Aurora: Are Android Malware Classifiers Reliable under Distribution Shift?

Alexander Herzog*
University College London and core64
London, United Kingdom
alexander.herzog.23@ucl.ac.uk
alex@core64.co.uk

Aliai Eusebi*
University College London
London, United Kingdom
aliai.eusebi.16@ucl.ac.uk

Lorenzo Cavallaro
University College London
London, United Kingdom
l.cavallaro@ucl.ac.uk

ABSTRACT

The performance figures of modern drift-adaptive malware classifiers appear promising, but does this translate to genuine operational reliability? The standard evaluation paradigm primarily focuses on baseline performance metrics, neglecting confidence-error alignment and operational stability. While Tesseract established the importance of temporal evaluation, we take a complementary direction by investigating whether malware classifiers maintain reliable confidence estimates under distribution shifts and exploring the tensions between scientific advancement and practical impacts when they do not. We propose Aurora, a framework to evaluate malware classifiers based on their confidence quality and operational resilience. Aurora subjects the confidence profile of a given model to verification to assess the reliability of its estimates. Unreliable confidence estimates erode operational trust, waste valuable annotation budget on non-informative samples for active learning, and leave error-prone instances undetected in selective classification. Aurora is further complemented by a set of metrics designed to go beyond point-in-time performance, striving towards a more holistic assessment of operational stability throughout temporal evaluation periods. The fragility we observe in state-of-the-art frameworks across datasets of varying drift severity suggests the need for a return to the whiteboard.

CCS CONCEPTS

• **Computing methodologies → Uncertainty in AI**; • **Security and privacy → Robustness**.

KEYWORDS

UQ Robustness, Selective Classification, Active Learning

*Both authors contributed equally to this research.

1 INTRODUCTION

Research contributions to malware classification under distribution drifts[1] center on accuracy and novelty, while marginalizing trustworthiness. As Goodhart famously stated, "When a measure becomes a target, it ceases to be a good measure." [16].

Malware classifiers experience performance degradation over time due to concept drift, caused by adversarial adaptation, the natural evolution of legitimate software, or imprecise feature spaces. This breaks the i.i.d. assumption upon which the manually engineered or learned representations depend. Therefore, existing state-of-the-art (SOTA) solutions to drift-adaptation aim to determine which observations at test time have most likely drifted, enabling targeted interventions to counteract performance degradation. Each solution implements its own distinct detection criteria. CADE [43] calculates the minimum class-centroid distance normalized by Median Absolute Deviation; Chen et al. [7] quantify embedding misalignment via averaged pairwise contrastive losses between test samples and nearest training neighbors; Transcendent [6] applies conformal prediction theory to evaluate sample nonconformity relative to calibration data of the predicted class.

Once drift has been detected, there are two possible response pathways: (1) selective classification, where a rejection threshold is applied to abstain from predicting on drifted observations [6], or (2) active learning in a continual learning set-up where observations are ranked by their "out-of-distribution (OOD) score" and routed to human labeling for subsequent model retraining [7, 43]. The selection in active learning typically follows either a budget-constrained method, where a predetermined number of top-ranked observations (e.g., 50, 100, 200) receive human labels [7], or a threshold-based method, where all observations exceeding a specified threshold are selected for annotation [43]. Selective classification prioritizes classification reliability at the cost of coverage (the proportion of accepted predictions), while active learning considers drift signals as informative for model adaptation.

These methods are typically evaluated using standard performance metrics, with performance improvements treated as sufficient evidence for model trustworthiness under realistic deployment conditions. Uncertainty calibration—the alignment between a model's confidence values and the true probability of correct decisions—is a property often neglected or sidelined. A model may perform well in terms of overall metrics, yet still exhibit poorly calibrated confidence estimates. This is problematic under distribution shifts, where predictive uncertainty becomes most consequential

---

[1]Without any loss of generality, we will refer to distribution shift and drift, and concept shift and drift, interchangeably throughout.

[34]. Indeed, deep neural networks are known to assign high confidence scores to incorrect predictions on OOD inputs [18].

Moreover, this form of gaming the metric obscures another critical flaw: OOD selection strategies themselves are rarely subjected to direct assessment. The approach fails to differentiate whether performance improvements result from optimal sample selection or from a robust adaptation model compensating for suboptimal selection. Performance gains are in fact often treated as an implicit confirmation that the underlying selection strategy is working effectively. This is an assumption that becomes even more questionable when selective strategies are based on confidence estimates that could be miscalibrated under distribution shift. This reliance on proof by indirect inference, where downstream metrics are taken to validate upstream decisions, has drawn increasing scrutiny in applied domains such as computer vision [39]. In response, reviewers require detailed ablation studies to isolate and quantify the individual contributions within a given framework.[2]

In the spirit of calls for more rigorous and contextualized understanding of existing methods, we propose Aurora, a framework to evaluate malware classifiers based on their confidence quality and operational resilience. Aurora subjects the confidence profile of a given model to verification using the Area Under the Risk Coverage Curve (AURC) to measure the quality of its confidence ranking [14]. We define a model's "confidence profile" as the behavior of both its out-of-distribution scoring function and its vanilla confidence function. In this way, we evaluate both the reliability of a classifier's out-of-distribution scoring function and its predictive uncertainty. The AURC quantifies how effectively confidence scores rank instances according to their actual probability of misclassification. Intuitively, a prediction with higher out-of-distribution score or predictive uncertainty should be more likely to be incorrect. Therefore, we expect performance to be monotonically non-decreasing when observations are rejected in order of decreasing confidence. This shows the model correctly quantifies both what samples fall outside its training distribution and when it is likely to make mistakes. If not, we are misdirecting limited annotation resources toward non-informative samples while allowing error-prone instances to remain undetected, relying on systems that do not know what they do not know. Aurora is further complemented by a set of metrics designed to move beyond point-in-time performance, enabling assessment of operational stability across temporal evaluation periods.

We used our framework to evaluate SOTA models across three different datasets with escalating drift severity. The brittleness we find compels us to reconsider whether the field has truly progressed as claimed. Our work aims to advance towards a more contextualized evaluation methodology for examining the trustworthiness of machine learning models under realistic deployment conditions.

2 MOTIVATION AND RELATED WORK

Existing solutions in Android malware detection have primarily proposed representations that improve performance under stationary conditions [3, 17, 29] or enhance robustness against temporal drift [6, 7, 33, 36, 42–44, 46]. Traditional methods leverage static or dynamic analysis to extract behavioral proxies from application code,

operating on the premise that these features preserve sufficient discriminative information. Drebin [3], a key contribution to the field, extracts eight feature sets through static analysis to capture hardware component access, permissions, app components, intents, API calls, and network addresses, encoding them as sparse binary vectors. Alternative approaches include APIGraph [44], which constructs and embeds relational API graphs to identify functionally similar clusters, and MaMaDroid [29], which models package transitions through Markov chains, employing transition probabilities as feature vectors. Emerging threats, however, introduce novel patterns that manually engineered representations frequently fail to capture due to inherent semantic limitations, driving research toward more robust representation learning techniques using deep neural networks [5, 17, 19, 25, 37].

Despite advances in representation learning [46], models usually require periodic retraining to accommodate distribution shifts [6, 7, 35]. However, this retraining process is both difficult and costly given the reliance on hard-to-obtain labels and the speed with which malware and goodware are produced. On a daily basis, for example, the AV-TEST institute identifies over 450,000 new malware samples and potentially unwanted applications [4], while Virus-Total receives more than 1 million unique software submissions [1]. This volume creates a fundamental bottleneck, as Miller et al. [30] estimated that an average organization's manual labeling capacity is limited to approximately 80 samples per day. In response to performance decay and operational constraints, SOTA malware detection frameworks adopt selective classification or continual learning methods to ensure reliable classification with minimal human intervention. [3]

Selective classification, also known as classification with rejection or learning with rejection, is a framework aiming to prevent misclassifications by providing an option not to make a prediction [20, 45]. The same problem, with theoretical foundations established by Chow in the early 1970s [8], is studied in the equivalent terms of misclassification detection [21], failure prediction [9, 47], and ordinal ranking [12, 31]. Key examples of malware detection frameworks with a rejection component include Transcendent [6], DroidEvolver [42], and CP-Reject. [27]. The main motivation underlying selective classification is to minimize the expected misclassification cost (selective risk), while maximizing coverage (the proportion of accepted predictions) [13]. Ideally, selective prediction employs a classifier equipped with a "dial" to precisely control the desired error rate while maximizing coverage [13]. This dial functions via a confidence estimator that is thresholded to decide whether a prediction is accepted.

Continual learning (CL) instead is the process of incrementally learning new information from a non-stationary stream of data. As the contents are provided incrementally over a lifetime, CL is also referred to as incremental learning or lifelong learning in much of the literature, without a strict distinction [41]. CL assumes abundant availability of labeled data throughout the learning process, an assumption that, as we mentioned earlier, rarely holds in practical settings. This has led to active continual learning (ACL), which performs active learning (AL) for CL. AL is used to select the most

---

[2]For a comprehensive discussion on the role of ablation studies in model evaluation, we refer the reader to Chapter 2.2 of [39].

[3]Note: There is conceptual and methodological overlap between selective classification and continual learning, as observations that are selected for rejection in the former can be valuable candidates for the latter.

informative subset that, once labeled, should be the most valuable for model learning [41]. The most popular AL query strategy is uncertainty sampling, which selects the observation with the most uncertain predictions, under the intuition that they are the most relevant to adjust decision boundaries. A natural baseline is to use the predicted probability of the underlying classifier, where observations with probabilities close to the decision boundary ($p \approx$ 0.5 in binary classification) fall into an "uncertainty region" where the model lacks confidence in class assignment.

Existing work in AL for malware detection each define their own "uncertainty" criteria to guide sample selection. The core idea is to select samples the model is least confident about or those that "deviate significantly from known data distribution." CADE [43], for example, computes the minimum normalized distance between a test sample and all class centroids (calculated from training samples) in a contrastive latent space, where distances are scaled by each class's Median Absolute Deviation (MAD) to account for the natural variation within classes. Instead of assessing whether the sample fits in *class A* better than *class B*, the authors assess how well the sample fits in *class A* compared to other training samples in *class A*. Samples above an empirically set threshold are considered out-of-distribution and selected for re-training a binary SVM classifier. Chen et al. (2023) [7] introduce a hierarchical contrastive learning scheme paired with a pseudo loss for sample selection. Their rationale is that contrastive learning maps similar samples to nearby vectors in embedding space, making it well-suited for security tasks where new malware or benign applications would appear dissimilar to known samples. However, there is no existing measure of uncertainty for a model trained with contrastive learning. The authors therefore use the classifier to predict the label of a test sample, then construct many pairs of samples that include the test sample and another training sample, compute the contrastive loss on each pair, and average these losses. A higher average loss value means the model is more uncertain about the test sample. We provide a more formal treatment of the above-mentioned criteria to detect uncertain observations in Appendix A.

While these confidence-based methods show promise, they rely on a fundamental assumption that remains largely untested: that the model's confidence scores meaningfully correlate with true predictive risk. In a well-calibrated system, instances assigned higher out-of-distribution scores or greater predictive uncertainty should correlate strongly with incorrect classifications, reflecting the model's ability to recognize when it operates beyond its training distribution or lacks sufficient confidence. The reliability of confidence-based sampling should, therefore, hinge on calibrated confidence quantification. However, when this assumption fails—as is particularly likely under distribution shift [18]—the consequences are significant: selective classification will mistakenly allow high-confidence errors to pass through while rejecting samples the model would have classified correctly; similarly, AL will direct scarce human annotation resources toward samples that appear uncertain but provide minimal model improvement, while overlooking genuinely informative instances because masked by overconfidence. To this end, we propose Aurora to quantify how effectively a classifier's predictive uncertainty or OOD scoring function rank observations according to their actual probability of misclassification. Furthermore, Aurora subjects classifiers and their associated confidence

functions to verification under simulated deployment conditions, assessing resilience across temporal shifts and operational constraints.

3 AURORA

We now formalize Aurora as an evaluation framework that extends beyond traditional uncertainty-aware model assessment by incorporating both uncertainty quantification and an operational perspective. Specifically, it evaluates how models perform under distributional drift when equipped with a reject option, offering a comprehensive analysis of both predictive confidence and deployment robustness. Our framework assesses confidence quality quantitatively through RC curves and qualitatively through AURC (3.1) while assessing operational stability via rejection rate variance (3.3.2), performance margin extremes (3.3.1), and normalized consistency metrics (3.2).

In this context, we use τ to denote the label budget in AL environments, which denotes the number of manually labeled samples at each evaluation cycle. Likewise, τ_{rej} indicates the target number of samples to be rejected per cycle, while τ_{real} represents the actual number of rejections when deploying a model with a target rejection rate. The parameter θ is a threshold used for confidence scores in selective classification, applicable in AL scenarios. Additionally, κ represents a model's confidence in its predictions, which may be derived from its predictive uncertainty or the OOD scoring function.

3.1 Risk-Coverage Curve

Any classifier with an associated confidence function κ that cannot reliably rank its predictions presents a fundamental problem: while it overall may have high performance in terms of accuracy, it fails to distinguish between correct and incorrect predictions. This deficiency (1) prevents effective selective classification, where low-confidence predictions are rejected to provide reliable classification, and (2) compromises the AL paradigm, where the acquisition functions depend on meaningful confidence rankings to identify informative samples for annotation [31].

Thus, we present the use of the Risk-Coverage (RC) and the associated Area Under the Risk Curve (AURC) metric [15] for verifying confidence functions, aimed at determining if a classifier can effectively gauge the trustworthiness of its predictions. In essence, we assess whether a classifier can convey when it is uncertain about its predictions or whether it "knows what it does not know".

The RC curve represents one of the most informative and practical representations of the overall uncertainty profile of a given model [11]. Please refer to Appendix A in [11] for the mathematical evidence supporting this claim. The performance of a selective classifier is quantified using *coverage* and *risk*. The former denotes the percentage of the inputs processed by the model without human intervention, and the latter indicates the level of risk in the corresponding predictions. Formally,

$$\text{coverage} = \frac{|X_h|}{|X|} \quad (1)$$

$$\text{risk} = L(\hat{Y}_h) \quad (2)$$

Figure 1: Temporal results for models trained on the *androzoo* dataset with $\tau = 50$ and for $\tau_{\text{rej.}} = 400$. The top-row presents the F1 score <u>after</u> rejection. The middle depicts the actual rejections on a monthly basis and in the bottom row is the improvement in F1 after rejection vs. the baseline of no rejections as Δ F1. The latter highlights that rejections does not always lead to improvements with respect to F1 scores for some methods.

where L is used to measure prediction quality. In many binary classification tasks, this is often represented as an "error" metric:

$$err = \frac{1}{N} \sum_{n=0}^{N} |\hat{y}_n - y_n| \qquad (3)$$

where $\hat{y}$ denotes the predicted labels and y the ground-truth labels, with both taking values in 0, 1.

To assess how prediction confidence relates to error, the RC (Risk-Coverage) curve is a valuable tool. It visualizes how the model's error evolves as we vary the coverage—that is, the proportion of predictions retained based on confidence. For instance, see Fig. 4. A derived metric, AURC (Area Under the Risk-Coverage Curve), summarizes this behavior by integrating the RC curve, yielding a single scalar that quantifies the model's average selective risk [15].

A well-calibrated confidence function κ yields an RC curve that increases smoothly with coverage. Ideally, as we admit less certain

predictions into consideration, the error rate also rises, forming a curve that is flat at first and then gradually climbs.

Model confidence can be analyzed through both ordinal ranking and calibration. One prominent calibration metric is the Expected Calibration Error (ECE), which evaluates the alignment between predicted confidence and actual accuracy. This is done by dividing predictions into M equal-sized confidence intervals and comparing each bin's average predicted confidence to its empirical accuracy [32]. While calibration offers useful insights, our primary focus lies in the more fundamental task of ranking uncertainties.

Indeed, calibration alone can be misleading as a proxy for predictive confidence quality. Calibration performance does not always reflect the model's classification accuracy or ranking fidelity [26].

Selective prediction and confidence calibration are not inherently correlated [11]. Consider two cases: (1) If all predictions have identical confidence (e.g., $r = 0.5$), and half are correct,

calibration is perfect (ECE = 0), yet ranking is impossible, making selective prediction useless. (2) In contrast, with predictions grouped into $r \in {0.9, 1.0}$, if all $r = 1.0$ predictions are correct and all $r = 0.9$ predictions are incorrect, selective prediction is optimal, but calibration is poor (ECE = 0.9).

As discussed in [13], any confidence scoring function can be converted into a selective classifier. This concept has also gained attention in the security domain; for instance, Figure 1 in [6] illustrates several approaches to constructing alternative confidence functions from an array of different models.

3.2 Coefficient of Variation

The *Coefficient of Variation* (C_V) quantifies the relative dispersion of a metric by calculating the ratio of its standard deviation to its mean ($C_V[M] = \sigma_M/\mu_M$) [2]. A lower value implies better temporal stability. Unlike standard deviation, C_V normalizes variation relative to the mean, offering two operational advantages: (1) direct comparison between models with different baseline performances, and (2) proportional context for interpreting variation magnitude. For operators, a 0.05 standard deviation in F1 score represents dramatically different reliability concerns when the mean F1 is 0.95 versus 0.45. $C_V[F1]$ quantifies the temporal stability of classification effectiveness, while $C_V[FPR]$ measures the consistency of error rates across evaluation cycles – particularly useful in domains with high false positive costs or regulatory constraints. These normalized metrics enable operators to determine whether performance fluctuations remain within acceptable operational thresholds.

3.3 Performance under Simulated Abstention

When deploying malware classifiers, operators may not always prioritize high-fidelity uncertainty estimation across the full coverage spectrum. Instead, they often focus on optimizing operational performance by abstaining from predictions on the most uncertain cases—for instance, rejecting the top 20% based on model uncertainty. These high-uncertainty observations are traditionally considered informative for AL, yet at inference time, abstaining from them can enhance downstream performance. By excluding the least confident predictions, the system yields more reliable classifications on the retained subset, aligning better with production needs such as those of malware detection APIs.

This dual-use of uncertainty—both for data acquisition and selective abstention—introduces a natural avenue for assessing classifiers under deployment-relevant constraints. We formalize this through a post-hoc rejection simulation framework, which allows evaluation of abstention strategies after training and model selection, using stored uncertainty scores. Specifically, this framework assesses how classifiers would perform if they rejected exactly a proportion τ_{rej} of test-time inputs based on their uncertainty estimates.

Unlike conventional risk-coverage analysis, which idealizes rejection in terms of sorted confidence without temporal considerations, our method calibrates rejection thresholds over a rolling time horizon. For each test month $M_i \in \mathcal{D}_{test}$, we simulate fixed-rate abstention based on uncertainty distributions observed in preceding months. This allows us to assess model robustness under realistic deployment dynamics, including distributional drift.

We distinguish between class-conditional calibration, used for softmax-derived uncertainties, and single-threshold calibration for OOD scores. Details are provided in Appendix A, with algorithm described in Alg. 1 and corresponding subprocedures (Alg. 2, 3).

Note that when this simulation is extended to new time windows, deviations in the rejection rate may occur due to temporal drift in the uncertainty distribution. Only when the test-time score distribution aligns with that of previous calibration months will the abstention rate precisely match the target τ_{rej} (see Fig. 1).

3.3.1 Performance Extrema. Inspired by financial risk analysis [28], we introduce the *Max Drawdown* to capture the worst-case F1 score degradation of a classifier over the entire evaluation period. This metric provides operators with insight into how severely a classifier's F1 performance could deteriorate. We define deterioration as the difference between pre- and post-rejection performance. See the bottom plot in Fig. 1 for example–here rejection does not always lead to performance improvements. In Month 7, for example, and for the SVC's confidence function, rejecting points actually leads to a $\sim 40\%$ drop in performance. While average metrics are useful, awareness of performance floor events is essential, as isolated failures can disproportionately damage confidence in systems with strong average F1 metrics. For risk-averse operators, this metric is particularly valuable, as operators may prefer trading off overall or average F1 performance for reduced risk in terms of worst-case scenarios.

3.3.2 Mean Absolute Percentage Deviation. The AURC quantifies the quality of the confidence function under a single distribution, but fails to capture its **stability** when subject to shift. Operators of classifiers reject-option may set a threshold value θ that constrains the model f to make predictions only when $\kappa(x, \hat{y}|f) > \theta$, i.e., when the confidence exceeds the threshold. This approach ensures that the model handles a desired percentage of the data (the coverage) with high confidence. Notably, the performance of this confidence threshold mechanism exhibits distribution dependence. A confidence function κ with fixed threshold θ calibrated to reject a target proportion of samples on one distribution may reject significantly different proportions when deployed in production and faced with temporal drift. Figure 7 in [6], for example, clearly illustrate this fluctuation in the number of 'quarantined' observations across evaluation periods. For this reason, we propose the Mean Absolute Percentage Deviation (MAPD), which acts as a stability metric to quantify the confidence function's consistency in maintaining the target rejection rate across evaluation cycles. We compute this metric by measuring the average relative difference between actual and target rejection rates, expressed as a percentage of the target rate; lower values indicate more consistent and accurate rejection behavior, while higher values reveal greater variability or systematic bias from the intended rejection threshold.

The MAPD metric quantifies the average relative deviation of the actual monthly rejection counts from the target rejection level. For each target rejection rate $\tau_{rej} > 0$ (we use $\tau_{rej} = \{100, 200, \ldots, 1500\}$ with increments of 100), and the corresponding actual rejection rates, the MAPD is calculated as:

$$\mathrm{MAPD}(\tau_{\mathrm{rej}}) = \frac{100}{\tau_{\mathrm{rej}}} \cdot \frac{1}{n} \sum_{i=1}^{n} \left| \tau_{\mathrm{real}} - \tau_{\mathrm{rej}} \right| \qquad (4)$$

where n is the total number of monthly values collected, and the factor 100 is to lift the metric from a decimal.

Temporal variations in rejection rates are not inherently problematic as some time periods might naturally contain more uncertain observations that legitimately warrant higher rejection rates. A well-functioning confidence function should accurately identify these periods of increased uncertainty. All the confidence functions in this study are evaluated under identical drift conditions; therefore, we can isolate stability differences that are attributable to the confidence functions themselves rather than to increased magnitude of drift. This evaluation directly addresses the practical question: "How much variance can I expect from deploying the model X with its confidence function κ using a reject option and a rejection threshold of τ_{rej}?"

3.3.3 Adaptation of Metrics for Simulated Abstention. To assess the impact of simulated abstention, we collect metrics such as F1, FNR, and FPR for each model and each rejection level $\tau_{\mathrm{rej}} \in \{100, 200, \ldots, 1500\}$. For each metric, we compute the mean across all τ_{rej} values, yielding an aggregate that reflects the average improvement or degradation in retained performance due to abstention. Importantly, performance does not necessarily improve with increased rejection—poorly calibrated confidence functions may exhibit the opposite trend. For example, in Fig. 5, the SVC's F1 score declines as τ_{rej} increases. To ensure robustness, all metrics are computed across multiple random seeds, with results concatenated rather than averaged per seed, thereby preserving full variability in the final aggregate.

4 τ-PROPORTIONAL SUBSAMPLING OF $\mathcal{D}_{\mathbf{init}}$

While reflecting on the evaluation of classifiers across multiple dimensions in Aurora, we observed a fundamental challenge in the AL paradigm: the tension between historical knowledge retention and adaptation to emerging patterns to confront temporal drift. *Note: See section 5.1 and 5.2 introducing the symbols and the data.* The canonical approach for AL methods for Android malware detection involves pretraining models using historical data, $\mathcal{D}_{\mathrm{init}}$, which may span varying time periods (commonly one year or more), serving as the initial training dataset. Subsequently, during each test period (monthly evaluation cycle), τ samples are selected according to some confidence-based criteria for labeling and incorporated into the training process. This integration follows either: (1) a cold-start approach, where after each set of τ samples is labeled, the model is fully retrained from scratch using both $\mathcal{D}_{\mathrm{init}}$ and all previously labeled samples; or (2) a warm-start approach, where the model undergoes incremental updates using only the set of labeled τ samples.

In a cold-start scenario, we hypothesize that not all data points from $\mathcal{D}_{\mathrm{init}}$ contribute equally to the learning objective. Our reasoning stems from the assumption that the τ samples identified during each test period contain critical signals for adapting the model to underlying data drift, yet their influence may be diminished when overwhelmed by historical data. The gradient signal during backpropagation may be dominated by this imbalance – where

the large reservoir of historical training samples overwhelms the comparatively small set of recent labeled examples from $\mathcal{D}_{\mathrm{test}}$.

To contextualize this imbalance, consider the dataset, proposed by [6], which we will introduce more formally in Section 5.2. Transcendent contains 57,740 samples in the first year: a monthly injection of 50 observations accounts for just 0.1% of the initial training data, and grows to only 4.1% after 48 test periods. We question whether this stark imbalance of $\tau \ll |\mathcal{D}_{\mathrm{init}}|$ leads to an impact on performance, potentially limiting the model's ability to adapt to emerging patterns in malware evolution. Hence, we test how subsampling of $\mathcal{D}_{\mathrm{init}}$ impacts the downstream performance of malware classifiers. We experiment with two distinct subsampling strategies: (1) stratified random sampling (*StratK-Sampling*), which maintains the binary label distribution, and (2) uncertainty-based sampling (*Uncertainty-Sampling*). The latter is motivated by the common sampling procedure of malware classifiers under drift where observations with high uncertainty are prioritized because they are informative to update the decision boundary.

We test various sample proportions of τ_{init}, with several values directly inspired by prior literature [7, 43]. For context, our training dataset spans a 12-month period, so $\tau_{\mathrm{init}} = 12$ corresponds to "1 selected observation per month" in D_{init}, while $\tau_{\mathrm{init}} = 4800$ represents "400 selected observations per month." We evaluated label budgets $\tau = \{50, 100, 200, 400\}$. We conduct this experiment using a neural network model, DeepDrebin, with 30 training epochs. Section 5.1 provides a more detailed description of this model. Each configuration of τ_{init} and τ was executed five times on the three distinct datasets of different levels of drift. For *Uncertainty-Sampling*, we split $\mathcal{D}_{\mathrm{init}}$ into $k = 6$ folds, train DeepDrebin on $k - 1$ and select the top-$\frac{\tau_{\mathrm{init}}}{k}$ most uncertain points across every k'th split.

Figure 2 reports the results averaged across the five runs. The rightmost points in each plot (at maximum τ_{init}) represent the F1 performance with no subsampling of D_{init}. As shown, average performance tends to increase as the label budget τ grows, with this effect being more pronounced for the Androzoo and Transcendent datasets compared to the APIGraph dataset. This illustrates the varying task complexity across datasets, with the APIGraph dataset being "easier" to learn as evidenced by its consistently high F1 scores even at low τ values. We find that for the APIGraph and Transcendent datasets, performance reaches near-maximum levels at relatively small τ_{init} values, with minimal improvements as the sample size increases. In contrast, the Androzoo dataset exhibits a distinct bell-shaped performance curve, peaking at $\tau_{\mathrm{init}} \approx 1\mathrm{k}$ and declining as the sample size increases further. In other words, performance is maximized at moderate sampling levels, with diminishing returns observed at extremes of the sampling spectrum. As evidenced by Figure 3, there is no performance benefit of performing *Uncertainty-Sampling* as a simple *StratK-Sampling* fully suffices. Note that due to the prohibitive computational demands for CADE and HCC, we were constrained to conducting the experiments within this section with DeepDrebin.

> **Takeaway:** (1) τ-proportional subsampling of $\mathcal{D}_{\mathrm{init}}$ can improve performance in terms of aggregate metrics such as F1. (2) Stratified random sampling of $\mathcal{D}_{\mathrm{init}}$ performs equivalently to uncertainty sampling on our datasets.

Figure 2: Results for *Uncertainty-Sampling*. Average Performance across $n = 5$ trials with *DeebDrebin* on selected datasets. For every τ (monthly label budget for $\mathcal{D}_{\text{test}}$) and τ_{init} (selected samples from $\mathcal{D}_{\text{init}}$) we run a full experiment on the all months in $\mathcal{D}_{\text{test}}$ and report the average monthly performance, excluding the first 6 months of the test-period as per standard-protocol [7, 43].

Figure 3: Comparison of *Uncertainty-Sampling* and *StratK-Sampling*. Results are additionally averaged across all τ's to allow for a direct comparison. We find that *StratK-Sampling* is on par with *Uncertainty-Sampling*.

5 EXPERIMENTAL SETUP

We introduce the reference frameworks under evaluation (Section 5.1) and a formal treatment of their associated confidence functions (whose details are provided in Appendix A due to space constraints). In Section 5.2, we detail the datasets adopted.

5.1 Reference Frameworks

We consider a number of high-profile malware classification frameworks, some of which were originally designed for AL schemes. We selected these methods because they represent a progression of increasingly sophisticated representation learning paradigms in malware classification. The first classifier is Drebin, a linear support vector machine (SVM) on high-dimensional binary feature vectors engineered with a lightweight static analysis [3]. The second classifier is DeepDrebin, a fully-connected feedforward neural network introduced [17] to mitigate adversarial attacks, later adopted for malware classification by [44]. We ultimately consider two contrastive learning frameworks which were designed to support active learning, CADE [43] and HCC [7]. CADE relies on contrastive learning to perform a distance-based feature transformation which

results in more homogeneous class clusters relative to which out-of-distribution observations are easier to detect. HCC implements a hierarchical contrastive loss function that enforces similarity constraints in the embedding space based on family-level relationships. For CADE, we use the version replicated and enhanced provided by [7]. This improved version replaces the original SVM classifier with a neural network classifier and uses the embeddings generated by the contrastive encoder as input features rather than the raw features used in the original implementation. All these methods can be easily replicated using the authors' publicly available code and offer a stable baseline. Detailed hyperparameter settings for each method are documented in Appendix C. To ensure reliability and consistency of our results, each method has been executed 5 times with different random seeds, with all the reported results representing the aggregated outcomes across runs.

We exclude DroidEvolver [42] as prior work [24] has demonstrated high risks of self-poisoning due to poorly calibrated models. Furthermore, DroidEvolver employs online learning mechanisms (relying on pseudo-labels for retraining), which fall outside the scope of our current research focus. We focus on approaches that leverage drift detection to drive additional processes, specifically, continual active learning [7] and explainability [43]. By contrast,

Transcendent [6] is a purely selective-classification method that uses rejection to handle likely misclassifications but does not harness drift to inform any subsequent tasks. While we were inspired by Transcendent's demonstration of low performance on rejected data, we leave the investigation of purely drift-focused or selective-classification approaches—and their robustness—outside the scope of this paper and plan to address them in future work.

5.2 Datasets

The experiments are conducted on three datasets: APIGraph and Androzoo from [7] and Transcendent from [6]. We selected these datasets as they are used in SOTA research, are well-established in the Android malware detection domain, and contain large sample collections with timestamps to simulate natural temporal drift. For the APIGraph dataset, the authors from [7] collected Android apps using hashes from APIGraph [44], but extracted Drebin features rather than using the feature space originally proposed in that work. When we reference the APIGraph dataset, we are specifically referring to the dataset introduced by Chen et al. [7]. The APIGraph dataset contains Android apps from 2012-2018 with a 9:1 malware-to-goodware ratio, while Androzoo spans 2019-2021 with the same ratio. Transcendent includes apps from 2014-2018, also maintaining the 9:1 ratio, as recommended by Pendlebury et al. [35]. We treat the first year as an initial training buffer $\mathcal{D}$init, where we assume full label availability. As common practice [7], we use a monthly evaluation- and retraining cycle which we denote as $\mathcal{D}$test. As in [7], we use the first 6 months from $\mathcal{D}$test to select the best hyperparameters for each method. The reported results exclude the initial 6-month period of $\mathcal{D}_{\text{test}}$ to avoid data snooping [10].

6 EXPERIMENTS

We begin by analyzing the risk-coverage trade-offs of all confidence functions using risk-coverage plots in Section 6.1. Subsequently, Section 6.2 presents the empirical performance of all evaluated methods, using the metrics introduced in our Aurora framework.

6.1 Risk-Coverage Plots

Every time a prediction is made with low confidence, it might lead to expensive actions, like requiring human assessment for labeling or defaulting to more cautious decisions. The risk level for each coverage point equates to the model's selective risk when it dismisses inputs not encompassed at that particular coverage level. In Figure 4 we report the RC curve for the selected datasets for $\tau = 50$. The RC plots illustrate how different confidence functions' error rates correlate with coverage. Ideally, the curve should be flat and monotonically increasing as more uncertain observations are included. For the Transcendent dataset, for example, one method has ~40% risk at low coverage, while others remain below ~10%. On the APIGraph dataset, peak risks translate to ~10% and ~12% at low coverage, with most confidence functions stabilizing below 2%. CADE's MSP shows severe miscalibration in comparison, particularly evident at low coverage (40% in Transcendent, 6% in Androzoo, and 12% in the APIGraph datasets). This indicates CADE frequently assigns high confidence to incorrect predictions. While CADE's OOD scoring functions (warm and cold setups) demonstrate increased reliability, they remain erratic compared to other confidence functions.

DeepDrebin's MSP confidence function is a very strong candidate in terms of robustness, maintaining near-zero risk across most coverage levels in the Transcendent and APIGraph datasets. SVC's confidence function shows complementary strengths, particularly in the Androzoo dataset, where its $\tau_{\text{init}} = 4800$ configuration maintains nearly 0% risk until 60% coverage (where it still outperforms most confidence functions despite increased variability). Surprisingly, **SVC and DeepDrebin** stand as **the most trustworthy options** in terms **of confidence calibration**, with their consistent performance across diverse datasets translating directly to reliable risk estimation, beating SOTA approaches regarding their confidence functions. Additionally, we find that subsampling of $\mathcal{D}_{\text{init}}$ often preserves or improves the quality of confidence across all coverage levels, particularly at lower coverage thresholds where uncertainty estimation is often most critical.

> **Takeaway:** (1) Simple methods outperform complex architecture in terms of uncertainty calibration, with DeepDrebin and SVC achieving low risk where others are mis-calibrated. (2) subsampling training data improves calibration quality by acting as a regularizer, particularly at critical low-coverage thresholds. (3) OOD scoring functions might select misleading samples while missing informative edge cases that would actually benefit model robustness against shift.

6.2 Numerical Results

Table 1 reports the performance comparison of our reference frameworks and associated confidence functions across the three datasets under various training configurations (τ and τ_{init}). We report the F1 score, the Coefficient of Variation ($C_V[F1]$) and the AURC for baseline performance; the adapted F1 score, and the MAPD for performance under simulated abstention. Please refer to Section 3 for a discussion on the metrics. Due to space constraints, the additional metrics are provided in Appendix E with relevant findings incorporated into this discussion here. The AURC analysis is not necessary under simulated abstention since evaluating performance under a fixed rejection threshold would represent a segment of the full RC curve corresponding to the non-rejected points, providing no additional insights beyond what is already contained in the standard AURC analysis.

6.2.1 Baseline Performance. We observe no single method dominating across all metrics and datasets. There exists a discernible hierarchy concerning dataset difficulty. Methods trained on the APIGraph dataset demonstrate the highest overall performance, followed by the moderately challenging Transcendent dataset, while the Androzoo dataset is identified as the toughest for drawing inferences. Notably, the varying the τ-parameter has the biggest impact in terms of performance improvements for the Androzoo dataset, where methods consistently show gradual improvements with increasing τ. For the other datasets, the improvement in performance is more marginal, especially the APIGraph dataset where the increase is barely noticeable.

DeepDrebin with subsampling ($\tau_{\text{init}} = 4800$) demonstrates competitive F1 performance compared to HCC (warm) across all datasets. For the Androzoo dataset, HCC (warm) shows a notable advantage

Risk–Coverage Curves ($\tau = 50$)

Figure 4: Risk-Coverage Plots for selected datasets and for a label-budget $\tau = 50$. The ideal curve has minimal error across the coverage-spectrum and a higher coverage or *acceptable uncertainty* correlates with a higher error. The dashed line refers to models trained with a sub-sampled initial data-set $\mathcal{D}_{\mathbf{init}}$ (with $\tau_{\mathbf{init}} = 4800$). See appendix D for results for $\tau \in \{100, 200, 400\}$.

at lower τ values (75% vs 69% at $\tau = 50$, a 6% difference; 77% vs 75% at $\tau = 100$, a 2% difference), while DeepDrebin outperforms at higher τ values (81% vs 78% at $\tau = 200$, a 3% difference) and matches HCC at $\tau = 400$ (both 84%). On the APIGraph dataset, DeepDrebin consistently achieves equal or better F1 scores across all τ settings (90% vs 89% at $\tau = 50$; 92% vs 90% at $\tau = 100$; 92% vs 91% at $\tau = 200$), with a substantial 7% advantage at $\tau = 400$ (93% vs 86%). On the Transcendent dataset, HCC performs slightly better at $\tau = 50$ (84% vs 82%, a 2% difference), while DeepDrebin shows a slight edge at higher τ values (86% vs 85% at $\tau = 100$; 88% vs 87% at $\tau = 200$; 90% vs 89% at $\tau = 400$), with differences of 1-2 percentage points.

For stability (measured by C_V [F1]), HCC outperforms Deep-Drebin on Androzoo at lower budgets, while on APIGraph, Deep-Drebin maintains significantly better stability at $\tau = 400$ (4) compared to HCC (27) (please refer to Figure 8 in Appendix G), and on Transcendent, both methods perform comparably, with HCC slightly better at $\tau = 200$.

Subsampling the training set with $\tau_{\mathrm{init}} = 4800$ observations yields substantially better uncertainty calibration (as measured by AURC) for DeepDrebin across all τ values on the Androzoo dataset (63-76% improvement), while SVC shows mixed results (improvement at $\tau = 50$ but degradation at higher τ values); AURC values remain comparable on APIGraph and Transcendent datasets regardless of whether the methods were trained with subsampling ($\tau_{\mathrm{init}} = 4800$) or not.

The MSP response from DeepDrebin with $\tau_{\mathrm{init}} = 4800$ outperforms both HCC confidence functions in terms of reliability across

all τ values on both the Transcendent (AURC: 3-6 vs 8-12 for Softmax and 16-21 for Pseudo-Loss) and APIGraph (AURC: 10-17 vs 22-54 for Softmax and 28-41 for Pseudo-Loss) datasets. On the Androzoo dataset, its quality is comparable or slightly lower (AURC: 11-27 vs 16-23 for both confidence functions).

> **Takeaway:** (1) No single methods dominates across all metrics and datasets, suggesting the importance of context-specific evaluation and consideration. (2) Subsampling the training dataset ($\tau_{\mathrm{init}} = 4800$) either improves DeepDrebin performance across all metrics or shows negligible impact compared to $\tau_{\mathrm{init}} = |\mathcal{D}_{\mathrm{init}}|$. (3) DeepDrebin, a simple feedforward neural-network outperforms more complex frameworks in terms of performance and confidence reliability, while requiring only binary labels and minimal computational resources. This evidence compels us to revisit established research paradigms and priorities. (4) Even when simpler models show performance metrics slightly lower than or on par with SOTA approaches, they frequently demonstrate superior results across other critical metrics, such as confidence quality, highlighting that "better" extends beyond primary performance indicators.

6.2.2 Performance under Simulated Abstention. While the AURC metric evaluates confidence functions across the entire confidence spectrum, operational deployment scenarios typically focus on the top percentage of uncertain observations due to human labeling

Table 1: Results for both the baseline evaluation and under simulated abstention . For each dataset, we standardize the AURC results column-wise by dividing them by the maximum AURC value and then multiplying by 100 to increase readability and facilitate the comparison of methods across various τ values. All experiments with simulated abstention use $\tau_{\text{rej}} \in \{100, 200, \dots, 1500\}$. A ↓ indicates that lower values are better. We perform all experiments with 5 random seeds. For additional results, please refer to table 2 in appendix E.

| | | | Dataset | | | | | | | | | | | | | | | |
| | | | androzoo | | | | | apigraph | | | | | transcendent | | | | |
τ	τ_{init}	Method	F1↑	C_V[F1]↓	AURC↓	F1↑	MAPD↓	F1↑	C_V[F1]↓	AURC↓	F1↑	MAPD↓	F1↑	C_V[F1]↓	AURC↓	F1↑	MAPD↓
50	$\|\mathcal{D}_{\text{init}}\|$	CADE (cold) - OOD	62%	37	89	74%	63	86%	6	46	64%	54	71%	20	24	69%	74
		CADE (cold) - Softmax	62%	37	72	73%	87	86%	6	100	88%	71	71%	20	30	73%	120
		CADE (warm) - OOD	66%	32	58	78%	53	84%	8	43	35%	73	60%	38	66	57%	91
		CADE (warm) - Softmax	66%	32	60	78%	46	84%	8	100	86%	42	60%	38	100	45%	86
		DeepDrebin (cold) - Softmax	54%	44	73	75%	50	90%	5	13	94%	36	84%	10	5	92%	66
		HCC (warm) - Pseudo-Loss	75%	25	23	69%	58	89%	5	41	91%	60	87%	10	21	87%	53
		HCC (warm) - Softmax	75%	25	23	70%	57	89%	5	31	91%	72	84%	10	12	85%	65
		SVC	63%	35	26	57%	43	88%	5	15	93%	38	63%	24	14	55%	33
	4800	DeepDrebin (cold) - Softmax	69%	34	27	80%	60	90%	5	17	94%	40	82%	14	6	89%	55
		SVC	69%	30	18	45%	45	88%	5	15	92%	38	64%	23	13	60%	39
100	$\|\mathcal{D}_{\text{init}}\|$	CADE (cold) - OOD	63%	36	100	72%	65	87%	6	43	66%	56	73%	20	22	71%	70
		CADE (cold) - Softmax	63%	36	80	72%	78	87%	6	86	90%	89	73%	20	22	69%	121
		CADE (warm) - OOD	62%	38	72	72%	58	86%	7	42	21%	54	62%	37	31	62%	91
		CADE (warm) - Softmax	62%	38	66	75%	45	86%	7	87	88%	47	62%	37	27	42%	104
		DeepDrebin (cold) - Softmax	57%	42	71	75%	49	91%	4	11	94%	36	86%	10	4	92%	63
		HCC (warm) - Pseudo-Loss	77%	25	19	73%	57	90%	5	33	93%	59	85%	9	21	87%	51
		HCC (warm) - Softmax	77%	25	19	75%	57	90%	5	26	92%	74	85%	9	12	85%	63
		SVC	68%	32	19	64%	43	88%	5	15	93%	38	69%	20	11	65%	41
	4800	DeepDrebin (cold) - Softmax	75%	29	23	84%	54	92%	4	14	95%	41	86%	9	5	91%	52
		SVC	66%	34	27	57%	49	89%	5	16	93%	39	69%	21	12	62%	40
200	$\|\mathcal{D}_{\text{init}}\|$	CADE (cold) - OOD	66%	36	73	73%	65	89%	5	37	70%	55	76%	16	21	75%	79
		CADE (cold) - Softmax	66%	36	75	75%	83	89%	5	79	91%	105	76%	16	21	75%	146
		CADE (warm) - OOD	63%	37	84	70%	59	87%	7	41	27%	67	52%	63	30	51%	74
		CADE (warm) - Softmax	63%	37	70	77%	51	87%	7	78	90%	102	52%	63	28	35%	99
		DeepDrebin (cold) - Softmax	59%	42	63	77%	50	92%	4	10	95%	33	87%	11	3	94%	65
		HCC (warm) - Pseudo-Loss	78%	24	18	75%	58	91%	4	29	94%	61	87%	8	16	89%	45
		HCC (warm) - Softmax	78%	24	18	76%	57	91%	4	22	94%	74	87%	8	9	88%	61
		SVC	69%	30	18	68%	42	89%	4	16	94%	36	69%	22	12	61%	44
	4800	DeepDrebin (cold) - Softmax	81%	21	15	91%	56	92%	4	10	95%	37	88%	9	4	93%	61
		SVC	68%	34	26	65%	48	89%	4	14	93%	39	70%	21	13	65%	51
400	$\|\mathcal{D}_{\text{init}}\|$	CADE (cold) - OOD	77%	24	64	76%	71	89%	5	45	72%	54	79%	16	19	79%	74
		CADE (cold) - Softmax	77%	24	67	80%	72	89%	5	90	91%	121	79%	16	16	81%	171
		CADE (warm) - OOD	70%	32	67	72%	56	89%	5	33	40%	74	58%	53	28	59%	86
		CADE (warm) - Softmax	70%	32	66	79%	53	89%	5	75	91%	124	58%	53	34	59%	147
		DeepDrebin (cold) - Softmax	70%	34	45	84%	52	93%	4	9	95%	30	90%	8	2	95%	60
		HCC (warm) - Pseudo-Loss	84%	13	16	77%	58	86%	27	28	94%	74	89%	8	17	90%	43
		HCC (warm) - Softmax	84%	13	16	79%	59	86%	27	54	87%	76	89%	8	8	89%	63
		SVC	72%	31	15	73%	43	89%	5	16	94%	35	72%	19	10	67%	54
	4800	DeepDrebin (cold) - Softmax	84%	17	11	94%	52	93%	4	10	96%	37	90%	8	3	94%	53
		SVC	71%	31	19	66%	45	89%	4	14	94%	37	71%	21	12	65%	54

capacity limitations and system up-time requirements, which motivates our design of post-hoc simulation experiments to assess classifier performance under practical rejection constraints.

DeepDrebin MSP often yields superior F1 score compared to all other methods (94% on Androzoo, 96% on APIGraph, and 94% on Transcendent at $\tau = 400$ with $\tau_{\text{init}} = 4800$), with clear performance scaling as label-budget increases (e.g., Androzoo: 80% → 94%). The advantage is the most pronounced for the Androzoo dataset, where the F1 performance gap between DeepDrebin with $\tau_{\text{init}} = 4800$ and non-DeepDrebin methods is 14% (91% vs. 77% at $\tau = 200$ and 94% vs. 80% at $\tau = 400$). DeepDrebin with $\tau_{\text{init}} = 4800$ shows improved temporal stability as budget increases (C_V[F1] decreasing from 30

to 15 on Androzoo) and achieves strong C_V[F1] on Transcendent (as low as 3 at $\tau = 400$). However, we occasionally observe some high Max Drawdown (F1) (34% at $\tau = 400$ on Androzoo). Initialization with $\tau_{\text{init}} = 4800$ significantly improves F1 scores (e.g., from 84% to 94% on Androzoo at $\tau = 400$) without corresponding MAPD improvements. MAPD performance is also dataset-specific: DeepDrebin with $\tau_{\text{init}} = |\mathcal{D}_{\text{init}}|$ achieves better MAPD on APIGraph (30 vs. 37 at $\tau = 400$) but trails behind SVC with $\tau_{\text{init}} = |\mathcal{D}_{\text{init}}|$ on Androzoo (52 vs. 43 at $\tau = 400$), while DeepDrebin with $\tau_{\text{init}} = 4800$ performs better on Transcendent (53 vs. 54 at $\tau = 400$) but still worse than HCC (Pseudo-Loss) (43 at $\tau = 400$).

Figure 5: Performance of classifiers <u>after</u> rejection, i.e. the performance of non-rejected test-points on the *androzoo* dataset. F1 scores for a range of selected rejection-thresholds $\tau_{\mathrm{rej.}} = \{0, 100, 200, \ldots, 1500\}$. The case of $\tau_{\mathrm{rej.}} = 0$ refers to the baseline of performance without rejection. See Appendix F for results on the transcendent- and apigraph datasets.

Notably, DeepDrebin trained with $\tau_{\mathrm{init}} = 4800$ seems over-proportionally performant regarding the aggregated F1 score of this simulated abstention experiment, especially once contextualized with the results of the RC plots (please refer to Figure 4) where the corresponding RC curve is outperformed by HCC. **This underscores a key point: even though the RC plot and AURC assess the total error, they fail to reveal the error distribution across classes, which is precisely where the F1 score shows sensitivity.** Figure 5 and Figure 7 in Appendix F illustrate the underlying data across the rejection thresholds of $\tau_{\mathrm{rej}} \in \{100, 200, \ldots, 1500\}$.

SVC with $\tau_{\mathrm{init}} = |\mathcal{D}_{\mathrm{init}}|$ achieves the lowest MAPD score across all label budgets ($\tau = 50, 100, 200, 400$) for the Androzoo dataset. For the Transcendent dataset, SVC has the lowest MAPD score for $\tau = 50$, $\tau = 100$, and $\tau = 200$, but for $\tau = 400$, HCC (warm) - Pseudo-Loss outperforms it. For the APIGraph dataset, DeepDrebin - Softmax consistently achieves lower MAPD scores than SVC across all label budgets.

Both HCC confidence functions exhibit very strong temporal stability with $C_V[F1]$ values as low as 3-4 on the APIGraph dataset at $\tau = 50\text{-}200$, in stark contrast to CADE (warm) - OOD which reaches 239 at $\tau = 200$. However, this stability diminishes at $\tau = 400$, where HCC (warm) - Softmax increases to 26. For rejection capabilities, HCC (Pseudo-Loss) generally outperforms its Softmax variant on MAPD metrics, particularly on Transcendent (43-53 vs. 61-65). While both HCC variants achieve strong F1 scores under simulated abstention (69-79% on Androzoo, 87-94% on APIGraph, and 85-90% on Transcendent), their Max Drawdown (F1) score vary by dataset—relatively low on Transcendent (4-13%) but higher on Androzoo (12-27%). Notably, HCC (Pseudo-Loss) consistently outperforms both SVC and DeepDrebin variants on Transcendent at $\tau = 400$ in terms of MAPD (43 vs. 53-61).

CADE configurations demonstrate inconsistent performance across datasets, with significant differences between cold and warm set-ups. On Androzoo, CADE (warm) - Softmax achieves moderate F1 scores (75%–79%), while exhibiting poor stability on Transcendent with Max Draw scores of 78%–86%. CADE (warm) - OOD shows the most severe performance instability, with high Max Drawdown (F1) reaching 74%–91% on Transcendent, coupled with particularly low F1 performance on APIGraph (dropping as low as 21% at $\tau = 100$). In contrast, CADE (cold) variants perform better than their warm counterparts on the Transcendent dataset, achieving higher F1 scores (69%–81% vs 35%–62%), but still demonstrate concerning Max Draw values (32%–88%) and extremely high MAPD (up to 171 for CADE (cold) - Softmax at $\tau = 400$). Between rejection strategies, Softmax generally outperforms OOD for both cold and warm CADE set-ups on Androzoo and APIGraph datasets in terms of F1 scores. Performance improvements from increased τ are minimal across all CADE configurations, with worst-case scenarios remaining problematic even at the highest τ values of 400, as evidenced by persistent high MAPD values and significant Max Drawdown scores.

> **Takeaway:** (1) While increased τ improve some methods substantially in terms of F1 score, others plateau quickly, revealing critical differences in resource utilization efficiency. (2) Methods with the highest F1 score can demonstrate inconsistent rejection stability (3) Softmax-based methods achieve higher F1 score but suffer from less stable rejection rates over time compared to alternative confidence functions. (4) Models with similar overall accuracy can have dramatically different failure patterns when selectively refusing predictions.

Figure 5 reports the F1 performance of the confidence functions within our reference frameworks when implementing selective classification, showing how performance on non-rejected samples changes as the rejection threshold increases. Ideally, the curve should be monotonically non-decreasing.

The SVC classifier shows the most dramatic decline in performance as rejection threshold increases, especially visible in all τ values. This suggests poor confidence calibration - SVC is likely rejecting both correct and incorrect predictions indiscriminately. DeepDrebin with $\tau_{\text{init}} = 4800$ often demonstrates a superior confidence calibration, particularly at $\tau = 200, 400$ where it achieves the highest F1 score with increasing rejection thresholds. The HCC classifier demonstrates strong initial calibration but degrading reliability as the rejection threshold increases beyond 500 observations. This indicates that HCC maintains reliable confidence estimation for its most certain predictions (around the first 500 rejected), but beyond this point, its confidence scores become increasingly misaligned with actual prediction accuracy.

Figure 1 shows how our reference frameworks perform over time on the Androzoo dataset under simulated abstention. We use $\tau = 50$ and $\tau_{\text{rej.}} = 400$. All methods exhibit pronounced declines in F1 scores in Month 7 and Month 13. DeepDrebin demonstrates more robustness compared to the other methods in response to these shifts. Different methods show varying recovery capabilities after performance drops - some bounce back quickly while others exhibit lingering effects. SVC performance starts declining with temporal distance to the training set. Subsampling mitigates this degradation.

In the middle chart, CADE variants frequently operate above the target rejection threshold. The most dramatic spike occurs in Month 3, where CADE retrained with observations derived from its softmax in cold set-up rejects over 1500 samples, nearly 5 times the target threshold of $\tau_{\text{rej.}} = 400$. Rejection patterns generally stabilize in later months, with fewer extreme spikes and better adherence to the target threshold. The bottom chart (Δ F1) shows that rejecting samples does not consistently improve performance. Some methods, especially SVC, actually see negative Δ F1 values in certain months, meaning the rejection strategy decreased their effectiveness. Some configurations appear to benefit more consistently from rejections, with DeepDrebin showing more positive Δ F1 across the timeline. SVC exhibits a massive performance spike in Month 12, reaching over 50 % improvement.

> **Takeaway:** (1) Rejections do not always lead to improvements with respect to F1 score for some methods. (2) Subsampling for DeepDrebin helps mitigating temporal shift.

7 LIMITATIONS

While we evaluated softmax alternatives for HCC and CADE, we did not test them as AL selection criteria, instead using each method's original confidence function. We omitted the Area Under Time (AUT) metric as it essentially duplicates average metrics. We excluded Population Stability Index (PSI) and Prediction Accuracy Index (PAI) because they require separate development data splits and lack intuitive interpretability compared to our selected metrics. While the AURC provides valuable insights into confidence reliability across our evaluation, we acknowledge that—like all metrics—it comes with inherent tensions [40]. We view these not as limitations but as considerations that researchers should explicitly factor when evaluating their models. Our extensive multi-dimensional approach ultimately aim to express precisely this philosophy: effective security evaluation demands a panoramic perspective of model performance rather then the tunnel vision of traditional metrics.

8 CONCLUSION

Our results challenge the prevailing SOTA paradigm in machine learning research for Android security: models are only as good as the metrics used to evaluate them. No single method consistently excels across all datasets and metrics, with each showing distinct trade-offs: DeepDrebin's strong F1 performance and improved AURC with subsampling, HCC's apparent temporal stability–still on par with if not challenged by a simpler DeepDrebin–at the cost of expensive and hard-to-obtain family labels for HCC, and soldid MAPD behavior of SVC. Crucially, simpler models often match or outperform complex frameworks while requiring fewer computational resources. Beyond proposing additional metrics, we advocate for a shift in security model evaluation that prioritizes multidimensional assessment over single-metric optimization, with particular emphasis on confidence reliability under drift. This holistic evaluation approach better reflects real-world security challenges where threats continuously evolve and would lead to more robust, deployable solutions than those optimized for accuracy under laboratory conditions. Ultimately, most models rely on the Drebin feature space [3], which may be approaching saturation, suggesting future research should explore alternative feature representations to break through current performance plateaus.

REFERENCES

[1] 2025. VirusTotal. https://www.virustotal.com Accessed: 2025-03-04.

[2] Hervé Abdi. 2010. Coefficient of variation. *Encyclopedia of research design* 1, 5 (2010), 169–171.

[3] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014.* The Internet Society. https://www.ndss-symposium.org/ndss2014/drebin-effective-and-explainable-detection-android-malware-your-pocket

[4] AV-TEST. [n. d.]. *Malware Statistics and Trends Report.* https://www.av-test.org/en/statistics/malware/ n.d..

[5] Amin Azmoodeh, Ali Dehghantanha, and Kim-Kwang Raymond Choo. 2018. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE transactions on sustainable computing* 4, 1 (2018), 88–95.

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

[7] Yizheng Chen, Zhoujie Ding, and David Wagner. 2023. Continuous Learning for Android Malware Detection. In *32nd USENIX Security Symposium (USENIX Security 23).* USENIX Association, Anaheim, CA, 1127–1144. https://www.usenix.org/conference/usenixsecurity23/presentation/chen-yizheng

[8] C Chow. 1970. On optimum recognition error and reject tradeoff. *IEEE Transactions on information theory* 16, 1 (1970), 41–46.

[9] Charles Corbiere, Nicolas Thome, Antoine Saporta, Tuan-Hung Vu, Matthieu Cord, and Patrick Perez. 2021. Confidence estimation via auxiliary models. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 10 (2021), 6043–6055.

[10] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2021. *Dos and Don'ts of Machine Learning in Computer Security* (2022 ed.). USENIX.

[11] Yukun Ding, Jinglan Liu, Jinjun Xiong, and Yiyu Shi. 2020. Revisiting the evaluation of uncertainty estimation and its application to explore model complexity-uncertainty trade-off. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops.* 4–5.

[12] Ido Galil, Mohammed Dabbah, and Ran El-Yaniv. 2023. What can we learn from the selective prediction and uncertainty estimation performance of 523 imagenet classifiers. *arXiv preprint arXiv:2302.11874* (2023).

[13] Yonatan Geifman and Ran El-Yaniv. 2017. Selective classification for deep neural networks. *Advances in neural information processing systems* 30 (2017).

[14] Yonatan Geifman, Guy Uziel, and Ran El-Yaniv. 2018. Bias-reduced uncertainty estimation for deep neural classifiers. *arXiv preprint arXiv:1805.08206* (2018).

[15] Yonatan Geifman, Guy Uziel, and Ran El-Yaniv. 2019. Bias-Reduced Uncertainty Estimation for Deep Neural Classifiers. In *International Conference on Learning Representations.* https://openreview.net/forum?id=SJfb5jCqKm

[16] Charles A. E. Goodhart. 1975. Monetary relationships: A view from Threadneedle Street. *Papers in Monetary Economics* 1 (1975).

[17] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial Examples for Malware Detection. In *Computer Security – ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, Cham, 62–79.

[18] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17).* JMLR.org, 1321–1330.

[19] William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. 2016. DL4MD: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Science (ICDATA).* The Steering Committee of The World Congress in Computer Science, Computer …, 61.

[20] Kilian Hendrickx, Lorenzo Perini, Dries Van der Plas, Wannes Meert, and Jesse Davis. 2024. Machine learning with a reject option: A survey. *Machine Learning* 113, 5 (2024), 3073–3110.

[21] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).

[22] Dan Hendrycks and Kevin Gimpel. 2017. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *International Conference on Learning Representations.* https://openreview.net/forum?id=Hkg4TI9xl

[23] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *26th USENIX Security Symposium.* USENIX Association, Vancouver, BC. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney USENIX Sec.

[24] Zeliang Kan, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2021. Investigating labelless drift adaptation for malware detection. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security.* 123–134.

[25] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. 2016. Deep learning for classification of malware system call sequences. In *AI 2016: Advances in Artificial Intelligence: 29th Australasian Joint Conference, Hobart, TAS, Australia, December 5-8, 2016, Proceedings 29.* Springer, 137–149.

[26] Ananya Kumar, Percy S Liang, and Tengyu Ma. 2019. Verified uncertainty calibration. *Advances in neural information processing systems* 32 (2019).

[27] Henrik Linusson, Ulf Johansson, Henrik Boström, and Tuve Löfström. 2018. Classification with reject option using conformal prediction. In *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part I 22.* Springer, 94–105.

[28] Malik Magdon-Ismail and Amir F Atiya. 2004. Maximum drawdown. *Risk Magazine* 17, 10 (2004), 99–102.

[29] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2016. Mamadroid: Detecting android malware by building markov chains of behavioral models. *arXiv preprint arXiv:1612.04433* (2016).

[30] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. 2016. Reviewer integration and performance measurement for malware detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13.* Springer, 122–141.

[31] Jooyoung Moon, Jihyo Kim, Younghak Shin, and Sangheum Hwang. 2020. Confidence-aware learning for deep neural networks. In *international conference on machine learning.* PMLR, 7034–7044.

[32] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. 2015. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 29.

[33] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. 2017. Context-aware, adaptive and scalable android malware detection through online learning (extended version). *arXiv preprint arXiv:1706.00947* (2017).

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

[35] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *28th USENIX Security Symposium (USENIX Security 19).* USENIX Association, Santa Clara, CA, 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury

[36] Xingzhi Qian, Xinran Zheng, Yiling He, Shuo Yang, and Lorenzo Cavallaro. 2025. LAMD: Context-driven Android Malware Detection and Classification with LLMs. arXiv:2502.13055 [cs.CR] https://arxiv.org/abs/2502.13055

[37] Joshua Saxe and Konstantin Berlin. 2017. eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *arXiv preprint arXiv:1702.08568* (2017).

[38] Glenn Shafer and Vladimir Vovk. 2008. A Tutorial on Conformal Prediction. *J. Mach. Learn. Res.* 9 (June 2008), 371–421.

[39] Sina Sheikholeslami. 2019. *Ablation Programming for Machine Learning.* Master's thesis. KTH, School of Electrical Engineering and Computer Science (EECS).

[40] Jeremias Traub, Till J Bungert, Carsten T Lüth, Michael Baumgartner, Klaus H Maier-Hein, Lena Maier-Hein, and Paul F Jaeger. 2024. Overcoming common flaws in the evaluation of selective classification systems. *arXiv preprint arXiv:2407.01032* (2024).

[41] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[42] Ke Xu, Yingjiu Li, Robert Deng, Kai Chen, and Jiayun Xu. 2019. Droidevolver: Self-evolving android malware detection system. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 47–62.

[43] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *30th USENIX Security Symposium (USENIX Security 21).* USENIX Association, 2327–2344. https://www.usenix.org/conference/usenixsecurity21/presentation/yang-limin

[44] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20).* Association for Computing Machinery, New York, NY, USA, 757–770. https://doi.org/10.1145/3372297.3417291

[45] Xu-Yao Zhang, Guo-Sen Xie, Xiuli Li, Tao Mei, and Cheng-Lin Liu. 2023. A survey on learning to reject. *Proc. IEEE* 111, 2 (2023), 185–215.

[46] Xinran Zheng, Shuo Yang, Edith C. H. Ngai, Suman Jana, and Lorenzo Cavallaro. 2025. Learning Temporal Invariance in Android Malware Detectors. arXiv:2502.05098 [cs.CR] https://arxiv.org/abs/2502.05098

[47] Fei Zhu, Zhen Cheng, Xu-Yao Zhang, and Cheng-Lin Liu. 2022. Rethinking confidence calibration for failure prediction. In *European conference on computer vision.* Springer, 518–536.

A BACKGROUND ON CONFIDENCE FUNCTIONS

For each model, we retrieve both its (1) native prediction confidence, specifically the maximum softmax probability (MSP) for neural network models and the distance to the hyperplane for SVC models, and (2) the out-of-distribution detection confidence where available. We refer to these as *confidence functions* (κ). As described in Section 2, each framework employs its own approach to detect out-of-distribution observations. These confidence functions serve as the foundation for the "uncertainty" criteria guiding the selection of samples the model is least confident about or those that "deviate significantly from the known data distribution." Here, we provide a more formal treatment of each criterion.

A.1 Softmax Uncertainty

For classification models with softmax at the last layer, the vanilla confidence score is the MSP. The adoption of MSP as a certainty measure is supported by findings in [22], where the authors demonstrate that correctly classified examples tend to have higher maximum softmax probabilities compared to misclassified or OOD examples. Given an input sample x, a neural network parameterized by θ outputs a vector of logits $f_\theta(x) \in \mathbb{R}^K$, where K represents the number of classes. In this work, we focus on the binary case where $K = 2$. The softmax function then transforms these logits into a probability distribution across the classes:

$$p_k(x) = \frac{\exp(f_{\theta,k}(x))}{\sum_j \exp(f_{\theta,j}(x))}, \quad k \in \{1, \ldots, K\} \tag{5}$$

The MSP, representing the model's most confident class prediction, is given by:

$$\hat{p}(x) = \max_k p_k(x) \tag{6}$$

Instead of directly using $\hat{p}(x)$ as a confidence measure, we define an uncertainty score that normalizes its deviation from 0.5 - the decision boundary in binary classification problems. This results in the following measure:

$$U(x) = 1 - \frac{|\hat{p}(x) - 0.5|}{0.5} \tag{7}$$

Expressing uncertainty this way is analogous to NCMs in conformal prediction [6, 23, 38] where $U(x) = 1$ expresses maximum uncertainty in the case where the model assigns equal probability to both classes, i.e. $\hat{p}(x) = 0.5$.

A.2 Distance to the Hyperplane

For SVC, we use the geometric distance to its hyperplane as the confidence measure. In a binary classification setting with hyperplane $f(x) = \mathbf{w}^T\mathbf{x} + b$, the signed distance from a sample $\mathbf{x}$ to the hyperplane is:

$$d(\mathbf{x}) = \frac{f(\mathbf{x})}{||\mathbf{w}||} = \frac{\mathbf{w}^T\mathbf{x} + b}{||\mathbf{w}||} \tag{8}$$

The magnitude of this distance represents confidence—samples further from the boundary are classified with higher certainty.

A.3 CADE OOD Score

The authors employ the Median Absolute Deviation (MAD) in a contrastively learned latent space. First, they map all training samples into the latent space using their encoder and calculate the centroid for each class i as c_i (by taking the mean value across each dimension in Euclidean space). For each class, they measure the typical variation in distances between samples and their class centroid by computing the median distance and the MAD, where $\text{MAD}_i = 1.4826 \cdot \text{median}(|d_i^{(j)} - \tilde{d}_i|)$. For a test sample, they calculate its score for each class as:

$$A_i^{(k)} = \frac{|d_i^{(k)} - \tilde{d}_i|}{\text{MAD}_i} \tag{9}$$

where $d_i^{(k)}$ is the distance to the class centroid, $\tilde{d}_i$ is the median distance within that class. The minimum score across all classes ($A^{(k)} = \min_i(A_i^{(k)})$) determines whether the sample is drifting; if this minimum score exceeds an empirically set threshold of 3.5, they classify the sample as OOD.

A.4 HCC OOD Score

The authors identify OOD samples using a pseudo-loss mechanism adapted specifically for contrastive learning settings. Uncertainty in contrastive learning is inherently relational, being determined by a sample's relative positioning within the embedding space compared to other samples. For each test sample x_i, they compute its embedding and find its $2N - 1$ nearest neighbors in the training set. They form a batch containing x_i and these neighbors, then assign a predicted binary label $\hat{y}_i$ to x_i as a pseudo label while using ground-truth labels for the training samples. Using this configuration, they define the pseudo-loss as:

$$\hat{L}_{hc}(i) = \frac{1}{|\hat{P}(i, \hat{y}_i)|} \sum_{j \in \hat{P}(i, \hat{y}_i)} \max(0, d_{ij} - m) + \\ \frac{1}{|N(i, \hat{y}_i)|} \sum_{j \in N(i, \hat{y}_i)} \max(0, 2m - d_{ij}) \tag{10}$$

The final uncertainty score combines this with a binary cross-entropy pseudo-loss:

$$\hat{L}(i) = \hat{L}_{hc}(i) + \lambda \hat{L}_{ce}(i) \tag{2}$$

At test time, they calculate uncertainty scores for all test samples using this equation, then prioritize those with the highest scores for AL selection.

B ALGORITHM FOR *POST-HOC SIMULATED ABSTENTION*

Algorithmic definitions of the *Post-Hoc Rejection Simulation* with associated subroutines for the single-value and class-specific uncertainty scores. The algorithm is designed to study the behavior of classifiers post-training under rejection. To perform this analysis, practitioners need to record the uncertainty scores S for every test month M_i in $\mathcal{D}_{\text{test}}$. In our evaluations, we experiment with rejection budgets of $\tau_{\text{rej.}} = \{0, 100, 200, \ldots, 1500\}$ in steps of 100.

Algorithm 1 Post-Hoc Rejection Routine

Require:
1: Test-Data $\mathcal{D}_{\text{test}}$ with test-months M_i (with $i = 0, \ldots, M$)
2: Rejection-Quota: τ_{rej}
3: Method: `ood` or `softmax`

4: Initialize empty array A
5: **for** each month M_i (with $i = 1, 2, \ldots, M$) **do**   ▷ Skip $M_{i=0}$
6:     **if** method is `softmax` **then**
7:         $S \leftarrow$ softmax scores for M_i ▷ Of the positive class
8:     **else**
9:         $S \leftarrow$ ood scores for M_i
10:    **end if**
11:    $T \leftarrow i \times \tau_{\text{rej}}$
12:    **if** $i = 1$ **then**   ▷ Skip Post-hoc for first Month
13:        Append S from M_0 to A
14:    **end if**
15:    **if** method is `ood` **then**
16:        $c \leftarrow$ `ood_threshold`(A, T)
17:        **for** each sample j in M_i **do**
18:            **if** $S[j] > c$ **then**
19:                Mark j as **rejected**
20:            **end if**
21:        **end for**
22:    **else**   ▷ softmax method
23:        $(l, u) \leftarrow$ `softmax_thresholds`(A, T)
24:        **for** each sample j in M_i **do**
25:            **if** $S[j]$ is between l and u **then**
26:                Mark j as **rejected**
27:            **end if**
28:        **end for**
29:    **end if**
30:    Append scores S to array A
31: **end for**

Algorithm 2 Binary Classification Rejection Thresholds

1: **function** `softmax_thresholds`(S, T)
2:     **Input:** Scores S; Total to reject T
3:     **Output:** Lower and upper thresholds (l, u)

4:     Sort S in descending order (most to least uncertain)
5:     Initialize $l \leftarrow 1.0$, $u \leftarrow 0.0$
6:     **for** $k = 0$ to $T - 1$ **do**
7:         $val \leftarrow S[k]$
8:         $l \leftarrow \min(l, val)$ ▷ Update lower threshold
9:         $u \leftarrow \max(u, val)$ ▷ Update upper threshold
10:    **end for**
11:    **return** (l, u) ▷ Rejection zone: $S \leq l$ or $S \geq u$
12: **end function**

Algorithm 3 Single Value Uncertainty Threshold

1: **function** `ood_threshold`(S, T)
2:     **Input:** Scores S; Total to reject T
3:     **Output:** Threshold for rejection

4:     Sort S in descending order (most to least uncertain)
5:     $idx \leftarrow$ index of the T-th sample in sorted array
6:     **return** $S[idx]$ ▷ Threshold value
7: **end function**

C HYPERPARAMETER CONFIGURATIONS

Drebin SVM: We conducted a search for the regularization parameter C from the set 0.001, 0.01, 0.1, 1, 10, 100, 1000. The best C is 1 for the Transcendent dataset, 0.1 for the APIGraph dataset, and 0.01 for the Androzoo dataset.

DeepDrebin MLP: For the DeepDrebin implementation, we adhered to the configuration described in [17], employing a batch size of 512, Adam optimizer with default parameters, and a dropout probability of $p = 0.5$. As part of our experimental design, we varied the number of epochs $e \in 30, 50$, though we found this choice to be relatively inconsequential, with $e = 30$ proving sufficient for convergence.

HCC: Our HCC implementation followed the architecture and training set-up described in the original paper [7]. The encoder subnetwork comprises fully connected layers with ReLU activation, progressively reducing the dimensionality from the input features to a 128-dimensional embedding space through layers of sizes 512-384-256-128. The classifier subnetwork employs two hidden layers with 100 neurons each, ReLU activation, and two output neurons normalized with Softmax. The authors used a batch size of 1,024. For hyperparameter selection, they considered two optimizers (SGD and Adam), four initial learning rates (0.001, 0.003, 0.005, 0.007), three learning rate schedulers (cosine annealing without restart, step-based decay by a factor of 0.95 every 10 epochs, and step-based decay by a factor of 0.5 every 10 epochs), and four choices for classifier epochs (100, 150, 200, 250). The warm start phase involved exploring two optimizers (SGD and Adam), two learning rate scales (1% and 5% of the initial learning rate), and two epoch counts (50 and 100). The best parameters for the APIGraph dataset are: SGD optimizer for the initial model with learning rate 0.003 and step-based decay by factor 0.95 every 10 epochs for 250 training epochs; Adam optimizer during warm start with learning rate 1.5×10^{-4} (5% of initial) for 100 epochs after each monthly update. For the Androzoo dataset are: SGD optimizer for the initial model with learning rate 0.001 and step-based decay by factor 0.5 every 10 epochs for 200 training epochs; Adam optimizer during warm start with learning rate 1×10^{-5} (1% of initial) for 50 epochs after each monthly update. For the Transcendent dataset, after performing a search over the parameter space reported in [7], we determined the optimal configuration to be: SGD optimizer for the initial model with learning rate 0.003 and step-based decay by factor 0.95 every 10 epochs for 250 training epochs; Adam optimizer during warm start with learning rate 3×10^{-5} (1% of initial) for 100 epochs after each monthly update.

CADE: For CADE, we used the exact same setup described in [7]. The authors in [7] adapt CADE OOD sample selector for MLP in both cold start and warm start. To have a fair comparison, the authors use the same encoder dimensions as their encoder, and mirror

that as the decoder in CADE. They use the same MLP structure as their classifier subnetwork. They use batch size 1,536. The authors fix the MLP learning rate (0.001) and training epochs (50), but perform grid search over the same set of parameters for the CADE autoencoder as described above. Note that the original active learning experiment in CADE did not tune hyperparameters (Section 6 in [43]). But they tune hyperparameters including optimizer, initial learning rate, learning rate scheduling, epochs to train the contrastive autoencoder model, warm start learning rate, and epochs.

The optimal cold-start configuration for the APIGraph dataset is: Adam optimizer with learning rate 0.001, step-based decay by factor 0.95 every 10 epochs, and 150 training epochs. For the Androzoo dataset is: Adam optimizer with learning rate 0.001, step-based decay by factor 0.5 every 10 epochs, and 100 training epochs. For the Transcendent dataset, after grid search over the parameter space reported in [7], we determined the optimal configuration to be: Adam optimizer, initial learning rate 0.003, cosine decay with a factor 1

every 10 epochs, and 100 training epochs. The optimal warm-start configuration for the APIGraph dataset is: Adam optimizer for both initial classifier and active learning; autoencoder with learning rate 0.001, cosine annealing decay without restart, and 250 initial training epochs; during active learning, both the autoencoder and MLP used 5% of the initial learning rate for 50 warm training epochs. For the Androzoo dataset is: Adam optimizer for both initial classifier and active learning; autoencoder with learning rate 0.001, cosine annealing decay without restart, and 100 initial training epochs; during active learning, both the autoencoder and MLP used 1% of the initial learning rate for 50 warm training epochs. For the Transcendent dataset, after grid search over the parameter space reported in [7], we determined the optimal configuration to be: Adam optimizer for both initial classifier and active learning; initial learning rate 0.007, step-based decay with a factor 0.95, and 200 initial training epochs; active learning: for both the autoencoder and MLP, initial learning rate 0.00035, and 50 warm training epochs.

D   RISK-COVERAGE CURVES

Figure 6: Risk-coverage curves for $\tau \in \{100, 200, 400\}$.

E ADDITIONAL NUMERICAL RESULTS

Table 2: Additional results for both the baseline evaluation and under simulated abstention . For each dataset, we report the False Negative Rate (FNR), False Positive Rate (FPR), and their coefficient of variation (CV). For metrics under simulated abstention, we also report Maximum Drawdown (F1) and CV[F1]. All experiments with simulated abstention use $\tau_{\mathbf{rej}} \in \{100, 200, \ldots, 1500\}$.

| | | | Dataset | | | | | | | | | | | | | | | |
| | | | androzoo | | | | | apigraph | | | | | transcendent | | | | |
τ	τ_{init}	Method	FNR↓	FPR↓	C_V[FNR]↓	CV[F1]↓	Max D.↓	FNR↓	FPR↓	C_V[FNR]↓	CV[F1]↓	Max D.↓	FNR↓	FPR↓	C_V[FNR]↓	CV[F1]↓	Max D.↓
50	$\lvert\mathcal{D}_{\text{init}}\rvert$	CADE (cold) - OOD	48%	0.6%	52	34	8%	17%	1.1%	47	27	4%	37%	1.5%	46	27	32%
		CADE (cold) - Softmax	48%	0.6%	52	35	20%	17%	1.1%	47	5	3%	37%	1.5%	46	43	81%
		CADE (warm) - OOD	45%	0.4%	53	30	29%	20%	1.0%	49	133	19%	38%	12.5%	59	58	87%
		CADE (warm) - Softmax	45%	0.4%	53	25	10%	20%	1.0%	49	7	4%	38%	12.5%	59	65	78%
		DeepDrebin (cold) - Softmax	58%	0.2%	42	38	14%	15%	0.4%	46	6	1%	20%	1.3%	46	7	7%
		HCC (warm) - Pseudo-Loss	34%	0.4%	66	57	24%	16%	0.6%	46	4	11%	19%	1.7%	50	10	4%
		HCC (warm) - Softmax	34%	0.4%	66	55	26%	16%	0.6%	46	4	0%	19%	1.7%	50	11	4%
		SVC	50%	0.3%	47	85	33%	18%	0.5%	43	5	0%	49%	0.8%	35	38	6%
	4800	DeepDrebin (cold) - Softmax	40%	0.4%	64	30	26%	15%	0.4%	50	6	1%	22%	1.4%	58	14	10%
		SVC	42%	0.3%	55	83	21%	17%	0.5%	44	6	2%	48%	0.7%	36	33	15%
100	$\lvert\mathcal{D}_{\text{init}}\rvert$	CADE (cold) - OOD	48%	0.6%	52	34	8%	16%	0.9%	47	25	4%	36%	1.4%	47	29	59%
		CADE (cold) - Softmax	48%	0.6%	52	35	24%	16%	0.9%	47	5	0%	36%	1.4%	47	57	88%
		CADE (warm) - OOD	50%	0.3%	50	37	26%	18%	0.8%	49	214	15%	45%	1.8%	50	64	91%
		CADE (warm) - Softmax	50%	0.3%	50	30	23%	18%	0.8%	49	6	0%	45%	1.8%	50	66	80%
		DeepDrebin (cold) - Softmax	55%	0.2%	46	37	21%	13%	0.3%	52	5	2%	17%	1.3%	52	7	7%
		HCC (warm) - Pseudo-Loss	32%	0.4%	70	51	12%	14%	0.4%	48	4	0%	17%	1.6%	51	10	7%
		HCC (warm) - Softmax	32%	0.4%	70	50	25%	14%	0.4%	48	4	0%	17%	1.6%	51	10	7%
		SVC	44%	0.3%	55	68	33%	17%	0.5%	43	5	0%	40%	1.3%	40	30	6%
	4800	DeepDrebin (cold) - Softmax	33%	0.4%	75	23	12%	12%	0.4%	53	5	1%	18%	1.2%	48	8	11%
		SVC	45%	0.3%	53	68	15%	16%	0.5%	43	5	1%	42%	0.8%	42	32	7%
200	$\lvert\mathcal{D}_{\text{init}}\rvert$	CADE (cold) - OOD	44%	0.5%	59	33	43%	14%	0.8%	50	23	23%	32%	1.3%	48	25	58%
		CADE (cold) - Softmax	44%	0.5%	59	33	14%	14%	0.8%	50	4	0%	32%	1.3%	48	56	87%
		CADE (warm) - OOD	48%	0.3%	53	38	34%	17%	0.7%	50	239	11%	53%	1.7%	58	84	90%
		CADE (warm) - Softmax	48%	0.3%	53	28	24%	17%	0.7%	50	6	0%	53%	1.7%	58	110	86%
		DeepDrebin (cold) - Softmax	53%	0.2%	48	32	22%	12%	0.3%	54	5	0%	15%	1.2%	69	6	2%
		HCC (warm) - Pseudo-Loss	31%	0.3%	72	49	27%	12%	0.5%	54	3	0%	14%	1.6%	45	11	7%
		HCC (warm) - Softmax	31%	0.3%	72	50	27%	12%	0.5%	54	3	0%	14%	1.6%	45	8	7%
		SVC	42%	0.3%	56	58	4%	16%	0.5%	42	5	0%	42%	0.7%	45	35	8%
	4800	DeepDrebin (cold) - Softmax	27%	0.3%	75	16	33%	12%	0.3%	54	4	1%	15%	1.2%	58	7	5%
		SVC	42%	0.4%	60	60	14%	15%	0.5%	44	5	1%	40%	0.9%	44	30	8%
400	$\lvert\mathcal{D}_{\text{init}}\rvert$	CADE (cold) - OOD	32%	0.5%	68	25	37%	11%	1.1%	59	20	9%	28%	1.3%	58	21	58%
		CADE (cold) - Softmax	32%	0.5%	68	23	24%	11%	1.1%	59	4	0%	28%	1.3%	58	46	86%
		CADE (warm) - OOD	40%	0.4%	62	33	35%	13%	0.9%	48	152	10%	48%	1.4%	62	61	78%
		CADE (warm) - Softmax	40%	0.4%	62	25	20%	13%	0.9%	48	4	0%	48%	1.4%	62	60	83%
		DeepDrebin (cold) - Softmax	41%	0.2%	61	24	12%	11%	0.3%	55	5	0%	12%	1.1%	48	3	4%
		HCC (warm) - Pseudo-Loss	23%	0.3%	71	48	26%	16%	1.3%	136	-	0%	10%	1.6%	42	9	7%
		HCC (warm) - Softmax	23%	0.3%	71	47	27%	16%	1.3%	136	26	0%	10%	1.6%	42	8	13%
		SVC	38%	0.3%	65	47	3%	16%	0.5%	43	5	1%	37%	1.3%	43	30	11%
	4800	DeepDrebin (cold) - Softmax	23%	0.3%	80	15	34%	10%	0.3%	55	4	0%	11%	1.1%	43	3	4%
		SVC	38%	0.5%	67	51	12%	16%	0.5%	42	4	1%	40%	0.8%	46	32	8%

F ADDITIONAL RESULTS FOR REJECTION SIMULATION

Figure 7: Results depicting performance of classifiers <u>after</u> rejection for the transcendent- and apigraph datasets.

G ADDITIONAL RESULTS AND COMMENTS

Figure 8: Temporal Instability in HCC (warm) beginning at month 61 - manifesting in large values for $C_V[\text{F1}]$ (see table 1). We repeated the experiments for HCC as described in [7] and repeated the experiment with 5 random trials (random seeds). The cause for the observed instability in HCC is unknown. Reported is average monthly performance across all 5 seeds.