CalmDroid: Core-Set Based Active Learning for Multi-Label Android Malware Detection

Minhong Dong[1,2], Liyuan Liu[1], Mengting Zhang[1], Sen Chen[3], Wenying He[4], Ze Wang[1,2], Yude Bai[1,2]
[1]School of Software, Tiangong University, Tianjin, China
[2]Tianjin Key Laboratory of Autonomous Intelligence Technology and Systems, Tiangong University, Tianjin, China
[3]College of Cryptology and Cyber Science, Nankai University, Tianjin, China
[4]School of Artificial Intelligence, Hebei University of Technology, Tianjin, China
{2231081178, 2111320207, 2211530206, wangze, baiyude}@tiangong.edu.cn,
tigersenchen@163.com, hwying1234@hebut.edu.cn

***Abstract*—One of the trends in the evolution of Android malware is the increasing diversity of malicious behaviors, such as SMS-related and Internet-related actions. Traditional binary or family-based classification methods are inadequate for fine-grained detection of these behaviors. Thus, multi-label classification is required to identify various malicious behaviors within a single malware sample. This paper employs an active learning strategy to add multi-behavior labels to large-scale datasets based on expert-annotated small-scale datasets. To address the issue of noisy labels (simulating real-world mislabeling), we propose CalmDroid, an active learning framework utilizing the core-set strategy, instead of the confuse-set strategy for updating the model with out-of-distribution (OOD) points. We evaluate CalmDroid's performance using the Drebin and VirusShare datasets. Experimental results demonstrate that CalmDroid achieves superior detection performance under varying noise conditions, with an accuracy improvement of up to 0.704 compared to the confuse-set strategy. In high-noise environments (15%), it reaches detection accuracy as high as 0.944. Additionally, we validate CalmDroid's capability to detect evolving malware. Despite behavioral evolution in Drebin malware across different time steps, CalmDroid consistently achieves detection rates above 70% in the newest time step.**

***Index Terms*—Android Malware, Multi-label Classification, Active Learning, Core-set Strategy, Label Noise**

I. Introduction

The adoption of smartphones has surged rapidly in recent years, with the number of global smartphone network users exceeding 6.9 billion in 2023 [1]. Among the various mobile operating systems, Android OS has become the most popular due to its open-source nature and extensive application ecosystem, holding a market share of 70% [2]. However, as the scale of Android applications grows, so does the prevalence of malware targeting this platform. Android malware not only threatens user privacy and data security but also causes significant economic losses to businesses and society.

Security experts have focused on malware detection as a countermeasure to this growing threat. Common malware detection methods include binary classification (malware vs. benign application) and family classification (family detection) [3, 5]. Both methods fall under single-label classification tasks, where each malware is assigned to a single category. However, these methods face limitations when dealing with the diverse and continuously evolving behaviors of modern malware. For instance, binary classification cannot handle samples exhibiting multiple malicious behaviors simultaneously, while family classification may overlook the characteristics of novel malware [4]. As the number of malware families increases and 0-day malware emerges, multi-label classification techniques, which can simultaneously identify multiple malicious behaviors in a single sample, have gained significant attention [13, 6, 42].

One of the challenges in Android malware multi-label classification is the data annotation problem. Well-known datasets such as Androzoo [7] and Drebin [11] are primarily single-label datasets. This work builds upon the anonymousCERT dataset, used in MLCDroid [13], which includes security reports and categorizes malware into six behavioral classes (e.g., SMS-related, Ads, etc.). A single malware instance can correspond to one or more of these classes. MLCDroid employed an active learning strategy to add multi-label annotations to the malware in the Drebin dataset, assuming that the annotations in both anonymousCERT and Drebin are accurate (i.e., noise-free).

However, the model update process in active learning is often dependent on expert decisions, typically using the confuse-set strategy to select out-of-distribution (OOD) points. This approach can lead to mislabeling issues, especially when processing large volumes of malware, thereby introducing noise. This noise can negatively impact active learning methods based on the confuse-set strategy [8], ultimately degrading model performance. To address this issue, we incorporate the core-set strategy, commonly used in code learning and image recognition research [45], into the active learning model. The core-set strategy inherently filters out noisy samples [9].

To address these challenges, we propose the CalmDroid (https://github.com/TonyDplus/CalmDroid), which utilizes a core-set-based active learning approach to tackle the multi-label classification of Android malware in noisy environments. First, we extract labeled malware samples from anonymousCERT and, using similarity-based methods, add multi-label annotations to unlabeled malware samples from datasets such as Drebin and Virushare [10]. To better simulate real-world conditions, CalmDroid introduces noise into these data to mimic potential classification errors. After labeling the malware samples, we

Yude Bai is the corresponding author. Email: baiyude@tiangong.edu.cn.

2643-7171/25/$31.00 ©2025 IEEE
DOI 10.1109/ICPC66645.2025.00013

use clustering to select representative samples (core points). These selected malware samples help the model maintain high accuracy and stability in noisy environments during the active learning process.

Furthermore, to simulate malware evolution, CalmDroid introduces temporal labeling to the dataset. Specifically, the training data is divided into multiple time steps, with data being progressively added to the model over time. Each time step corresponds to a specific time period, ensuring that the training process aligns with the real-world scenario of malware evolving over time. During the active learning phase, CalmDroid integrates basic classifiers (BC) and multi-label classifiers (MLC), evaluating and adjusting the model at each time step based on classification performance. Through experimental comparisons between the core-set and confuse-set strategies, we demonstrate that CalmDroid exhibits more stable performance under varying noise conditions. Notably, CalmDroid's accuracy is up to 0.704 higher than that of the confuse-set strategy when handling complex multi-label classification tasks.

The contributions of this study are as follows:

- The CalmDroid is the first to apply the core-set strategy to the multi-label classification of Android malware. By selecting the most representative core malware samples, the model enhances the efficiency and accuracy of active learning.
- CalmDroid exhibits strong robustness to noise. Even in high-noise environments (with pre-noising applied to the dataset), it maintains stable classification performance, achieving accuracy up to 0.944.
- Extensive experiments confirm that the CalmDroid performs well on real-world malware datasets with temporal evolution, further demonstrating its high applicability in real-world scenarios.

II. Approach

This section introduces the complete workflow of the CalmDroid, divided into three primary phases: noise injection into labeled Android malware datasets, selection of Core points, and time-evolving active learning. We will first describe the overall framework and subsequently dive into the technical details of each model component.

A. Overall Framework

As depicted in Fig. 1, the CalmDroid operates through the following key steps:

Noise Injection into Labeled Data: Initially, labeled samples are extracted from the Android malware database. A proportion of noise is injected into these labeled datasets to simulate real-world classification challenges, such as mislabeling or data corruption.

Core Point Selection: Once the noisy labeled data is generated, the next phase focuses on selecting the Core points. Given that CalmDroid leverages the Core-Set strategy, it is essential to identify the most representative Core points for each multi-label dataset. Specifically, clustering is employed to group data samples within each class, and the most representative points—those with high similarity to other points within their cluster—are selected as Core points. These Core points are considered to be the most influential and representative samples within the dataset, playing a pivotal role in the subsequent active learning process.

Temporal Partitioning and Active Learning: After selecting the Core points, the dataset is partitioned based on temporal sequences to ensure that the training data aligns with the temporal evolution of real-world malware. The data is divided into multiple time steps, with each step corresponding to a specific time interval. During active learning, new training data is progressively introduced in temporal order at each training iteration. This temporal partitioning enables CalmDroid to simulate the Android malware evolution over time.

In the active learning phase, the CalmDroid incorporates both BC and MLC. After new data is added at each time step, the model undergoes training and testing to evaluate its classification performance. The active learning strategy then adjusts the model parameters based on the test results and continues adding new data for subsequent training. This iterative process repeats until all time-partitioned data has been fully utilized.

The following sections will delve into the technical specifics of each phase, including the methods for noise injection, the implementation of clustering, the approach to temporal partitioning, and the detailed mechanisms of the active learning strategy.

B. Noise Injection into Labeled Data

*1) **Acquisition of Labeled Android Malware Datasets**:* In this study, several benchmark datasets were provided by our industry partner, anonymousCERT. It is important to note that the security reports have been thoroughly verified and identified by anonymousCERT. Based on their analysis, six types of malicious behaviors were identified and categorized. To match these six labels, we referred to [17] and [18], selecting three widely-used feature categories: API calls, permissions, and intents. By integrating insights from previous research and technical documentation [19][20][21], we consolidated a total of 531 features and 6 types of malware labels.

Due to the limited scale of the benchmark dataset provided by anonymousCERT, we plan to conduct more extensive experiments on larger datasets to comprehensively evaluate the performance of the CalmDroid. Additionally, given the current lack of mature multi-label classification tools, CalmDroid leverages the anonymousCERT benchmark dataset as a reference to perform multi-label classification on other large-scale Android malware datasets.

As illustrated in Fig. 2, for Android malware data not provided by anonymousCERT, we employed source code parsing and regularization methods for feature extraction within the CalmDroid framework. Once the features were obtained, we performed similarity calculations between the unlabeled data and the benchmark data from anonymousCERT that had validated reports. The unlabeled data was assigned multi-label

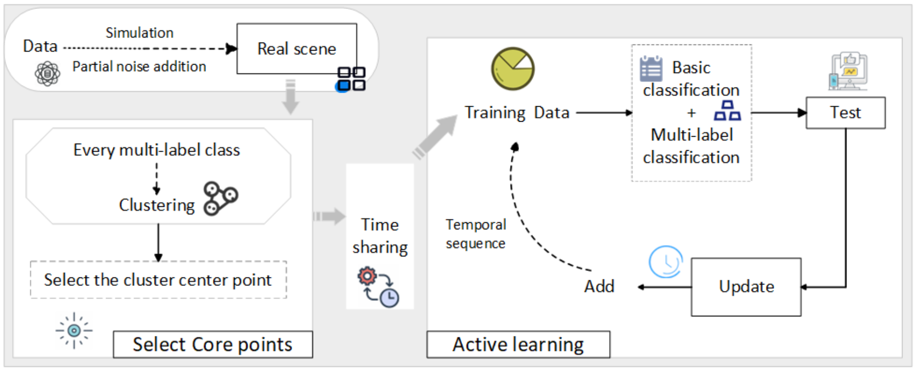

Fig. 1: The Framework of CalmDroid

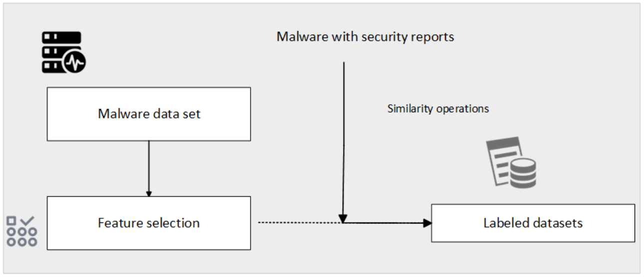

Fig. 2: The data process of Noise Injection into Labeled Data

results based on the most similar baseline data, thus generating a labeled Android malware dataset.

*2) **Noise Injection**:* Although various tools exist for labeling malware, misclassification remains a common issue. Both manual and automated methods can result in errors, especially when processing large datasets. In this study, we classify such misclassifications as noise data.

To ensure the initial labeled Android malware dataset is correctly partitioned, we used the anonymousCERT dataset with validated reports as the noise-free baseline. To introduce noise, we randomly altered the labels of a certain proportion of the data while keeping the features unchanged, thereby simulating misclassifications that occur in real-world environments. By setting different noise ratios, we generated datasets with varying noise levels, specifically 5%, 10%, and 15% noisy datasets.

The process of noise injection can be expressed as follows:

$$y' = (1 - p) \cdot y + p \cdot \eta \tag{1}$$

where y represents the original label, p is the noise ratio, and η denotes the noise.

C. Core Point Selection

*1) **Cluster algorithm**:* To obtain the Core point set required for the Core-Set strategy in CalmDroid, clustering is first applied to the noisy dataset to identify the Core points. Based on multi-label classification, the dataset is divided into multiple candidate multi-label classes, and the similarity between samples is calculated to form a similarity matrix. Clustering methods, such as KNN clustering, are effective for identifying Core points. In this study, KNN clustering was employed as the hierarchical clustering method. Given a similarity matrix S_{ij} and a predefined number of clusters K, each sample x_i determines its neighbors based on its similarity values S_{ij}, where j represents the index of the other samples, using the following formula:

$$N_i = \{x_j \in D \mid S_{ij} > \tau\} \tag{2}$$

where τ is the similarity threshold, and D is the dataset.

KNN clustering is a supervised clustering method, and thus requires the selection of an appropriate K-value, i.e., the number of clusters. Too many or too few clusters can negatively impact performance. To address this issue, we propose an adaptive cluster selection algorithm to determine the optimal number of clusters for a given similarity matrix.

Algorithm 1 Adaptive Cluster Number Selection

```
 1: **Input:** Similarity Divergence Matrix
 2: **Output:** Optimal Number of Clusters and Cluster Labels
 3: normalized_matrix ← Normalize Matrix
 4: Initialize silhouette_scores, calinski_scores
 5: for num_clusters in range do
 6:     labels ← KNN Cluster
 7:     silhouette_avg ← Compute silhouette score
 8:     calinski_score ← Compute Calinski-Harabasz score
 9:     Append scores
10: end for
11: final_scores ← Normalize($\alpha$ · silhouette_scores + $\beta$ · calinski_scores)
12: best_num_clusters ← argmax(final_scores)
13: best_labels ← Final Clustering
14: Return best_num_clusters, best_labels
```

The proposed algorithm 1 takes the similarity matrix as input and performs normalization to ensure that data points are on a consistent scale. Key variables (best score, optimal number of clusters) are initialized, and silhouette scores and Calinski-Harabasz scores are stored for further evaluation. Within a predefined range of cluster numbers, hierarchical clustering is iteratively performed, and scores are calculated and normalized. The optimal number of clusters is identified by calculating a weighted sum of the evaluation metrics, which generates the cluster labels. This algorithm enhances the robustness of clustering results by integrating multiple evaluation metrics, providing a systematic approach to clustering analysis.

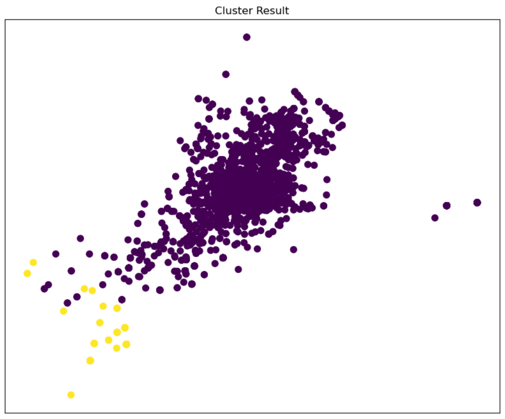

Fig. 3: The example of the cluster visualization.

Silhouette scores and Calinski-Harabasz scores serve as the evaluation metrics for the clustering process. The silhouette score measures the internal similarity of clusters by computing the ratio between the average distance of each point to the points within the same cluster (intra-cluster distance) and the average distance to points in the nearest other cluster (inter-cluster distance). Meanwhile, the Calinski-Harabasz score evaluates the ratio of between-cluster separation to within-cluster compactness. By combining these two metrics in the adaptive algorithm, the clustering results can be optimized to produce well-separated and cohesive clusters.

*2) **Acquisition Core Point:*** After determining the optimal number of clusters and obtaining the clustering results, the next step is to select the most representative point from each cluster to be included in the Core set. Specifically, for each cluster C_k, we select the representative point $x_i \in C_k$ as the Core point. This selection is based on the following formula:

$$x_{core} = \arg\max_{x_i \in C_k} \frac{1}{|C_k|} \sum_{x_j \in C_k} S(x_i, x_j) \tag{3}$$

As illustrated in Fig. 3, different colored points represent different clusters. Using the adaptive cluster selection algorithm, the multi-label class is optimally divided into two clusters. The points in the clusters are distributed across core and peripheral regions, where points in the core region have higher similarity to other points and are selected as Core points. Meanwhile, the peripheral points, which have lower similarity, are classified into the Confuse set under the traditional Confuse-set strategy. These Confuse points are likely to introduce ambiguity into the classification process, contrasting with the Core points.

D. Time-Evolving Active Learning

*1) **Temporal Partitioning of Datasets:*** To better reflect the impact of the temporal evolution of Android malware on classification model training, the CalmDroid orders all training samples chronologically after obtaining the Core points and Confuse points required by traditional models. The training data is then fed to the active learning model in a time-sequential manner, with each update introducing training samples that correspond to a subsequent point on the timeline.

*2) **Active Learning:*** In this study, the active learning model primarily employs a combination of BC and MLC for multi-label classification tasks.

BC refers to traditional classification algorithms used for single-label classification tasks, where each classifier is designed to predict a single label. This is formally defined as:

$$y_i = f_{BC}(x) \in \{0, 1\} \tag{4}$$

where x means the input features, and y_i is the output label.

MLC , on the other hand, handle multi-label classification tasks where each sample can be assigned multiple labels. This can be expressed as:

$$Y = f_{MLC}(x) \subseteq \{y_1, y_2, \ldots, y_n\} \tag{5}$$

The combination of BC and MLC forms the core of the CalmDroid active learning process, as multi-label classification problems can often be reduced to a series of single-label classification tasks. By combining base classifiers, such as Random Forest or ensemble learning methods, the model's robustness and accuracy are significantly enhanced. Current machine learning frameworks and tools provide comprehensive support for base classifiers and multi-label classification, making the integration of these methods convenient for constructing and deploying complex multi-label models.

After training on a given set of data, the CalmDroid outputs the training results for that specific time period and updates its internal model parameters. Following this, the temporal partitioning module provides the next set of updated samples, which are added to the training set. This iterative training process continues until all samples have been introduced into the model, at which point the final testing results are generated.

III. Evaluation

This section evaluates the performance of the proposed model, CalmDroid, on real-world datasets that evolve over time. First, we introduce the datasets, evaluation metrics, and experimental setup. Next, we conduct a detailed analysis of the model's improvement through various experiments combining different MLC algorithms and BC. Finally, we validate the effectiveness of the model under different noise conditions on the VirusShare by selecting the best-performing MLC and BC combinations from the Drebin.

A. Datasets

In this experiment, we utilized two well-known Android malware datasets: DREBIN [11] and VirusShare [12]. The DREBIN contains 5,560 Android applications collected from August 2010 to October 2012, covering 179 distinct malware families. For this study, we selected 5,456 samples from the dataset for analysis. Additionally, we collected 5,500 Android malware samples from the VirusShare, spanning from 2019 to 2022, which represent more modern malware behaviors. By using these two datasets, we can evaluate the model's performance in a time-evolving, real-world scenario when conducting active learning under various noise conditions.

B. Evaluation Metrics

Let L denote the set of six predefined labels, where $|L| = 6$ represents the total number of labels. The test dataset used to evaluate the pre-trained models is denoted as D, with $|D|$ representing the total number of samples. Each sample in D is denoted by x_i.

For each sample x_i, the true labels are represented as a boolean vector Y_i of length $|L|$, where $Y_i = (Y_{i,1}, \ldots, Y_{i,j}, \ldots, Y_{i,|L|})$. Each $Y_{i,j}$ takes a value of either 1 or 0, indicating whether sample x_i is relevant to the malicious behavior associated with label j.

Similarly, the classification results for sample x_i are represented by a boolean vector $Z_i = (Z_{i,1}, \ldots, Z_{i,j}, \ldots, Z_{i,|L|})$, where $Z_{i,j}$ is 1 if the pre-trained model assigns label j to the sample, and 0 otherwise.

*1) **Sample-ACC**:* Sample-ACC [13] is used to evaluate the overall accuracy of the pre-trained model on the test dataset and serves as a benchmark metric reflecting the effectiveness of the multi-label classification approach. As the primary metric, Sample-ACC helps offer valuable insights for subsequent analysis, with supplementary metrics supporting it.

In MLC, ensuring that all $|L|$ labels of a given sample are correctly classified is highly challenging, particularly when the labeled dataset is limited. Therefore, we adopt an alternative definition for classification correctness. A sample x_i is considered correctly classified if its classification result satisfies the following two conditions:

- At least one malicious behavior is correctly detected (true positive: $Y_{i,j} = 1 \wedge Z_{i,j} = 1$).
- No false positives are predicted for nonexistent behaviors (false positive: $Y_{i,j} = 0 \wedge Z_{i,j} = 0$).

The mathematical definition of Sample-ACC is provided as follows:

$$\begin{cases} C_i = \{Z_{i,j} \mid \neg((Y_{i,j} = 0) \wedge (Z_{i,j} = 1)), j \in [1, |L|]\}, \\ C = \{x_i \mid \exists C_i \ni 1, x_i \in D, i \in [1, |D|]\}, \\ \text{Sample-ACC} = \frac{|C|}{|D|} \end{cases} \tag{6}$$

Where C_i represents the classification results of sample x_i after excluding false positives, and C denotes the set of correctly classified samples where each sample has at least one malicious behavior detected. Sample-ACC is then defined as the ratio of correctly classified samples to the total number of samples in the test dataset.

*2) **Label-ACC**:* To assess the model's detection capability for each individual label, we introduce the Label-ACC metric. This metric measures the model's accuracy for a specific label, with higher values indicating better performance for that label. Due to the limited and imbalanced nature of labeled data, overall accuracy may obscure the model's deficiencies on labels with fewer samples. Label-ACC, however, is able to reveal such shortcomings.

$$\begin{cases} C_j = \{x_i | \forall x_i \in D, i \in N \cap (0, |D|], \\ ((Y_{i,j} = 1) \wedge (Z_{i,j} = 1)) \vee ((Y_{i,j} = 0) \wedge (Z_{i,j} = 0))\}, \\ D_j = \{x_i | \forall x_i \in D, Y_{i,j} = 1, i \in N \cap (0, |D|]\}, \\ \text{label-ACC} = \frac{|C_j|}{|D_j|}; \end{cases} \tag{7}$$

Here, C_j represents the set of correctly classified samples for label j, while D_j denotes the set of all samples associated with label j.

*3) **Auxiliary Metrics**:* In addition to the two primary metrics for classification tasks, particularly for multi-label classification, we also utilize several auxiliary metrics: Hamming loss, Zero-one loss, and F1-score [47, 48, 49]. Hamming loss represents the proportion of misclassified labels among all samples; the smaller the Hamming loss, the better the model's performance. Zero-one loss reflects the proportion of misclassified samples in the entire dataset; similarly, a lower Zero-one loss indicates better model performance. F1-score captures both classification accuracy and completeness, with higher F1-scores signifying better model effectiveness.

C. Experimental Enviroment

All experiments were conducted on a Lenovo G5000 IRH8, equipped with a 13th Gen Intel(R) Core(TM) i7-13700H processor (clock speed 2.40 GHz) and 16GB RAM. Python was the primary programming language for our implementation. In this experiment, we selected MEKA v1.9.2 as the core toolkit for training and evaluating the model. MEKA is designed to provide a variety of algorithms and evaluation metrics specifically for multi-label classification problems. MEKA [14] is an open-source project based on WEKA [15]. By changing the base BC and MLC used in the CalmDroid within MEKA, we further validated the model's capabilities. For all experiments, the training-to-test set ratio was set to 8:2. The training set comprised the complete data prior to the strategy-based point selection. To ensure fairness in the experimental results, the number of points selected for both strategies was set to approximately half of the training set.

D. Results on Drebin

1) Experimental Setup: We tested various combinations of MLC and BC and ultimately selected five well-performing combinations to serve as the foundation for the CalmDroid. To validate the improvements of our active learning strategy under noisy environments, compared to the traditional confuse-point selection strategy, we created different noise levels: 5%, 10%, and 15%. By comparing the performance of CalmDroid using the Core-set strategy against the Confuse-set strategy under these varying noise intensities, we aimed to demonstrate the superiority of the proposed method in handling time-evolving noisy environments.

2) Comparison of Core-set and Confuse-set Strategies on the Drebin under 5% Noise: As shown in Table I, under the 5% noise condition, the performance of the Core-set strategy remains relatively stable across five combinations of BC and

TABLE I: 5% noise detailed comparison of metrics for different algorithm combinations in different strategies on Drebin.

Strategy	Core (with MLC)					Confuse (with MLC)				
	CDN	CDN	BR	CC	CDN	CDN	CDN	BR	CC	CDN
Basic Classifier	LMT	J48	Random Forest	REP Tree	REP Tree	LMT	J48	Random Forest	REP Tree	REP Tree
Hamming Loss	0.087	0.146	**0.034**	0.041	0.114	0.454	0.495	**0.045**	0.126	0.291
Zero-One Loss	0.263	0.422	**0.180**	0.185	0.385	0.726	0.928	**0.200**	0.344	0.572
F1-Score	0.739	0.485	0.780	**0.859**	0.667	0.275	0.397	**0.895**	0.702	0.303
Sample-ACC	0.837	0.824	**0.941**	0.865	0.833	0.467	0.120	**0.911**	0.726	0.670
Avg. Label-ACC	0.913	0.854	**0.966**	0.959	0.886	0.546	0.505	**0.955**	0.874	0.709
+-ACC	+0.367	**+0.704**	+0.030	+0.139	+0.163	-0.367	-0.704	**-0.030**	-0.139	-0.163

(a) Avg. Label-ACC. is the average of all labels
(b) +-ACC represent the difference in Sample-ACC between the core-set and the confusion-set

MLC. All combinations yield a Sample-ACC greater than 0.824 and an Avg. Label-ACC above 0.854. This indicates that the Core-set strategy is highly adaptable in low-noise environments, maintaining consistent performance. In contrast, the Confuse-set strategy shows much greater performance fluctuations under the same conditions, with notably less stability. When comparing the same combinations, Core-set consistently outperforms Confuse-set in terms of Sample-ACC. Although the gap between the two strategies narrows to a mere 0.030 in the Random Forest + BR combination, Core-set still surpasses Confuse-set. The largest performance gap is observed in the CDN + J48 combination, where the difference reaches an impressive 0.704, further demonstrating the superiority of Core-set in low-noise environments.

In the Core-set strategy, the best-performing combination is Random Forest + BR, with a Sample-ACC of 0.941 and an Avg. Label-ACC of 0.966. Comparatively, the best-performing combination under the Confuse-set strategy is Random Forest + BR, with a Sample-ACC of 0.911 and an Avg. Label-ACC of 0.955. Although Confuse-set performs well in certain combinations, overall, it falls short compared to Core-set, particularly in the key metrics of Sample-ACC and Avg. Label-ACC.

When analyzing the worst-performing combinations, even the least effective combination in the Core-set strategy, LMT + CDN, still yields a Sample-ACC of 0.824 and an Avg. Label-ACC of 0.854. This remains superior to the worst-performing combination in the Confuse-set strategy, J48 + CDN, which produces a Sample-ACC of just 0.120 and an Avg. Label-ACC of 0.505. This stark contrast highlights the higher risk of misclassification with the Confuse-set strategy, especially when handling complex multi-label classification tasks in low-noise environments.

From the perspective of misclassification control, the Core-set strategy's Random Forest + CDN combination performs best in terms of Hamming Loss, achieving a low value of 0.034, indicating a smaller proportion of misclassified labels. While Confuse-set shows some promise, its best-performing combination, Random Forest + BR, has a slightly higher Hamming Loss of 0.045, suggesting inferior performance in terms of misclassification. Additionally, Confuse-set struggles with Zero-One Loss, particularly with the J48 + CDN combination, which records a high value of 0.928. This suggests that Confuse-set has weaker control over misclassifications in low-noise environments.

In summary, under 5% noise conditions, the Core-set strategy outperforms the Confuse-set strategy not only in Sample-ACC and Avg. Label-ACC but also in misclassification control. Particularly in the Random Forest + CDN combination, the Core-set strategy's performance significantly exceeds that of the Confuse-set strategy, proving its effectiveness in selecting representative samples and enhancing classification performance, especially in stable, low-noise environments.

3) ***Comparison of Core-set and Confuse-set Strategies on the Drebin under 10% Noise:*** As the noise level increases to 10%, in Table II the performance differences between the Core-set and Confuse-set strategies become more pronounced. Similar to the 5% noise scenario, the Core-set strategy continues to demonstrate stability across various metrics. Although there is some fluctuation due to the increased noise, all five combinations of BC and MLC in the Core-set strategy maintain a Sample-ACC above 0.770, with the highest value being 0.937, achieved by the Random Forest + BR combination. On the other hand, the Confuse-set strategy still performs well in some combinations, such as Random Forest + BR, where the Sample-ACC reaches 0.896. However, the other combinations show more significant variation, with the lowest Sample-ACC being just 0.380, indicating inconsistent performance.

Even when comparing the same methods, Core-set consistently outperforms Confuse-set. The smallest gap between the two strategies is 0.041 in the Random Forest + BR combination, slightly larger than the difference under the 5% noise condition. In the largest gap, found in the CDN + REPtree combination, the difference reaches 0.451, reflecting the continued superior performance of the Core-set strategy in environments with moderate noise levels.

In terms of Avg. Label-ACC, the Core-set strategy retains high accuracy across all combinations, with values exceeding 0.746, and the highest being 0.972, again achieved by the Random Forest + BR combination. In contrast, the Confuse-set strategy shows more volatility, with the lowest Avg. Label-ACC recorded at 0.603, and only the Random Forest + BR combination achieving a value of 0.948. This highlights the Confuse-set strategy's instability under higher noise levels.

From the misclassification perspective, the Core-set strategy continues to excel in controlling misclassified labels. The lowest Hamming Loss is 0.028, achieved by the Random Forest + BR combination, whereas the best Hamming Loss under the

TABLE II: 10% noise detailed comparison of metrics for different algorithm combinations in different strategies on Drebin.

Strategy	Core (with MLC)					Confuse (with MLC)				
	CDN	CDN	BR	CC	CDN	CDN	CDN	BR	CC	CDN
Basic Classifier	LMT	J48	Random Forest	REP Tree	REP Tree	LMT	J48	Random Forest	REP Tree	REP Tree
Hamming Loss	0.120	0.254	**0.028**	0.045	0.113	0.227	0.372	**0.052**	0.160	0.397
Zero-One Loss	0.331	0.487	**0.146**	0.191	0.352	0.624	0.707	**0.202**	0.319	0.756
F1-Score	0.687	0.284	0.790	**0.858**	0.571	0.642	0.252	**0.836**	0.652	0.330
Sample-ACC	0.781	0.770	**0.937**	0.860	0.831	0.489	0.543	**0.896**	0.726	0.380
Avg. Label-ACC	0.880	0.746	**0.972**	0.955	0.887	0.773	0.628	**0.948**	0.840	0.603
+-ACC	+0.292	+0.227	+0.041	+0.134	**+0.451**	-0.292	-0.227	**-0.041**	-0.134	-0.451

Confuse-set strategy is 0.052. Misclassification rates are particularly high in the J48 + CDN combination under the Confuse-set strategy, with a Hamming Loss of 0.372, indicating greater susceptibility to noise interference. Additionally, the Zero-One Loss under the Confuse-set strategy's worst combination reaches 0.756, significantly higher than the Core-set strategy's highest value of 0.487.

In terms of F1-score, the Core-set strategy's REP Tree + CC combination performs best, with a score of 0.858, while the best-performing combination under the Confuse-set strategy is Random Forest + BR, with a score of 0.836. Despite some favorable results in the Confuse-set strategy, the Core-set strategy remains more stable as noise increases.

Overall, as noise increases to 10%, the Core-set strategy maintains consistent performance across multiple key metrics, while the Confuse-set strategy exhibits larger fluctuations, especially in certain combinations. This demonstrates the superior adaptability of the Core-set strategy in noisy environments, where its representative sample selection plays a crucial role.

*4) **Comparison of Core-set and Confuse-set Strategies on the Drebin under 15% Noise**:* As noise intensity further increases to 15%, the performance gap between the Core-set and Confuse-set strategies becomes even more evident. According to the data in Table III, despite the higher uncertainty introduced by the noise, the Core-set strategy continues to perform well across key metrics such as Sample-ACC and Avg. Label-ACC. In the Core-set strategy, all combinations maintain a Sample-ACC above 0.591, with only one combination falling below 0.78. The highest value, 0.944, is achieved by the Random Forest + BR combination. By contrast, the best Sample-ACC in the Confuse-set strategy is 0.689 (achieved by the REP Tree + BR combination), while the lowest value is 0.356.

When comparing the same methods, Core-set again consistently outperforms Confuse-set. The smallest gap, observed in the CC + REPtree combination, is 0.187, slightly larger than under the 10% noise condition. In the largest gap, found in the CDN + J48 combination, the difference is 0.387, which, although reduced from the 10% noise condition, remains significant. This reflects the continued strong performance of the Core-set strategy in a highly noisy environment.

In terms of Avg. Label-ACC, the Core-set strategy remains more stable, with all combinations achieving values above 0.675 and only one combination below 0.850. The highest value is 0.966, while the Confuse-set strategy shows much greater variability, with the lowest Avg. Label-ACC being 0.656 and the highest only 0.907. These results suggest that even with 15% noise, the Core-set strategy still has a distinct advantage in label-level classification accuracy.

Hamming Loss and Zero-One Loss analyses further support this trend. The Core-set strategy's Random Forest + BR combination achieves the lowest Hamming Loss at 0.034, while the best Hamming Loss under the Confuse-set strategy is 0.093. Misclassification instability is particularly pronounced in the LMT + CDN combination under the Confuse-set strategy, where the Hamming Loss reaches 0.344, showing that the Confuse-set strategy is more vulnerable to noise interference at higher levels. In terms of Zero-One Loss, the worst-performing combination in the Core-set strategy, LMT + CDN, still only reaches 0.613, while the highest value under the Confuse-set strategy is 0.785, further illustrating the Confuse-set strategy's sharp decline in performance when handling noisier data.

Regarding F1-score, the Core-set strategy's Random Forest + BR combination again performs best, achieving 0.789, while the Confuse-set strategy's best combination, Random Forest + BR, scores 0.744. Although the Confuse-set strategy performs well in certain combinations, its overall stability is weaker.

In conclusion, as the noise intensity rises to 15%, the Core-set strategy excels in selecting representative samples and controlling misclassifications, maintaining high classification performance. While the Confuse-set strategy demonstrates potential in certain combinations, its overall volatility increases, particularly when noise is 15%.

*5) **Comprehensive Analysis of Two Strategies in the Drebin Under Noisy Conditions**:* In the CalmDroid, the performance of the Core-set and Confuse-set strategies shows significant differences under 5%, 10%, and 15% noise conditions. Overall, the Core-set strategy exhibits greater robustness and consistency in the face of noise interference, while the Confuse-set strategy demonstrates a marked decline in stability as noise intensity increases.

Robustness and Stability: At lower noise levels (5%), the Core-set strategy remains particularly stable, with almost all combinations of Sample-ACC and Avg. Label-ACC maintaining high levels. As noise increases to 10% and 15%, although classification performance declines, the Core-set strategy continues to achieve high classification accuracy, with its lowest Sample-ACC and Avg. Label-ACC remaining within acceptable ranges. This indicates that the Core-set strategy is better at identifying representative samples amidst noisy data, ensuring satisfactory classification performance even in high-noise environments.

TABLE III: 15% noise detailed comparison of metrics for different algorithm combinations in different strategies on Drebin.

Strategy	Core (with MLC)					Confuse (with MLC)				
	CDN	CDN	BR	CC	CDN	CDN	CDN	BR	CC	CDN
Basic Classifier	LMT	J48	Random Forest	REP Tree	REP Tree	LMT	J48	Random Forest	REP Tree	REP Tree
Hamming Loss	0.325	0.150	**0.034**	0.053	0.067	0.310	0.344	**0.093**	0.214	0.303
Zero-One Loss	0.613	0.423	**0.167**	0.307	0.256	0.759	0.785	**0.420**	0.348	0.548
F1-Score	0.267	0.555	**0.789**	0.754	0.737	0.529	0.388	**0.744**	0.603	0.371
Sample-ACC	0.591	0.789	**0.944**	0.876	0.804	0.356	0.402	0.669	**0.689**	0.635
Avg. Label-ACC	0.675	0.850	**0.966**	0.947	0.933	0.690	0.656	**0.907**	0.786	0.697
+-ACC	+0.235	**+0.387**	+0.275	+0.187	+0.169	-0.235	-0.387	-0.275	**-0.187**	-0.169

In contrast, the Confuse-set strategy achieves relatively good results at 5% noise but exhibits significant fluctuations as noise intensity increases, especially under 10% and 15% conditions. Notably, in certain combinations, such as LMT + CDN, the Sample-ACC of the Confuse-set drops sharply to 0.356, indicating its poor robustness and inability to effectively tackle the challenges posed by noisy data.

Misclassification Control: The ability to control misclassification is a key metric for assessing the robustness of active learning strategies. Analysis of Hamming Loss and Zero-One Loss reveals that the Core-set strategy maintains a lower misclassification rate across various noise levels, particularly outperforming the Confuse-set strategy under high noise (15%). Conversely, the Confuse-set strategy experiences a significant increase in misclassification risk as noise intensity rises, with some combinations exceeding a Zero-One Loss of 0.750 at 15% noise, indicating a propensity for substantial misclassification in high-noise environments.

Overall Trend: As noise levels increase from 5% to 15%, the Core-set strategy consistently outperforms the Confuse-set strategy across all noise conditions, particularly demonstrating better stability and misclassification control in high-noise scenarios. This trend suggests that the Core-set strategy effectively selects representative samples to mitigate the adverse effects of noise on model performance, thereby enhancing the model's generalization capability.

The fundamental reason for this performance difference lies in the sample selection mechanisms of the two strategies. The Core-set strategy employs a clustering approach to identify the most representative core points in the dataset. Even under noisy conditions, this strategy prioritizes selecting core samples that are most similar to other samples of the same class, thereby reducing noise interference in the classification task and ensuring model robustness. The clustering method can identify intra-class structures, filtering out noisy points and enhancing the model's generalization capacity.

In contrast, the Confuse-set strategy, by selecting ambiguous samples, tends to favor points with higher uncertainty. However, in noisy environments, the feature overlap between noisy samples and genuine samples increases, leading the Confuse-set to more readily select noisy points as training samples. This sensitivity to noise directly affects the classification performance of the Confuse-set, resulting in significant fluctuations and an increased risk of misclassification in high-noise conditions.

Thus, from the perspective of active learning, the Core-set strategy more precisely selects representative samples, effectively mitigating noise interference during model training. In contrast, the Confuse-set strategy, due to its sample selection mechanism, is more susceptible to noise interference, exhibiting greater instability.

*6) **Analysis of the Core-set Strategy Over Time in the Drebin Under 5% Noise:*** Under 5% noise conditions, the performance of the Core-set strategy in the CalmDroid demonstrates the positive impact of temporal evolution on classification performance. By analyzing Sample-ACC and Avg. Label-ACC across five different classifier combinations, we observe changes in overall performance over time.

In the Core-set strategy, as shown in Fig. 4 (a), the Sample-ACC and Avg. Label-ACC for the CDN + J48 combination exhibit a steady upward trend throughout the timeline. At Step 1, the Sample-ACC begins at a low point of approximately 0.5, but significant improvements are evident by Step 2, culminating in stability near 0.8 by Step 3. The Avg. Label-ACC starts at 0.6 and steadily increases to 0.9 by Step 3. This trend indicates that this combination is sensitive to the increasing quantity of training data over time, effectively leveraging data volume to enhance classification capabilities.

Similarly, in Fig. 4 (b), the CDN + REPTree combination shows a steady increase in both metrics over time. Sample-ACC rises from about 0.6 at Steps 1 and 2, reaching nearly 0.8 and 0.9 by Step 3. This temporal improvement further validates that, under low noise conditions, the Core-set strategy can enhance model performance as data accumulates.

In Fig. 4 (c), the performance of the BR + RandomForest combination under the Core-set strategy mirrors the previous combinations but shows more pronounced improvement. Both metrics consistently rise throughout the timeline, with Sample-ACC and Avg. Label-ACC exceeding 0.9 at Step 3. This trend reinforces the idea that the Core-set strategy optimizes training set quality over time through representative sample selection.

The CDN + LMT combination in Fig. 4 (d) exhibits different volatility compared to the prior combinations. Although there is a noticeable upward trend in Sample-ACC and Avg. Label-ACC during Steps 1 and 2, fluctuations occur at Step 3, particularly with Sample-ACC experiencing a decline and Avg. Label-ACC showing variability, yet both metrics remain above 0.8 and 0.9 overall. This volatility may relate to the complexity of label changes and the combination's lower sensitivity to data volume. Nonetheless, the Core-set strategy maintains high performance.

In Fig. 4 (e), the CC + REPTree combination displays a persistent upward trend in Avg. Label-ACC similar to Figures

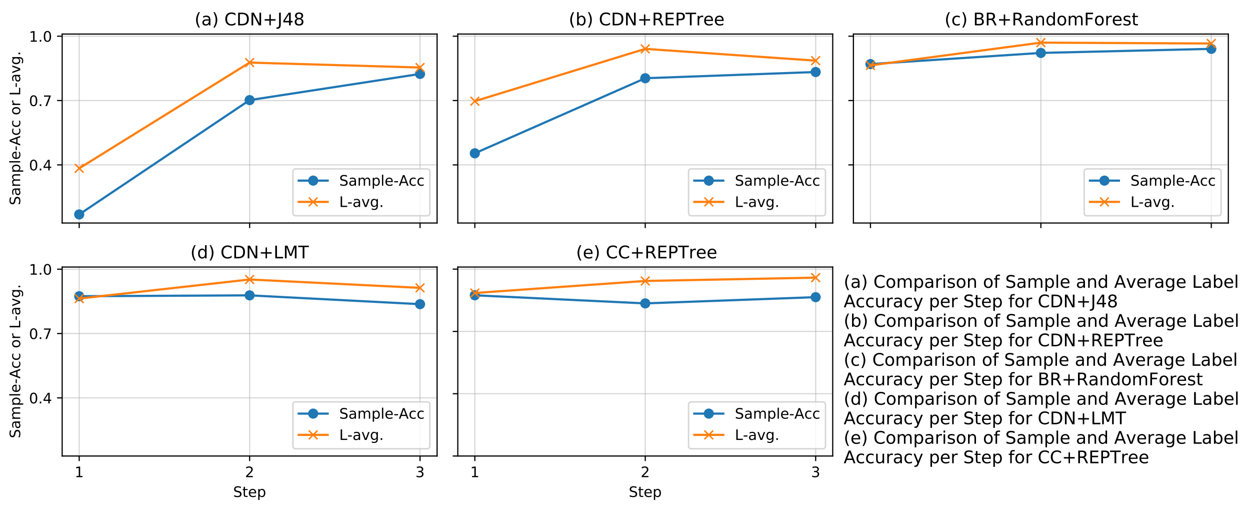

Fig. 4: Core-set Strategy Over Time in the Drebin Under 5% Noise.

a, b, and c, while Sample-ACC experiences a dip followed by recovery. Although Sample-ACC slightly decreases at Step 1, it rebounds above 0.8 by Step 3, with Avg. Label-ACC consistently rising to nearly 0.9. This pattern suggests initial insensitivity to data changes, which adjusts as the dataset grows over time.

The significant increases observed in Figures a and b indicate these combinations are sensitive to changes in data volume, particularly as sample sizes in the training set increase with time. Over the timeline, the active learning strategy expands the training set, peaking at Step 3, leading to marked improvements in Sample-ACC and Avg. Label-ACC. In contrast, Figures c, d, and e show relative insensitivity to initial data volume, displaying a more stable trend with minor impacts from increased data quantity.

Furthermore, the temporal trends of Sample-ACC in Figures a, b, and c illustrate that overall performance improves over time, supporting the positive correlation between data volume and model performance. Conversely, the fluctuations in Figures d and e indicate sensitivity to label changes within the training set. Notably, the label proportions in Steps 1 and 2 undergo significant alterations, reverting to a distribution similar to Step 1 by Step 3. This fluctuation is similar to findings that Android malware behaviors and labels evolve over time[16], reflecting specific patterns that manifest in classifier performance variations.

E. Experimental Results on the VirusShare

*1) **Experimental Setup**:* Based on our earlier experiments with the Drebin, the BR + RandomForest combination performed well under both the Core-set and Confuse-set strategies. Therefore, we utilize this combination as the model's BC and MLC in the VirusShare with varying noise levels.

*2) **Comprehensive Analysis of Two Strategies in the VirusShare Under Noisy Conditions**:* As shown in Table IV, under the Core-set strategy, Sample-ACC displays a relatively stable trend with increasing noise. At 5% noise, Sample-ACC reaches 0.945, rising to 0.961 at 10% noise, and slightly decreasing to 0.950 at 15%. This trend indicates that the Core-set strategy effectively mitigates noise interference, maintaining high classification accuracy even in higher noise environments.

The Avg. Label-ACC follows a similar trend, steadily increasing from 0.953 at 5% noise to 0.962 at 10%, then slightly dropping to 0.960 at 15%. This suggests that the Core-set strategy retains good label classification performance even under high noise. Additionally, Hamming Loss exhibits slight fluctuations, decreasing from 0.047 at 5% noise to 0.038 at 10%, then rising to 0.040 at 15%, with overall minor variations. Zero-One Loss remains low, demonstrating effective misclassification control, while the F1 Score remains stable with minimal variation.

Conversely, the Confuse-set strategy shows pronounced sensitivity to increasing noise levels. Sample-ACC drops sharply from 0.921 at 5% noise to 0.845 at 10%, and further declines to 0.784 at 15%, highlighting its instability in high-noise environments. Similarly, Avg. Label-ACC decreases from 0.957 at 5% noise to 0.950 at 10%, and then to 0.921 at 15%. As noise increases, the Confuse-set strategy experiences significant deterioration in label classification performance, particularly in high-noise conditions.

In terms of misclassification control, both Hamming Loss and Zero-One Loss significantly increase with noise, indicating a higher misclassification risk for the Confuse-set strategy under high noise. Notably, at 15% noise, Zero-One Loss reaches 0.261, significantly exceeding that of the Core-set strategy. Additionally, the F1 Score trends downward, reflecting the deterioration of the Confuse-set strategy's performance in label classification tasks as noise increases.

The Core-set strategy consistently outperforms the Confuse-set strategy under all noise conditions, particularly at 10% and 15% noise, where the Core-set strategy maintains a Sample-ACC around 0.95, while the Confuse-set drops to 0.784. This confirms the superior robustness of the Core-set strategy in

TABLE IV: Performance of BR Random Forest on Core and Confuse strategies under different noise levels on VirusShare.

Metric	Core Strategy			Confuse Strategy		
	5% Noise	10% Noise	15% Noise	5% Noise	10% Noise	15% Noise
Sample-ACC	0.945	0.961	0.95	0.921	0.845	0.784
Hamming Loss	0.047	0.038	0.040	0.043	0.050	0.079
F1 Score	0.437	0.432	0.438	0.413	0.403	0.387
0/1 Loss	0.242	0.174	0.192	0.152	0.211	0.261
Label Accuracy	-	-	-	-	-	-
L1	0.989	0.984	0.987	0.966	0.974	0.955
L2	1	1	1	0.997	1	0.968
L3	0.8	0.863	0.842	0.895	0.908	0.887
L4	1	1	1	0.997	1	0.976
L5	0.992	0.992	0.992	0.995	0.997	0.966
L6	0.937	0.934	0.937	0.889	0.824	0.774
Avg. Label-ACC	0.953	0.962	0.960	0.957	0.950	0.921

sustaining high classification accuracy in noisy environments.

Avg. Label-ACC comparisons reveal a similar trend, with the Core-set strategy maintaining higher average label classification accuracy across all noise levels. At 15% noise, the Confuse-set's Avg. Label-ACC declines to 0.921, while the Core-set remains at 0.960, further validating the advantages of the Core-set strategy in multi-label classification tasks.

Comparing Hamming Loss and Zero-One Loss demonstrates the Core-set strategy's excellent misclassification control, particularly in high-noise conditions, where its Hamming Loss remains consistently lower than that of the Confuse-set strategy, which experiences significant instability as noise increases, especially at 15% noise when its Zero-One Loss peaks.

At the label level, the Core-set strategy consistently achieves near-perfect accuracy (approaching 1) for labels such as L1, L2, and L4, whereas the Confuse-set exhibits substantial fluctuations as noise increases. Notably, the accuracy for label L6 drops to 0.774 under 15% noise for the Confuse-set, while the Core-set sustains 0.937. This indicates the Core-set strategy's superior robustness in handling complex labels and its effectiveness in mitigating label-level noise interference.

Overall, the Core-set strategy employed by the CalmDroid significantly outperforms the Confuse-set strategy across varying noise environments, particularly in terms of Sample-ACC and Avg. Label-ACC. Even under higher noise levels, the Core-set strategy maintains high classification accuracy and effective misclassification control, while the Confuse-set strategy is highly sensitive to noise, exhibiting marked performance deterioration with increasing noise. This disparity highlights the superior robustness and adaptability of the CalmDroid utilizing the Core-set strategy in high-noise conditions.

IV. Threat to Validity

The external validity in this work comes from the selected datasets. Although anonymousCERT contains only 180 malware samples, all labels undergo verification and voting by all co-authors. The labels are reliable, and supported by detailed security reports. The multi-label labels in Drebin and VirusShare result from clustering. We state that these may contain noise. However, CalmDroid is designed to tolerate noisy data. Experimental results confirm this conclusion. The predictive performance of Drebin and VirusShare also briefly demonstrates the robustness of CalmDroid.

Internal validity relates to feature selection. We extract static features of malware, such as API, permission, etc. These static features are derived from the security reports in anonymousCERT. All 531 static features, including 6 labels (e.g., message and ad), are linked to understanding malware behavior based on source code. From this perspective, we consider this work a significant contribution to program analysis and comprehension. Additionally, Li et al. [46] also demonstrate the importance of static features in comprehending malicious behaviors.

V. Related Work

To identify whether an Android application is malicious, feature extraction is the fundamental step. Malware detection can be categorized into three main types: static analysis [22, 23, 24, 25, 26, 27, 28]; dynamic analysis [29, 30] ; and hybrid approaches [31, 32, 33]. However, basic detection methods offer limited alerts for various malware types, necessitating finer granularity in classification.

Currently, Android malware classification is divided into single-label [34, 35, 36] and multi-label classification [13]. In single-label classification, [37] Hassen introduced a neural network representation method for open set recognition, while [38] Jiang et al. proposed a one-vs-all ensemble method to enhance accuracy by establishing strict decision boundaries. Research [39] presented a continuous learning framework that uses active learning to update models for newly emerging malware. Multi-label classification, as explored by [13] Qiao et al., faces additional challenges, employing active learning to categorize unlabeled samples from small, labeled datasets.

Active learning models, particularly the confuse-set strategy [40, 41, 43], have gained interest. [44] Guerra-Manzanares utilized this strategy to select uncertain samples for expert labeling, improving IoT botnet detection accuracy while reducing labeling resource demands. Conversely, the core-set strategy has seen limited application in Android malware classification but has proven effective in other domains. Qiang et al. [45] applied this strategy to intelligently select informative code samples, significantly enhancing training efficiency and reducing the need for manual annotation.

VI. Conclusion

This work presents the CalmDroid, which integrates Core-Set strategy and active learning for multi-label Android malware classification in noisy environments. We show that, in this challenging setting, CalmDroid achieves superior performance compared to conventional strategies like the confuse-set approach, delivering more stable and accurate classification despite noise interference. The model also adapts to the evolving nature of Android malware over time, making it a promising solution for future security tasks.

ACKNOWLEDGMENT

This work was partially supported by the National Natural Science Foundation of China (62302148, 62472309), Hebei Natural Science Foundation (F2024202076) and Hebei Yanzhao HuangJintai Talents Program (Postdoctoral Platform) (B2024003002).

REFERENCES

[1] Number of smartphone mobile network subscriptions worldwide from 2016 to 2023, with forecasts from 2023 to 2028, [Online], Accessed October, 2024. https://www.statista.com/statistics/330695/.

[2] Global smartphone sales share by operating system, [Online], Accessed October, 2024. https://www.counterpointresearch.com/insights/global-smartphone-os-market-share/.

[3] Y. Shen, P.-A. Vervier, G. Stringhini, A large-scale temporal measurement of android malicious apps: Persistence, migration, and lessons learned, in: 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 1167–1184.

[4] Y. Bai, S. Chen, Z. Xing, X. Li, Argusdroid: detecting android malware variants by mining permission-api knowledge graph, Science China Information Sciences 66 (2023) 192101.

[5] B. Wu, S. Chen, C. Gao, L. Fan, Y. Liu, W. Wen, and M. R. Lyu, ”Why an android app is classified as malware: Toward malware classification interpretation,” ACM Trans. Softw. Eng. Methodol., vol. 30, no. 2, pp. 1-29, 2021.

[6] C. Wu, S. Chen, J. Li, R. Chai, L. Fan, X. Xie, and R. Feng, ”Beyond Decision: Android Malware Description Generation through Profiling Malicious Behavior Trajectory,” ACM Trans. Softw. Eng. Methodol., accepted on 10 January 2025, to be published.

[7] K. Allix, T. F. Bissyandé, J. Klein, Y. Le Traon, Androzoo: Collecting millions of android apps for the research community, in: Proceedings of the 13th international conference on mining software repositories, 2016, pp. 468–471.

[8] H. Li, G. Xu, L. Wang, X. Xiao, X. Luo, G. Xu, H. Wang, Malcertain: Enhancing deep neural network based android malware detection by tackling prediction uncertainty, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE ’24, 2024.

[9] J. Yang, S. Ma, Z. Zhang, Y. Li, S. Xiao, J. Wen, W. Lu, X. Gao, Say no to redundant information: Unsupervised redundant feature elimination for active learning, IEEE Transactions on Multimedia 26 (2024) 7721–7733.

[10] VirusShare, [Online], Accessed October, 2024. https://virusshare.com.

[11] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket.,” in Ndss, 2014, pp. 23–26. Accessed: Sep. 30, 2024.

[12] VirusShare.com - Because Sharing is Caring. [Online]. Available: https://virusshare.com/

[13] Q. Qiao, R. Feng, S. Chen, F. Zhang, and X. Li, “Multi-label classification for android malware based on active learning,” IEEE Transactions on Dependable and Secure Computing, 2022, Accessed: Sep. 30, 2024.

[14] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes, “Meka: a multi-label/multi-target extension to weka,” Journal of Machine Learning Research, vol. 17, no. 21, pp. 1–5, 2016.

[15] Meka project. [Online]. Available: https://github.com/Waikato/meka/tree/f0cc96133399afa4c80d2b8a6342913fe9057fb0

[16] A. M. R. AlSobeh, K. Gaber, M. M. Hammad, M. Nuser, and A. Shatnawi, “Android malware detection using time-aware machine learning approach,” Cluster Comput, vol. 27, no. 9, pp. 12627–12648, Dec. 2024.

[17] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, “A performance-sensitive malware detection system using deep learning on mobile devices,” IEEE Transactions on Information Forensics and Security, vol. 16, pp. 1563–1578, 2020.

[18] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, “Androdialysis: Analysis of android intent effectiveness in malware detection,” computers & security, vol. 65, pp. 121–134, 2017.

[19] Y. Bai, Z. Xing, X. Li, Z. Feng, and D. Ma, “Unsuccessful story about few shot malware family classification and siamese network to the rescue,” in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul South Korea: ACM, Jun. 2020, pp. 1560–1571.

[20] Android developer docs. [Online]. Available: https://developer.android.google.cn/reference

[21] S. Chen et al., “An empirical assessment of security risks of global Android banking apps,” in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul South Korea: ACM, Jun. 2020, pp. 1310–1322.

[22] B. Wang, C. Yang, and J. Ma, “Iafdroid: Demystifying collusion attacks in android ecosystem via precise inter-app analysis,” IEEE Transactions on Information Forensics and Security, vol. 18, pp. 2883–2898, 2023.

[23] K. O. Elish, M. O. Elish, and H. M. Almohri, “Lightweight, effective detection and characterization of mobile malware families,” IEEE Transactions on Computers, vol. 71, no. 11, pp. 2982–2995, 2022.

[24] H. Gao, S. Cheng, and W. Zhang, “GDroid: Android malware detection and classification with graph convolutional network,” Computers & Security, vol. 106, p. 102264,

2021.

[25] G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, “Towards an interpretable deep learning model for mobile malware detection and family identification,” Computers & Security, vol. 105, p. 102198, 2021.

[26] Y. Wu, S. Dou, D. Zou, W. Yang, W. Qiang, and H. Jin, “Contrastive learning for robust android malware familial classification,” IEEE Transactions on Dependable and Secure Computing, 2022, Accessed: Sep. 30, 2024.

[27] L. Shi et al., “VAHunt: Warding Off New Repackaged Android Malware in App-Virtualization’s Clothing,” in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event USA: ACM, Oct. 2020, pp. 535–549.

[28] Bai, Y., Xing, Z., Li, X., Feng, Z., & Ma, D. Unsuccessful story about few shot malware family classification and siamese network to the rescue. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE), 2020, pp. 1560-1571.

[29] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, “DL-Droid: Deep learning based android malware detection using real devices,” Computers & Security, vol. 89, p. 101663, 2020.

[30] E. Amer, I. Zelinka, and S. El-Sappagh, “A multi-perspective malware detection approach through behavioral fusion of api call sequence,” Computers & Security, vol. 110, p. 102449, 2021.

[31] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian, “Ec2: Ensemble clustering and classification for predicting android malware families,” IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 2, pp. 262–277, 2017.

[32] Q. Han, V. S. Subrahmanian, and Y. Xiong, “Android malware detection via (somewhat) robust irreversible feature transformations,” IEEE Transactions on Information Forensics and Security, vol. 15, pp. 3511–3525, 2020.

[33] E. Amer and S. El-Sappagh, “Robust deep learning early alarm prediction model based on the behavioural smell for android malware,” Computers & Security, vol. 116, p. 102670, 2022.

[34] T. Lu and J. Wang, “F2DC: Android malware classification based on raw traffic and neural networks,” Computer Networks, vol. 217, p. 109320, 2022.

[35] J. Tang, R. Li, Y. Jiang, X. Gu, and Y. Li, “Android malware obfuscation variants detection method based on multi-granularity opcode features,” Future Generation Computer Systems, vol. 129, pp. 141–151, 2022.

[36] B. Yuan, J. Wang, D. Liu, W. Guo, P. Wu, and X. Bao, “Byte-level malware classification based on markov images and deep learning,” Computers & Security, vol. 92, p. 101740, 2020.

[37] M. Hassen and P. K. Chan, “Learning a neural-network-based representation for open set recognition,” In Proceedings of the 2020 SIAM International Conference on Data Mining, 2020, pp. 154-162. Society for Industrial and Applied Mathematics.

[38] Jang, Jaeyeon, and Chang Ouk Kim. ”Collective decision of one-vs-rest networks for open-set recognition.” IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 2 , pp. 2327-2338, 2022.

[39] Y. Chen, Z. Ding, and D. Wagner, “Continuous learning for android malware detection,” in 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 1127–1144. Accessed: Sep. 30, 2024.

[40] A. Chaudhary, A. Anastasopoulos, Z. Sheikh, and G. Neubig, “Reducing confusion in active learning for part-of-speech tagging,” Transactions of the Association for Computational Linguistics, vol. 9, 2021, Accessed: Sep. 30, 2024.

[41] D. Arp et al., “Dos and don’ts of machine learning in computer security,” in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 3971–3988. Accessed: Sep. 30, 2024.

[42] R. Feng, S. Li, S. Chen, M. Ge, X. Li, and X. Li, ”Unmasking the Lurking: Malicious Behavior Detection for IoT Malware with Multi-label Classification,” in Proc. 25th ACM SIGPLAN/SIGBED Int. Conf. Languages, Compilers, and Tools for Embedded Syst., Jun. 2024, pp. 95-106.

[43] D. Yoo and I. S. Kweon, “Learning loss for active learning,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 93–102. Accessed: Sep. 30, 2024.

[44] A. Guerra-Manzanares and H. Bahsi, “On the application of active learning for efficient and effective IoT botnet detection,” Future Generation Computer Systems, vol. 141, pp. 40–53, 2023.

[45] Q. Hu et al., “Active Code Learning: Benchmarking Sample-Efficient Training of Code Models,” IEEE Transactions on Software Engineering, 2024, Accessed: Sep. 30, 2024.

[46] Md Shamsujjoha, John Grundy, Li Li, Hourieh Khalajzadeh, Qinghua Lu, Checking app behavior against app descriptions: What if there are no app descriptions?, in: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), 2021, pp. 422–432.

[47] Wang C, Yu H, Li X, et al. Dependency-Aware Microservice Deployment for Edge Computing: A Deep Reinforcement Learning Approach with Network Representation[J]. IEEE Transactions on Mobile Computing (TMC), 2024.

[48] Wang X, Dong Y, Jin D, et al. Augmenting affective dependency graph via iterative incongruity graph learning for sarcasm detection[C]. In Proceedings of the AAAI conference on artificial intelligence (AAAI). 2023, 37(4): 4702-4710.

[49] Zhang, J., He, R., Guo, F., & Liu, C. (2024, March). Quantum Interference Model for Semantic Biases of Glosses in Word Sense Disambiguation. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2024, pp. 19551-19559.