the credibility p-values from conformal evaluation, with the computational complexity of full TCE being addressed by using the small fixed-size app buffer: drift identification should be effective so long as the app buffer is representative of the overall data population. Due to this relationship, it is informative to compare against TRANSCENDENT.

Results. Figure 8 shows the F_1 performance of CP-Reject and DroidEvolver trained and calibrated on the first year of the dataset and tested on the two subsequent years at monthly intervals. This can both be compared to the first 24 months of ICE and CCE results of Figures 7b and 7c. For CP-Reject, the similar F_1 performance for kept, rejected, and baseline predictions indicate that it is unable to distinguish between drifting and non-drifting points. Although it may effectively reject low-quality predictions in a stationary environment, conformal prediction relies heavily on the exchangeability assumption, which is violated in this dataset. To obtain a prediction, CP-Reject follows conformal prediction principles and outputs the class with the highest credibility (see §III-C), but we argue this output is not trustworthy under drifting conditions. Conversely in conformal *evaluation*, by decoupling the prediction of the underlying classifier from the rejection mechanism and directly interpreting the credibility as a measurement of drift when comparing it to the calibrated thresholds, we can more effectively detect poor quality predictions.

While the detection performance of DroidEvolver is mediocre on this dataset, the pseudo-labeling update mechanism manages to stabilize the system against the impact of drift up until the last four months. After this, performance deteriorates due to the poor quality of pseudo-labels used for updating the online models—as DroidEvolver uses predicted labels as pseudo-labels, the negative feedback loop is difficult to recover from. Surprisingly, the drift identification mechanism rejects more correct predictions than it keeps for each test period. We posit that the small app buffer fails to sufficiently represent the true app population, which may in turn lead to the negative feedback loop in the later months. Although much more extreme here, this informational inefficiency is also responsible for the variability we see when using ICEs—different dataset splits may be more or less representative of the true distribution and result in better or worse accuracy, a phenomenon that is mitigated by using a CCE. These limitations, among others, have recently been explored in concurrent work by Kan et al. [21].

F. Beyond Android Malware and SVMs

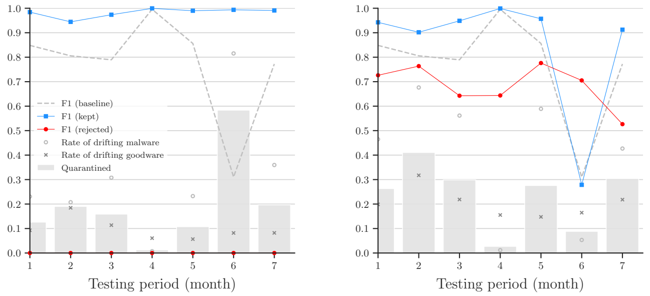
While TRANSCENDENT and conformal evaluation are agnostic to the underlying classifier and feature space, we have so far focused on detecting Android malware with a linear SVM. Here we demonstrate the performance beyond this setting. To simplify the axes of comparison, we apply an ICE to each setting, using credibility p-values and random search for threshold calibration with the same constraints as before.

Windows PE malware with GBDT. We take examples from the EMBER v2 dataset [3] spanning 2017, containing 47,888 benign and 69,202 malicious executables (labeled as having 40+ VirusTotal AV detections). The feature space contains a diverse set of features which can be categorized as either parsed features (e.g., header information), histograms (e.g., byte-value histograms), and printable strings (e.g., URL frequency). As the underlying classifier, we use gradient boosted decision trees (GBDT) [19] as in Anderson and Roth [3], and for the NCM we use the output probability for the predicted class, negated for positive predictions. We train on executables from the first five months and test on the remaining.

PDF malware with RF. We use examples from the Hidost dataset [41] spanning five weeks in Aug–Sep 2012, consisting of 181,792 benign and 7,163 malicious files (labeled as having 5+ VirusTotal AV detections). The feature space is created by statically parsing the PDF files to extract structural paths in the PDF hierarchy that map to boolean or numeric feature values, such as the presence of certain PDF objects or metadata such as the number of pages. As the underlying classifier we use a random forest (RF) classifier following Srndic and Laskov [41]. As the NCM we use the proportion of decision trees that disagree with the prediction of the ensemble (as illustrated in Figure 1e). Interestingly, a major contribution of the Hidost feature space in contrast to prior approaches [e.g., 40] is that similar features are consolidated in order to be *more robust to drift*. This means the distribution should be relatively stationary compared to the Android dataset and will allow us to test whether TRANSCENDENT is able to make effective decisions on prediction quality when drift is less severe.

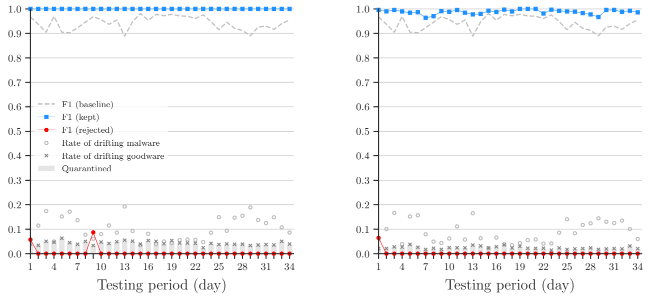
Note that we are unable to find authoritative measurements for the expected class balance for PE and PDF malware in the wild as we are for Android malware, so we defer to the class balance in the original datasets. This may result in a slight spatial bias if the class balance is unrealistic [35], however all approaches will be affected equally. Additionally, we can here examine whether the class balance affects the ability for TRANSCENDENT to identify low quality predictions.

Results. The results for Windows PE malware (Figure 9) are consistent with those on Android data. TRANSCENDENT outperforms probabilities alone which tend to reject many otherwise correct predictions. In particular, a large spike in drifting malware affects month six which probabilities are unable to cope with, while TRANSCENDENT raises the rate of rejections accordingly without making any additional errors.



(a) ICE, credibility (b) ICE, probabilities

Fig. 9: F_1-Score, EMBER Windows PE malware [3] and GBDT.



(a) ICE, credibility (b) ICE, probabilities

Fig. 10: F_1-Score, Hidost PDF malware [41] and RBF SVM.

As noted earlier, the PDF dataset gives us the opportunity to evaluate TRANSCENDENT on a relatively stationary distribution. As expected, thresholding using probabilities is much more effective than it is in a drifting setting, however it under-rejects compared to TRANSCENDENT, which is able to find thresholds that push the F_1 of kept predictions to 1.0 while rejecting almost entirely incorrect predictions. Exceptions to this are months one and nine, in which a small quantity of true positive predictions are rejected. However this is anomalous (i.e., it does not continue as drift increases) and could be mitigated by calibrating with a constraint on the F_1 of rejected samples rather than the F_1 of kept examples alone. From this we conclude that TRANSCENDENT is useful for maximizing the potential of a high-quality robust classifier, and does not rely on relatively severe drift—as present in the Android dataset—to detect low quality decisions. To this end TRANSCENDENT can be combined with robust feature spaces which is an orthogonal direction to combating concept drift.

As an additional result we observe that TRANSCENDENT outperforms CP-Reject for both domains, with more details reported in Appendix B.

VII. OPERATIONAL CONSIDERATIONS

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

TABLE III: Performance of optimal thresholds discovered using a full grid search vs. random search. Random search discovers thresholds equivalent to the full grid search but with two orders of magnitude fewer trials (§VI-D).

	FPs	FNs	Prec.	Rec.	F_1	#Trials
No rejection	3,529	19,486	0.98	0.92	0.95	N/A
Full grid	2,187	0	0.99	1.00	0.99	1,317,520
Random	3,259	0	0.98	1.00	0.99	10,000

Transcendent in a Detection Pipeline. Transcendent has particular applications in detection tasks where there is a high cost of individual False Positives, (e.g., spam [31], malware [6, 15, 40], fake accounts [13, 14]). 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 (§VI), we make the following recommendations for Transcendent deployments:

- Transcendent 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 is 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 (§VI-C), it requires greater calibration time.
- Random search is preferred over grid search as it finds similarly effective thresholds at significantly lower cost.
- Rising rejection rates should be interpreted as a signal that the underlying model is degrading. This signal can be used to trigger model retraining or other remediation strategies.
- Guidance on tuning calibration thresholds and an example of using alternative constraints is presented in Appendix E.

VIII. Related Work

Conformal evaluation is based on conformal prediction theory, a mechanism for producing predictions that are correct with some guaranteed confidence [38]. Additionally, the ICE and CCE are inspired by inductive [32, 46, 47] and cross-conformal predictors [48], 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. [47] and the conformal evaluation of Jordaney et al. [20] and show how the violation of conformal prediction's assumptions is detected and exploited by Transcend [20] to detect concept drift.

That 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 introduced experimental bias [7, 35] (§VI). 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 originally employed was not practical for real-world deployments, which we rectify by proposing the ICE and CCE.

CADE [51] focuses on *explaining* drift and relies on contrastive learning to perform a distance-based feature transformation which results in more homogenous class clusters relative to which outliers are easier to detect. They compare against using conformal evaluation as an active learning query strategy using an arbitrary NCM and *without* Transcend [20] thresholding. We believe CADE and Transcendent are orthogonal and that such feature transformations can be used to devise stronger NCMs—we leave the development of optimal NCMs for active learning as future work.

Other works have explored alternative solutions to tackling concept drift. As described in §VI-E, DroidEvolver [21, 50] is a malware detection system motivated by Transcend [20] 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. However, we find the drift identification mechanism is inferior to Transcendent and leads to a negative feedback loop. Other solutions solely adapt to concept drift without using rejection: DroidOL [29] and Casandra [30] use online learning to continually retrain the models, with API call graphs as features. Like all online-trained neural networks, these are susceptible to catastrophic forgetting [18], where performance degrades on older examples as the model attempts to adapt to the new distribution. Pendlebury et al. [35] present a comparison of different strategies for combating concept drift, including rejection, incremental retraining, and online learning, illustrating the advantages and disadvantages of each. Semi-supervised techniques may be used to reduce the labeling burden of such strategies as drift increases, as recently demonstrated for intrusion detection by Andresini et al. [5].

The related task of detecting *adversarial examples* [11, 36, 44] is addressed by Sotgiu et al. [39], who propose a rejection strategy for neural network-based classifiers that identifies anomalies in an input's latent feature representation at different layers of the neural network. Additionally, Papernot and McDaniel [33] combine a conformal predictor with a k-Nearest Neighbor algorithm to identify low-quality predictions that are indicative of adversarial inputs. However, both methods are restricted to deep learning-based image classification.

IX. Conclusion

We provide a thorough formal treatment of Transcend [20] which acts as the *missing link* between conformal prediction and conformal evaluation. We propose TRANSCENDENT, a superset of the original framework which includes novel conformal evaluators that match or surpass the original performance while significantly decreasing the computational cost. We show TRANSCENDENT outperforms the existing state-of-the-art approaches while generalizing across different malware domains and exploring realistic operational settings.

We envision these improvements will enable researchers and practitioners alike to make use of conformal evaluation to build rejection strategies to improve their security detection pipelines. To this end, we release our implementation of TRANSCENDENT, making Transcend [20] and conformal evaluation available to the community for the first time.

Acknowledgements

This research has been partially sponsored by the UK EP/P009301/1 EPSRC research grant.

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 *International Symposium on Engineering Secure Software and Systems (ESSoS)*, 2015.

[2] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. Androzoo: Collecting millions of android apps for the research community. In *ACM International Conference on Mining Software Repositories (MSR)*, 2016.

[3] H. S. Anderson and P. Roth. EMBER: an open dataset for training static PE malware machine learning models. *CoRR*, abs/1804.04637, 2018.

[4] 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.

[5] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro. INSOMNIA: towards concept-drift robustness in network intrusion detection. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, 2021.

[6] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: effective and explainable detection of android malware in your pocket. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[7] 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. In *USENIX Security Symposium*, 2022.

[8] P. L. Bartlett and M. H. Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research (JMLR)*, 2008.

[9] R. Bellman. *Adaptive Control Processes - A Guided Tour (Reprint from 1961)*. Princeton University Press, 2015.

[10] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)*, 2012.

[11] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 2018.

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

[13] 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 *Network and Distributed System Security Symposium (NDSS)*, 2015.

[14] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[15] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert. ZOZZLE: fast and precise in-browser javascript malware detection. In *USENIX Security Symposium*, 2011.

[16] 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 Security and Privacy Workshops (SPW)*, 2016.

[17] S. Edunov, M. Ott, M. Auli, and D. Grangier. Understanding back-translation at scale. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[18] R. M. French and N. Chater. Using noise to compute error surfaces in connectionist networks: A novel means of reducing catastrophic forgetting. *Neural Computation*, 2002.

[19] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

[20] 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*, 2017.

[21] Z. Kan, F. Pendlebury, F. Pierazzi, and L. Cavallaro. Investigating labelless drift adaptation for malware detection. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, 2021.

[22] 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 *ACM Workshop on Artificial Intelligence and Security (AISec)*, 2015.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 2017.

[24] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 1951.

[25] M. Lindorfer, M. Neugschwandtner, and C. Platzer. MARVIN: efficient and comprehensive mobile app classification through static and dynamic analysis. In *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2015.

[26] H. Linusson, U. Johansson, H. Boström, and T. Löfström. Classification with reject option using conformal prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. Springer, 2018.

[27] 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 *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

[28] J. G. Moreno-Torres, T. Raeder, R. Alaíz-Rodríguez, N. V. Chawla, and F. Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 2012.

[29] A. Narayanan, Y. Liu, L. Chen, and J. Liu. Adaptive and scalable android malware detection through online learning. In *International Joint Conference on Neural Network (IJCNN)*, 2016.

[30] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence (TETCI)*, 2017.

[31] 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 (CCS)*, 2017.

[32] H. Papadopoulos. Inductive conformal prediction: Theory and application to neural networks. In *Tools in Artificial Intelligence*. 2008.

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

[34] F. Pendlebury. *Machine Learning for Security in Hostile Environments*. PhD thesis, University of London, 2021.

[35] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSERACT: eliminating experimental bias in malware classification across space and time. In *USENIX Security Symposium*, 2019.

[36] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. Intriguing properties of adversarial ML attacks in the problem space. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.
[37] Plato. *The Symposium*. c. 385–370 BC. Penguin Classics edition published 1999, translated by Christopher Gill.
[38] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research (JMLR)*, 2008.
[39] A. Sotgiu, A. Demontis, M. Melis, B. Biggio, G. Fumera, X. Feng, and F. Roli. Deep neural rejection against adversarial examples. *EURASIP Journal on Information Security*, 2020.
[40] N. Srndic and P. Laskov. Detection of malicious PDF files based on hierarchical document structure. In *Network and Distributed System Security Symposium (NDSS)*, 2013.
[41] N. Srndic and P. Laskov. Hidost: a static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security*, 2016.
[42] 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 Systems With Applications*, 2014.
[43] 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 *ACM Conference on Data and Applications Security and Privacy (CODASPY)*, 2017.
[44] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR (Poster)*, 2014.
[45] 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. In *USENIX Security Symposium*, 2019.
[46] V. Vovk. Conditional validity of inductive conformal predictors. *Journal of Machine Learning Research (JMLR)*, 2013.
[47] V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic learning in a random world*. Springer-verlag New York Inc., 2010.
[48] V. Vovk, I. Nouretdinov, V. Manokhin, and A. Gammerman. Cross-conformal predictive distributions. In *Workshop on Conformal Prediction and its Applications (COPA)*, 2018.
[49] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, and J. Lu. Deepintent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.
[50] K. Xu, Y. Li, R. H. Deng, K. Chen, and J. Xu. Droidevolver: Self-evolving android malware detection system. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.
[51] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang. CADE: detecting and explaining concept drift samples for security applications. In *USENIX Security Symposium*, 2021.
[52] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *International Conference on Software Engineering (ICSE)*, 2015.
[53] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang. Enhancing State-of-the-Art Classifiers with API Semantics to Detect Evolved Android Malware. In *ACM Conference on Computer and Communications Security (CCS)*, 2020.

Appendix

A. Symbol Table

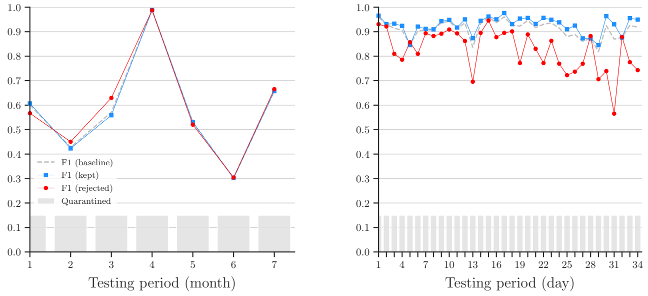
Table IV reports the major symbols and abbreviations used throughout the paper.

B. Additional CP-Reject Results

In §VI-F we demonstrate how Transcendent can apply to other classifiers and domains, comparing the performance of an ICE using credibility against using probabilities alone, for both PE and PDF malware (Figures 9 and 10). Here we show in Figure 11 additional results to compare against the prior rejection approach CP-Reject (we exclude DroidEvolver as it is specific to Android malware). Similar to the results on the Android dataset (§VI-E), the overall ability for CP-Reject to distinguish between drifting and non-drifting points is poor on the PE malware dataset. For the PDF malware dataset, which exhibits much less drift, CP-Reject is significantly more effective, which supports the hypothesis that it is the violation of conformal prediction's exchangeability assumption which results in the lower performance on the Android and PE datasets. Nevertheless, Transcendent with credibility (and even probabilities) outperforms CP-Reject in this setting also (cf. Figure 10).

TABLE IV: Table of symbols and abbreviations.

Symbol	Description
$\mathcal{X}$	Feature space $\mathcal{X} \subseteq \mathbb{R}^n$.
$\mathcal{Y}$	Label space.
z	Example pair $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$.
z^*	Previously unseen test example.
$\hat{y}$	Predicted class $g(z^*)$.
a_z	Nonconformity score output by an NCM for z.
p_z	Statistical p-value for z.
p_z^y	Statistical p-value for z, calculated with respect to class $y \in \mathcal{Y}$ (used in *label conditional* calculations).
τ_y	A 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 $\wr z_1, z_2, ..., z_n \wr$.
d	Distance function $d(z, z')$.
$\hat{z}$	Point predictor $\hat{z}(B)$.
A	Nonconformity measure (NCM) usually composed of a distance function and point predictor.
S	Collection of nonconformity scores computed in elements of B, relative to other elements in B, $S = \wr A(B \setminus \wr z \wr, z) : z \in B \wr$.
g	Classifier $g : \mathcal{X} \longrightarrow \mathcal{Y}$ that assigns object $\boldsymbol{x} \in \mathcal{X}$ to class $y \in \mathcal{Y}$. Also known as the *decision function*.
ε	Significance level used in conformal prediction to define prediction region with confidence guarantees.
NCM	Nonconformity measure.
TCE	Transductive Conformal Evaluator.
ICE	Inductive Conformal Evaluator.
CCE	Cross-Conformal Evaluator.



(a) EMBER Windows PE malware [3]

(b) Hidost PDF malware [41]

Fig. 11: F_1-Score of CP-Reject [26] on alternative malware datasets.

C. Full Vanilla TCE on EMBER Subset

A full scale comparison to the original TCE is not possible due to its computational complexity—recall that one classifier must be trained for each example in the training set. However, it is informative to perform a small-scale experiment as there may be settings where the vanilla TCE *is* viable, and we wish

TABLE V: AUT(F_1, $7m$) comparing vanilla TCE to our novel conformal evaluators on Windows PE malware data. To be computationally viable, 10% of the training data was randomly sampled to use for training and calibration.

	TCE	Approx-TCE	ICE	CCE
Baseline	0.68	0.70	0.45	0.69
Kept Elements	0.97	0.97	0.94	1.00
Rejected Elements	0.00	0.00	0.00	0.21

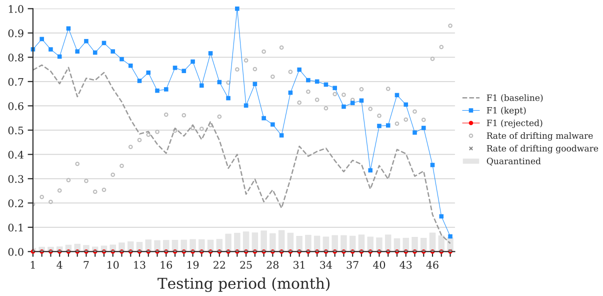


Fig. 12: F_1-Score of an ICE optimized to find calibration thresholds that minimize the rejection rate with F_1-Score no less than 0.8. These settings keep the rejection rate low (below 10%) while sacrificing the F_1 performance on kept elements (cf. Figure 7b).

to ensure that there is no significant performance difference between vanilla TCE and our novel conformal evaluators.

We perform an experiment on the Windows PE malware dataset, where 10% of the training data is randomly sampled to use for training and calibrating the evaluators (this is the largest subsample we can take given our resource constraints). We choose the PE dataset over the Android dataset due to the high dimensionality of the Android feature space that may cause instability when the number of examples is very low, and over the PDF dataset which is relatively stationary and may make it harder to discern performance differences between the different evaluators. One caveat of this subsampling is the reduced performance of the baseline for the ICE, which is due to the reduced data available to the proper training set, although TRANSCENDENT appears unaffected by this.

Table V summarizes the F_1 performance over the seven month-long test periods using the area-under-time (AUT) metric [35]. The performance difference between TCE and our evaluators in terms of distinguishing between drifting and non-drifting examples is negligible, shown by the very high AUT of kept elements and very low AUT of rejected elements. That is, there is little to no performance sacrifice when using our evaluators over the vanilla TCE. The overall trends otherwise follow those in our main Android experiments (cf. Figure 7).

D. Analysis of CCE Tuning

Here we revisit the majority vote conditions for the CCE applied to the Android malware dataset in §VI-B. The size of the quorum for the CCE affects how conservative the CCE is in accepting test examples. Figure 13 shows the performance over time summarized using the AUT metric for F_1 (a), Precision (b), and Recall (c). Note that Figure 13 omits the setting where the majority vote must be unanimous, as the CCE eventually rejects every example—causing F_1, Precision, and Recall to be undefined for kept elements. As more folds of the CCE are required to agree with each other before a decision is accepted, the CCE will reject more elements. If less folds are required, more elements will be accepted. Similarly, the quality of the rejection lessens: more elements are rejected on which the underlying classifier would not have made a mistake. Tuning the majority vote conditions on the calibration set can help find the sweet spot between the performance of kept elements, and the quality—and volume—of rejections.

E. Guidance for Choosing Calibration Constraints

In §V-D we formally describe the threshold calibration as an optimization problem in which one metric of interest is maximized or minimized given constraints on another metric. Throughout our evaluation we focus on maximizing the F_1 of kept elements, while keeping a reasonably low rejection rate. We choose 15% after taking into account the size of our dataset and using guidance from Miller et al. [27] to estimate a reasonable labeling capacity.

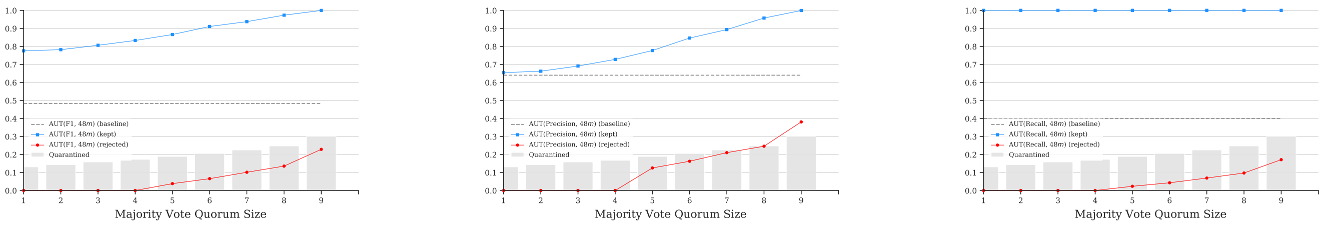
Recall that the calibration constraints are with respect to the calibration set which ideally exhibits minimal drift. It is clear from our evaluation that as concept drift becomes more severe during a deployment, constraints such as those on the rejection rate will be surpassed to some degree. This is the desired outcome—so long as the performance on rejected elements remains low (i.e., they would likely be misclassified) we would rather reject drifting examples.

Figure 12 presents an alternative to the optimization used in our previous experiments which is more appropriate if the rejection rate must be kept low. By finding thresholds that minimize the rejection rate on the calibration set with F_1-Score no less than 0.8, during deployment the rejection rate stays much lower, consistently staying below 10% even as the drift increases. Similar to how the rejection rate begins close to the calibration constraint and then increases in our previous experiments, in this setting the F_1 begins close to the calibration constraint, and then decreases. The overall effect here is that the ICE is more conservative in its rejections: while the F_1 of kept elements decreases as more incorrect predictions are accepted, the ICE rejects only those predictions that are most likely to be incorrect, keeping the F_1 of rejected elements at 0.

In summary, to estimate how many rejections will be acceptable, we advise practitioners to consider the expected volume of incoming samples, the available resources for processing quarantined examples, and the lifetime of the classifier before being retrained (as drift will likely increase during this period). Next they should identify which metrics are most important, or nonnegotiable, and use these to balance the threshold optimization. As the emergence of concept drift will likely result in the calibration values being surpassed, a ballpark is more important than the exact values.

F. Formal Calibration Algorithms

We present algorithms for our random search calibration and calibration and test procedures for TCEs, ICEs, and CCEs.



(a) AUT(F_1, 48m)

(b) AUT(Precision, 48m)

(c) AUT(Recall, 48m)

Fig. 13: AUT of performance metrics showing the effect of tuning the quorum size k of the majority vote in a CCE.

Algorithm 1: Random search threshold calibration

```
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

$\boldsymbol{t^*} \leftarrow \boldsymbol{0}$
counter $\leftarrow 0$
while counter ¡ $m$ do
    $\boldsymbol{t} \xleftarrow{\$} [0,1]^{|\mathcal{Y}|}$        ▷ Pick random thresholds
    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
        $\boldsymbol{t^*} \leftarrow \boldsymbol{t}$
    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
        $\boldsymbol{t^*} \leftarrow \boldsymbol{t}$
    counter $\leftarrow$ counter + 1
end
return $\boldsymbol{t^*}$
```

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

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

Calibration Phase
$\boldsymbol{P} \leftarrow \boldsymbol{0}$
$i \leftarrow 0$
partition $Z$ equally into $Z^{part} \leftarrow \{ Z_0', Z_1', \ldots, Z_{k-1}' \}$
foreach partition $Z'$ of $Z^{part}$ do
    $Z'' \leftarrow Z \setminus Z'$
    $g \leftarrow$ Fit $(Z'')$
    foreach $z'$ of $Z'$ do
        ▷ Predicted label
        $\hat{y} \leftarrow g(z')$
        ▷ Bag of examples with same label
        $Z_{\hat{y}}' \leftarrow \wr z \in Z' : z.y = \hat{y} \wr$
        ▷ Nonconformity score
        $\alpha_{z'} \leftarrow A(Z_{\hat{y}}', z')$
        ▷ Nonconformity scores for bag elements
        $S \leftarrow \wr A(Z_{\hat{y}}' \setminus \wr z \wr) : z \in Z_{\hat{y}}' \wr$
        ▷ Credibility p-value
        $p_{z'} \leftarrow \frac{|\alpha \in S : \alpha >= \alpha_{z'}|}{|S|}$
        $\boldsymbol{P}_i \leftarrow p_{z'}$
        $i \leftarrow i + 1$
    end
end
$\boldsymbol{t}^* \leftarrow$ Transcend.FindThresholds $(Z, \hat{\boldsymbol{Y}}, \boldsymbol{P})$

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

Algorithm 3: Inductive Conformal Evaluator (ICE)

```
Input: $Z = \Lbag z_0, z_1, \ldots, z_{n-1} \Rbag$, $n$ training examples
       $Z^* = \Lbag z_0^*, z_1^*, \ldots \Rbag$, stream of test 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
$\boldsymbol{P} \leftarrow \hat{\boldsymbol{Y}} \leftarrow \boldsymbol{0}$
$i \leftarrow 0$
$Z^{tr} \leftarrow \Lbag z_0, z_1, \ldots, z_{n-m-1} \Rbag$
$Z^{cal} \leftarrow \Lbag z_{n-m}, z_{n-m+1}, \ldots, z_{n-1} \Rbag$
foreach $z'$ of $Z^{cal}$ do
    $g \leftarrow$ Fit $(Z^{cal} \setminus \Lbag z' \Rbag)$
    ▷ Predicted label
    $\hat{y} \leftarrow \hat{\boldsymbol{Y}}_i \leftarrow g(z')$
    ▷ Bag of examples with same label
    $Z^{cal}_{\hat{y}} \leftarrow \Lbag z \in Z^{cal} : z.y = \hat{y} \Rbag$
    ▷ Nonconformity score
    $\alpha_{z'} \leftarrow A(Z^{cal}_{\hat{y}}, z')$
    ▷ Nonconformity scores for bag elements
    $S \leftarrow \Lbag A(Z^{cal}_{\hat{y}} \setminus \Lbag z \Rbag) : z \in Z^{cal}_{\hat{y}} \Rbag$
    ▷ Credibility p-value
    g $p_{z'} \leftarrow \frac{|\alpha \in S \,:\, \alpha >= \alpha_{z'}|}{|S|}$
    $\boldsymbol{P}_i \leftarrow p_{z'}$
    $i \leftarrow i + 1$
end
$\boldsymbol{t}^* \leftarrow$Transcend.FindThresholds $(Z, \hat{\boldsymbol{Y}}, \boldsymbol{P})$

Test Phase
$g \leftarrow$ Fit $(Z^{tr})$
foreach $z^*$ of $Z^*$ do
    ▷ Predicted label for test example
    $\hat{y} \leftarrow g(z^*)$
    ▷ Bag of training examples with same label
    $Z^{cal}_{\hat{y}} \leftarrow \Lbag z \in Z^{cal} : z.y = \hat{y} \Rbag$
    ▷ Nonconformity score
    $\alpha_{z^*} \leftarrow A(Z^{cal}_{\hat{y}}, z^*)$
    ▷ Nonconformity scores for bag elements
    $S \leftarrow \Lbag A(Z^{cal}_{\hat{y}} \setminus \Lbag z \Rbag) : z \in Z^{cal}_{\hat{y}} \Rbag$
    ▷ Credibility p-value
    $p_{z^*} \leftarrow \frac{|\alpha \in S \,:\, \alpha >= \alpha_{z^*}|}{|S|}$
    if $P_{z^*} < \boldsymbol{t}^*_{\hat{y}}$ then emit $0$ else emit $1$
end
```

Algorithm 4: Cross-Conformal Evaluator (CCE)

```
Input: $Z = \Lbag z_0, z_1, \ldots, z_{n-1} \Rbag$, $n$ training examples
       $Z^* = \Lbag z_0^*, z_1^*, \ldots \Rbag$, stream of test 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
$\boldsymbol{P} \leftarrow \hat{\boldsymbol{Y}} \leftarrow \boldsymbol{G} \leftarrow \boldsymbol{t}^* \leftarrow \boldsymbol{0}$
$i \leftarrow j \leftarrow 0$
partition $Z$ equally into $\{ Z'_0, Z'_1, \ldots, Z'_{k-1} \}$
foreach $j$ of $\{ 0, 1, \ldots, k-1 \}$ do
    foreach $z'$ of $Z'_j$ do
        $g \leftarrow$ Fit $(Z'_j \setminus \Lbag z' \Rbag)$
        ▷ Predicted label
        $\hat{y} \leftarrow \hat{\boldsymbol{Y}}_{j,i} \leftarrow g(z')$
        ▷ Bag of examples with same label
        $Z'_{j_{\hat{y}}} \leftarrow \Lbag z \in Z'_j : z.y = \hat{y} \Rbag$
        ▷ Nonconformity score
        $\alpha_{z'} \leftarrow A(Z'_{j_{\hat{y}}}, z')$
        ▷ Nonconformity scores for bag elements
        $S \leftarrow \Lbag A(Z'_{j_{\hat{y}}} \setminus \Lbag z \Rbag) : z \in Z'_{j_{\hat{y}}} \Rbag$
        ▷ Credibility p-value
        $\boldsymbol{P}_{j,i} \leftarrow \frac{|\alpha \in S \,:\, \alpha >= \alpha_{z'}|}{|S|}$
        $i \leftarrow i + 1$
    end
    $\boldsymbol{G}_j \leftarrow$ Fit $(Z \setminus Z'_j)$
    $\boldsymbol{T}^*_j \leftarrow$Transcend.FindThresholds $(Z'_j, \hat{\boldsymbol{Y}}_{\boldsymbol{j}}, \boldsymbol{P}_j)$
end

Test Phase
$s \leftarrow 0$
foreach $z^*$ of $Z^*$ do
    foreach $j$ of $\{ 0, 1, \ldots, k-1 \}$ do
        ▷ Predicted label for test example
        $\hat{y} \leftarrow G_j(z^*)$
        ▷ Bag of training examples with same label
        $Z'_{j_{\hat{y}}} \leftarrow \Lbag z \in Z'_j : z.y = \hat{y} \Rbag$
        ▷ Nonconformity score
        $\alpha_{z^*} \leftarrow A(Z'_{j_{\hat{y}}}, z^*)$
        ▷ Nonconformity scores for bag elements
        $S \leftarrow \Lbag A(Z'_{j_{\hat{y}}} \setminus \Lbag z \Rbag) : z \in Z'_{j_{\hat{y}}} \Rbag$
        ▷ Credibility p-value
        $p_{z^*} \leftarrow \frac{|\alpha \in S \,:\, \alpha >= \alpha_{z^*}|}{|S|}$
        ▷ Track positive evaluations
        if $P_{z^*} \geq \boldsymbol{T}^*_{j_{\hat{y}}}$ then $s \leftarrow s + 1$
    end
    ▷ Majority vote for final decision
    if $s < k/2$ then emit $0$ else emit $1$
end
```