# Active learning-based mobile malware detection utilizing auto-labeling and data drift detection

Zhe Deng*, Arthur Hubert†, Sadok Ben Yahia*, Hayretdin Bahsi*‡
{zhe.deng, sadok.ben, hayretdin.bahsi}@taltech.ee, arthur.hubert@uni.lu
*Department of Software Science, Tallinn University of Technology, Estonia
†Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg
‡School of Informatics, Computing, and Cyber Systems, Northern Arizona University, USA

2024 IEEE International Conference on Cyber Security and Resilience (CSR) | 979-8-3503-7536-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSR61664.2024.10679343

***Abstract*—Machine learning-based detection methods have demonstrated high performance in mobile malware detection. However, changes in mobile malware over time induce a significant challenge for malware detection systems running in operational environments. The development of non-stationary models to address concept drift in the threat landscape has garnered significant research interest. Auto-labeling is using automated algorithms to assign labels to data without manual intervention. This study explores the method for introducing auto-labeling in active learning with data drift detection. It achieves satisfying results for the lowest possible cost while succeeding in adapting to changes in the data for a long period.**

## I. Introduction

Mobile devices are ubiquitous in our daily lives today. It is critical to detect attacks against them, ensuring their safety and privacy. Machine learning-based approaches have proven effective in mobile malware detection [1]. Generally, we train and test machine learning models using static data from a specific time interval. However, mobile malware's relevant dynamic and static characteristics are constantly changing. Active learning methods, such as pool-based learning [2], can choose to retrain the model periodically based on available resources and update it based on changes in data characteristics.

Malware detection models are useful because they automatically distinguish between malicious and benign files. It is hard to find a large amount of recently labeled cyber security data due to privacy concerns and because the cost of labeling such data is high [3]. Auto-labeling is important for detecting mobile malware because it saves time and money compared to manually labeling data. Moreover, extensive research remains to enhance the accuracy of detection and operational efficiency by optimally integrating auto-labeling systems with active learning frameworks. Since it is easier to find unlabeled data, active learning relies on using unlabeled data and gradually labeling as few of them as possible. Besides, threats in the cyber security world are always evolving. Malicious actors often change and adapt their strategies to find new ways to attack, meaning that the features of malware applications change with time. This implies that we must continuously update detection models with new malware data, thus keeping them away from data drift.

In this paper, we adapt to new changes in the cyber threat environment by introducing a pool-based active learning method that can use auto-labeling to update the detection model for uninformative instances and overcome data drift simultaneously. This method has been tested under several training processes and strategies, utilizing the dataset [4] with basic balancing techniques. It achieves a $97.9\%$ F1 score using only $2.37\%$ of the labels, surpassing the established benchmarks. This improvement underscores the efficacy of integrating auto-labeling with data drift detection, maintaining high detection accuracy with minimal labeled data. The novelty of our work lies in its ability to adapt to evolving mobile malware threats through data drift detection, while simultaneously reducing the labeling burden via an innovative active learning framework.

The paper is organized as follows: Section II is background information and the related work. The methodology is explained in Section III. Our experiments' results are presented in Section IV. Section V concludes the paper.

## II. Background and Related Work

Active learning [5] is a semi-supervised strategy that is particularly useful when collecting data is easy but labeling data is expensive. Unlike a typical supervised machine learning strategy, it works when limited labeled data is available. This data is used to train our model by selecting informative samples from a large, unlabeled pool. In this context, our objective is to create a model as accurately as possible and at the cheapest cost. This technique works with the help of an *oracle*, an expert on our data, who will be able to assist the training process by labeling the data we ask them to. In our strategy's real-life use case, that oracle would be a cyber security expert, and every time we need their help, it counts as a cost. This process relies on the constant interaction between humans and machines.

The pool-based active learning framework [6] follows the following steps: 1) *Initialization* and *Model Training*: Start with a pool of unlabeled data points and a limited number of labeled datasets. Train an initial machine learning model (classifier) using the labeled data. 2) *Query Strategy* and *Data Selection*: Selecting data points from the unlabeled pool that are expected to provide the most information gain when labeled. The most commonly used query strategy is *Uncertainty sampling* which is based on the least confident

prediction. if $y*$ is the most likely label predicted for the sample $x$, then the uncertainty score $U(x)$ of that sample will be: $U(x) = 1 - P(y * |x)$ With this strategy, the selected sample will be the one with the highest uncertainty, which is the one our model is least confident in classifying; 3) *Labeling (Annotation)*: The oracle provides labels or annotations for the selected data points; 4) *Model Update* and *Evaluation*: Put the newly labeled data points into the training set and re-train; Evaluate the performance of the updated model on a validation or test set. 5) *Iterations*: Repeat 2-4 until the stopping criterion is met. Next, we conduct a final evaluation of the model.

Research on the application of active learning to drift detection has explored approaches for processing incoming data streams. One is processing consecutive batches of data instances when sufficient data storage and computational resources are available. The other one is online learning [7], which handles each data instance individually when computational resources are limited. Labeling the entire incoming data stream is considered costly for both approaches. Therefore, active learning aims to minimize the labeling effort by selecting the most informative instances. Despite size constraints, a pool of unlabeled data is available for sampling when using data batches. On the other hand, in an online learning environment where we process samples individually, we make decisions for each incoming instance independently, lacking access to a comparative data pool. The active learning approach has been adapted to online settings where drifts occur in data streams with unlabeled data [8]. To deal with sudden and gradual drifts in an online setting, frameworks have been suggested to keep an ensemble of models up to date using active learning [9].

Attacks also target active learning-based models. Attackers can add bad samples to the data pool, an oracle could purposely give wrong labels, and security attacks for machine learning that target the training and testing phases can be changed to work in active learning situations [10]. Also, generative adversarial networks can use active learning to make adversarial samples to hide from intrusion detection systems if they can't get to the training data [11]. Active learning has been used in intrusion detection. Network intrusion detection can be enhanced by minimizing the number of labeled samples needed while maximizing detection quality [12]. Although active learning has found applications in various problem domains, its exploration of mobile malware detection is limited, particularly in non-stationary settings. Dynamic features, such as system calls, are used to build well-performing detection models [13], [14]. Hybrid features that combine static and dynamic analysis are proven to improve Android malware detection [15]. Using human-machine interaction in active learning can make models for finding IoT botnets that work much better with less data than the usual ways of making machine learning-based detection systems[16].

## III. Methods

### A. Dataset

Our study's dataset is labeled, fully timestamped, and named *KronoDroid* [4]. Real device data was chosen for its substantial size, consisting of $41,382$ malware samples and $36,755$ benign apps: system calls, permissions, and timestamps. We employed class labels as input features to order samples along the historical Android timeline and for class identification. The timestamp provides information on when the detection system first received the sample, enabling the simulation of a continuous data stream for a realistic malware detection scenario. Three sets of input features, static (permissions), dynamic (system calls), and hybrid (system calls and permissions), with lengths of $166, 288$, and $454$, respectively, were used for model induction, each composed of various variable types.

It allows us to compare the performance of these various types of features separately. We divide the dataset into several smaller sub-datasets following the timeline of the samples, which we will refer to as periods $t_0, t_1, \ldots, t_n$, which contain two months' data. We excluded periods with too few elements, leaving 44. We divide each period into a training part, a pool of unlabeled data for active learning and testing. It is essential to exercise caution when utilizing the dataset due to its significant imbalance, which can adversely affect the uncertainty sampling strategy[17]. Following the idea [16], we integrate a balancing method into our main training process and apply balancing strategies to each period before batch retraining. In active learning, whether employing uncertainty or random sampling, we utilize a balancing strategy exclusively on $t_0$, as this initial period is used to establish the model. We choose **two basic balancing techniques** for each experiment:

- *Oversampling* creates copies of the samples from the minority until the same amount as the majority.
- *Undersampling* deletes samples from the majority until the same amount as the minority.

### B. Implementation

*1) Query strategy:* The primary objective of the query strategy in active learning is to rank the usefulness of data within the unlabeled pool, which is crucial for determining which elements to label. This ranking then guides the selection of the most informative data for labeling. The distinction among various existing strategies lies in the parameters used to establish this ranking distinguish it from various existing strategies.

While the query selection process is the most critical factor for optimizing active learning, we also examined the roles of the initial training pool, the size of the full unlabeled pool, and the supervised model chosen for training. As we talked about before, initiating the active learning process requires a small sample of data from the unlabeled pool to form the training set. However, without an existing model, identifying the most useful data for the training set is challenging. The size of the initial training set is crucial: a set that is too small necessitates more queries during the active learning cycle, while a set that is too large incurs higher labeling costs due to the random selection of initial data compared to more strategically selected subsequent data.

In most datasets, there is a significant imbalance between benign and malware data. For example, in the *KronoDroid*

dataset used frequently in this study, the ratio can change over different periods. We conducted experiments to determine whether the composition of the initial pool should reflect this bias. In future experiments, we will simulate the active learning process with an expert using labeled datasets, although our models will initially operate without labels. The dataset will serve as the *oracle*, providing labels for each query. We randomly selected samples for the initial training pool from the unlabeled pool, ensuring the inclusion of at least one minority label sample. Training more iterations gradually improves the model's performance. We can continue until we achieve a specific performance or until we have labeled all the samples we can afford.

*2) Training strategy:* **Three training processes** are used several times each: permissions only, system calls only, and hybrid features that combine both of them. This way, we can compare which type of feature is the most informative in training with active learning.

To provide a reference for the performance of our active learning training strategy, we ran **three types of training**:

- *Active learning with uncertainty sampling*: This is our main strategy. We start the model by training on $t_0$ using pool-based active learning steps. The model continues to train, employing uncertainty sampling to identify the most informative sample within this unlabeled pool. During this time, we use the testing pool associated with $t_0$ to measure the performance. Once the performance reaches the given condition, we pause the training and move one period forward using the testing set of $t_1$. The training continues using the unlabeled pool $t_1$. We repeat the same operation for the next period, and we continue this process until we have trained across all periods. Throughout the entire training process, we continuously add more training data from each of the $t_i$ periods to the training set of our model. Whenever we switch from $t_i$ to $t_{i+1}$, all the samples not queried in $t_i$ are discarded.
- *Batch retraining*: This strategy serves as an upper limit we aim to reach. We induce a supervised model trained by all samples in the current and previous periods, assuming that we know their labels. We initiate our model on $t_0$, but with all the samples and without using any active learning queries. Then, after measuring the model performance with the testing set of $t_1$, we retrain by adding all the data from $t_1$ and repeat. While training on $t_i$, our model's training set contains every sample from $t_0, t_1, \ldots, t_i$.
- *Active learning with random sampling*: This can be a lower limit. The query strategy employs random sampling, which uses no criteria to select the most informative data and selects a random sample from the unlabeled pool at each iteration. This part is mostly useful for assessing the effectiveness of our model's query strategy.

Our main evaluation metrics for our experiments are the F1 score and the accuracy. We measure the performance at the end of each training period on its current testing time. Then, we observe the average of these performances across all periods. We consider the amount of data labeled by the *oracle* as the cost we try to reduce.

*3) Data drift:* Our research aims to adapt to data changes over time effectively. We regularly retrain our model by selecting the most informative data to label. However, an alternative approach is to detect them before significant changes appear in the data distribution. Efficient data drift detection methods based on labeled data do not apply to unlabeled data. Even if we detect data drift beforehand, we need to set a clear bar to adjust the training process to improve over the regular method. We set the training threshold as the goal to reach within a given period, and we estimated drift by measuring the average number of queries needed to achieve it. Periods requiring many queries are costly and indicate significant data changes, making retraining difficult. On the other hand, periods with few or no queries suggest minimal data changes. However, this method is post-training, which limits its use for real-time training adaptation. Before beginning a new training period, we should implement an effective drift detection method. Nonetheless, this post-training method can serve as a benchmark for evaluating the efficiency of drift detection strategies.

*4) **Auto-labelling**:* To enhance our model, we aim to develop a strategy that reduces the number of labels requested from the oracle. Instead of solely relying on the oracle for labels, we seek to obtain them at no additional cost.

Our current active learning querying strategy involves identifying data points where the model exhibits uncertainty and requesting labels for these from the oracle. This approach focuses on acquiring highly informative data at minimal cost. However, we can also automatically label data points for which the model has high confidence, thus expanding the training pool without additional input from the *oracle*. Although these automatically labeled data points may not be as valuable as those obtained through the querying strategy, they provide free labeled data to enhance our training pool. On the other hand, it is important to remember that high-confidence instances in a stationary model may not improve its performance. However, in a non-stationary model, the decision boundary may change as time passes. A highly confident instance, expected to stay far away from the decision boundary in one time period, can be close to the decision boundary in another period. Thus, it is possible that including them in training set in advance in the previous time periods without any cost may reduce the required number of queries to reach the expected detection performance in the later time periods. We maintain our non-stationary model with an active learning strategy, but in each iteration of the active learning cycle, we supplement the query selection with a few automatically labeled samples that the model has high confidence in.

Although this technique can provide a high amount of free-labeled data, it also creates the risk of giving the wrong label to some samples, as we do not know what the high confidence threshold is for each time period. Training on mislabeled data can significantly degrade the model's performance. Therefore, we only automatically label data points when the model's

confidence in its predictions is very high. Our goal is to strike a balance between reducing labeling costs and lowering the risk of performance deterioration, which depends on the empirical confidence threshold we choose. Because this threshold is a critical parameter, we experimented with various decision thresholds and will discuss the advantages of each in detail.

## IV. Results

We conduct benchmark training experiments to establish the baseline. We first experimented by selecting a constant number of queries as the condition, and then evaluated the performance. Then we switched the condition to a certain threshold in performance (95% accuracy or 98% F1) and carried out several experiments to test three training processes, three types of training, and two basic balancing techniques mentioned before. We set the benchmark for our baseline active learning strategy shown in Table I, aiming to either use less than 6.9% of the labels or achieve an F1 score exceeding 97.4%, for training with hybrid features.

### A. Auto-labelling with threshold

*1) Static threshold:* To determine the level of uncertainty associated with each sample, ranging from 0.5 to 1, a higher value signifies greater confidence in the sample's label by the model. We experiment with various threshold values within this range to identify how confident our labeling is. Subsequently, all samples surpassing the designated threshold are automatically labeled. The outcomes of this process are illustrated in Table II. We notice that the lower the threshold, the more data are automatically labeled and the fewer queries we need, going down to 3% of the data that were queried at the lowest. However, as expected, this also translates to a higher amount of miss labeling and a direct decrease in the average F1 lower than 90% and accuracy score from 95.7% to 91.2%.

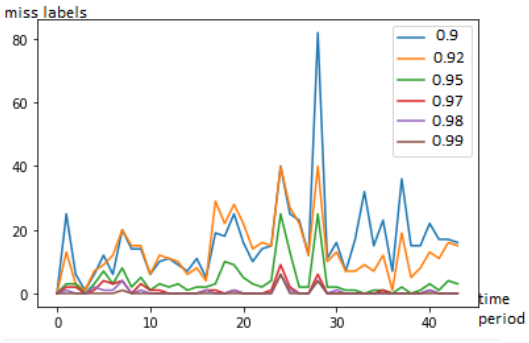


Fig. 1: Miss labels for static threshold

Figure 1 shows the number of miss-labels through all periods. Some periods (19, 24, 28, 37 ...) tend to produce more spikes in missed labels than others, even if the spike went down when we increased the threshold for confidence from 0.90 to 0.99. Although the experiment with a static threshold of 0.97 achieves an acceptable performance of 95.7% accuracy, 94.3% F1 is still low. We realized that a fixed threshold might not be the best solution. As the model trains, the value of uncertainty it gives to each sample in the unlabeled pool will evolve. We next experimented with more precise control over the amount of auto-labeled data throughout the training, which may prove beneficial.

*2) Time dynamic threshold:* While we hypothesized that a dynamic threshold might enhance learning performance compared to a fixed one, determining whether to increase or decrease the threshold over time gradually posed a challenge. As the model undergoes training, its predictive confidence typically improves, suggesting a potential benefit in lowering the threshold step by step to incorporate more data via auto-labeling without compromising accuracy. Increasing the auto-labeling threshold to prioritize active learning might be beneficial. As a result, we explored various functions to model the evolution of the threshold, aiming to assess the impact of a dynamic threshold on training efficacy. Upon entering a new period, we adjust the auto-labeling threshold. The threshold went linearly from 0.9 to 0.99 in the case of the increasing or decreasing threshold, as shown in Table III. The best scenario is the decreasing threshold with oversampling that performed best with a 3.6% of queried data, a 94.3% F1 score, and the lowest miss labeling amount at 181.

While testing with dynamic thresholds, we discovered a specific scenario: allowing too many samples to be auto-labeled at the start of training runs the risk of causing a significant spike in miss-labeling. This is because the model has not yet received a lot of training data, and it may mislabel multiple samples consecutively. This will trigger a reaction where the model trains with inverse labels, leading to a near-zero accuracy rate. To avoid this edge case, we disable auto-labeling in the first period.

*3) Iteration dynamic threshold:* The time dynamic threshold can be improved. Moreover, it failed to account for the non-linear nature of our model's training process. It gradually increases the performance in one period but drops when data drifts in consecutive periods. To better align the auto-labeling

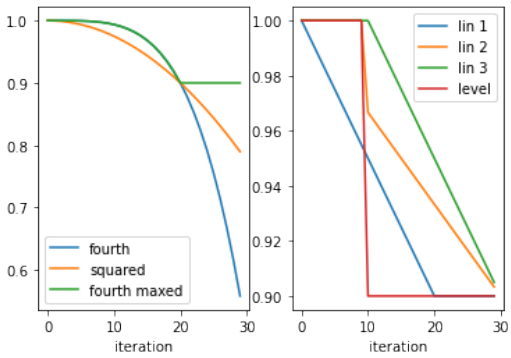


Fig. 2: Shapes of functions for iteration dynamic threshold

threshold with the active learning process, we propose the iteration dynamic threshold function based on the number of queries within one period rather than the previous periods. To avoid the risk of too many false labels at the beginning, we

TABLE I: Baseline: benchmark training results

| Feature | Balancing Method | Training Strategy | Label | | F1(%) | Accuracy(%) |
|---|---|---|---|---|---|---|
| | | | Numbers | Proportion(%) | | |
| Permission | Oversampling | Batch | 31723 | 100.0 | 99.0 | 98.1 |
| | | Random | 3020 | 9.5 | 93.9 | 95.2 |
| | | Uncertainty | 1501 | 4.7 | 95.0 | 96.0 |
| | Undersampling | Batch | 31723 | 100.0 | 99.0 | 98.1 |
| | | Random | 3292 | 10.4 | 93.6 | 95.6 |
| | | Uncertainty | 1983 | 6.2 | 94.3 | 95.8 |
| System call | Oversampling | Batch | 31723 | 100.0 | 98.1 | 96.5 |
| | | Random | 9841 | 31.0 | 89.6 | 90.5 |
| | | Uncertainty | 6694 | 21.1 | 91.2 | 92.7 |
| | Undersampling | Batch | 31723 | 100.0 | 98.1 | 96.5 |
| | | Random | 10108 | 31.8 | 89.1 | 89.6 |
| | | Uncertainty | 7470 | 23.5 | 90.9 | 92.2 |
| Hybrid | Oversampling | Batch | 31723 | 100.0 | 99.3 | 98.6 |
| | | Random | 5073 | 16.0 | 95.1 | 95.9 |
| | | Uncertainty | 2264 | 7.1 | 96.9 | 96.8 |
| | Undersampling | Batch | 31723 | 100.0 | 99.3 | 98.6 |
| | | Random | 4496 | 14.2 | 96.2 | 96.4 |
| | | Uncertainty | 2182 | **6.9** | **97.4** | 97.2 |

TABLE II: Training results for different static thresholds values (Hybrid, Undersampling)

| Static Threshold | Label | | F1(%) | Accuracy(%) | Auto-label | Miss-label |
|---|---|---|---|---|---|---|
| | Numbers | Proportion(%) | | | Numbers | Numbers |
| 0.90 | 946 | 2.98 | 89.9 | 91.2 | 27550 | 972 |
| 0.92 | 1079 | 3.40 | 91.0 | 92.5 | 26392 | 698 |
| 0.95 | 1673 | 5.27 | 92.4 | 94.1 | 21825 | 419 |
| 0.97 | 1655 | 5.21 | 94.3 | 95.7 | 13790 | 55 |

TABLE III: Training results for thresholds increasing or decreasing through time (Hybrid)

| Balancing Method | Dynamic Threshold | Label | | F1(%) | Accuracy(%) | Auto-label | Miss-label |
|---|---|---|---|---|---|---|---|
| | | Numbers | Proportion(%) | | | Numbers | Numbers |
| Oversampling | Ascending | 1439 | 4.53 | 92.0 | 93.3 | 27466 | 592 |
| | Descending | 1145 | 3.60 | **94.3** | **95.1** | 10306 | **181** |
| Undersampling | Ascending | 1405 | 4.60 | 92.6 | 94.0 | 24792 | 387 |
| | Descending | 927 | 3.00 | 90.9 | 94.0 | 18751 | 323 |

always start with a confidence threshold of 1, meaning there is no auto-labeling during the first query in each period. Starting from this, we tried multiple different shapes of function shown in Figure 2 for a dynamic threshold. For each of these shapes, we try to optimize their different parameters before comparing them. Of all the types of decreasing functions we tested, these shapes are the ones that performed the best in general. We separated these functions into two types shown in Figure 2: the affine functions and the curved functions. We tuned and optimized the parameters and found different sets of optimal values for each shape.

TABLE IV: Comparison of optimized shape results

| Shapes | F1 (%) | Accuracy (%) | Label | |
|---|---|---|---|---|
| | | | Numbers | Proportion(%) |
| no auto-labeling | 98.30 | 97.40 | 1334 | 4.27 |
| *fourth* | 97.20 | 96.00 | 502 | 1.61 |
| *fourth maxed* | 97.70 | 96.70 | 718 | 2.30 |
| *squared* | 97.30 | 96.10 | 531 | 1.70 |
| *lin 1* | 97.50 | 96.30 | 711 | 2.28 |
| *lin 2* | 97.70 | 96.60 | 716 | 2.29 |
| *lin 3* | 97.70 | 96.60 | 576 | 1.85 |
| *level* | **97.97** | 96.90 | 662 | 2.12 |
| *T_desc* | 97.40 | 96.30 | 897 | 2.87 |

As a reference, we added the active learning model without auto-labeling as well as the *T_desc* function, which is the decreasing time dynamic threshold from the previous subsection. We can see that the algorithm without auto-labeling remains above in terms of F1 score and accuracy (Table IV), which shows the impact this process has on performances. However, there has been a significant improvement in the query quantity. If we focus only on the lowest number of labels, then the best auto-label function is the polynomial function of degree four without a minimal threshold value: *fourth*. Moreover, the *level* function reaches the highest F1 score of 97.97%. Although it would depend on the use case to determine which one is better, they all achieve similar performance to our initial active learning process, but with a much lower label cost.

### *B. Auto-labelling driven by data drift detection*

This study presents an auto-labeling strategy with a dynamic threshold. To improve it further, we consider adding a drift detection method as a criterion for dynamically adjusting the auto-labeling threshold.

The problem with auto-labeling is that it increases the number of free labels but induces missed labels. Drift detection helps reduce the number of missed labels, as it allows us to anticipate big changes in the data. As the model trains through periods, the more overconfident it becomes. Since auto-labeling relies on how confident the model is, if a big drift in data occurs later in the training when the model is overconfident, it is much more likely to create mislabeling. To reduce this problem, we try to considerably reduce the amount of auto-labeling, which means increasing the auto-labeling threshold when a major drift happens.

Our methodology incorporates a validated auto-labeling dynamic threshold mechanism with iterative updates. We raise the auto-labeling threshold to 1 and pause the auto-labeling process if we detect a significant drift during an observation period. This adjustment aims to prompt the algorithm to acquire new, accurately labeled instances via uncertainty sampling, reducing the risk of mislabeling a large dataset. Data drift detection is achieved using the *Evidently AI*[1] which calculates the proportion of features displaying potential distributional changes between consecutive periods. We evaluate our approach through multiple experiment iterations, adjusting tolerance thresholds for drift magnitude that triggers auto-labeling suspension. An auxiliary drift threshold of $0.5$ is introduced to maintain uninterrupted auto-labeling. This comparative analysis benchmarks the effectiveness of concept drift detection in enhancing our algorithm's predictive capabilities.

TABLE V: Auto-labeling driven by drift detection (*level*)

| Drift Threshold | F1 (%) | Accuracy (%) | Label | |
|---|---|---|---|---|
| | | | Numbers | Proportion (%) |
| 0.20 | **97.8** | 96.7 | 699 | **2.24** |
| 0.25 | **97.9** | **96.9** | 741 | **2.37** |
| 0.30 | **97.7** | 96.6 | 734 | **2.35** |
| 0.50 | **97.8** | 96.7 | 691 | **2.21** |

According to Table V, with a drift threshold of $0.25$, the model achieves the best F1 score ($97.9\%$) and accuracy ($96.8\%$) using $2.37\%$ labels. This exceeds the baseline; however, changing different drift detection thresholds had a less significant impact. Labeled data quantity analysis suggests minimal effects. Given pre-existing measures from our optimal auto-labeling strategy at training onset, halting auto-labeling during challenging periods is ineffective. This strategy, prioritizing active learning queries with a higher initial auto-labeling threshold, showed limited efficacy. Consequently, the focus on refining the active learning process is important. Active learning via uncertainty sampling does not create an extensively trained model, as they tried to minimize the labeled data. To mitigate this, our approach diversifies model data queries to help detect significant changes. When detecting data drift, we replace the initial query with a random sample, expanding exposure to neglected data.

## V. Conclusion

In mobile malware detection, the primary objective of active learning extends beyond merely enhancing model performance; it also encompasses the significant reduction of labeling costs. This approach focuses on acquiring highly informative data at minimal cost. Additionally, we automatically label high-confidence data points to expand the training set without further oracle input, which can be particularly beneficial in non-stationary environments where decision boundaries shift over time. Our auto-labeling method provides an outstanding reduction in model cost while maintaining the detection model's performance. Meanwhile, it successfully adapts to the changes in data over a long period. However, this approach necessitates careful management of the threshold to avoid performance degradation due to potential mislabeling, and we have empirically tested various thresholds to optimize this balance. In the future, the data drift prediction method needs improvement. One possible way is to continuously monitor the statistical properties of incoming data and compare them with the original training data distributions. Another option would be to combine more sophisticated methods, such as the adaptive windowing algorithm, with active learning. Both should enable timely model updates, guaranteeing the detection system's accuracy and effectiveness despite evolving malware patterns.

[1]https://github.com/evidentlyai/evidently

## References


[1] Zhenxiang Chen, Qiben Yan, Hongbo Han, Shanshan Wang, Lizhi Peng, Lin Wang, and Bo Yang. Machine learning based mobile malware detection using highly imbalanced network traffic. *Information Sciences*, 433-434:346–364, 2018.

[2] Takafumi Kanamori. Pool-based active learning with optimal sampling distribution and its information geometrical interpretation. *Neurocomputing*, 71(1):353–362, 2007. Dedicated Hardware Architectures for Intelligent Systems Advances on Neural Networks for Speech and Audio Processing.

[3] Fabricio Ceschin, Felipe Pinage, Marcos Castilho, David Menotti, Luiz S Oliveira, and Andre Gregio. The need for speed: An analysis of brazilian malware classifiers. *IEEE Security & Privacy*, 16(6):31–41, 2018.

[4] Alejandro Guerra-Manzanares, Hayretdin Bahsi, and Sven Nõmm. Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization. *Computers & Security*, 110:102399, 2021.

[5] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[6] Burr Settles. From theories to queries: Active learning in practice. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, volume 16, pages 1–18, 2011.

[7] Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey, 2018.

[8] Saad Mohamad, M. Sayed Mouchaweh, and Hamid Bouchachia. Active learning for classifying data streams with unknown number of classes. *Neural Networks*, 98, 10 2017.

[9] Weike Liu, Hang Zhang, Zhaoyun Ding, Qingbao Liu, and Cheng Zhu. A comprehensive active learning method for multiclass imbalanced data streams with concept drift. *Knowledge-Based Systems*, 215:106778, 2021.

[10] Brad Miller, Alex Kantchelian, Sadia Afroz, Rekha Bachwani, Edwin Dauber, Ling Huang, Michael Tschantz, Anthony Joseph, and J.D.Tygar. Adversarial active learning. volume 2014, 11 2014.

[11] Ning Wang, Yimin Chen, Yang Hu, Wenjing Lou, and Y. Thomas Hou. Manda: On adversarial example detection for network intrusion detection system. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, page 1–10. IEEE Press, 2021.

[12] Amir Ziai. Active learning for network intrusion detection, 2019.

[13] Xi Xiao, Shaofeng Zhang, Francesco Mercaldo, Guangwu Hu, and Arun Kumar. Android malware detection based on system call sequences and lstm. *Multimedia Tools and Applications*, 78:1–21, 02 2019.

[14] Vinod P., Akka Zemmari, and Mauro Conti. A machine learning based approach to detect malicious android apps using discriminant system calls. *Future Generation Computer Systems*, 94:333–350, 2019.

[15] Yung-Ching Shyong, Tzung-Han Jeng, and Yi-Ming Chen. Combining static permissions and dynamic packet analysis to improve android malware detection. pages 75–81, 06 2020.

[16] Alejandro Guerra-Manzanares and Hayretdin Bahsi. On the application of active learning to handle data evolution in android malware detection. In *International Conference on Digital Forensics and Cyber Crime*, pages 256–273. Springer, 2022.

[17] Burcu Sayin, Evgeny Krivosheev, Jie Yang, Andrea Passerini, and Fabio Casati. A review and experimental analysis of active learning over crowdsourced data. *Artificial Intelligence Review*, 54:5283–5305, 2021.