Multi-label Classification for Android Malware Based on Active Learning

Qijing Qiao, Ruitao Feng, Sen Chen, *Member, IEEE*, Fei Zhang, Xiaohong Li, *Member, IEEE*

Abstract—The existing malware classification approaches (i.e., binary and family classification) can barely benefit subsequent analysis with their outputs. Even the family classification approaches suffer from lacking a formal naming standard and an incomplete definition of malicious behaviors. More importantly, the existing approaches are powerless for one malware with multiple malicious behaviors, while this is a very common phenomenon for Android malware in the wild. So that both of them actually cannot provide researchers with a direct and comprehensive enough understanding of malware.
In this paper, we propose *MLCDroid*, an ML-based multi-label classification approach that can directly indicate the existence of pre-defined malicious behaviors. With an in-depth analysis, we summarize 6 basic malicious behaviors from real-world malware with security reports and construct a labeled dataset. We compare the results of 70 algorithm combinations to evaluate the effectiveness (best at 73.3%). Faced with the challenge of the expensive cost of data annotation, we further propose an active learning approach based on data augmentation, which can improve the overall accuracy to 86.7% with a data augmentation of 5,000+ high-quality samples from an unlabeled malware dataset. This is the first multi-label Android malware classification approach intending to provide more information on fine-grained malicious behaviors.

Index Terms—Android malware, multi-label classification, active learning, malicious behavior analysis

1 Introduction

Currently, Android is the most popular operating system (OS) on mobile devices with over 2.5 billion active users spanning over 190 countries [1]–[3]. In 2021, the total number of applications (apps) on Google Play has reached 2.8 million [4]. However, various apps brought not only convenience but also security threats through Android malware. A recent statistic shows the number of new malware amounted to 0.48 million per month [5]. That large number of malware directly or indirectly causes immeasurable harm to users' privacy and property [6], [7]. So the classification of Android malware is becoming more and more important.

Existing approaches for Android malware detection include traditional solutions (signature-based method [8], behavior-based method [9], [10], and data flow analysis-based method [11]) and machine learning (ML)-based solutions [12]–[24]. Compared with the former one, ML-based method is considered as one of the most promising techniques and achieves high detection accuracy. ML-based method can be mainly divided into two categories, binary classification [12]–[15], [17], [21], [22] and family classification [16], [19], [20]. Through the classic binary classification, we can successfully identify whether a certain application is malicious or benign. However, even if it is correctly classified, we still have no idea about its specific attack chain and corresponding malicious behaviors. In other words, the detection result can hardly provide any clue for security analysts, who aim at completing the knowledge of the detected 0-day attack efficiently and further provide potential victims with alerts and emergency protections. On the other hand, family classification's focal point is classifying malware into a certain family, whose definition can tell the key malicious behavior the malware may perform and the potential hazards the malware may have. However, along with the quick iterations on malware, existing family classification methods have two fatal limitations on their engineering basis, which severely undermine their capability towards providing more detailed and useful information for further analysis. Firstly, by investigating public malware family information [25]–[29], we notice there is no common agreement on malware naming or review procedure to oversee the family names. In other words, different agencies may categorize the same malware into different families. It may confuse researchers who attempt to provide a quick response to fight against the malware. Secondly, when analyzing the detailed malicious behaviors in real cases, we have also observed the behaviors in many malware are actually out of their family definitions. The reason behind this phenomenon is that these are mostly defined according to the original 0-day malware, which may only contain a specific malicious behavior. Hence, when some more variants are developed, even a successful family classification can become useless since most malware family definitions can cover only a specific and limited set of malicious behaviors. For those widely existing variants constructed with multiple behaviors, the result of family classification can fail to provide expected information. If we want to understand all the involved malicious behaviors, a great effort in the additional analysis is always necessary.

Considering all the above limitations, we conduct a survey on learning-based classification problems in other

- *Q. Qiao and R.Feng (co-first) contributed equally to this paper.*
- *S. Chen and X. Li are the corresponding authors.*
- *Q. Qiao, S. Chen, F. Zhang, X. Li are with the College of Intelligence and Computing, Tianjin University, China.*
  E-mail:{qqj, senchen, zhangfei, xiaohongli}@tju.edu.cn
- *R.Feng is with the CSCRC, University of New South Wales, Australia, and the SPIRIT, Nanyang Technological University, Singapore.*
  E-mail:ruitao.feng@unsw.edu.au, feng0082@ntu.edu.sg

domains, such as computer vision, to seek potential inspirations that can enable a stronger capability on providing more detailed malicious information in classification results, especially, for those malware containing multiple malicious behaviors. We find our task, which is locating the multiple malicious behaviors in malware as fully as possible, is quite similar to the multi-target detection [30], [31], which aims to locate all wanted targets in a picture. Both of them have the same task which is accurately locating multiple uncorrelated targets in a problem space. In the field of multiple malicious behavior detection, the "uncorrelated targets" are the malicious behaviors, and the "problem space" is the malware. Therefore, referring to the method used in multi-target detection, we try to adopt a multi-label learning method to provide a more comprehensive classification that has the potential to overcome the above limitations. In a multi-label classification (MLC) problem, multiple labels are assigned to each classifiable instance [32]. That means a single malware can be associated with a set of labels that represents different malicious behaviors simultaneously. So, if we can summarize these labels as reasonable and scientific as possible, a direct and full understanding on the malicious behaviors of a certain malware can be promised through the classification results. In addition to the exciting prospect, applying multi-label classification will also pose new challenges. Firstly, as far as we investigated, there is no formal general standard that aims to help in categorizing the detailed malicious program behaviors. That means we have to make relatively precise definitions at the program level for the multi-label classification. Secondly, despite there are usable Android malware datasets, however, the effort on analyzing and labeling them with precise labels at the program level can be extremely costly. Hence, achieving reasonable usability with a small labeled dataset and discovering an effective way to expand the size of the usable dataset are the other essential problems that need to be handled.

In this paper, we propose *MLCDroid*, an ML-based multi-label malware classification approach that can point out 6 types of fine-grained malicious behaviors in the classification results. To ensure the effectiveness of the predefined labels, we first summarize 6 types of basic malicious behaviors with the help of an in-depth analysis of a dataset of 180 malware samples with security reports. We then propose 6 inductive labels with a summarized feature dictionary and construct a new labeled dataset. Further, in order to enhance the completeness of the feature dictionary, we supplement it with relevant knowledge from various research papers and technical documents and finally obtain 531 features associated with 6 types of malicious behaviors. To investigate the effectiveness of potential multi-label ML algorithms, we evaluate 70 combinations of multi-label classification and basic classification algorithms on the labeled dataset. To address the challenge of data shortage and further enhance the effectiveness and reliability of our approach, we propose a method called *Detection-Training* by leveraging active learning. According to the evaluation results on diverse ML algorithms, we select 10 trained ML models that yield the best results and use them as the base models. By adopting a data augmentation method using the base models and unlabeled datasets, we not only enlarge the labeled dataset and obtain a better MLC model, but also validate the possibility of active learning in the malware classification domain towards understanding and explaining the inner malicious behaviors. Through our approach, we finally obtain a relatively high accuracy (i.e., CDN+J48: 86.7% on the DREBIN dataset and 83.3% on the VirusShare dataset) in our multi-label classification task. Compared with the accuracy of the base model constructed with the same algorithms (73.3%), the proposed Detection-Training outperforms with the help of a successful data augmentation of 4,840 additional unlabeled malware samples from the DREBIN dataset and 4,992 from VirusShare dataset.

In summary, we make the following main contributions:

- Generally, we propose *MLCDroid*, an approach that can perform multi-label classification on Android malware which can categorize them into fine-grained malicious behaviors.
- We perform an in-depth study in the field of Android malware to get a well pre-understanding of the malicious behaviors. With that in mind, we summarize 6 types of basic malicious behaviors and define a set of 6 inductive labels by manually analyzing 180 malware with security reports.[1]
- We basically select the combinations of multi-label algorithm and basic machine learning algorithm which are most suitable for our task by evaluating 70 combinations on the labeled dataset.
- Facing the challenges of the expensive cost on data annotation, we propose an active learning method, called Detection-Training, to enhance the classification capability with data augmentation from the unlabeled dataset. In this way, we can not only improve the effectiveness but also obtain auto-labeled high-quality malware.

2 Approach

1 presents the workflow of *MLCDroid* can be divided into 3 main phases: (1) Behavior analysis, feature selection, and data annotation: We propose 6 different behavior labels as a set with an in-depth analysis on the dataset of malware with security reports. Next, we summarize the feature dictionary with the help of static analysis and supplement it according to the relevant research papers and technical documents. After that, we manually label the malware with substantial efforts according to the observed malicious behaviors. (2) Base MLC model construction: By comparing the effectiveness of various multi-label classification and basic ML classification algorithms on the labeled dataset, we obtain a base model which performs the best among all candidates. (3) Detection-Training: With the obtained base model, we adopt an active learning method based on data augmentation to enlarge the labeled dataset and further improve the model accuracy. The details of the above three phases are introduced in the following sections respectively.

2.1 Behavior Analysis, Feature Selection, and Data Annotation

In order to gain a better understanding and further provide a comprehensive representation of the various malicious

1. We release the labeled malware as a benchmark (https://github.com/qqj1130247885/MLC-for-Android-Malware).

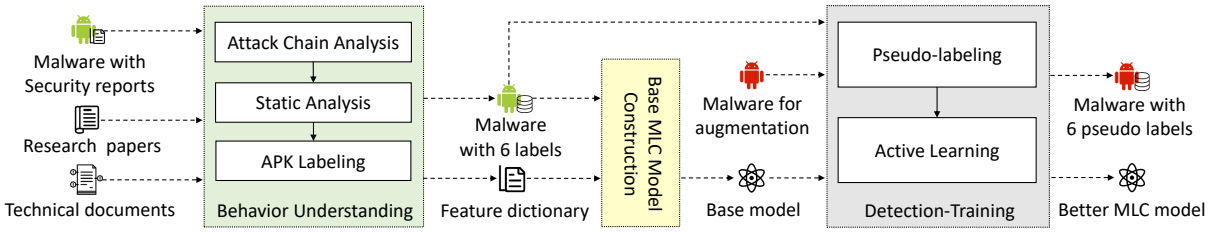


Fig. 1: An overview of *MLCDroid*

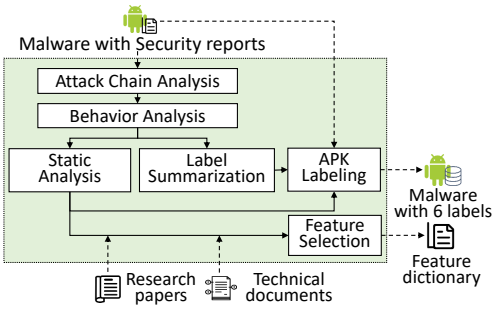


Fig. 2: Behavior analysis and data annotation

behaviors, we first conduct an in-depth analysis of real-world malware with the help of their security reports. These malware were collected in the period of 2008 to 2018. We collect these data from our industrial partner, anonymousCERT. Note that the security reports have been thoroughly validated and identified by the employees in anonymousCERT. Based on the analysis results, we then summarize 6 types of malicious behaviors. By applying an additional supplementation according to relevant research papers and technical documents, we propose a novel multi-label classification benchmark [33] with a summarized feature dictionary. The detailed workflow is shown in Fig. 2.

2.1.1 Attack chain and behavior analysis

To investigate every detail of malicious behaviors for ensuring the coverage of various types, we first study their attack chains by analyzing the corresponding security reports attached to the malware dataset. In this step, we mainly focus on the triggering events, violated assets, and suspicious actions which can help us understand the attack mechanism and actual flow in the implementation. Next, according to this key information, we further categorize the malicious behaviors with their functionalities and associated elements.

While performing behavior analysis, we also refer to the family classification results from VirusTotal [34]. By comparing our behavior analysis with those results, we notice an interesting conflict, which is the malicious behaviors in many malware that are actually out of their family definition. For example, a malware named "personal identity electronic certificate" is classified into the family named "Android:SMSThief" by Avast, AVG, and Alibaba, it is also classified into the family named "Android.SmsSpy" by DrWeb and F-secure. Generally, malware belonging to "Android:SMSThief" may contain the following malicious behaviors, such as "read mobile phone status", "receive, read, write or send text messages", "access network connections", "access to contact information", etc. Malware belonging to "Android.SmsSpy" contains similar malicious behaviors to those belonging to "Android:SMSThief", and can also access the WiFi network status information of the mobile phone, etc. However, by comparing with our analysis results, we find it also performs other malicious behaviors out of its family definition. In Listing. 1, the highlighted contents show that this malware can monitor the incoming and outgoing phone calls and make phone calls at run-time without the victim's authorization. Besides, it can also set up call forwarding. Obviously, this kind of situation may greatly affect family classification methods on their effectiveness, and even seriously mislead further analysis and behavioral interpretation. Therefore, a new classification standard that can provide an essential explanation on more precise knowledge of basic malicious behaviors is much needed.

2.1.2 Label definition

With the in-depth behavior analysis on the collected malicious apps and their security reports, we summarize 6 types of basic fine-grained malicious behaviors and define 6 inductive labels, which are "**SMS-related**", "**Internet-related**", "**Telephony-related**", "**Lock-in**", "**Ads**", and "**Re-infection**". A brief description of each label is provided in Table 1. These 6 types are the most representative and can cover all the malicious behaviors that occur in our dataset. The detailed introduction of their mechanisms and potential threats are shown as follows:

- **SMS-related:** This type of behavior can cause various cyber-attacks conducted via SMS service, such as privacy leakage and service charge. The first case type can silently steal the victim's privacy, such as personal information, digital identities to properties, and contents of the user's private conversations, by sending them from the infected device to a remote server via SMS service. The second case type can cause unaware service charges by bypassing a two-factor authentication (2FA), such as silently confirming a subscription payment by reading the verification code in messages at runtime. The third case type can perform SMS-based malicious behaviors with remote control commands received from a remote server via SMS service. The fourth case type can delete the messages in the user's SMS box.
- **Internet-related:** This type of behavior can send the user's privacy from a local device to the attacker's remote server by requesting network connections. In our identified cases, the attack actions are performed

```
//Relevant permissions extracted from AndroidManifest.XML
android.permission.READ_PHONE_STATE
android.permission.RECEIVE_SMS
android.permission.READ_SMS
android.permission.WRITE_SMS
android.permission.SEND_SMS
android.permission.CALL_PHONE
android.permission.INTERNET
android.permission.READ_CONTACTS

//Relevant implementation extracted from source code
//Behavior of retrieving user's SMS
private void getSMS(){
    Cruse v10=this.getContentResolver().qury(Uri.parse(''Content
     ://sms/inbox'')),v2,(String)v2),v2,(String)v2));
    ...
}
//Behavior of uploading user's SMS to a remote server
private void run(){
    ...
    Log.i(MainActivity.this.TAG, MainActivity.this.httpPOST(
     CommonMoudle.Url7,((List)v0)));
    ...
}
//Behavior of monitoring user's outgoing calls
public void onReceive(Context arg7,Intent arg8){
    ...
    if("Android.intent.action.NewOutgoingCall".equal(arg8.
     getAction())){
        ...
    }
    ...
}
//Behavior of making phone call
private void quickCall(){
    ...
    v2.setAction("android.intent.action.CALL");
    ...
}
//Behavior of setting up call forwarding
private void transferredCall(){
    ...
    v4.invoke(this.telephonyManager,null).endCall();
    ...
    v2.setAction("Android.intent.action.Call");
    ...
}
```

Listing 1: A sample: multiple behaviors in one malware

according to the control instructions received from the server.

- **Telephony-related:** This type of behavior can perform telephone call-related attacks which can cause privacy leakage or economic losses. The first case type can steal the victim's telephone call-related information, such as incoming & outgoing call history and saved contact information. The second case type can monitor the incoming & outgoing calls at runtime. A highly representative case is that some ransomware only allows victims to answer the designated call on the infected device.
- **Lock-in:** This type of behavior can extort money by blocking the victim's control behaviors. The first case type can bind its own layout on the top layer of the UI (a.k.a., user interface) after installation unless the victim pays the ransom according to their instructions. The second case type can modify the lock screen password without the user's authorization. Same as the first case, it also asks the user to transfer money to a designated e-money wallet. Once the payment is confirmed, the unlock password will be sent to the victim. Besides, some variants in this type can change the password and lock the phone repeatedly, even though the user has paid the ransom many times.

TABLE 1: An overview of ground-truth dataset

Label	Description	# of samples
SMS-related	This type of behavior can cause various cyber attacks conducted via SMS service, such as privacy leakage and service charge.	113
Internet-related	This type of behavior can send the user's privacy from a local device to the attacker's remote server by requesting network connections.	50
Telephony-related	This type of behavior can perform telephone call-related attacks which can cause privacy leakage or economic losses.	10
Lock-in	This type of behavior can extort money by blocking the victim's control behaviors.	73
Ads	This type of behavior usually distributes annoying advertisements, such as pornographic and gambling websites, to victims and possibly causes economic losses.	13
Re-infection	This type of behavior can prevent itself from being uninstalled so that it can keep performing attack actions on the infected device.	67

- **Ads:** This type of behavior usually distributes annoying advertisements, such as pornographic and gambling websites, to victims and possibly causes economic losses. The first case type displays advertisements at the top layer of the UI by frequently generating pop-up windows. Once the victim misclicks them, it will automatically switch to another app or website. The second case type can send the download URL to all saved contacts via group SMS service.
- **Re-infection:** This type of behavior can prevent itself from being uninstalled so that it can keep performing attack actions on the infected device. In our investigated cases, it always exists in conjunction with other types of malicious behaviors. When installed by the victims, the malware can induce them to activate the device management authority and hide the desktop icon, so that the program cannot be uninstalled normally. At the same time, some variants will also monitor operations like mobile phone startup, screen brightening, text messages, phone calls, etc., and keep running in the background after self-starting, so as to achieve the purpose of persistence.

2.1.3 *Feature selection*

In order to extract the features which can represent malicious behaviors in malware properly, we build a feature dictionary. We select 3 types of widely-used features [22], [35], which are API, used permission, and intent. API is most important in our approach since it is the core component in malicious behaviors' demonstration and a procedure call interface to operating system resources. Used permission is the relevant access right that needs to be applied before performing malicious behaviors. Intent provides a mechanism to assist in the interaction and communication between activities, acting as an intermediary.

With the knowledge of the 6 defined types of malicious behaviors, we first summarize the key program-level

```
//Relevant permissions extracted from AndroidManifest.XML
android.permission.READ_SMS
android.permission.WRITE_SMS
android.permission.SEND_SMS

//Relevant implementation extracted from source code
public void onReceive(Context arg0, Intent arg11)
{
    ...
    else if(arg11.getAction().equals("
     android.intent.action.SMS_RECEIVED"){
        ...
        StringBuffer v1=new StringBuffer(v8[v7].
     getOriginatingAddress());
        if(v1.toString().contains("10658166")){
            this.abortBroadcast();
            SmsManager.getDefault().sendTextMessage(v1.
     toString(),v2,"Y",(PendingIntent)v2),(PendingIntent)v2)
     );
        ...
        }
    }
}
```

Listing 2: A sample: an SMS-related malicious behavior

components of each type from the security reports. Since the analysis-oriented and discriminating basis of relevant features is highly associated with the knowledge of malicious behaviors and defined behavior labels, it is essential to validate the concrete implementation step by step, in case any mistake or missing information which may cause fatal damage to the usability and effectiveness of our approach. Thus, we further perform a validation based on static analysis. After decompiling the samples back into source code, we first locate the potential candidates of the malicious behaviors by searching the key APIs or the statement of permission requests and intent actions. To further ensure the completeness and correctness of our summarized key features, we then validate the candidates by analyzing the program flow of control at the level of API call. After this process, the size of the feature dictionary is 401, including 109 APIs, 59 used permissions, and 23 intents.

Listing. 2 demonstrates a malicious code snippet about the malicious behavior that sending confirmation SMS without authorization in the malware named "passion movie". It first obtains the permissions related to SMS, and then keeps monitoring the SMS inbox by continuously checking the actions performed by the victim's device. If the current captured action equals to *android.intent.action.SMS_RECEIVED*, the malware is aware that there is an SMS arrived at the inbox. Once received any message, it captures sender's number by calling the function *getOriginatingAddress()*. If the number equals to the designated number "10658166", it will call the function *abortBroadcast()* to block the incoming SMS notification, and then replies a "Y" as a confirmation message silently. Through these above actions, the service subscription will be confirmed without the user's knowledge. By analyzing the behaviors along the malware's attack chain, we can successfully summarize and supplement the relevant permissions, intents, and APIs to our feature dictionary.

While analyzing detailed cases, we find there exist diverse implementations for a same type of malicious behaviors, which result in different key features. For example, the malicious behavior of sending SMS may be achieved by

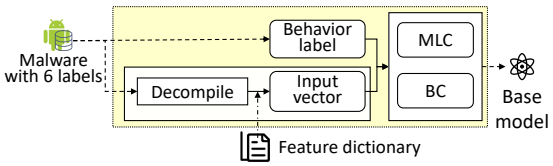


Fig. 3: Base MLC model construction with labeled dataset

calling *SendSMS()*, *sendTextMessage()*, *sendMultipartTextMessage()*, or others. However, the limited number of malware with reports seriously weakens the completeness of our summary on the characteristics of malicious behaviors. To handle this problem, we supplement our feature dictionary by referring to some relevant research papers and technical documents [16], [20], [22], [36], [37]. For research papers, we locate the relevant behaviors corresponding to the 6 defined labels, and supplement those missing features to our dictionary. For technical documents, we check the descriptions of relevant functionalities to seek potential similar implementation. Once we find any, we will then analyze the relevant features, and add them to the feature dictionary as a supplementation. During our study, we find some API differences caused by different Android versions and make some updates to the APIs in the feature dictionary. By complementing the feature set with research papers and technical documents, the size of the feature dictionary increases by 130, among which, APIs increase by 80, permission increases by 20, and intent increases by 30. Finally, we obtain a relatively comprehensive feature dictionary of 531 features, which includes 189 APIs, 79 used permissions, and 263 intents (more details [38]).

2.1.4 Data annotation

With the best understanding of the malicious behaviors, we then label the malware samples according to the analysis results. Each sample has a six-dimensional vector representing the 6 labels. Each dimension of the vector corresponds to a type of malicious behavior which is defined in § 2.1.2. If the corresponding behavior type to the dimension exists in the malware sample, the dimension will be marked as "1", otherwise, it will remain at "0". Table. 1 shows the number of malware samples in each label category. In this work, this well-labeled dataset will serve as the ground truth afterward.

2.2 Base MLC Model Construction

The most well-known two categories of methods in MLC are algorithm adaptation (AA) and problem transformation (PT) [32], [39]. AA method uses a variety of algorithms to convert a single-label learning model into a multi-label learning model. PT methods transform a multi-label learning problem into multiple single-label learning tasks. In order to obtain the best base practice, we totally adopt 70 different combinations of MLC and basic classification algorithms to construct ML-based MLC models. In our work, we adopt 10 different multi-label classification algorithms in total. Table 2 shows the acronym of each algorithm and the type of its belonging method. Totally, there are one algorithm belonging to the AA method and 9 algorithms

TABLE 2: MLC algorithms compared in our work

Algorithm Name	Acronym	Type
Binary Relevance	BR	PT
Classifier Chain	CC	PT
Random k-Label Disjoint Pruned Sets	RAkELd	PT
Pruned Sets	PS	PT
Pruned Sets with Threshold	PSt	PT
Multi-Label Back Propagation Neural Network	ML-BPNN	AA
Ranking and Threshold	RT	PT
Conditional Dependency Networks	CDN	PT
Conditional Dependency Trellis	CDT	PT
Classifier Trellis	CT	PT

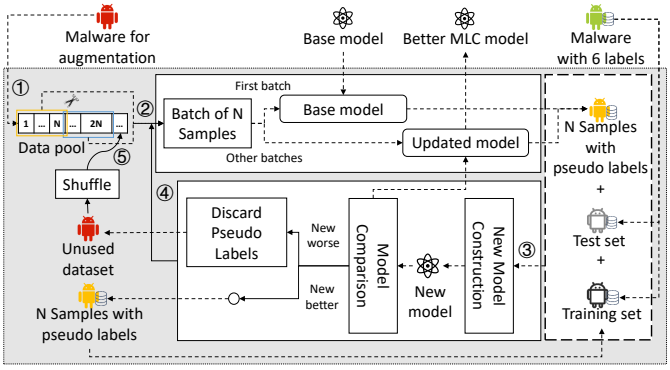


Fig. 4: Detection-Training: an active learning framework based on data augmentation

belonging to the PT method. To the best of our knowledge, these MLC algorithms have never been adopted on any task in Android malware classification. Besides, we pick 7 different kinds of basic classification algorithms, which are J48, LMT, RandomForest, OneR, PART, RandomTree, and REPTree, and combine them with those MLC algorithms individually.

The brief workflow is shown in Fig. 3. We first extract feature vectors by decompiling and analyzing the apps in our labeled dataset, according to the feature dictionary constructed in the previous phase. Together with the predefined labels, the input vectors are then fed into each preimplemented combination of MLC and basic classification (a.k.a, BC) algorithms to train the candidates for the base model.

2.3 Detection-Training

For applications in the data science field, a large-scale high-quality dataset is always the most fundamental basis to ensure completing the established tasks at a promising quality. However, collecting and organizing a new labeled dataset is time-consuming and labor-intensive. In the cybersecurity domain, it can be even harder and more expensive. For the solutions in other domains, such as computer vision, when we are dealing with the task of labeling pictures, we can distribute it to anyone non-professional but willing to participate and then validate the result with a simple random sampling. Back to our task of labeling malware, the first challenge is that very fewer people have the qualification to finish the task at a reasonable confidence level. Secondly, even if we have the support of well-trained security analysts, collecting and labeling thousands of malware samples according to their behaviors may take too much time. Hence, despite of there exist some public malware datasets, expanding the size of the labeled dataset is still a critical challenge that we can barely withstand its cost. So that we design and propose an alternative solution, namely *Detection-Training*, inspired by the research work of both active learning [40] and data augmentations [41], [42] in other domains.

Fig 4 shows the workflow of Detection-Training, which is an active learning framework based on data augmentation. The framework takes the labeled dataset from the first phase described in § 2.1, the base model from the second phase described in § 2.2, and an unlabeled malware dataset as inputs. The feature extraction step with our constructed dictionary is considered as default and omitted in Fig. 4. The output is an MLC model which outperforms the base model. The detailed procedures are illustrated as follows:

- In step ①, the malware dataset without our predefined labels, namely "Malware for data augmentation" is divided into several batches in size N.
- In the next step ②, the current first batch is picked and removed from the data pool. Then, the current best MLC model, which is the base model for the first batch of data and updated model for other batches, takes the picked batch as input and generates a classification result for each malware sample in this batch. In this way, these results are temporarily determined as basic facts and further paired with the N samples as pseudo labels.
- Next, we combine the batch of data with the current training set to train a new model in step ③. The test set is remaining unchanged for the purpose of evaluating the quality of this new model by comparing it to the current best model. When determining the better model, we mainly compare the classification accuracy of the model. In the first epoch, we update the current model only if the new model has a higher accuracy for improving the accuracy as much as possible. In the next epochs, the models with the same accuracy as the current model are also considered in order to obtain a large data augmentation. The specific idea behind adopting different methods for the first epoch and others is that improving the capability of classification is always the first-order target when there is any potential for accuracy improvement. Besides, an MLC model with higher accuracy can also classify the batches with higher confidence. Through the method, the better new model is used as the updated model to label the next batch of N samples. The associated malware samples are added to the training set permanently. Otherwise, the new model and pseudo labels will be discarded. The discarded batch of N samples will be marked as unused data.
- In step ④, the workflow loops along step ② and step ③ until all the malware samples in the data pool have been picked. Because the classification capability of the updated model is increased after many cycles, the updated model will also have a higher possibility to conduct more precise classifications on those unused data and further enhance the detection capability as well.
- Besides, in order to avoid the potential influence from the fixed order in the data pool, the unused dataset is shuffled before returning to the end of the data pool in step ⑤ and

fed to the updated model in step ② to start next epoch until all the samples are labeled or the accuracy and scale of data augmentation are no longer improved.

Through all the procedures in this framework, both the classification capability of our MLC model and the size of the usable dataset can be significantly improved.

3 EVALUATION

In this section, we first introduce the used datasets, the experiment environment, and the metrics selected for evaluating the classification capability of the MLC model. After that, We evaluate the effectiveness of different MLC and BC algorithm combinations. Later, we investigate the effectiveness of Detection-Training. At last, we perform an experiment for validating the authenticity of the pseudos labels collected through Detection-Training.

3.1 Used Datasets

3.1.1 Manually labeled malware

This labeled dataset is constructed on 180 malware samples with manual analysis reports, which are obtained from an anonymousCERT. With our effort, each sample is labeled according to the summarized 6 types of malicious behaviors. A more detailed description is shown in Table 1. We release these labeled data as a benchmark [33].

3.1.2 DREBIN dataset

We choose the classic Android malware dataset DREBIN [13] as the first unlabeled dataset for our experiment of data augmentation. This dataset was collected in the period of August 2010 to October 2012, and it contains 5,560 applications from 179 different malware families. However, during decompiling, these malware and extracting feature vectors, some malware samples failed in prepossessing. The number of usable samples in our experiments is 5,456 in total.

3.1.3 VirusShare dataset

We also collected 5,500 Android malware samples from VirusShare [43] as the second unlabeled dataset. These samples have been collected in the period of 2018 to 2022. The process of building the VirusShare dataset is shown in § 3.5.1.2.

3.2 Experimental Environment

All experiments are conducted on the Ubuntu 18.04 server with 36 Intel Xeon E5-2699 v3 CPUs and 192GB RAM.

The based language of most implementations in our work is Python 3.6. MEKA v1.9.2 [44], [45], which is designed to provide a series of algorithms and evaluation indicators to solve multi-label classification problems, is chosen as the basic toolkit in training and evaluating the models. It is an open-source project based on the WEKA [46].

TABLE 3: A summary of basic notations

Notation	Definition	Value/Denotation
L	The predefined labels	$\|L\| = 6$
D	The set of test data	N/A
x_i	The ith sample in test dataset	$x_i \in D$
$Y_{i,j}$	The jth value in the actual label vector of sample x_i	$Y_{i,j} \in \{0, 1\}$
Y_i	The actual label vector of sample x_i	$(Y_{i,1}, \cdots, Y_{i,\|L\|})$
$Z_{i,j}$	The jth result in the classification result vector of sample x_i	$Z_{i,j} \in \{0, 1\}$
Z_i	The classification result vector of sample x_i	$(Z_{i,1}, \cdots, Z_{i,\|L\|})$

TABLE 4: The conditions for calculating confusion matrix

Label \ Pred.	Pos.	Neg.
Pos.	$(Y_{i,j} = 1) \wedge (Z_{i,j} = 1)$	$(Y_{i,j} = 1) \wedge (Z_{i,j} = 0)$
Neg.	$(Y_{i,j} = 0) \wedge (Z_{i,j} = 1)$	$(Y_{i,j} = 0) \wedge (Z_{i,j} = 0)$

3.3 Evaluation Metrics

Generally, the evaluation of the multi-label classification approach is much more complicated than binary or family classification [47], since there are multiple classes associated with each sample. Besides, for the active learning method, the effectiveness of the method is highly dependent on the correctness of feedback (i.e., generated pseudo labels). Therefore, we adopt 2 categories of metrics for different purposes, which are 4 sample-based metrics for the overall effectiveness of the trained model and a label-based metric for the effectiveness on different labels [48]. Overall, we take the sample-ACC as the first-order metric in all experiments, and the rest are used as auxiliary ones for determining the capability of models towards various factors.

To help in understanding the mathematical definitions of the metrics, we first introduce the definitions of all used notations with their values or denotations if exist as shown in Table 3. In order to make it easier to be understood, we explain each notation in more detail and its interrelationship with other notations in the following paragraph.

L denotes the 6 predefined labels, and $|L|$ represents the total number of labels (i.e., 6). D is the test data set used to evaluate the pre-trained models, and $|D|$ represents its total number of samples. x_i represents the ith sample in D. Y_i is an L-dimension boolean vector $(Y_{i,1}, \cdots, Y_{i,j}, \cdots, Y_{i,|L|})$ which represents the $|L|$ labels of sample x_i in vector space. $Y_{i,j}$ could be either 1 or 0, which means the sample is actually relevant or irrelevant to the malicious behavior associated with label j. Z_i is also an L-dimension boolean vector $(Z_{i,1}, \cdots, Z_{i,j}, \cdots, Z_{i,|L|})$ which represents the classification result of sample x_i. $Z_{i,j}$ could be either 1 or 0, which means the sample is classified into label j by the pre-trained model, or not. Besides, in Table 4, we also present the basic logic conditions while calculating confusion matrix for the sample x_i on label j, according to the actual label $Y_{i,j}$ and classification result $Z_{i,j}$.

3.3.1 Sample-based metrics

In all, we adopt 4 sample-based evaluation metrics, they are hamming loss, zero-one loss, F1-score, and sample-ACC.

Hamming loss represents the fraction of the misclassified labels to the total label number among all samples. The

smaller the value of hamming loss is, the stronger the classification ability the MLC model has. It can be calculated with the following equation:

$$\text{hammingLoss} = \frac{1}{|D| \cdot |L|} \sum_{i=1}^{|D|} \sum_{j=1}^{|L|} Y_{i,j} \oplus Z_{i,j};$$

Where $\oplus$ stands for the XOR operation in Boolean logic.

Zero-one loss is a metric that evaluates the fraction of misclassified samples to the entire dataset. For a given input x_i, this metric considers the classifier makes a correct classification, only if all labels in the classification results Z_i equal to the ground truth Y_i. Otherwise, if any single label is different, the classification is considered as a failure. The smaller the value of zero-one loss, the better the performance. It can be calculated with the following equations:

$$\begin{cases} loss_i = \begin{cases} 1, Y_i \neq Z_i, \\ 0, Y_i = Z_i, \end{cases} \\ \text{zero-one loss} = \dfrac{1}{|D|} \displaystyle\sum_{i=1}^{|D|} loss_i; \end{cases}$$

Where $loss_i$ denotes the classification result of the sample x_i. It could be either 1 or 0, which means the sample is correctly predicted or incorrectly predicted.

F1-score measures the accuracy and completeness of the classification. The F1-score adopted in evaluating the proposed multi-label classification approach is calculated by the F-measure averaging on each sample. A higher F1-score means better performance in classification. It can be calculated with the following equation:

$$\text{F1-score} = \frac{1}{|D|} \sum_{i=0}^{|D|} \frac{2 \sum_{j=0}^{|L|} Y_{i,j} \wedge Z_{i,j}}{\sum_{j=0}^{|L|} Y_{i,j} + \sum_{j=0}^{|L|} Z_{i,j}};$$

Sample-ACC is adopted to evaluate the overall accuracy of the pre-trained model on the test dataset. The idea that we are trying to explore is providing as much explanation as possible in order to provide some usable and correct clues for future analysis. However, different from the method used in binary or family classification tasks, ensuring to correctly classify all $|L|$ labels of the given sample could be extremely challenging in an MLC task. Especially, the limited size of the labeled dataset will definitely make this target become a "mission impossible". Hence, we use an alternate definition of the correctness of the classification result. We consider the sample x_i to be correctly classified, as long as its classification result satisfies the following two conditions:

- Detect at least one malicious behavior (true positive: $(Y_{i,j} = 1) \wedge (Z_{i,j} = 1)$) correctly.
- Predict nonexistent malicious behaviors (false positive: $(Y_{i,j} = 0) \wedge (Z_{i,j} = 1)$) to none.

Otherwise, the sample is misclassified. In all, the goal is to predict the malicious behavior of the sample as much as possible, meanwhile, avoiding misleading with incorrect information. The mathematical definition of the sample-ACC is shown as follows:

$$\begin{cases} C_i = \{Z_{i,j} | \neg((Y_{i,j} = 0) \wedge (Z_{i,j} = 1)), j \in N \cap (0, |L|]\}, \\ C = \{x_i | \forall x_i \in D, C_i \ni 1, i \in N \cap (0, |D|]\}, \\ \text{sample-ACC} = \dfrac{|C|}{|D|}; \end{cases}$$

Where C_i denotes the set of classification results of the sample x_i after removing false positives; and C denotes the set of correctly classified samples, while each sample contains at least one successfully detected malicious behavior. Sample-ACC represents the proportion of correct samples in the entire test dataset.

3.3.2 Label-based metric

To evaluate the detection ability of the MLC model on each label, we also select a label-based metric, namely label-ACC.

Label-ACC aims to evaluate the accuracy of the pre-trained model on the selected labels. In other words, the larger the label-ACC, the better the model performs on the label. The idea behind this metric is that, because of the limited and imbalanced labeled data, the overall accuracy calculated on all samples may hide the poor effectiveness of the classifier on those labels that contain fewer samples. The mathematical definition of the label-ACC is shown as follows:

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

Where C_j denotes the set of correctly classified samples on label j. Here, the basis of judgment includes both successfully detected samples containing malicious behavior and successfully detected samples that do not contain malicious behavior. D_j denotes the set of samples on label j.

3.4 RQ1: Which combination of MLC and BC algorithms is best for multi-label classification of malware?

In this experiment, to find out the best combinations of multi-label classification and basic classification algorithms, we evaluate the effectiveness of diverse machine learning models on the metrics proposed in § 3.3.

3.4.1 Dataset

As shown in Table 1, we use the dataset of 180 labeled malware samples as the ground truth. Due to the limited size of the dataset, using a common data split ratio (i.e., 8:2) like other approaches which perform on large-scale datasets can seriously undermine the validity of the evaluation result. Especially, for the labels with a very small number of samples, such as "Telephony-related", if the test set has only 2 samples, it will be quite hard to judge the actual classification ability since there are only 3 possible test results (i.e., 0%, 50% or 100%). Besides, because of the imbalanced data distribution, samples under this label may not exist in the test set, if we adopt a global random data partitioning as usual. Hence, when dividing the dataset, we ensure that the proportions of samples under each label in

TABLE 5: The results of top 10 models according to the sample-based metrics

Model id	1	2	3	4	5	6	7	8	9	10
MLC algorithm	CDN	CDN	BR	CDN	RT	PS	RAkELd	CC	CC	CT
basic classifier	J48	LMT	Random Forest	REPTree	LMT	Random Forest	Random Forest	Random Forest	REPTree	Random Forest
hamming loss	0.133	0.128	0.139	0.133	0.161	**0.122**	0.133	0.156	0.133	0.156
zero-one loss	0.533	0.467	0.533	0.500	0.533	**0.433**	0.500	0.533	0.533	0.533
F1-score	0.696	0.723	0.702	0.704	0.707	**0.741**	0.719	0.702	0.693	0.702
sample-ACC	**0.733**	**0.733**	0.700	0.700	0.700	0.700	0.700	0.667	0.667	0.667

*To make the table more readable, we highlight the locally optimal results (for sample-based metrics) with bold font and the key determinant(s) of the overall best model with gray color. (also for Table 9)

TABLE 6: The label-based evaluation results of top 10 models

Model id	1	2	3	4	5	6	7	8	9	10
MLC algorithm	CDN	CDN	BR	CDN	RT	PS	RAkELd	CC	CC	CT
basic classifier	J48	LMT	Random Forest	REPTree	LMT	Random Forest	Random Forest	Random Forest	REPTree	Random Forest
L1	0.833	**0.867**	0.833	0.800	0.833	0.833	0.833	0.767	0.800	0.767
L2	0.800	0.800	**0.833**	**0.833**	0.800	0.800	0.800	0.800	**0.833**	0.800
L3	**0.933**	**0.933**	0.867	**0.933**	0.867	**0.933**	**0.933**	0.867	**0.933**	0.867
L4	0.933	0.900	0.933	0.933	0.867	**0.967**	**0.967**	0.933	0.933	0.933
L5	**0.900**	**0.900**	**0.900**	**0.900**	**0.900**	**0.900**	**0.900**	**0.900**	**0.900**	**0.900**
L6	0.800	**0.833**	0.800	0.800	0.767	**0.833**	0.767	0.800	0.800	0.800
L-avg.	0.867	**0.872**	0.861	0.867	0.839	0.867	0.867	0.845	0.867	0.845

*L1-6 refers to the label-ACC of 6 predefined labels (i.e., "SMS-related", "lock in", "Re-infection", "Telephony-related", "Ads", "Internet-related") and L-avg. is the average of all L1-L6.
*To make the table more readable, we highlight the locally optimal results (for the label-ACC of each label) with bold font and the key determinant (L-avg.) of the overall best model with gray color.

the training set and test set are the same. We split the 180 labeled dataset into a training set of 150 samples and a test set of 30 samples.

3.4.2 Experiment setup

To evaluate the effectiveness of different multi-label classification (MLC) and basic classification (BC) algorithms, we adopt 70 different combinations of MLC and BC algorithms and train each with the same data configuration. By comparing the results by calculating the selected sample-based and label-based evaluation metrics, we then determine the most suitable combinations for our task.

3.4.3 Results

We demonstrate the top 10 results from 2 aspects referring to sample-based and label-based metrics, respectively. The full results could be found on the website [38].

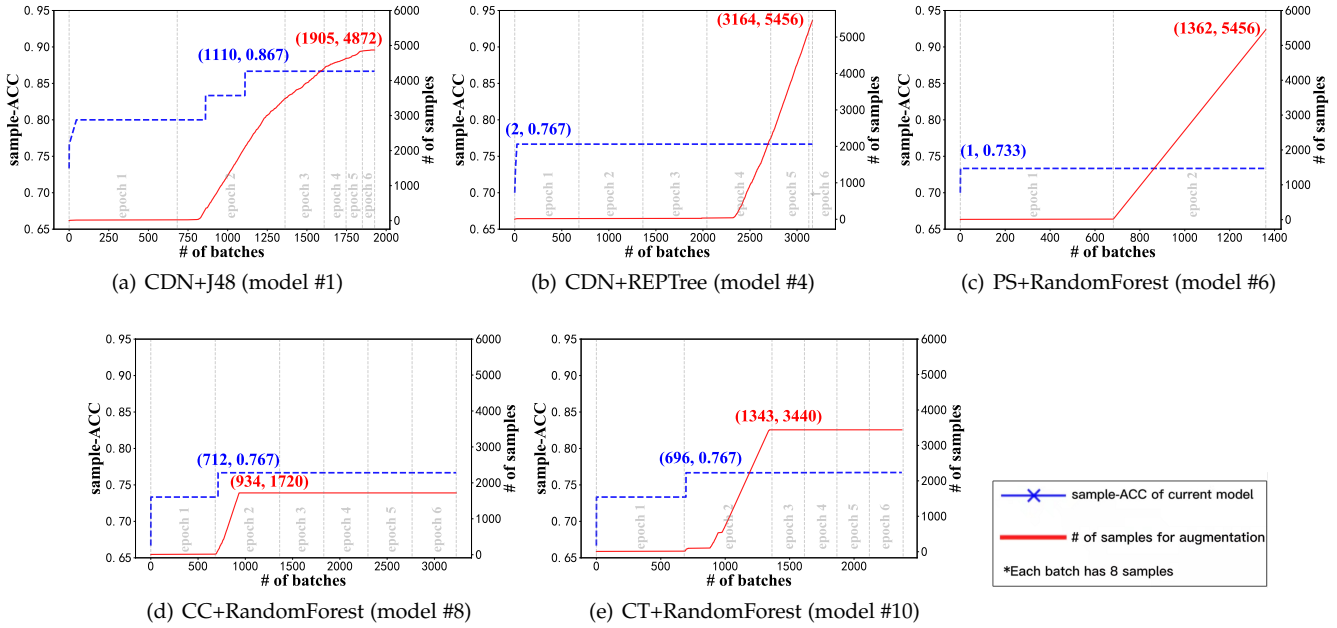
3.4.3.1 **Sample-based evaluation**: As shown in Table 5, the sample-based results of the top 10 models are sorted by the sample-ACC in descending order. Overall, model #1 and #2 outperform other candidates at 0.733 on the sample-ACC. Between them, model #2 using CDN as MLC algorithm and LMT as BC algorithm achieves a better result in terms of higher sample-ACC and F1-score, which are 0.733 and 0.723, and lower hamming loss and zero-one loss, which are 0.128 and 0.467, respectively. This evaluation result reveals that the above combination is able to predict as many malicious behaviors as possible and is less likely to make false classifications. Besides the best 2 combinations, model #3-7 have the second highest sample-ACC at 0.7, model #8-10 have the third highest sample-ACC at 0.667. Due to the limitation of article length and weaker importance of results, the evaluation results of the rest 60 combinations are released on the website [38] We can also notice that model #6, which uses PS as MLC algorithm and RandomForest as the BC algorithm, outperforms among all models in terms of hamming loss, zero-one loss, and F1-score at 0.122, 0.433, and 0.741, respectively. That reveals the prediction results given by this combination are relatively more comprehensive, and it will make fewer false classifications. Through this evaluation, CDN obviously shows their stronger effectiveness in our task since 3 of the top 10 models are using CDN as their MLC algorithm.

3.4.3.2 **Label-based evaluation**: To further compare the effectiveness of different algorithm combinations towards each specific malicious behavior, we further evaluate the accuracy of the top 10 models on each predefined label. As shown in Table 6, we use L1-6 to denote the label-ACC of 6 types of malicious behaviors, which are "SMS-related", "Internet-related", "Telephony-related", "Lock-in", "Ads" and "Re-infection" in order, and use L-avg. to denote the averaged label-ACC (a.k.a., label-based accuracy). From the results, we can see that all MLC models have the same L5 (a.k.a., the label-ACC of the 5th malicious behavior, namely "Ads") at 0.9. We notice that model #2, which has the best result in the sample-based evaluation, also outperforms others in the label-based evaluation. The L-avg. of model #2 is 0.872. In detail, it reaches the best results on L1,3,5,6 at 0.867, 0.933, 0.9, and 0.833, respectively. These results reveal that the combination of algorithms CDN and LMT has higher confidence in these 4 labels in our task. Besides, sorted by L-avg. in descending order, the rest are model #1,4,6,7,9, #3, #8,10 and #5 at 0.867, 0.861, 0.845, and 0.839. Among them, model #3,4 achieve the highest L2 at 0.833, and model #6,7 achieve the highest L4 at 0.967.

TABLE 7: The results of Detection-Training with top 10 algorithm combinations on the DREBIN dataset

model id	1	2	3	4	5	6	7	8	9	10
MLC algorithm	CDN	CDN	BR	CDN	RT	PS	RAkELd	CC	CC	CT
basic clsssifier	J48	LMT	Random Forest	REPTree	LMT	Random Forest	Random Forest	Random Forest	REPTree	Random Forest
# of samples	**4,872**	312	16	**5,456**	472	**5,456**	16	**1,720**	72	**3,440**
hamming loss	**0.106**	0.122	0.111	0.144	0.172	0.139	0.122	0.128	0.128	0.122
zero-one loss	**0.467**	0.500	0.433	0.500	0.533	0.533	0.467	**0.467**	0.500	**0.467**
F1-score	**0.779**	0.742	0.758	0.729	0.696	0.730	0.748	0.752	0.734	0.769
sample-ACC	**0.867**	0.833	0.767	0.767	0.800	0.733	0.767	0.767	0.800	0.767
Δsample-ACC	**0.134**	0.100	0.067	0.067	0.100	0.033	0.067	0.100	0.133	0.100
Avg. label-ACC	**0.894**	0.878	0.889	0.856	0.828	0.861	0.878	0.872	0.872	0.878

*To make the table more readable, we highlight the locally optimal results (only if # of samples > 1k, the model is considered; best one(s) for other metrics) with bold font and the key determinants of the overall best models with gray color. (also for Table 8, 10 and 11)



(a) CDN+J48 (model #1) (b) CDN+REPTree (model #4) (c) PS+RandomForest (model #6)

(d) CC+RandomForest (model #8) (e) CT+RandomForest (model #10)

Fig. 5: The plots of accuracy improvement and data augmentation on the DREBIN dataset

Conclusion: *By training with 70 different algorithm combinations, we basically confirm the candidate MLC and BC algorithms that could achieve better results for our task. The best practice reaches 0.733 and 0.872 on sample-ACC and averaged label-ACC.*

3.5 RQ2: Can Detection-Training effectively augment the dataset? What is the best batch-size for enhancement?

In this section, we investigate the effectiveness of Detection-Training from 2 aspects. First, we conduct an experiment to see which ML algorithm can perform well with Detection-Training. Second, we further tune the parameters defined in Detection-Training to achieve a better result on accuracy improvement and data augmentation.

3.5.1 Capability Evaluation of Detection-Training on Accuracy Improvement and Data Augmentation

To find out the most suitable algorithm combinations in our task, we conduct the experiment that trains 10 combinations on 2 different malware datasets. We evaluate the capability according to 2 factors, such as the # of samples for data augmentation and the accuracy improvement of the MLC model.

3.5.1.1 **Evaluation on the DREBIN datset**: **Dataset.** In this experiment, we adopt a setting on the labeled dataset as same as § 3.4.1, because it is necessary to keep the test set remaining unchanged while comparing the accuracy between the base model and better models, which obtained through Detection-Training. Besides, two more datasets without manual labeling are used as the "Malware for data augmentation" in the Detection-Training module which is introduced in § 2.3. The first one is a widely used public dataset, namely DREBIN. As introduced in § 3.1.2, the malware samples in DREBIN were collected in the period of 2010 to 2012. During the past decade, the attack principle of malware also changes with the improvement of the Android system mechanism and implementation. Hence, the evaluation results on DREBIN may be affected by the evolution of the malicious behaviors implemented in malware.

Experiment setup. To evaluate the capability of the pro-

posed active learning framework on the accuracy improvement and data augmentation by using the results of base models in § 3.4 as the baseline, we adopt 10 algorithm combinations as same as those of the top 10 best base models. In the experiment, we follow the steps described in § 2.3. We set 8 as the default batch size, which means 8 samples from the "data pool" are fed into the "base/updated model" in each round of workflow, and 6 as the default epoch number, which means traversing "data pool" for 6 times by adding the shuffled the unused data in step ⑤ at the end of each epoch.

Result on the DREBIN dataset. Table 7 presents the evaluation results of improving the classification accuracy and size of the usable dataset by applying Detection-Training to the DREBIN dataset. Overall, the accuracy increase (i.e., Δsample-ACC) can reach up to 0.134 on the test set by comparing to the base models, even the worst case has an increase at 0.033. Among the 10 selected algorithm combinations, model #1, which uses CDN as MLC algorithm and J48 as BC algorithm, achieves the best result, which shows a significant improvement on all evaluation metrics with a successful data augmentation. Specifically, model #1 reaches the lowest hamming and zero-one loss at 0.106 and 0.467, the highest F1-score, sample-ACC, and averaged label-ACC at 0.779, 0.867, and 0.894, respectively, and the most successful accuracy improvement at 0.134 based on a data augmentation with 4,872 malware samples. Besides, we notice that there are 4 more combinations of MLC and BC algorithms, which are used in constructing model #4, 6, 8, 10, and can also expand the dataset with thousands of additional malware samples from DREBIN. Among them, model #4 and #6 have increased the training set to the largest size with 5,456 samples. Model #4 improves the sample-ACC to 0.733, which is 0.033 higher than that of the base model with the same algorithm combination. The averaged label-ACC of it is 0.856, which has a little decrease by comparing to the result of its base model (0.867). Model #6 also improves the sample-ACC to 0.733, which is 0.033 higher than that of the base model with the same algorithm combination.

To investigate the precise improvement in both accuracy and data augmentation during the entire active learning progress, we also plot the detailed results in Fig. 5. Generally, the plots present the run-time sample-ACC and the total number of involved samples by augmentation at each batch. We pick 6 as the epoch number in the experiment, because we observe the accuracy and number of samples always remaining unchanged after 6 epochs. Note that each epoch accumulates batches based on the previous epoch, the number of batches is not set to zero after one epoch ends, but it still keeps increasing. Totally, there are 5 models (i.e., model #1, 4, 6, 8, and 10) that obtained a data augmentation of more than one thousand samples. Fig. 5(a) shows the result of model #1, which uses CDN and J48 as MLC and BC algorithms. We can see that in the early stage of Detection-Training, along with the increasing # of batches, the sample-ACC and # of augmented samples keep increasing. After the # of batches reaches 1,110, the sample-ACC remains unchanged at 0.876. And, the # of samples for augmentation is still able to increase. In epoch 6, when 1,905 batches have been fed to the base/update

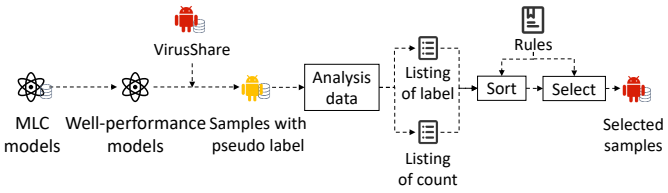


Fig. 6: The process of selecting samples from VirusShare dataset

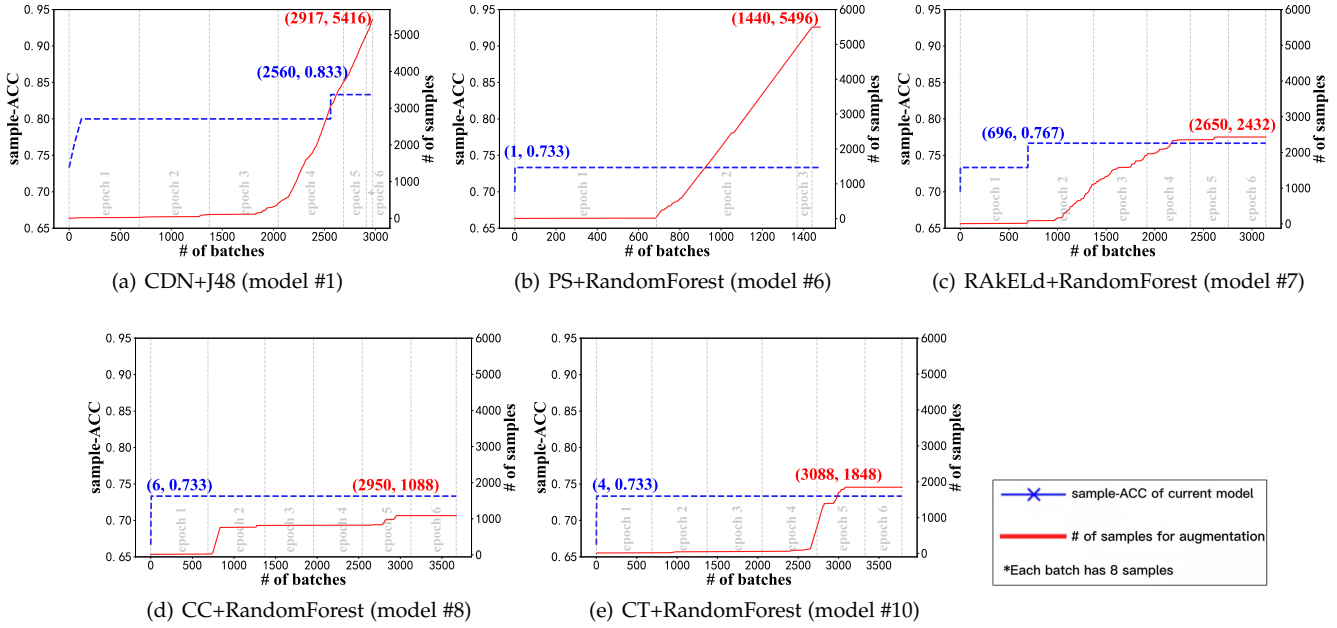
models, the # of samples reaches 4,872. After that, the # of augmented data hardly ever grows significantly. As shown in Fig. 5(d) and Fig. 5(e), model #8 and #10, which were trained with the combinations of CC and RandomForest and CT and RandomForest, also have a similar tendency. For these above 3 combinations, there is always such a situation observed, when the batches of involved samples reach a certain number, the classification capability of the model and # of samples for augmentation can no longer increase. For model #8 and #10, the lift of the curves stops in epoch 2. For model #4, when the # of batches reaches 2, the sample-ACC of the current model reaches the peak at 0.767. In epoch 4, # of samples for augmentation starts to increase significantly, and in epoch 6, when # of batches equals to 3,164, all 5,456 samples in the DREBIN dataset have been labeled with pseudo labels and added to the training set. A similar situation happens to model #6, which uses the combinations of PS and RandomForest. Fig 5(c) shows that in epoch 2, it also adds all 5,456 samples from the DREBIN dataset to the training set.

3.5.1.2 **Evaluation on the VirusShare dataset**:
Dataset. VirusShare has tens of thousands of Android malware between 2018 and 2022, the malicious behavior of these samples is varied. In order to balance the data on each label, we propose a method for selecting samples, which selects a subset of samples from VirusShare. The specific data selection process is as shown in Fig. 6.

First, we select 3 well-performing models from existing MLC models. Among them, one model comes from the basic models which are constructed in § 3.4. As Table 5 and Table 6 show, the algorithm combination of CDN and LMT (model #2) achieves the best results. The other two models are built in the process of detection-training using the DREBIN dataset in § 3.5.1.1, they are trained with the algorithm combinations of CDN and J48 (model #1), CDN and REPTree (model #4) shown in Table 7. We then use these three well-performing models to predict pseudo-labels for each of the 100,000 samples from VirusShare and get three pseudo-labeled sample datasets. Note that, for some same samples, the pseudo-labels predicted by the three models may be different. After that, we look into the statistics of each sample referring to these three pseudo-labeled sample datasets and mark each sample with two vectors with lengths at 6, namely *label* and *count*, respectively. For the vector *label*, the index i represents the number of models whose i-th pseudo-label of the sample is equal to 1. For the vector *count*, count[j] indicates the number of models that totally predict j labels for this sample. After getting the *label* and *count* vectors of all samples, we next sort them and pick samples based on the following 2 rules. Rule #1

TABLE 8: The results of Detection-Training with top 10 algorithm combinations on VirusShare dataset

model id	1	2	3	4	5	6	7	8	9	10
MLC algorithm	CDN	CDN	BR	CDN	RT	PS	RAkELd	CC	CC	CT
basic clsssifier	J48	LMT	Random Forest	REPTree	LMT	Random Forest	Random Forest	Random Forest	REPTree	Random Forest
# of samples	**5416**	536	72	120	960	**5496**	**2432**	**1088**	24	**1848**
hamming loss	0.133	0.111	0.117	0.139	0.189	0.139	0.144	**0.128**	0.128	**0.128**
zero-one loss	0.500	0.433	0.467	0.500	0.600	0.533	0.533	0.500	0.533	**0.467**
F1-score	0.733	0.753	0.747	0.729	0.641	0.730	0.720	**0.736**	0.746	0.719
sample-ACC	**0.833**	0.833	0.767	0.767	0.767	0.733	0.767	0.733	0.8	0.733
Δsample-ACC	**0.100**	0.100	0.067	0.067	0.067	0.033	0.067	0.066	0.133	0.066
Avg. label-ACC	0.867	0.889	0.883	0.861	0.811	0.861	0.856	**0.872**	0.872	**0.872**



(a) CDN+J48 (model #1) (b) PS+RandomForest (model #6) (c) RAkELd+RandomForest (model #7)

(d) CC+RandomForest (model #8) (e) CT+RandomForest (model #10)

Fig. 7: The plots of accuracy improvement and data augmentation on VirusShare dataset

is we want to have at least 1,000 samples on each label. Rule #2 is it would be better for the selected samples to have as many labels equal to 1 as possible because we want the selected samples to have diverse kinds of behaviors. We first sort them according to the value of label[i]. The larger the label[i] is, the more likely the sample actually has malicious behavior related to this label. For each label, we select the top 1,000 samples. Considering the value range of label[i] is [0, 3], all the samples with a high probability of an equal label[i]. When such a conflict occurs, we refer to Rule #2 and perform a secondary analysis on them based on the *count* vector. Specifically, we compare the maximum index of each sample whose count[j] is not equal to zero and select a sample with a larger index, which means that it may have more types of malicious behaviors. When the indexes of two samples are the same, we compare the value of their count[j] and select the one with a larger count[j], which indicates that it has higher credibility of having actual malicious behavior under label j. Through the above steps, we finally select 5,500 samples from VirusShare [43] and construct a relatively balanced VirusShare dataset.

Experiment setup. In this experiment, we adopt an experiment setup as same as § 3.5.1.1

Result on the VirusShare dataset. Table 8 presents the evaluation results of Detection-Training with top 10 algorithm combinations on VirusShare dataset. The top results are consistent with the markup of the Table 7. In general, the sample-ACC of these models comparerevised to those of base models can be improved $\geq$ 0.33 on the test set, and the best case has an increase at 0.1. There are totally 5 combinations of MLC and BC algorithms, which are used in constructing model #1, 6, 7, 8, 10, and can expand the train set with thousands of additional malware samples from the VirusShare dataset. Among all of them, model #1, which uses the algorithm combination of CDN and J48, achieves the best result. Its sample-ACC is 0.833, which is 0.1 more than that of its base model. Model #1 also enlarge the training set to the second largest size at 5,416. Besides, model #6 has increased the training set to the largest size with 5,496 samples and improved the sample-ACC to 0.733, which is 0.033 higher than that of its base model. The averaged label-ACC of it is 0.861, which has a little decrease by comparing to that of its base model (0.867).

We also plot the detailed results of model #1, 6, 7, 8, 10 in Fig. 7. The x-axis represents # of batches used in Detection-Training. Generally, the plots present the run-time sample-

ACC and the total number of involved samples by augmentation at each batch. As a default, we set the epoch number to 6. In Fig. 7(a) which shows the result of the combination of CDN and J48 (model #1), along with the increasing # of batches, the sample-ACC and # of augmented samples keep increasing at the early stage of Detection-Training. when the # of batches reaches 2,560, the sample-ACC reaches a peak at 0.833. And after that, the # of samples for augmentation is still able to increase. In epoch 6, when 2,917 batches have been fed into the base/update models, the # of samples reaches 5,416 and the # of augmented data hardly ever grows significantly after that. As shown in Fig. 7(c), Fig. 7(d) and Fig. 7(e), model #7, #8 and #10, which are trained with the combinations of RAkELd and RandomForest, CC and RandomForest, CT and RandomForest, also have a similar tendency as model #1. For these 3 models, there is always such a situation that both the classification capability of the model and # of samples for augmentation has no longer increased when the # of batches reaches a certain number. For model #7, #8, and #10, the lift of the curves stops in epoch 5. As shown in Fig. 7(b), for model #6, after the first batch is added, the sample-ACC remains unchanged at 0.733. In epoch 2, # of samples for augmentation starts to increase significantly. When the # of batches equals 1,440 in epoch 3, 5,496 samples from the VirusShare dataset have been labeled with pseudo labels and added to the training set. Because the rest of the 4 samples other than the samples used for data augmentation cannot be successfully added, we consider the process of Detection-Training for this algorithm combination stops in 3 epochs. In general, all these 5 processes of Detection-Training have proof of accuracy improvement and data augmentation, and can both be finished within 5 epochs, which means an acceptable efficiency.

3.5.1.3 **Comparison between the DREBIN dataset and VirusShare dataset**: Generally, the results on the two datasets reveal that the proposed active learning framework can not only improve the capability of MLC classifier but also construct a larger dataset with the help of the limited number of labeled malware samples. Besides the proof of accuracy improvement and data augmentation, we can observe that the processes of using the proposed Detection-Training on all five selected algorithm combinations can be finished within 6 epochs, which means an acceptable efficiency.

In detail, by comparing the results in Table. 7 and 8, 4 algorithm combinations, which are CDN and J48 (model #1), PS and RandomForest (model #6), CC and RandomForest (model #8), CT and RandomForest (model #10), perform good on both of the two selected datasets. From the aspect of improving accuracy, these 4 combinations achieve similar results with a difference of less than 0.034. From the aspect of data augmentation, the combinations of CDN and J48 (model #1), PS, and RandomForest (model #6) achieve similar results on 2 datasets with differences at 544 and 40 samples, which only occupy 0.112 and 0.007 of the total on the lesser side (on DREBIN). However, the sizes of data augmentation achieved by the other two combinations on two datasets show a big difference at 1,592 and 632, which occupy 0.861 and 0.581 of the total on the lesser side (on the VirusShare dataset). Besides, the combination of CDN and REPTree (model #4), RAkELd, and RandomForest (model #7) demonstrated significantly different effects on different datasets. These phenomena are quite interesting and worth future exploration. Through this work which is still in the early stages of this new direction, the biggest potentially possible reason is the variability of the datasets. As we introduced in previous sections (§ 1 and § 2.3), it is quite hard and expensive to manually validate the malicious behaviors in malware, even with a detailed security report. In § 5, we will discuss some engineering approaches, which can potentially simplify the analysis process and are expected to be validated in future work.

Conclusion 2.1: *By conducting the process of Detection-Training on the DREBIN dataset and VirusShare dataset, we confirm the effectiveness of our method. Specifically, for the DREBIN dataset, the best practice has enlarged the dataset to a size of 4,872, and the sample-ACC reaches 0.867. For the VirusShare dataset, # of samples of the best practice is 5,416 and its sample-ACC is 0.833.*

3.5.2 Evaluation on Parameters Tuning in Detection-Training

To find out the best parameter setting of Detection-Training for better enhancing the effect of data augmentation and improving the classification capability on each selected algorithm combination, we conduct the experiments of tuning the batch size N on 2 datasets.

3.5.2.1 **Dataset**: The dataset configuration used in this experiment is same as § 3.5.1.1 and § 3.5.1.2

3.5.2.2 **Experiment setup**: As introduced in § 3.5.1.1, the default batch size and number of epochs are 8 and 6. In order to further explore the potential impact of different parameter settings, we pick 4, 8, 16, and 32 as the candidate batch sizes in this experiment, and determine the best number of epochs by plotting the result on accuracy improvement and data augmentation. We pick 5 algorithm combinations for each unlabeled dataset (i.e., the DREIN dataset and VirusShare dataset), which has been initially proved to enlarge the training set with thousands of new malware samples as data augmentation, according to the results in § 3.5.1.1 and § 3.5.1.2.

3.5.2.3 **Result on the DREBIN dataset**: We demonstrate and detail the results on our website [38]. Overall, we can find that adopting 8 as batch size can achieve the highest classification accuracy and the largest size of data augmentation to the best practice for 3 combinations, such as CDN and J48, CDN and REPTree, PS, and RandomForest. For the other 2 combinations, the best batch size is 32. Besides, among all the models, whose data augmentation contains more than a thousand samples, model #1.2 has the best overall result in terms of all evaluation metrics.

3.5.2.4 **Result on VirusShare dataset**: We also demonstrate and detail the results on our website [38]. In general, we can find that adopting 8 as batch size can achieve the highest classification accuracy and the largest size of data augmentation to the best practice for 4 combinations, such as CDN and J48, PS and RandomForest, CC and RandomForest, CT and RandomForest. This is the same as what we found in the experiment on the DREBIN dataset. For the combination of RAkELd and RandomForest, the best batch size is 32. Besides, among all the models, whose

data augmentation contains more than a thousand samples, model #1.2 has the best result in terms of both # of samples and sample-ACC, proved to be an overall best model.

Conclusion 2.2: *By tuning the parameter (i.e., batch size) in Detection-Training, we can achieve the best practice of each algorithm combination on both data augmentation and classification capability. For most situations, when batch size is equal to 8, it can achieve excellent results within 6 epochs.*

3.6 RQ3: How reliable is the pseudo-label for the sample through Detection-Training?

In this section, to verify the authenticity of the pseudo labels which are generated in the process of Detection-Training, we evaluate the effectiveness of MLC models which are trained with the pseudo-labeled samples on all sampled-based and label-based metrics.

3.6.1 Dataset

The dataset used in this experiment is same as § 3.5.2.1. However, we adopt an alternate dataset configuration in order to check the validity of the pseudo labels obtained within data augmentation. For each selected algorithm combination, we train the MLC classifier with a new training set, which is built from those pseudo-labeled malware samples obtained through Detection-Training (§ 3.5.1), and then test it with the manually labeled ground-truth dataset. Please do note that the pseudo-labeled malware samples used on each combination are strictly restricted to the generated data augmentation by the same algorithm combination.

3.6.2 Experiment setup

The algorithm combinations used in this experiment are same as § 3.5.2.2.

3.6.3 Results

In this section, we present the experiment results on the DREBIN dataset and VirusShare dataset referring to sample-based metrics and label-based metrics. We also discuss some insightful conclusions for relevant approaches using active learning and data augmentation.

3.6.3.1 **Evaluation on the DREBIN dataset**: As shown on the left side of Table 9, model #2, which is trained with the combination of CDN and REPTree algorithms on the 5,456 pseudo-labeled samples from DREBIN, outperforms on all evaluation metrics. Compared with the result presents in Table. 5, it performs better on all metrics than the base model which uses the same algorithm combination. Besides, compared with the updated model with the same combination (i.e., model #4 in Table. 7) whose classification capability has been improved a lot through Detection-Training, it also achieves a better result in terms of all metrics. In detail, the hamming loss and zero-one loss have been reduced by 0.053 and 0.107, and the sample-ACC and averaged label-ACC have been improved by 0.014 and 0.053. So that we can make a conclusion that the 5,456 samples which are labeled using this combination during Detection-Training are well predicted. In other words, the generated pseudo labels are highly credible. Model #1, which uses the combination of CDN and J48 that has the best result in § 3.5.1.1, also performs well as the second best in this evaluation with the second lowest hamming loss and zero-one loss, and the second highest F1-score, sample-ACC and averaged label-ACC.

We selected 20 samples and manually verify the authenticity of their pseudo-labels, the details are shown in Appendix B.

3.6.3.2 **Evaluation on VirusShare dataset**: As shown on the right side of Table 9, model #2, which is trained with the combination of PS and RandomForest algorithms on the 5,496 pseudo-labeled samples from the VirusShare dataset, outperforms on all evaluation metrics. Compared with the result presents in Table 5, the sample-ACC is 0.003 less than its base model and the Avg.label-ACC is 0.057 less than that of its base model. Besides, compared with the updated model of the same combination (i.e., model #6 in Table. 8), whose classification capability has been improved a lot through Detection-Training, it also has a lower classification accuracy. The reason for this phenomenon possibly is the samples from the VirusShare dataset collected in recent years (i.e. 2018-2022) may include some variants which use new methods to perform malicious behaviors. Model #3, which uses the combination of RAkELd and RandomForest that has the second best result in § 3.5.1.2, also performs well as the second best in this evaluation with a sample-ACC at 0.669. Model #1, which uses the combination of CDN and J48 that has the best result in § 3.5.1.2, performs as the third best in this evaluation with the second lowest hamming loss and zero-one loss, the second highest F1-score and averaged label-ACC, and the third highest sample-ACC. We also select 20 samples and manually verify the authenticity of their pseudo-labels, the details are shown in Appendix B.

3.6.3.3 **Comparison between DREBIN and VirusShare dataset**: By comparing the results shown in Table 9 with the results of the base model in Table 5, we observe that the overall effectiveness of the proposed approach is validated to be successful because we achieve a significant improvement in all evaluation metrics with the model of CDN and RandomForest trained on pseudo-labeled samples from DREBIN. The result of best practice outperforms its based model by adopting pseudo-labeled samples as a training set and the manually labeled dataset as a test set.

Besides, we also notice some interesting results. For instance, the results on DREBIN are better than that on the VirusShare dataset for most same algorithm combinations, except the combination of PS and RandomForest. And, for the combinations of CDN and RandomForest, RAkELd, and RandomForest, the results of Detection-Training are diametrically opposed between the two datasets. The most possible reason that causes this phenomenon could be the distribution of the dataset on our defined malicious behaviors since DREBIN is a more established family dataset with higher diversity. Another one is about the adaptability of MLC and BC algorithms to our task. Another example is that the sample-ACC of model #2 is a little lower than the base model which uses the same algorithm combination. One of the possible reasons behind this phenomenon could be the samples from the VirusShare dataset collected in recent years (i.e. 2018-2022) may include some variants which use

TABLE 9: Evaluation of models which use samples from the DREBIN dataset and VirusShare dataset

data source	DREBIN					VirusShare				
model id	1	2	3	4	5	1	2	3	4	5
MLC algorithm	CDN	CDN	PS	CC	CT	CDN	PS	RAkELd	CC	CT
basic classifier	J48	REPTree	Random Forest	Random Forest	Random Forest	J48	Random Forest	Random Forest	Random Forest	Random Forest
hamming loss	0.169	**0.091**	0.343	0.237	0.218	0.214	**0.190**	0.248	0.245	0.229
zero-one loss	0.646	**0.393**	0.882	0.888	0.787	0.843	**0.612**	0.899	0.876	0.870
F1-score	0.709	**0.839**	0.354	0.495	0.544	0.622	**0.641**	0.465	0.292	0.351
sample-ACC	0.635	**0.781**	0.343	0.646	0.618	0.523	**0.697**	0.669	0.365	0.461
Avg.label-ACC	0.831	**0.909**	0.657	0.763	0.782	0.786	**0.810**	0.752	0.755	0.772

new methods to perform malicious behaviors. Hence, the features of these behaviors may be not included in our feature set, leading to a miss classification. In § 5, we will discuss some potential directions that can further explore the reasons behind these phenomena and the underlying principles.

Conclusion: *By evaluating the effectiveness of MLC models which are trained with pseudo-labeled samples, we can tell the confidence of generated pseudo labels. The best practice (0.781), which is trained on pseudo-labeled samples from the DREBIN dataset, can even outperform the best-based models (0.733), which is trained on the ground-truth dataset.*

4 Related Work

4.1 Android Malware Classification

Facing the increasing number of malicious apps, malware classification is becoming extremely important. Malware Classification can refer either to the classification of binaries as malicious or benign, or the classification of malware samples into different known malware families. Towards android malware binary classification, many machine learning methods have achieved great success in Android malware [12], [14], [17], [49]–[51]. Many machine learning algorithms such as SVM [13] and Random forest [52] were used to detect malware and have proven to be effective. The method presented by Yerima et al. [53] detect Android malware based on Bayesian Classification models obtained from API calls, system commands, and permissions. Wu et al. [54] used data-flow APIs as classification features to detect Android malware. With the popularity of deep neural networks, researchers utilized the deep neural network models for malware detection [18], [21], [22], [55], [56].

For malware family classification, which classifies malware with common features into malware families, has been proposed as an effective malware analysis method. H. Peng et al. [57] proposed a family classification method using permissions as features, they introduced a probabilistic generative model to rank the risks of Android applications. M. Zhang et al. [58] proposed a novel semantic-based approach that classifies Android malware via dependency graphs. M. Fan et al. [16] proposed an approach that constructs frequent subgraphs to represent the common behaviors of malware samples that belong to the same family for enhancing the correctness of family classification. Y. Bai et al. [20] proposed a novel siamese-network based learning method for malware family classification.

Unlike the existing approaches on binary and family classification above, we highlight that this is the first multi-label Android malware classification approach for each malware which intends to provide more information on fine-grained malicious behaviors through detection results.

4.2 Multi-label Classification

Generally, multi-label classification (MLC) aims to attach multiple relevant labels to an input instance simultaneously through machine learning. During the past decades, multi-label classification has been widely adopted by communities in various domains [59], such as natural language processing, web mining, computer vision, bioinformatics, and multimedia. MLC has mainly engaged the attention of researchers working on text categorization [60], [61], as each member of a document collection usually belongs to more than one semantic category. In web mining, various problems brought by the massive information are also discovered with MLC, such as page categorization and tag suggestion [62]–[64]. In image semantic classification, a common scenario may contain multiple objects. With MLC, researchers can annotate the given scene with multiple labels, which makes scene classification more comprehensive [30], [31]. MLC is also applied to the field of protein function classification, as there are proteins that have multiple properties at the same time or exist in multiple subcellular organelles. Researchers use multiple labels on those data for proteins and achieve promising results with MLC [65]. In the field of multimedia, for the tasks that retrieve relevant (categories) emotions associated with a given piece of music, MLC can achieve success by learning about the music with multiple emotional labels [66], [67]. In our task, there exists the same situation as the above problems from other fields, which is one malware may have multiple malicious behaviors and some of them beyond their malware families' definitions. The original idea of our work is inspired by the above MLC-based work from other domains.

4.3 Data Augmentation and Active Learning

Most supervised learning-based methods tend to require immense amounts of computational and human resources for data annotation so designing effective methods that can enlarge a small labeled dataset without too much effort on annotations is a fundamental research challenge in many classification tasks based on big data. In the past decade, many presented research work using semi-supervised methods mainly focusing on algorithm innovation, engineering

practice, and new application exploration. Along with this style of research and development, recently, data augmentation and active learning have been proved that it can successfully alleviate the data-deficient scenarios in many research fields, such as natural language processing [68] and medical image analysis [69]. For data augmentation, SPADE [70] used a two-phase data augmentation process to enrich a dataset before training a deep-learning classifier. D.H. Lee et al. [41] picked the labels with the largest predicted probability to generate pseudo labels for unlabeled data. Mixmatch [71] applied the mix-up technique to both pseudo-labeled data and unlabeled samples, then use these data to iteratively train the network. L. Samuli [72] formed a consensus prediction of the unknown labels using the outputs of the network-in-training on different epochs. T. Antti [42] took a weight-average of the parameters of the model and used this weight-averaged parameter to generate pseudo labels. For active learning, J. Zhang et al. [73] applied active learning and data augmentation to the field of anomaly detection, the NAF-AL method they proposed uses the augmented samples incorporated with normal samples to train a better anomaly detector. Z. Zhou et al. [74] proposed an active learning approach based on data augmentation for reducing annotation efforts in medical image analysis. In the ED2 [75], active learning was applied in the error detection field, which aims to solve a fundamental problem in data science.

Data augmentation and Active learning also have several applications in the field of malware detection. For data augmentation, Ding et al. [76] proposed a Grad-CAM algorithm to find the raw data representing malware features and generated malware family samples. Chen et al. [77] first transformed malware into images and then used the generative adversarial networks to generate data. The generated data can be used to balance and expand the original dataset. For Active learning, Chen et al. [78] leveraged the Active learning by learning (ALBL) technique based on the experts' feedback. So that the trained models can be updated to reduce the classification error rate.

As we introduced in § 1 and § 2.3, we also face a similar problem in that the number of usable labeled data is limited and the cost of effort in data annotation is unbearable. Inspired by the above work from other domains, we take the idea of active learning and data augmentation to generate pseudo labels for unlabeled samples, iteratively retrain models to validate the correctness of pseudo labels as well as improve the capability of classification.

5 Discussion and Future Work

As we discussed in the introduction (§ 1), there exist (1) malicious behaviors that may be achieved with various key features; (2) malware that has more diverse kinds of malicious behaviors beyond the 6 label; (3) the best classification effectiveness cannot be achieved with API, permission, intent against the implementation of some behaviors; (4) changing of some APIs within the update of Android SDK. All these reasons lead to the incomplete of our defined label set (as well as the used features), which is the first threat in our work. To tackle this threat, we will (1) supplement the malicious behavior label categories and organize a summary of more potential ways to achieve identical malicious behaviors by analyzing more malware collected in recent years; (2) try other feature representation methods, such as call graph, etc, for more context information, which is beneficial to overcome the current limitation of "feature overlap"; (3) systematically update APIs in the feature dictionary to meet different Android system versions. As the experiment of the authenticity validation on pseudo labels (§ 3.6) shows, the pseudo labels of involved samples from the VirusShare dataset show less credibility compared to those from the DREBIN dataset. That is possible because (1) the VirusShare data are collected in recent years, which may include variants which use new approaches to perform malicious behaviors; (2) the dataset may be imbalanced on different labels; (3) training the base model with limited data can make the collected pseudo-labeled samples imbalanced. This is the second threat in our consideration. To face this, we will look into the technologies from other domains for (1) saving efforts on manual analysis and data annotation for constructing a relatively balanced and original dataset; (2) automatically verifying the labels in the process of Detection-Training with open-access security reports (e.g., confidence) and the changes in numerical information (e.g., variance, entropy) during active learning. Besides the threats, we will also try to further explore the potential application of multi-label classification in the field of malware detection, such as AI malicious behaviors interpretation based on MLC.

6 Conclusion

This paper is an initial attempt to study the multi-label classification problem of malicious behaviors in Android malware classification. We come up with a label set of 6 types of malicious behaviors, and manually label 180 malware with an in-depth study, which is released as a benchmark. To address the challenge in data annotation, we demonstrate a novel active learning method based on data augmentation and achieve a promising result.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China under Grant No. 62102284, No. 61872262, and Smart Platform Infrastructure Research on Integrative Technology (SPIRIT) under Grant CSA-CSEC-DC-20-083.

References

[1] (2021) Android Statistics (2021). [Online]. Available: https://www.businessofapps.com/data/android-statistics/
[2] S. Chen, L. Fan, C. Chen, T. Su, W. Li, Y. Liu, and L. Xu, "Storydroid: Automated generation of storyboard for Android apps," in *International Conference on Software Engineering (ICSE)*. IEEE, 2019.
[3] S. Chen, L. Fan, C. Chen, and Y. Liu, "Automatically distilling storyboard with rich features for Android apps," in *IEEE Transactions on Software Engineering (TSE)*. IEEE, 2022.
[4] (2021) Android and Google Play statistics. [Online]. Available: https://www.appbrain.com/stats
[5] (2021) Development of new Android malware worldwide from June 2016 to March 2020. [Online]. Available: https://www.statista.com/statistics/680705/global-android-malware-volume/
[6] (2021) 44 Worrying Malware Statistics to Take Seriously in 2022. [Online]. Available: https://legaljobs.io/blog/malware-statistics/

[7] C. Tang, S. Chen, L. Fan, L. Xu, Y. Liu, Z. Tang, and L. Dou, "A large-scale empirical study on industrial fake apps," in *International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 2019.

[8] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of "piggybacked" mobile applications," in *ACM Conference on Data and Application Security and Privacy*, 2013.

[9] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of Android malware behaviors." in *Annual Network & Distributed System Security Symposium*, 2015.

[10] G. Meng, R. Feng, G. Bai, K. Chen, and Y. Liu, "Droidecho: An in-depth dissection of malicious behaviors in Android applications," *Cybersecurity*, 2018.

[11] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN conference on Programming Language Design and Implementation*, 2014.

[12] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in Android," in *International conference on security and privacy in communication systems*, 2013.

[13] 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 *Annual Network & Distributed System Security Symposium*, 2014.

[14] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "Stormdroid: A streaminglized machine learning-based system for detecting android malware," in *ACM ASIA Conference on Computer and Communications Security*, 2016.

[15] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting Android malware by building markov chains of behavioral models," *ACM Transactions on Privacy and Security*, 2016.

[16] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Transactions on Information Forensics and Security*, 2018.

[17] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *computers & security*, 2018.

[18] R. Feng, S. Chen, X. Xie, L. Ma, G. Meng, Y. Liu, and S.-W. Lin, "MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform," in *2019 24th International Conference on Engineering of Complex Computer Systems*, 2019.

[19] M. Fan, J. Liu, X. Luo, K. Chen, T. Chen, Z. Tian, X. Zhang, Q. Zheng, and T. Liu, "Frequent subgraph based familial classification of Android malware," in *International Symposium on Software Reliability Engineering*, 2016.

[20] 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 *International Conference on Software Engineering*, 2020.

[21] R. Feng, J. Q. Lim, S. Chen, S.-W. Lin, and Y. Liu, "SeqMobile: An Efficient Sequence-Based Malware Detection System Using RNN on Mobile Devices," in *2020 25th International Conference on Engineering of Complex Computer Systems*, 2020.

[22] 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*, 2020.

[23] Y. BAI, S. CHEN, Z. XING, and X. LI, "Argusdroid: Detecting Android malware variants by mining permission-api knowledge graph."

[24] 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 Transactions on Software Engineering and Methodology (TOSEM)*, 2021.

[25] F. Maggi, A. Bellini, G. Salvaneschi, and S. Zanero, "Finding non-trivial malware naming inconsistencies," in *International Conference on Information Systems Security*, 2011.

[26] T. Kelchner, "The (in)consistent naming of malcode," *Computer Fraud & Security*, 2010.

[27] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *Research in Attacks, Intrusions, and Defenses*, F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, Eds. Cham: Springer International Publishing, 2016, pp. 230–253.

[28] M. Hurier, G. Suarez-Tangil, S. Kumar, T. Bissyandé, and L. Cavallaro, "Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware," in *IEEE/ACM 14th International Conference on Mining Software Repositories*, 2017.

[29] M. Hurier, K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds. Cham: Springer International Publishing, 2016, pp. 142–162.

[30] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern recognition*, 2004.

[31] F. Kang, R. Jin, and R. Sukthankar, "Correlated label propagation with application to multi-label learning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[32] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining*, 2007.

[33] MLC for Android Malware. [Online]. Available: https://github.com/qqj1130247885/MLC-for-Android-Malware

[34] VirusTotal. [Online]. Available: http://www.virustotal.com/

[35] Ali, Feizollah, Nor, Badrul, Anuar, Rosli, Salleh, Guillermo, Suarez-Tangil, and Steven, "Androdialysis: Analysis of android intent effectiveness in malware detection," *Computers & Security*, 2017.

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

[37] S. Chen, L. Fan, G. Meng, T. Su, M. Xue, Y. Xue, Y. Liu, and L. Xu, "An empirical assessment of security risks of global Android banking apps," in *International Conference on Software Engineering*, 2020.

[38] (2022) Overview of MLC for Android malware. [Online]. Available: https://sites.google.com/view/tdsc-mlc

[39] A. G. de Sá, C. G. Pimenta, G. L. Pappa, and A. A. Freitas, "A robust experimental evaluation of automated multi-label classification methods(supplemental material)," in *Genetic and Evolutionary Computation Conference*, 2020.

[40] T. Tran, T.-T. Do, I. Reid, and G. Carneiro, "Bayesian generative active deep learning," in *International Conference on Machine Learning*, 2019.

[41] D.-H. Lee *et al.*, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on challenges in representation learning, ICML*, 2013.

[42] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," *Advances in neural information processing systems*, 2017.

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

[44] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes, "MEKA: A multi-label/multi-target extension to Weka," *Journal of Machine Learning Research*, 2016.

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

[46] B. Pfahringer, P. Reutemann, I. H. Witten, M. Hall, E. Frank, and G. Holmes, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, 2009.

[47] Metrics Matter, examples from Binary and Multilabel Classication. [Online]. Available: http://sanmi.cs.illinois.edu/documents/koyejo-metrics.pdf

[48] X.-Z. Wu and Z.-H. Zhou, "A unified view of multi-label performance measures," in *International Conference on Machine Learning*, 2017.

[49] S. Chen, M. Xue, and L. Xu, "Towards adversarial detection of mobile malware: poster," in *MobiCom*, 2016.

[50] L. Fan, M. Xue, S. Chen, L. Xu, and H. Zhu, "Poster: Accuracy vs. time cost: Detecting android malware through pareto ensemble pruning," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.

[51] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: Android malware detection using static analysis of applications," in *International Conference on New Technologies, Mobility and Security*, 2016.

[52] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: evaluating android anti-malware against transformation attacks," in *ACM ASIA Conference on Computer and Communications Security*, 2013.
[53] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using bayesian classification," in *International Conference on Advanced Information Networking and Applications*, 2013.
[54] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of android malware based on the usage of data flow apis and machine learning," *Information and software technology*, 2016.
[55] R. Feng, Y. Liu, and S. Lin, "A performance-sensitive malware detection system on mobile platform," in *Formal Methods and Software Engineering*, Y. Ait-Ameur and S. Qin, Eds. Cham: Springer International Publishing, 2019, pp. 493–497.
[56] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, 2018.
[57] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 241–252.
[58] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual api dependency graphs," in *ACM ASIA Conference on Computer and Communications Security*, 2014.
[59] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, 2014.
[60] N. Ueda and K. Saito, "Parametric mixture models for multi-labeled text," in *Advances in neural information processing systems*, 2003.
[61] H. Kazawa, T. Izumitani, H. Taira, and E. Maeda, "Maximal margin labeling for multi-topic text categorization," in *Advances in neural information processing systems*, 2005.
[62] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Multilabel text classification for automated tag suggestion," in *Proceedings of the ECML/PKDD*, 2008.
[63] Y. Song, L. Zhang, and C. L. Giles, "A sparse gaussian processes classification framework for fast tag suggestions," in *ACM International Conference on Information and Knowledge Management*, 2008.
[64] L. Tang, S. Rajan, and V. K. Narayanan, "Proceedings of the 18th international conference on world wide web," in *WWW*, 2009.
[65] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *European conference on principles of data mining and knowledge discovery*, 2001.
[66] T. Li and M. Ogihara, "Detecting emotion in music," 2003.
[67] K. Trohidis, G. Tsoumakas, G. Kalliris, I. P. Vlahavas *et al.*, "Multi-label classification of music into emotions." in *International Symposium on Music Information Retrieval*, 2008.
[68] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy, "A survey of data augmentation approaches for NLP," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, Aug. 2021, pp. 968–988. [Online]. Available: https://aclanthology.org/2021.findings-acl.84
[69] S. Budd, E. C. Robinson, and B. Kainz, "A survey on active learning and human-in-the-loop deep learning for medical image analysis," *Medical Image Analysis*, 2021.
[70] M. Pham, C. A. Knoblock, M. Chen, B. Vu, and J. Pujara, "Spade: A semi-supervised probabilistic approach for detecting errors in tables," 2021.
[71] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. A. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," *Advances in neural information processing systems*, 2019.
[72] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *International Conference on Learning Representations*, 2017.
[73] J. Zhang, K. Saleeby, T. Feldhausen, S. Bi, A. Plotkowski, and D. Womble, "Self-supervised anomaly detection via neural autoregressive flows with active learning," in *NIPS 21 Workshop on Deep Generative Models and Downstream Applications*, 2021.
[74] Z. Zhou, J. Y. Shin, S. R. Gurudu, M. B. Gotway, and J. Liang, "Active, continual fine tuning of convolutional neural networks for reducing annotation efforts," *Medical image analysis*, 2021.
[75] F. Neutatz, M. Mahdavi, and Z. Abedjan, "Ed2: A case for active learning in error detection," in *ACM International Conference on Information and Knowledge Management*, 2019.
[76] D. Yuxin, W. Guangbin, M. Yubin, and D. Haoxuan, "Data augmentation in training deep learning models for malware family classification," in *2021 International Conference on Machine Learning and Cybernetics (ICMLC)*, 2021, pp. 1–6.
[77] Y.-M. Chen, C.-H. Yang, and G.-C. Chen, "Using generative adversarial networks for data augmentation in Android malware detection," in *2021 IEEE Conference on Dependable and Secure Computing (DSC)*, 2021, pp. 1–8.
[78] C.-W. Chen, C.-H. Su, K.-W. Lee, and P.-H. Bair, "Malware family classification using Active learning by learning," in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, 2020, pp. 590–595.

Qijing Qiao received her bachelor degree from Tianjin Normal University in 2020. She is currently a M.Sc student in the College of Intelligence and Computing, Tianjin University, since Sep 2020. Her research interests include Malware detection and malicious behavior analysis.

Ruitao Feng received his Ph.D. degree from the Nanyang Technological University in 2021. He is now a senior research associate in University of New South Wales, and an adjunct researcher in NTU. Previously, he was a research fellow (2021-2022) and a research assistant (2014-2021) in NTU. His research interests include discovering and solving security problems on mobile platform, IoT system and AI-based cybersecurity system.

Sen Chen (Member, IEEE) is an Associate Professor at Tianjin University, China. Before that, he was a Research Assistant Professor at Nanyang Technological University, and a Research Assistant of NTU from 2016 to 2019 and a Research Fellow from 2019-2020. He received his Ph.D. degree from East China Normal University, China, in 2019. His research focuses on Security and Software Engineering.

Fei Zhang is now an undergraduate student at TianJin University since 2018. He has received the National Encouragement Scholarship in 2020 and will start to pursue M.Sc on cybersecurity at Tianjin University in early Sep 2022.

Xiaohong Li (Member, IEEE) received her Ph.D. degree from Tianjin University, China. She is a full Professor with the College of Intelligence and Computing and the director of the Institute of Software and Information Security Engineering, Tianjin University. She is mainly engaged in computer science and computer application, software engineering and security software engineering, high-assurance software and network security and other information security fields.

APPENDIX A
EVALUATION ON PARAMETERS TUNING IN DETECTION-TRAINING

A.1 Result on DREBIN dataset

Table 10 shows the evaluation results of applying Detection-Training with 4 different batch sizes on the DREBIN dataset. In general, we observe that Detection-Training can always achieve a final accuracy improvement and data augmentation before/at epoch 6 with every batch size for algorithm combinations. Unless Detection-Training fails in the task (e.g., model #1.4 in Table 10).

For the combination of CDN and J48, we can observe that model #1.2, which adopts 8 as its batch size, outperforms the best result. Specifically, it has the lowest hamming loss and zero-one loss at 0.106 and 0.467, it also has the highest F1-score, sample-ACC, averaged label-ACC, and # of samples at 0.779, 0.867, 0.894, and 4,872. For the combination of CDN and REPTree, there are 2 models that can enlarge the dataset with thousand pseudo-labeled samples, they are model #4.2 and model #4.3, which both successfully get 5,456 new samples as data augmentation. More specifically, model #4.2 outperforms model #4.3 in terms of the results on all other 6 selected evaluation metrics. For the combination of PS and RandomForest, when 4, 8, and 16 are chosen as the batch size, the 4 final models obtained applying Detection-Training can achieve the same best results on the test dataset in terms of all evaluation metrics. The best three (i.e., model #6.1-6.3) can enlarge the training set with a data augmentation at 5,456 pseudo-labeled samples, and their sample-ACC is 0.733. For the combination of CC and RandomForest, the model #8.4, uses 32 as its batch size in Detection-Training. Its data augmentation involves 5,440 pseudo-labeled samples. It has the same best sample-ACC at 0.767 compared to model #8.2, which uses 8 as its batch size. For the combination of CT and RandomForest, there are 2 models, which are model #10.2 and 10.4, can enlarge the dataset with over one thousand pseudo-labeled samples. The data augmentation of the model #10.4 involves 5,440 pseudo-labeled samples, which is 2,000 more than that of model #10.2. Model #10.4 has a lower zero-one loss at 0.433 and a higher F1-score at 0.78, which makes it the best.

A.2 Result on VirusShare dataset

Table 11 shows the evaluation results of applying Detection-Training with 4 different batch sizes on the dataset from VirusShare. In general, we observe that Detection-Training can always achieve a final accuracy improvement and data augmentation before/at epoch 6 with every batch size for algorithm combinations. Unless Detection-Training fails in the task (e.g., model #1.3 in Table 11).

For the combination of CDN and J48, we can observe t/hat model #1.2, which adopts 8 as its batch size, outperforms the best result. Specifically, it has the lowest hamming loss and zero-one loss, it also has the highest F1-score, sample-ACC, averaged label-ACC, and # of samples. For the combination of PS and RandomForest, all these 4 models can enlarge the dataset with thousand pseudo-labeled samples, they successfully get more than 5,000 new samples as data augmentation. These 4 models also have an equal sample-ACC at 0.733, which is 0.033 more than that of their base model. However, model #6.2 outperforms the other 3 models in terms of the results on other metrics except for hamming loss. For the combination of RAkELd and RandomForest, when 8, 16, and 32 are chosen as the batch size, the 3 final models obtained by applying Detection-Training can enlarge the training set with more than a thousand samples. Among them, Model #7.4 can outperform the other 2 models by achieving the largest size of data augmentation at 5,472. Besides, its sample-ACC is 0.767. For the combination of CC and RandomForest, model #8.2, which uses 8 as its batch size in Detection-Training has the best result. Its data augmentation involves 1,088 pseudo-labeled samples. It has a sample-ACC at 0.733 and averaged label-ACC at 0.872. For the combination of CT and RandomForest, the model #10.2 can enlarge the dataset with more than one thousand pseudo-labeled samples at 1,848. The sample-ACC of it is 0.733, which is 0.66 more than that of its base model.

APPENDIX B
CASE STUDY

B.1 Summary

We randomly select 40 pseudo-labeled samples from our expanded dataset, 20 from the DREBIN dataset, and 20 from the VirusShare dataset. We manually check their actual behaviors to verify the authenticity of their pseudo-labels. All these 40 samples are augmented by the best models (model #1.2 in Table 10 and model 1.2 in Table 11). Samples are randomly selected on the basis of making sure that there are at least 5 samples on each malicious type. Note that, for the samples from the VirusShare dataset, considering different Antivirus engines may disagree on the malware family name, the result of a single detection may be unreliable. Hence, we refer to multiple family labels provided by different virus detection engines.

Table 12 presents the results of the case study. We introduce 3 metrics to evaluate the manual study. When the pseudo labels of a sample are exactly as same as its actual malicious behaviors, the sample will be marked as "Correct". The digits under the "Correct" column show the total number of confirmed samples. Once a sample is mislabeled with a malicious behavior that it doesn't perform, it will be categorized as "Misclassified". If the pseudo labels of a sample miss any behaviors belonging to it, it will be marked as "Missing". In the "Cause" column, we list the main reasons for the wrongly classified cases.

For the DREBIN dataset, 18 samples are correctly classified on L1 (SMS-related) and 2 samples are misclassified. For the VirusShare dataset, 5 samples are misclassified on the L1 label. We find that these samples legitimately perform SMS-related functions. However, there are many overlaps between the features of the programs in the legitimate use of the SMS function and the features of the programs when the SMS function is executed maliciously, such as "sendSMS()". This leads to misclassification of the model.

The same is true for L6 (Internet-related). Android apps usually use the network along with their normal behaviors such as downloading or uploading something from the remote server. At the same time, malicious Internet-related

TABLE 10: The result of assessing the impact of batch size N on DREBIN

MLC algorithm	CDN				CDN				PS				CC				CT			
basic classifier	J48				REPTree				RandomForest				RandomForest				RandomForest			
model id	1.1	1.2	1.3	1.4	4.1	4.2	4.3	4.4	6.1	6.2	6.3	6.4	8.1	8.2	8.3	8.4	10.1	10.2	10.3	10.4
batch size	4	8	16	32	4	8	16	32	4	8	16	32	4	8	16	32	4	8	16	32
stopped epoch	2	6	1	6	5	6	5	5	3	2	2	3	2	2	3	3	3	2	2	4
# of samples	32	**4872**	16	0	148	**5456**	**5456**	608	**5456**	**5456**	**5456**	**5440**	136	**1720**	768	**5440**	192	**3440**	768	**5440**
hamming loss	0.133	**0.106**	0.133	0.133	0.122	**0.144**	0.189	0.135	**0.139**	**0.139**	**0.139**	**0.139**	0.122	**0.128**	0.122	**0.128**	0.122	**0.122**	0.122	**0.122**
zero-one loss	0.533	**0.467**	0.500	0.533	0.500	**0.500**	0.633	0.467	**0.533**	**0.533**	**0.533**	**0.533**	0.500	**0.467**	0.467	**0.467**	0.467	0.467	0.467	**0.433**
F1-score	0.736	**0.779**	0.697	0.696	0.716	**0.729**	0.644	0.740	**0.730**	**0.730**	**0.730**	**0.730**	0.736	**0.752**	0.769	**0.752**	0.747	0.769	0.769	**0.780**
sample-ACC	0.833	**0.867**	0.767	0.733	0.800	**0.767**	0.700	0.767	**0.733**	**0.733**	**0.733**	**0.733**	0.733	**0.767**	0.767	**0.767**	0.733	**0.767**	0.767	**0.767**
Δsample-ACC	0.100	**0.134**	0.034	0.000	0.100	**0.067**	0.000	0.067	**0.033**	**0.033**	**0.033**	**0.033**	0.066	**0.100**	0.100	**0.100**	0.066	**0.100**	0.100	**0.100**
Avg.label-ACC	0.867	**0.894**	0.867	0.867	0.878	**0.856**	0.811	0.867	**0.861**	**0.861**	**0.861**	**0.861**	0.878	**0.872**	0.878	**0.872**	0.878	**0.878**	0.878	**0.878**

TABLE 11: The result of assessing the impact of batch size N on VirusShare dataset

MLC algorithm	CDN				PS				RAkELd				CC				CT			
basic classifier	J48				RandomForest				RandomForest				RandomForest				RandomForest			
model id	1.1	1.2	1.3	1.4	6.1	6.2	6.3	6.4	7.1	7.2	7.3	7.4	8.1	8.2	8.3	8.4	10.1	10.2	10.3	10.4
batch size	4	8	16	32	4	8	16	32	4	8	16	32	4	8	16	32	4	8	16	32
stopped epoch	2	6	3	5	3	3	3	3	2	5	2	3	4	5	5	2	3	5	3	3
# of samples	108	**5416**	96	**4544**	**5500**	**5496**	**5488**	**5472**	508	**2432**	**1344**	**5472**	200	**1088**	368	416	400	**1848**	480	544
hamming loss	0.106	**0.133**	0.156	0.150	0.122	0.139	0.144	0.144	0.139	**0.144**	**0.144**	**0.144**	0.117	**0.128**	0.133	0.133	0.128	**0.128**	0.128	0.128
zero-one loss	0.367	**0.500**	0.600	0.667	0.533	**0.533**	0.533	0.533	0.500	**0.533**	**0.533**	**0.533**	0.443	**0.500**	0.500	0.467	0.500	**0.467**	0.467	0.433
F1-score	0.780	**0.733**	0.669	0.697	0.716	**0.730**	0.708	0.708	0.731	**0.720**	**0.720**	**0.720**	0.769	**0.736**	0.736	0.719	0.736	**0.719**	0.747	0.730
sample-ACC	0.833	**0.833**	0.767	0.767	0.733	**0.733**	0.733	0.733	0.767	**0.767**	**0.767**	**0.767**	0.767	**0.733**	0.733	0.733	0.733	**0.733**	0.733	0.733
Δsample-ACC	0.100	**0.100**	0.034	0.034	0.033	**0.033**	0.033	0.033	0.067	**0.067**	**0.067**	**0.067**	0.100	**0.066**	0.066	0.066	0.066	**0.066**	0.066	0.066
Avg.label-ACC	0.894	**0.867**	0.844	0.867	0.856	**0.861**	0.856	0.856	0.861	**0.856**	**0.856**	**0.856**	0.883	**0.872**	0.867	0.867	0.872	**0.872**	0.872	0.872

TABLE 12: The result of case study

		Correct	Misclassified	Missing	Cause
DREBIN	sample	12	4	8	
	L1	18	2	0	Feature overlap
	L2	17	1	2	Malicious codes in native packages
	L3	16	1	3	Other ways to achieve
	L4	17	0	3	Too few samples in benchmark
	L5	18	1	1	Too few samples in benchmark
	L6	18	2	0	Feature overlap
VirusShare	sample	11	8	9	
	L1	14	5	1	Feature overlap
	L2	16	2	2	Malicious code in native package
	L3	19	0	1	API evolution
	L4	16	2	2	Too few samples in benchmark
	L5	17	1	2	Too few samples in benchmark
	L6	16	3	1	Feature overlap

A sample may have both "missing" and "misclassified" cases.
*L1-6 refers to the label-ACC of 6 predefined labels (i.e., "SMS-related", "lock in", "Re-infection", "Telephony-related", "Ads", "Internet-related").

behaviors are often executed in conjunction with other malicious behaviors. Therefore, the feature overlap between the models leads to misclassification.

For L2 (Lock-in), there is 1 sample from the DREBIN dataset in "misclassified" and 2 samples in "missing". For the VirusShare dataset, there are 2 samples in "misclassified" and 2 samples in "missing". We find that in these cases, the malicious codes are located in the native packages. However, in this work, we didn't consider this kind of malware since only API and manifest properties are used as features. As a result, the model cannot correctly classify these cases.

For L3 (Re-infection), there is 1 sample from the DREBIN dataset in "misclassified" and 3 samples in "missing". We find that the implementation of Re-infection for these cases, such as rooting it first, then disguising itself as a system application to prevent it from being uninstalled, is not seen in our benchmark. For the wrongly classified cases in the VirusShare dataset, there exists an API evolution between different Android systems. Therefore, it is difficult for the model to classify correctly in such cases. In the VirusShare dataset, there is 1 sample in "missing".

For L4 (Telephony-related), there are 3 samples from the DREBIN dataset in "missing". For the VirusShare dataset, 2 samples are in "misclassified" and 2 in "missing". A possible reason is that L4 has only 10 samples in the manually labeled dataset, and the model has learned too little useful information. In subsequent iterations of Detection-Training, the feature information related to L4 is gradually diluted, which in turn leads to a decrease in the classification ability of the model. There exists the same situation in the results of L5 (Ads).

B.2 Detailed Analysis

We present detailed analyses of 4 representative samples from the DREBIN dataset and analyses of 3 representative samples from the VirusShare dataset as follows.

A malware named "Dynamic Wallpaper" is from the DREBIN dataset. It was augmented to our dataset and pseudo-labeled by the combination CDN and J48 (model #1.2 in Table 8) as [1,0,0,0,0,1], which means it is predicted to have the behaviors related to SMS and Internet. It belongs to "kmin" malware family. Generally, malware belonging to "kmin" may contain the following malicious behaviors after installation, it lurks in the user's mobile phone system and steals the user's private information through Internet, which is belonging to the label "Internet-related". At the same time, it sends a deduction SMS at a very short interval, which will cause users to generate huge bills unknowingly in a very short period of time. The kind of malware will also actively intercept WAP, SMS and MMS text messages to cover up its malicious behavior and this behavior is belonging to the label "MS-related". The pseudo-labels are exactly the same as the actual behavior of the samples.

A malware named "certificado" is from the DREBIN dataset. It was augmented to our dataset and pseudo-labeled by the combination CDN and J48 (model #1.2 in Table 8) as [1,0,0,0,0,0], which means it is predicted to have the behaviors related to SMS. It belongs to the malware family "Zitmo". Generally, malware belongs to "ZitMo" search messages containing the mTAN code which is sent by the bank in the user's inbox. After that, it sends the mTAN code to the mobile phone of the cybercriminal by SMS. The cybercriminal uses the mTAN code to complete the transaction and steal the user's bank savings. This type of behavior belongs to the label "SMS-related". The pseudo-labels are exactly the same as the actual behavior of the samples.

"AdFree" is a malware from the DREBIN dataset. It is belonging to the malware family "gamex". Generally, once the malware of this family is installed, it will silently install various malware carried by it without the user's permission. These malware will be privately connected to the Internet, and other malicious applications will be downloaded secretly, which makes the user's mobile phone a paradise for malware. These malicious behaviors is belonging to the label "Ads" and "Internet-related". It was pseudo-labeled by the combination CDN and J48 (model #1.2 in Table 8) as [1,0,0,0,1,1], which represent "SMS-related", "Ads", "Internet-related". Compared with its actual behaviors, we can find it was mispredicted to have behaviors related to SMS. The possible reason is that the sample normally uses SMS-related APIs, permissions, or intents, and our model incorrectly classifies these legal SMS behaviors into malicious behaviors. In response to this problem, we will adjust the granularity of features in our future work. Specifically, we can construct features with sequential execution or bundled relationship into a feature group, which will help distinguish the legitimate behavior of the app from malicious behavior.

"Madcoaster Free Fan App" is a malware from the DREBIN dataset and it is belonging to the malware family "Fakeapp". Malware in "Fakeapp" can send text messages privately, which is belonging to the label "SMS-related". They can also dial specific numbers privately, which is belonging to the label "Telephony-related". Besides, they also display advertisements on the device and this behavior is belonging to the label "Ads". After installation, they hide their desktop icons from being uninstalled, this type of behavior is belonging to the label "Re-infection". "Madcoaster Free Fan App" was pseudo-labeled by the combination CDN and J48 (model #1.2 in Table 8) as [0,0,0,0,1,1], which represent "Ads", "Internet-related". Comparing the pseudo-label with its actual behaviors, we can find that it was misclassified into the category "Internet-related". There may be two reasons for this misclassification. First, when the sample performs other types of malicious behavior, such as obtaining information of ads, it involves interacting with a remote server. In this process, Internet-related features are called, which leads to the misclassification of the model. Second, Internet-related malicious behaviors usually accompany other types of malicious behaviors, and the boundaries are blurred, making it easy to misclassify. In order to solve the case of misclassification, we can further subdivide the sample features and combine related features into a group, so as to more clearly define the malicious behavior with blurred boundaries. We also noticed that the pseudo-labels given by the model are missing, "SMS-related", "Telephony-related", and "Re-infection" are not accurately predicted. That is because it was collected in 2012, whereas nearly half of our manually labeled data was collected in 2008. There is a version difference between the APIs, the features of newer APIs may not be fully learned by the model, thus resulting in missing predictions. In future work, we will systematically update APIs to the feature dictionary to meet different Android system versions and avoid the problem of missing label prediction.

"Setting" is a malware from the VirusShare dataset. It was augmented and pseudo-labeled by the algorithm combination of CDN and J48 (model #1.2 in Table 9) as [1,0,0,0,1,1], which means it is predicted to have behaviors related to SMS, Ads, and Internet. We refer to the family classification results from VirusTotal to check its real behavior. It is classified into the family "Hiddapp" by more than 11 antivirus engines, such as Antiy-AVL, AegisLab, Avira, etc. Generally, malware belonging to "Hiddapp" may contain the following malicious behaviors, such as hiding the icon after installation and hiding it in the system folder by using superuser permissions. Users cannot uninstall it, which belongs to the label "Re-infection". It also uses different methods to show users as many ads as possible, including installing new hidden adware, which is belonging to the label "Ads". Except referring to the results given by VirusTotal, we also decompiled it and made a static analysis. By analyzing its program flow of control at the level of API call, we checked the behaviors defined by its family "Hiddapp" and found that it may read databases like contact or SMS, get information like IMEI, phone number, or OS version and send them to a remote service. These behaviors belong to the label "SMS-Related" and "Internet". Comparing the true behaviors of this malware and its pseudo-label, we can find that the pseudo-label contains part of the real malicious behavior of the malware, including "SMS-related", "Ads" and "Internet-related", but

the malicious behavior under the label "Re-infection" is not revealed. That is mainly because samples in the VirusShare dataset were collected between 2018 - 2012, whereas nearly half of our manually labeled data was collected in 2008. There is a version difference between the APIs, the features of newer APIs may not be fully learned by the model, thus resulting in missing predictions. In future work, we will systematically update APIs to the feature dictionary to meet different Android system versions and avoid the problem of missing label prediction.

"Master Lu evaluation" is a malware from the VirusShare dataset. It was pseudo-labeled by the algorithm combination of CDN and J48 (model #1.2 in Table 9) as [1,0,0,0,1,1], which means it is predicted to have behaviors related to SMS, Ads, and Internet. According to the detection results from VirusTotal, it is classified into the family "Adware" by "K7GW" and "DrWeb". Malware belonging to "Adware" could download and install other applications without the user's permission. It also can be bundled with other software and advertised with pop-up ads. These behaviors belong to the label "Ads" and were correctly predicted. It was classified into the malware family "Trojan.Generic.gwswr" by "Jiangmin". Malware belonging to "Jiangmin" could force the infected device to actively connect to the remote server, obtain the local information of the device and send it to the web server designated by the malicious attacker. These behaviors fall under the label "Internet-related" and were correctly predicted. Comparing the pseudo-labels and their actual behaviors, we can find that it was misclassified into "SMS-related". The possible reason is that the sample normally uses SMS-related APIs, permissions, or intents, and our model incorrectly classifies these legal SMS behaviors into malicious behaviors. In response to this problem, we will adjust the granularity of features in our future work.

"Install" is a malware from the VirusShare dataset and it was pseudo-labeled by the algorithm combination of CDN and J48 (model #1.2 in Table 9) as [0,0,0,0,1,0], which means it is predicted to have the behaviors related to Ads. According to the detection results from VirusTotal, it is classified into the family "Android:Cerberus-AR [Bank]" by "Avast", "AVG" etc. Malware in this family could hide the desktop icon and then wait for the instructions sent by the remote service to complete some specific operations, such as forwarding text messages, accessing device information, etc. This malicious behavior belongs to the label "SMS-related", "Internet-related", and "re-infected'. Comparing with its actual behaviors, we find it was misclassified into the label "Ads" and its actual behaviors were not correctly predicted. The reason behind this situation is possible because samples from the VirusShare dataset were collected in recent years (i.e.2018-2022) and this sample may be a variant that uses new methods to perform malicious behaviors. Hence, the features of these behaviors may be not included in our feature set, leading to a miss classification. In future work, we will continue analyzing more malware collected in recent years, and try to complete our feature dictionary using other feature representation methods, such as call graph, etc.