Received May 21, 2021, accepted June 21, 2021, date of publication June 29, 2021, date of current version July 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3093366

An Adaptive Behavioral-Based Incremental Batch Learning Malware Variants Detection Model Using Concept Drift Detection and Sequential Deep Learning

ABDULBASIT A. DAREM[1], (Member, IEEE), FUAD A. GHALEB[2,3], ASMA A. AL-HASHMI[1], JEMAL H. ABAWAJY[4], (Senior Member, IEEE), SULTAN M. ALANAZI[1], AND AFRAH Y. AL-REZAMI[5,6]

[1]Department of Computer Science, Northern Border University, Arar 91431, Saudi Arabia
[2]Faculty of Engineering, School of Computing, Universiti Teknologi Malaysia (UTM), Johor Bahru 81310, Malaysia
[3]Department of Computer and Electronic Engineering, Sana'a Community College, Sana'a, Yemen
[4]Cybersecurity Research and Innovation Centre, Deakin University, Burwood, VIC 3217, Australia
[5]Mathematics Department, Prince Sattam Bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia
[6]Department of Statistics and Information, Sana'a University, Sana'a, Yemen

Corresponding authors: Abdulbasit A. Darem (basit.darem@nbu.edu.sa) and Fuad A. Ghaleb (abdulgaleel@utm.my)

This work was supported by the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia through project 1385.

ABSTRACT Malware variants are the major emerging threats that face cybersecurity due to the potential damage to computer systems. Many solutions have been proposed for detecting malware variants. However, accurate detection is challenging due to the constantly evolving nature of the malware variants that cause concept drift. Existing malware detection solutions assume that the mapping learned from historical malware features will be valid for new and future malware. The relationship between input features and the class label has been considered stationary, which doesn't hold for the ever-evolving nature of malware variants. Malware features change dynamically due to code obfuscations, mutations, and the modification made by malware authors to change the features' distribution and thus evade the detection rendering the detection model obsolete and ineffective. This study presents an Adaptive behavioral-based Incremental Batch Learning Malware Variants Detection model using concept drift detection and sequential deep learning (AIBL-MVD) to accommodate the new malware variants. Malware behaviors were extracted using dynamic analysis by running the malware files in a sandbox environment and collecting their Application Programming Interface (API) traces. According to the malware first-time appearance, the malware samples were sorted to capture the malware variants' change characteristics. The base classifier was then trained based on a subset of historical malware samples using a sequential deep learning model. The new malware samples were mixed with a subset of old data and gradually introduced to the learning model in an adaptive batch size incremental learning manner to address the catastrophic forgetting dilemma of incremental learning. The statistical process control technique has been used to detect the concept drift as an indication for incrementally updating the model as well as reducing the frequency of model updates. Results from extensive experiments show that the proposed model is superior in terms of detection rate and efficiency compared with the static model, periodic retraining approaches, and the fixed batch size incremental learning approach. The model maintains an average of 99.41% detection accuracy of new and variants malware with a low updating frequency of 1.35 times per month.

INDEX TERMS Malware variant detection, adaptive incremental batch learning, concept drift detection, deep learning, statistical process control.

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

I. INTRODUCTION

Recently, malware threats have been dramatically increased due to the increasing of internet users, the proliferation of malware creation tools, and the use of obfuscation techniques by malware authors [1], [2]. Malware is a general term that refers to malicious software or any unwanted software which is design to gain unauthorized access, manipulate computer content, violate privacy, steal valuable information, and/or disrupt normal operation [1]–[4]. Malware can be found in many types such as viruses, worms, botnet, backdoors, trojan horses, ransomware, rootkit among many other families [3]. Each type of malware attacks and functions differently. The consequence of malware also varies according to the type of malware, type of the infected target, and the purpose of the attacks [4], [5].

Malware attacks are spreading exponentially in volume, complexity, and severity. In terms of volume, recently, malware proliferated to huge figures due to the availability of public hacking tools and the presence of some enabling services such as Malware-as-a-Service (MaaS). These services make it easy for even novice attackers such as script kiddies to develop and spread malware over the internet [6]. According to the AV report [7], 350,000 malware are detected every day, 7 million malware attacks were reported in 2019, and the total number of malware worldwide will exceed 1.13 billion by the end of 2020. Most of the malware files are propagated through the internet and most of the new malware is malware variants [1], [8], [9]. Malware authors increase the complexity of the previous malware files and generate new variants to evade detection. Many malware variants are generated using sophisticated tools including code-reuse techniques, artificial intelligent-based tools, obfuscation techniques, encryption, and packing among many others are used by malware authors to hide the malicious characteristics of the malware and make the analysis difficult [9], [10]. Malware variants are ranked as the major emerging threat faced by internet security [11]. Therefore, detecting malware variants is crucial to safeguards cyberspace users and reduce the number of victims of cyber-attacks [5].

Malware analysis can be performed using either a static or dynamic analysis approach. In static analysis, the malware features are extracted from the portable executable file (e.g., EXE or DLL file in the Windows operating system) without executing the malware file. However, static analysis is vulnerable to code obfuscation, packing, and malware authors. That is, malware authors use obfuscation and packing techniques, or more advanced approaches such as polymorphism or metamorphism to generate malware variants to evade detection [12]. Meanwhile, in dynamic malware analysis, behavioral features are extracted by executing the malware and monitoring its interaction with the computer system. Unlike the limitation of static analysis in the detection of obfuscated malware and malware variants, in the case of dynamic malware analysis, it is difficult for the malware author to hide the actual malicious intent of the malware variant. That is, the newly emerging malware variants still have the same malicious behavior as the original malware [1]. Accordingly, many researchers use a dynamic analysis approach to extract the behavioral characteristics of the malware variants [9], [10], [12]–[14]. Some researchers [14] combine the features extracted from both static and dynamic analysis to complement each other's limitations. More specifically, to identify the malware that exploits the vulnerability of the dynamic analysis environment against anti-malware analyst techniques.

Two approaches have been used to detect malware files in the literature, namely, signature-based and behavioral-based [39]. Signature-based detection aims at detecting malware files by matching the file signature (hash value) of the subject malware file with one stored in the malware hash database. Although such an approach can thwart many malware files, it is ineffective for new malware variants or zero-day attack types. These methods change the binary of the malware, and thus its hash, but leave its behavior unmodified. Behavioral-based malware detection has been extensively studied in the literature [13], [15]. Machine learning techniques were used to train a model based on features collected from dynamic analysis. Although such approaches show their effectiveness in detecting malware variants comparing with the static analysis-based features, existing behavioral-based solutions suffer from two main limitations. First, it needs domain experts and feature engineering expertise for extraction and selection of the significant features which is time-consuming and the extracted features may not perfect due to the limitation of human capabilities and the ever-changing malware variants nature. The deep learning approach, which has become the recent trend for constructing malware detection models, can extract the features automatically and more effectively than the human crafted-based features if enough data were introduced to the model for learning [1]. Second, existing machine learning-based approaches including deep learning assume a static relationship between the input features and the output class. That is, the mapping between the input and output is considered stationary which does not hold for the evolved nature of malware and malware variants. The detection model becomes obsolete and the weights of the features drifted gradually with the malware variants and suddenly drifted with new types of malware. Such a problem renders existing solutions that were developed for malware detection ineffective for detecting future malware and malware variants.

Some existing approaches suggested overcoming such a problem by retraining the model from scratch once new malware samples are introduced. However, such an updating method has two main limitations. With the huge amount of malware files research is needed to identify when such a model should be retrained. Late training left the system vulnerable to new attacks while early retraining is inefficient way due to the enormous amounts of malware files scanned every day. Thus, existing malware detection solutions are ineffective to detect malware variants due to the inconsideration of the concept drift caused by the malware variants. Therefore, an effective and efficient updating strategy of the malware detection model is needed to improve detection performance.

To this end, this study proposes an Adaptive Behavioral-Based Incremental Batch Learning Malware Variants Detection Model using Concept Drift Detection and Sequential Deep Learning (AIBL-MVD). The proposed model continuously monitors the concept drift and accordingly is updated using a dynamic batch-size buffer containing behavioral features extracted from the API calls sequence. The model is updated when the concept drifts only to avoid unnecessary updates that increase the model complexity and affect the detection performance.

Our work makes the following original contributions:

1) An adaptive batch window size incremental deep learning model has been designed and developed for detecting malware variants. The detection model is incrementally updated based on concept drift detection and API traces extracted from both new and old malware samples to avoid the forgetting behavior of the new model.
2) An approach based on the sequential deep learning model is proposed to extract the representative hidden features automatically according to a given malware variant API's sequence. Sequential deep learning can effectively learn to recognize the behavioral patterns of the malware samples.
3) A concept drift technique has been designed and developed using the statistical process control method to monitor and detect the concept drift utilizing the classification error rate. That is, once concept drift is detected, the model adapts to the change by performing incremental learning without forgetting its existing knowledge.
4) Extensive experiments were conducted to evaluate and validate the proposed model. The results show that the proposed model achieves an average of 99.41% detection accuracy and a lower updating frequency of 1.35 times per month.

The rest of the paper is organized as follows. The related work is presented in Section II. The proposed adaptive behavioral-based malware variants detection model is described in Section III. The experimental setup is described in Section IV. Section V presents the results and discussion. Section VI concludes this study.

II. RELATED WORK

The fight against malware seems to be a never-ending and cyclical arms race: as security analysts and researchers improve their defenses, malware developers continue to innovate, find new infection vectors and enhance their obfuscation and anti-malware analysis techniques [2]. There are two approaches to analyze the malware samples in the literature: static or dynamic. In the static approach, the malware features are extracted from the physical structure of the malware files such as the format of the malware binary file such as the portable executable file (or EXE files and DLL files in Microsoft Windows platforms). Examples of static features that are extracted from malware samples include the operation codes [11], the imports [16], the strings [17], function call graph [18] among many others. Although statistical-based features play important role in identifying malware, such features are ineffective for detecting malware variants where the malware authors use obfuscation techniques to evade detection.

Dynamic analysis, on the other hand, focuses on extracting the behavioral features. The analysis is conducted while the malware is running and its behavior is recorded. Many researchers argue that dynamic malware analysis is more effective than static analysis for detecting malware variants. The idea is that malware authors can use polymorphic and metamorphic techniques to mutate the code of the malware samples to circumvent the detection yet the malicious behavior is difficult to hide. That is, the original functionality of the malware is intact. Accordingly, dynamic analysis is the typical approach for malware variants detection. A wide range of features can be extracted from dynamic analysis including API call sequence [16], [19], network-based behavior [20], [21], file access control behavior [22], memory structure [23], CPU registers values [24], and Windows registry keys [25].

In both dynamic and static analysis, malware analysts use the extracted features to construct an automated solution to identify malicious software. Many techniques have been used for constructing malware classifiers such as machine learning, statistical and rule-based techniques [2], [6], [23]. Machine learning techniques were the most widely used in the literature. Two approaches were commonly used in the literature to construct the detection model: signature-based or behavioral-based. In the signature-based approach, a signature is created for each malware file using cryptographic algorithms such as hashing programs such as the Message-Digest Algorithm 5 (MD5) and the Secure Hash Algorithm 1 (SHA-1). The detection is performed by matching the known signature or hash of malware files with the hash of the new programs. Traditional anti-virus programs maintain a database that contains signatures of all known malware. The signature-based approach is suitable for detecting known malware but it is ineffective for new malware or malware variants. Some researchers construct even more advanced signatures that can be generalized for a group of malware. That is, utilizing features extracted from static analysis, a common attack pattern or attack signature is extracted to identify a group of malware that have similar characteristics. Such an approach is more effective and efficient than the single signature; however, it also fails to identify the new attacks and malware variants. In the behavioral-based detection approach, both static and dynamic features were used to construct a model that can distinguish between benign and malicious programs, however, the model constructed using dynamic analysis is more effective for detecting malware variants due to the difficulties of hiding malicious behavior of the malicious program in dynamic analysis.

Al-rimy, *et al.* [4] introduced an incremental bagging method to reflect the progression of the malware files in the early stages of their execution. Then, the most informative features for each subspace were selected to build a pool of base classifiers. Finally, the detection of malware was conducted using a majority voting strategy. This method shows its effectiveness in the detection of ransomware samples in their early stages. However, the training was conducted offline where the unknown malware and malware variants were not considered. Lee, *et al.* [26] used the batch learning method to update a malware classifier constructed using the Support Virtual Machine (SVM) technique. However, it is not clear how and when the model was constructed, trained, and updated. Nevertheless, retrain the model from scratch is the main drawback of their work.

Concept drift handling was intensively studied in the literature in many fields including and not limited to fraud detection, anomaly detection, spam detection; however, surprisingly, the impact of such a problem has not yet been investigated in depth in the malware analysis field. Malware authors continuously try to drift the concept to evade detection by creating malware variants. Singh, *et al.* [27] propose a technique for tracking concept drifts using static features and variable sliding windows. The cosine similarity was used to measure the temporal change of summarized features extracted from sample source codes. These summarized features were called meta-features which include the n-gram unique terms and the terms with the highest frequency. The samples were ordered based on their timestamps to reflect the temporal change in the feature's changes. The idea was that the similarity will degrade over time. However, the proposed technique has three main drawbacks. First, the study considered static features only which is subject to obfuscation by malware authors. That is, the actual malware features may not be extracted. Besides, the study was focused on particular malware families where malware families are evolving and new families occur. Second, no online training was proposed. All the studied classifiers were constructed offline. Third, the study is limited to three types of malware families which may not be generalizable to other families.

An ensemble learning-based malware mitigation algorithm was proposed by Wang, *et al.* [28] to demonstrate the concept drift problem in malware analysis. The statistical p-value was introduced which combines the vertical life-cycle algorithm with the horizontal traffic similarity algorithm for correlation learning and gradual concept drift detection. However, the study focuses on extracting trends of single concept drift in a particular malware attack which may not be generalizable for other types of malware. Also, the detection model was constructed using offline training, thus no model update was considered once a drift is detected.

A two-tier architecture malware detection was proposed by Yan, *et al.* [29] in which an incremental learning method was devised to update the model online. The model consists of two layers one for premilitary detection using a random forest algorithm and the second layer for enhancing the detection by training a classifier based on a dataset constructed from the wrongly classified samples or the samples with low detection confidence in the first layer. A random forest algorithm was also used for training in the second layer. The model is updated by retraining the two layers from scratch utilizing the new samples and the final model is constructed by combining the decision trees that achieve high classification accuracy. There are two main shortcomings of this model as follows. The power of diversity in that random forest may be violated when some trees that have important features are deleted during the updating phase. In most cases, the malware datasets are imbalanced thus the trees which contain the minority may be deleted which may cause a catastrophic forgetting problem. Also, the update is conducted using batch training where the time of updating will last a long time, meaning the model will not be updated for a long time. Moreover, the model is blindly updated without detecting the concept drift which is unnecessary and may degrade the detection accuracy.

Dai, *et al.* [30] proposed malware detection based on ensemble learning. Multiple features were used to construct the ensemble model. API call sequences were combined with hardware features extracted from the performance counter and memory dump. The random forest algorithm was integrated with the multilayer perceptron to train the proposed model. Although this model shows its advantage compared to the other tested models, the dynamic change in malware variants features has not been considered as the model will become absolute over time. Table 1 depicts the drawbacks of the existing related work.

To sum up, most of the existing malware detection models were constructed based on statistical estimation with the underlying assumption of a stationary population of the malware data. The relation mapping between input features and the class label was always assumed static which does not hold for malware samples. The malware authors continuously change their malware structures to evade detection which renders the malware population nonstationary i.e., changes over time. The nonstationary population is called concept drift in machine learning [31]. Therefore, the mapping between inputs and outputs should be updated as new malware samples or variants are introduced or whenever the relation change. However, concept drift mitigation in the malware domain was not investigated in-depth in the literature. Most of the existing solutions are either based on one-time training or based on model retraining which is neither effective nor efficient for the ever-evolved malware. Similarly, instance-based incremental learning will suffer from updating overhead and catastrophic forgetting. Thus, in this study, an Adaptive Incremental Batch Deep Learning model for detecting malware variants (AIBL-MVD) based on the API call sequence to accommodate the new malware variants is introduced. Unlike previous models which assume a static relationship of class features and stationary distribution of feature representations, the proposed model can adapt to the change in malware characteristics by firstly tracking the malware behavior in terms of

TABLE 1. Issues found in the related work.

Author	Features	Method	Issues
Al-rimy, et al. [4]	API calls – Dynamic Analysis	Incremental Bagging with Random Forest	• Concept drift was not considered. • It can't cope with malware variants and zero-day attacks due to offline training.
Lee, et al. [26]	API calls – Dynamic Analysis	Support Vector Machine (SVM)	• Updating through retraining from scratch
Singh, et al. [27]	API functions – Static Analysis	Relative temporal similarity - cosine similarity	• Static analysis can't effectively reveal the actual malware behavior • Offline model training • Limited to specific families
Wang, et al. [28]	Network Traffic– Dynamic Analysis	The statistical P-value	• Assume the Concept drift has a stationery trend • No model update was proposed • Limited to botnet detection only
Yan, et al. [29]	Network Traffic– Dynamic Analysis	Incremental learning with Random Forest	• Regular retraining from scratch contains the concept drift. • High false alarms due to the catastrophic forgetting problem.
Dai, et al. [30]	API calls + Memory Dump + Performance Counters	Ensemble Learning by integrating Random Forest and Multilayer Perceptron	• Assume static relationship between input features and class label • No model update is proposed

its API call sequence. Such features restrict malware authors' ability to obfuscate malware behavior. The evolved change in malware behavior is also captured and the necessary model updating is conducted through monitoring the concept drift and accordingly update the model through incremental learning. If the current concept deviates much from the previous one, then the model is incrementally updated. Two types of updates are conducted: a minor and major update. In the minor update, a small batch size is used while in the major update the larger batch size is used. The larger batch is collected from new and old malware samples to include the new malware variants features and reduce the problem of the catastrophic forgetting of incremental learning. The detailed methodology conducted for constructing the proposed model is discussed in the following section.

III. THE PROPOSED AIBL-MVD MODEL

In this section, the proposed Adaptive Incremental Batch Deep Learning model for detecting malware variants (AIBL-MVD) model is presented and discussed. The AIBL-MVD model has been designed in three phases, namely, base classifier construction, online operation, and incremental learning phase. Figure 1 presents the architecture of the proposed model and the colored arrows show the flow of each phase. In the first phase, the base model is trained using sequential deep learning. The second phase is the classification and monitoring phase. The third phase is the detection of concept drifts and the incremental batch learning phase. In Figure 1, the black, red, and blue solid arrows refer to offline model construction, the online classification, and incremental learning, respectively, while the dashed arrows refer to the base classifier that is constructed in the offline phase.

A. BASE CLASSIFIER CONSTRUCTION PHASE

This phase consists of five main steps as follows. Feature extraction, features representation, features selection, imbalanced data resampling,and training the base classifier.

1) FEATURES EXTRACTION METHOD

Malware features can be extracted through either static analysis or dynamic analysis (or through a combination of both types of analysis). Because the representative malicious features that are extracted from the static analysis can be obfuscated. In this study, the features are extracted through dynamic analysis. The idea behind selecting dynamic analysis is that the behavior of the malware variant does not change much compared with the original malware behavior if the malicious payload is executed. Dynamic analysis methods are effective for detecting obfuscated malware because it is difficult for malware authors to hide the actual intent of the program. During the analysis, several types of data can be collected such as API calls, memory, CPU registers, processes, file system, registry, and network traffic. The API calls sequences are the most type of data that have been used for monitoring malware behaviors in the previous malware analysis. Therefore, the malware sample, usually a Portable Executable file (or the.exe file), is firstly executed in a controlled environment to obtain its real behavior. The API call sequences invoked by malware during the execution are used to represent malware behavior.

The API calls of each malware are ordered sequentially according to their appearance during the execution and used to represent malware behavior. Because the API calls can be used by either malware and benign samples, short sequences of the API calls are extracted using the n-gram model with $n = 2$ to increase the distinguishability between benign and

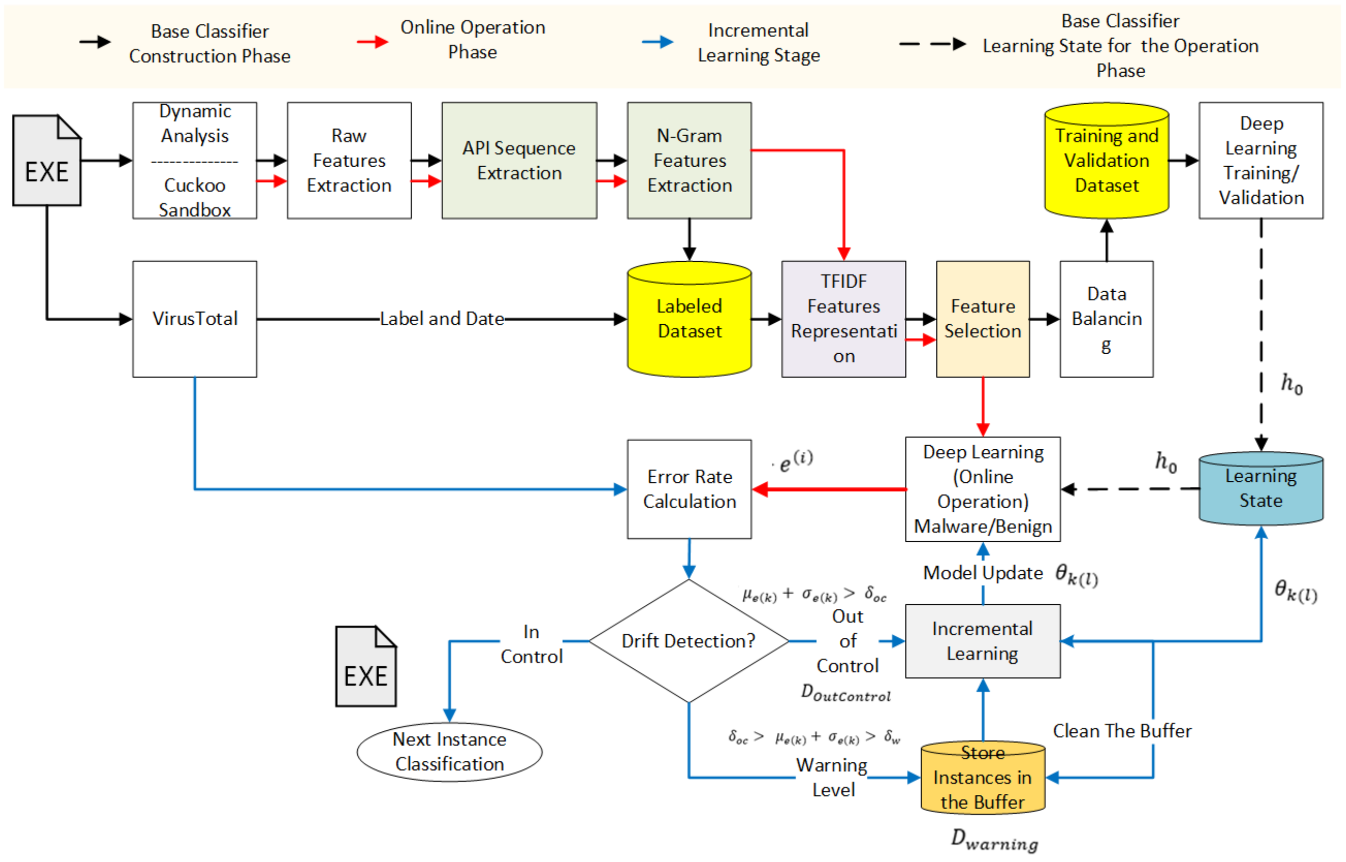
Base Classifier Construction Phase
Online Operation Phase
Incremental Learning Stage
Base Classifier Learning State for the Operation Phase
EXE
Dynamic Analysis
Cuckoo Sandbox
Raw Features Extraction
API Sequence Extraction
N-Gram Features Extraction
Training and Validation Dataset
Deep Learning Training/ Validation
VirusTotal
Label and Date
Labeled Dataset
TFIDF Features Representation
Feature Selection
Data Balancing
h_0
Error Rate Calculation
$e^{(i)}$
Deep Learning (Online Operation) Malware/Benign
Learning State
Model Update $\theta_{k(l)}$
$\theta_{k(l)}$
$\mu_{e(k)} + \sigma_{e(k)} > \delta_{oc}$
Out of Control
$D_{OutControl}$
Drift Detection?
In Control
EXE
Incremental Learning
Next Instance Classification
$\delta_{oc} > \mu_{e(k)} + \sigma_{e(k)} > \delta_w$
Warning Level
Clean The Buffer
Store Instances in the Buffer
$D_{warning}$

FIGURE 1. The architecture of the proposed malware variants detection model (AIBL-MVD).

malware files. Thus, the features vector consists of a collection of a single API call and API call sequence represented by two API calls that are occurred subsequently.

2) FEATURES REPRESENTATION TECHNIQUE

As the features extracted from binary files are in a textual form which are the names of the API call functions, the textual data must be converted to numerical values to be ready for pattern extraction and classification. Therefore, the Term Frequency - the Inverse Document Frequency technique (TF-IDF) is used to convert the textual data to numerical representation. TF-IDF is the statistical measure used to calculate the cross-entropy of the features between the malware sample and the population. Term frequency is the number of times that an API-based feature occurs in a malware trace file (or benign trace file). Meanwhile, the inverse documents frequency is the inverse number of times an API call function occurs across the entire dataset samples. IDF measures the importance of the features in the dataset by increasing the weights of the rare API call in the datasets and diminishing the API call that frequently occurred in the dataset. Accordingly, each API sequence was used as the features while their corresponding TF-IDFs values were used as numerical representations.

$$w\,(api_s) = tf\,(api_s)\,.log\frac{N}{df\,(api_s)} \tag{1}$$

where $w\,(api_s)$ is the TF-IDF value of the API sequence api_s, $tf\,(api_s)$ is the term frequency, N is the number of samples in the dataset, and $df\,(api_s)$ is the document frequency.

3) FEATURES SELECTION TECHNIQUE

The purpose of the feature selection is to reduce the dimensionality of the feature space. Although deep learning models can extract the hidden malware features, online learning can tend to consume the resources of the host machine leading to undesirable effects on the performance such as power and memory consumption problems. Therefore, it is crucial to remove redundant and irrelevant features and select only the most significant features. In this study, the eXtreme Gradient Boosting (XGBoost) algorithm [32] has been used to score the features according to their importance in a given classification task. XGBoost is a tree-based ensemble learning algorithm that uses Gini importance to measure the reduction of node impurity. Gini importance, also known as the Mean Decrease in Impurity (MDI) evaluates the feature based on the sum of the number of splits that include the feature across all trees. The optimal features were selected by the trade-off of the classification performance and the number of the selected features. Both high performance and fewer features are desirable. When the classification performance decreases, more features are included until the performance becomes stable.

4) IMBALANCED DATA RESAMPLING METHOD

One of the important issues of malware classification is the class imbalance problem [12]. Most of the available datasets are biased to malware samples due to the availability of huge malware datasets samples and the lack of existence of the gold standard benign dataset [2]. In this study, the minor class is identified in the offline and online dataset, and accordingly, the minor class (the benign samples) is resampled using the synthetic minority oversampling technique SMOTE [33]. The artificial instances created by the oversampling lead to improve classification accuracy. According to Fitkov-Norris and Folorunso [34], SMOTE synthetic oversampling expands the problem space and accordingly offers better performance than the original data.

5) CONSTRUCTION OF THE BASE CLASSIFIER (OFFLINE OPERATIONS)

The base classifier has been constructed offline using the deep sequential model. A sequential model was used to capture the ever-evolving nature of the malware variants. Sequential deep learning was selected as the base classifier because it can effectively learn to recognize the behavioral patterns of the malware samples. It also can effectively extract the representative hidden features automatically according to a given malware variant API's sequence. The date when the malware sample occurred is firstly identified and included in the training set, as the malware file timestamp that is available in the header of the binary file (PE header) may not be accurate or obfuscated by the malware author. The date when the malware was first seen is obtained from the virustotal web portal (https://www.virustotal.com/). The malware sample is submitted to the virustotal web portal and the first seen is extracted and included in the dataset. Then, the malware files were sorted according to their occurrence date in the dataset. Then, the dataset is divided into two subsets. The first subset was used to construct the base classifier while the second subset was used for online operation (for the incremental learning). The first subset contains the malware that occurred within six months of the first year in the dataset. This subset (the oldest subset) was further divided into two sets training and testing sets. The training set was used to build the base classifier and the testing set was used to evaluate the classifier accuracy. The learning state is frozen to continue learning the model if the new malware samples cause concept drift. The base classifier is then trained based on the data collected from the oldest subset.

$$D_{train} = \left\{\left(x^{(i)}, y^{(i)}\right) \mid i \in \{1, \ldots, j\}\right\} \tag{2}$$

$$h_0 = train\left(D_{train}\right) \tag{3}$$

The trained model h_0, its training state $\{\ell, \omega\}_0$, and testing error summary $\{\mu_e, \sigma_e\}_0$ is kept to be used for the online classification and monitoring phase.

B. CLASSIFICATION AND MONITORING PHASE (ONLINE OPERATIONS)

After training the base classifier h_0, the test is conducted to create a baseline to be used in the concept drift detection phase. Let D_{test} be the test data as follows.

$$D_{test} = \left\{\left(x^{(i)}, y^{(i)}\right) \mid i \in \{1, \ldots, k\}\right\} \tag{4}$$

The predicted class $\hat{y}^{(i)}$ can be obtained as the following.

$$\forall x^{(i)} \in D_{test} : \hat{y}^{(i)} = h_0\left(x^{(i)}\right) \tag{5}$$

The testing error $e^{(i)}$ can be calculated as the following.

$$\forall i \in \{1, \ldots, k\}\, e^{(i)} = \hat{y}^{(i)} - y^{(i)} : y \in \{0, 1\} \tag{6}$$

The baseline concept characteristics of the model h_0 can be created by calculating the $\mu_{e(0)}$, and $\sigma_{e(0)}$ which are the average and standard deviation of testing classification error e respectively as follows.

$$\mu_{e(0)} = \frac{\sum_{i=1}^{k} e^{(i)}}{k}, \quad \sigma_{e(0)} = \sqrt{\frac{\sum_{i=1}^{k} \left(e^{(i)}\right)^2}{k-1}} \tag{7}$$

C. CONSTRUCTION OF THE INCREMENTAL MODEL (ONLINE LEARNING)

In the absence of concept drift, the distribution $p(y|x)$ learned in the offline operation remains stationary. Thus, the learned relationship mapping between input x and class output y is fixed and the classifier that is trained on an old dataset continues relevant for new samples. However, with the concept drift occurring in the malware due to the malware variants, the performance of the base classifier degrades as new malware variants are introduced. The base classifier must learn how to classify the emerged malware variants. Neither batch learning nor instance-based incremental learning can address the problem in the malware variants domain. Malware variants can be a mutex of older ones. With such recurrent behavior of Malware variants, both old and new malware characteristics can present due to the mutation, and parents' genes can reoccur. Unfortunately, the fixed batch-based incremental learning [26] may need a long time to be collected, which depends on the size of the batch, and the model must be constructed from scratch which is an inefficient updating approach. In contrast, instance-based incremental learning [29] can be timely updated. However, incremental learning suffers from the catastrophic forgetting problem. The learned mapping between the input feature and class label drifts and the model becomes biased to new malware trends. Therefore, in this study, incremental learning without forgetting strategy is developed to address such an issue. The learning is conducted online based on the proposed concept drift detection techniques described in the following sections.

1) CONCEPT DRIFT DETECTION

The concept drift is defined as the phenomenon when the statistical properties of the dependent variable change over time [35]. This is the case in malware instances, where the representative features of the malware samples change with time due to the use of code obfuscations techniques by malware authors. Therefore, the Statistical Process Control

(SPC) technique is used to detect the concept of drift and activate incremental learning. The actual class label in binary classification is either zero or one. However, the output of the classification is a value between zero and one. When the value is approaching zero, then the predicted sample is assumed benign. In contrast, when the prediction value approaches one, the predicted sample is considered malware. In the case of an incorrect prediction, the error is calculated based on the difference between the actual class label and the predicted one.

Three states can be defined based on the distribution of the classification error: in-control state (or control level), warning level state, and out of control state. In the in-control state, the average error rate should remain within the known error distribution. When the mean of the prediction error strays outside the known distribution, then this state is called the warning level. In the warning level state, the malware samples that occur in this period are collected and stored in a small buffer size and used for incremental learning. The incremental learning is continuous with small window sizes and small batches until the error decreases as it was at the control level. Once the mean of the error rate deviates much from the reference error that is obtained before the warning level then this state is called out of control. In this case, a larger batch size is used for batch incremental modeling. The batch contains the misclassified samples that arose during the warning level until the current instance when the out-of-control error occurs. The model is incremented using a large batch, in this case, to avoid the bias that may occur due to small size incremental batch learning. If the out-of-control alarm occurred, an old sample collected from previously seen malware is mixed with the new batch to avoid the catastrophic forgetting problem of incremental learning.

To detect the concept drift, let $e^{(i)}$ be a variable that contains the classification error of instance i and μ_e, and σ_e is its incremental average and the standard deviation is taken within sliding window w. Two types of change are considered: light deviation (warning level) and heavy deviation (out of control level). The warning level is detected according to the following equations.

First, the incremental average $\mu_{e(k)}$ for each time epoch, k is computed as follows.

$$\mu_{e(k)} \leftarrow \mu_{e(k-1)} + \frac{e^{(i)} - \mu_{e(k-1)}}{n} \sum e^{(i)} \tag{8}$$

Then, the incremental variance $S_{e(k)}$ for each time epoch, k is computed as follows.

$$S_{e(k)} \leftarrow S_{e(k-1)} + (e^{(i)} - \mu_{e(k-1)})(e^{(i)} - \mu_{e(k-1)} \tag{9}$$

Next, the standard deviation $\sigma_{e(k)}$ is calculated for each time epoch k as follows.

$$\sigma_{e(k)} \leftarrow \sqrt{\frac{S_{e(k)}}{n}} \tag{10}$$

Thus, the warning level δ_w and out of control level δ_{oc} thresholds are calculated as the following equations, respectively.

$$\delta_w \leftarrow \mu_{e(0)} + \propto \times \sigma_{e(k)}, \quad \delta_{oc} \leftarrow \mu_{e(0)} + \beta \times \sigma_{e(k)} \tag{11}$$

where $\propto$ is set to two and β is set to three. The warning level mode is raised if $\delta_{oc} > \mu_{e(k)} + \sigma_{e(k)} > \delta_w$ while the out-of-control level is raised if $\mu_{e(k)} + \sigma_{e(k)} > \delta_{oc}$ is satisfied or the buffer $D_{warning}$ specified for warning level is full. The case where $\mu_{e(k)} + \sigma_{e(k)} < \delta_w$ is called in control level. The incremental batching learning in both warning and out-of-control levels is explained in the following section.

2) INCREMENTAL BATCH LEARNING

Let $D_{warning}$ be a dataset that contains the features collected during the warning level of malware detection online operation, L is the set of the layers in the model h_0 and $x^{(i)}$ is the features vector of sample i and $y^{(i)}$is the class label of that sample as it is assigned after manual analysis. Let $\hat{y}^{(i)}$ be the predicted class of the new samples as predicted by the last trained/incremented model h_0. Let $\theta_{k(l)}$ is a vector that contains the weights of the neurons in each layer (l) and learning rate (α). Then, in the warning level (minor update), the weights $\theta_{k(l)}$ is updated according to the following formula.

$$\forall x^{(i)} \in D_{warning} \quad and\ l \in L\ do$$
$$\theta_{k(l)} \leftarrow \theta_{k-1} - \alpha(e^{(i)}).x^{(i)} \tag{12}$$

Similarly, in the out-of-control level (major update), the weights $\theta_{k(l)}$ is updated is calculated using the same approach. The only difference is that the batch D_{oc} is used instead of $D_{warning}$. D_{oc} contains all the samples collected during the warning level $D_{warning}$ as well as samples selected from old seen data D_{old}. The reason for using the old data is to reduce the impact of the forgetting behavior of incremental learning. Algorithm 1 in Figure 2 shows the pseudocode of the proposed concept detection and incremental batch learning techniques.

IV. PERFORMANCE EVALUATION

In this section, the process of evaluation of the proposed model is described. We describe the experimental environment setup, the dataset used, and the performance metrics.

A. EXPERIMENTAL SETUP

Malware behavior in terms of API calls and sequences are extracted in runtime from a dynamic analysis environment. Because running the malware can bring danger to the analyst operating system and network, an isolated and controlled virtual environment was constructed for this study. All the experiments in this study were carried out in a sandbox environment with host computer CPU Intel (R) Core i7 @ 3.20 GH, the RAM is 16.0 GB and the host operating system is Linux Ubuntu 18.04, and Windows 7 guest operating system was used as a victim machine. Sandboxes are tools that

Algorithm 1: The proposed Concept Drift-based Incremental Learning Algorithm

```
Initialization: $h_0$ last trained classifier, $x^{(i)}$ Extracted features, $\mu_{e(0)}$ average of testing classification error, $\sigma_{e(i)}$ deviation of testing classification error, $D_{old}$ old samples
1: While True Do
2: $\forall x^{(i)} \in D_{recent}$ Classify $x^{(i)}$ using $h_0$, $\hat{y}^{(i)} \leftarrow h_0(x^{(i)})$
3: Get actual class $y^{(i)}$ and calculate the error $e^{(i)}, e^{(i)} \leftarrow y^{(i)} - \hat{y}^{(i)}$
//Concept − Drift Detection Using SCS Technique
4: Calculate the incremental average
       $\mu_{e(k)} \leftarrow \mu_{e(k-1)} + \frac{e^{(i)} - \mu_{e(k-1)}}{n} \sum e^{(i)},$
5: Calculate the incremental variance
       $S_{e(k)} \leftarrow S_{e(k-1)} + (e^{(i)} - \mu_{e(k-1)})(e^{(i)} - \mu_{e(k-1)})$
6: Calculate the standar deviation
       $\sigma_{e(k)} \leftarrow \sqrt{\frac{S_{e(k)}}{n}}$
7: Let $\delta_w \leftarrow \mu_{e(0)} + \propto \times \sigma_{e(k)}$
8: Let $\delta_{oc} \leftarrow \mu_{e(0)} + \beta \times \sigma_{e(k)}$
// Warning level
9: if $\delta_{oc} > \mu_{e(k)} + \sigma_{e(k)} > \delta_w$ then
10:     $D_{warning} \xleftarrow{append} (x^{(i)}, \hat{y}^{(i)})$
11:     $\forall x^{(i)} \in D_{recent}$ and $l \in L : x^{(i)}$ do
12:         $\theta_{k(l)} \leftarrow \theta_{k-1} - \alpha(e^{(i)}).\, x^{(i)}$
13:         Store $learning_{buffer} \leftarrow \theta_{k(l)}$
14:         Update $(h_0)$
15:     $\mu_{e(0)} \leftarrow \mu_{e(k)}, S_{e(0)} \leftarrow S_{e(k)}$
// Out of Control level
16: else if $\mu_{e(k)} + \sigma_{e(k)} > \delta_{oc}$ or $Size(D_{warning}) > T$ then
17:     $D_{OutControl} \leftarrow D_{warning} \cup D_{old}$
18:     $\forall x^{(i)} \in D_{OutControl}$ and $l \in L : x^{(i)}$ do
19:         $\theta_{k(l)} \leftarrow \theta_{k-1} - \alpha(e^{(i)}).\, x^{(i)}$
20:         Store $learning_{buffer} \leftarrow \theta_{k(l)}$
21:     $\mu_{e(0)} \leftarrow \mu_{e(k)}, S_{e(0)} \leftarrow S_{e(k)}$
22:     Update $(h_0)$
23:     Reset $D_{warning}, D_{OutControl}$
24: endif
25: end
```

FIGURE 2. The proposed concept drift-based incremental learning algorithm.

are commonly used by malware analysts and researchers to conduct dynamic analysis [36], [37]. They provide a means of detecting windows APIs invoked by a malware instance at the run time in a process called API hooking and DLL injection [38]. Cuckoo sandbox tools were used with the virtual box to create an isolated controlled virtual environment for Malware analysis.

The sandbox architecture was configured according to the guidelines presented in [39]. In the virtual machine, the guest Windows 7 operating system was installed and a configured and clean slate screenshot was created. In the guest operating system where the malware samples were run, to make the guest operating system more realistic to the evasive malware sample, many applications have been installed and some dummy files and folders have been created and internet access has been allowed as well. As shown in Figure 3, malware

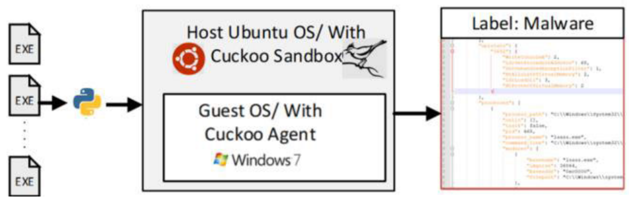

FIGURE 3. Sample files submitted to the sandbox.

and benign instances are submitted to the cuckoo sandbox in the host machine. The cuckoo agent in the guest operating system executes the submitted binary files and hooks their API calls and process. Each sample in the dataset is run one time. After the API calls and the process are hooked, they are timely dumbed to XML type trace files. The cuckoo agent sends the trace file back to the cuckoo instance in the host machine through the local network between the host and guest machine. Thus, each binary sample in the dataset has a dedicated trace file containing its API calls behavioral patterns. The API calls are used to examine the potential features of the malware. For each submitted malware or benign file for analysis, the virtual machine is restarted and the original operating system is restored to start the new analysis with a clean version of the guest operating system (clean slate). Finally, the API call sequences were extracted from the XML reports using Python programming packages.

B. DATASET DESCRIPTION

The dataset that was used in this study consists of Windows binary files belonging to malware and benign portable executable programs. The malware binary files were downloaded from the public repository Vxheaven (https://www.vxheaven.org). Vxheaven dataset is a public repository that is common used by previous malware analysis studies such as in [14], [36], [37], [40]–[42]. The malware dataset contains different types of malware families such as trojans, adware, backdoors, ransomware, viruses, and worms among many others. The Windows benign binary files were collected from the fresh installed Windows operating system. A total of 19,076 malware samples were randomly selected from the Vxheaven dataset. Thus, the used dataset in this study consists of 23,070 samples, 19,076 are malware samples and 3,994 are benign samples. All malware files were scanned using VirusTotal service which is a popular malware scanning web service to confirm its class label and extract information related to its first occurring. VirusTotal uses 72 antivirus software and a dynamic sandbox environment to scan the submitted files [36], [37], [43].

The instances of the dataset were sorted according to their created times extracted from malware binaries and confirmed by the VirusTotal web portal. The date when the malware file was first seen by the VirusTotal was considered. Then, for test purposes, the dataset was split into 11 subsets each containing the malware that occurred within 6 months of each year. Table 2 shows the datasets used in the experiment. Because

TABLE 2. Description of the used dataset.

Datasets	Year	Instances	Malware	Benign
DS	All instances	23070	19076	3994
DS1	< 2009	1864	932	932
DS2	2010-1	3092	1546	1546
DS3	2010-2	4120	2060	2060
DS4	2011-1	6604	3302	3302
DS5	2011-2	5804	2902	2902
DS6	2012-1	7832	3916	3916
DS7	2012-2	4660	2330	2330
DS8	2013-1	2564	1282	1282
DS9	2013-2	1444	722	722
DS10	2014-1	1364	682	682
DS11	2014-2	1364	547	547

the benign instances are lower than the malware instances, the benign sample has been oversampled using the synthetic minority oversampling technique SMOTE [33] and a random subset equivalent to the malware subtle size was used in each splitter dataset as shown in Table 2.

C. PERFORMANCE METRICS

Four main performance evaluation measures were used to evaluate the effectiveness of the proposed model. The detection accuracy (ACC), the false positive rate (FPR), detection rate (or recall) (DR), and F-measures (F1). The detection accuracy (ACC) is the percentage of the benign samples correctly classified to all the classified samples. The detection rate (DR) is the fraction of the malware samples that are correctly classified. The false-positive rate (FPR), is the percentage of the instances that are incorrectly classified as malware samples. F-measures (F1) is the harmonic mean and calculated as in Equation (13) where the TP is the number of malware samples that are correctly classified, FP number of benign samples that are wrongly classified, and FN number of malware samples that are wrongly classified. The performance of the proposed model was also evaluated using three additional measures, namely the average classification error, updating frequency, and the average update time. The average classification error is the root mean square error where the error is the difference between the class label and the score calculated by the activation value of the NEURON in the output layer of the proposed model. The updating frequency is the number of times the concept of drift detection invoked incremental learning. Finally, the average update time is the mean of the elapsed time during updating the model.

$$F_1 = \frac{2 \times TP}{2 \times TP + FN + FP} \tag{13}$$

V. EXPERIMENTAL RESULTS

We carried out several experiments to evaluate and validate the components of the proposed model including the feature selection, base classifier selection, and incremental learning. The API calls sequences that were extracted from the dynamic analysis and enriched by the n-gram model, were introduced to the feature extraction method to reduce the complexity of the detection model concerning the time and resources. The selected feature technique XGboost [32] which uses the Gini index to select the important features for the classification is compared with two widely used feature selection techniques for malware analysis (mutual information and chi-square feature selection methods). Figure (4) presents the results of the comparison. Several sets of features were experimented ranging from 10 best features to 200 features. The XGboost-based feature selection technique produces the best accuracy with the smallest number of features. Thus, the 100 best features suggested by XGboost were selected to train the base classifier.

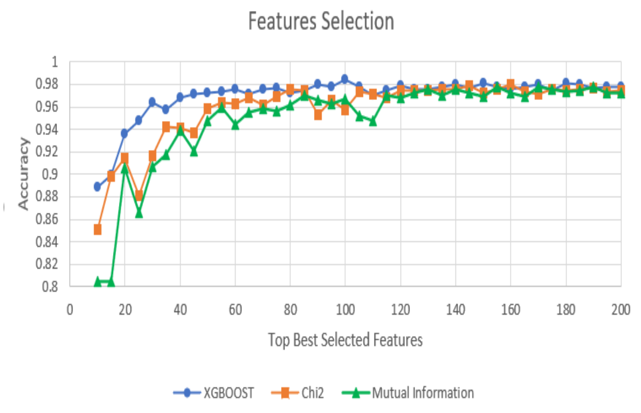

FIGURE 4. Comparison of feature selection techniques.

Another set of experiments were conducted to select the base classifier. The commonly used classifiers in the existing related works were implemented in this study for the comparison which are Support Vector Machine (SVM) [44], Naïve Bayes (NB) [45], Logistic Regression [46], Random Forest (RF) [47], XGBoost [9] and Deep Learning (DL). These classifiers were trained based on the dataset denoted by DS1 in Table 1. DS1 was divided into two subsets training and testing set. The classifiers were trained using the top 100 best features selected in the previous experiments. As shown in Figure 5, SVM, LR, XGBoost, and SDL achieved higher classification accuracy than NB and RF algorithms. Sequential Deep Learning (SDL) achieved the highest

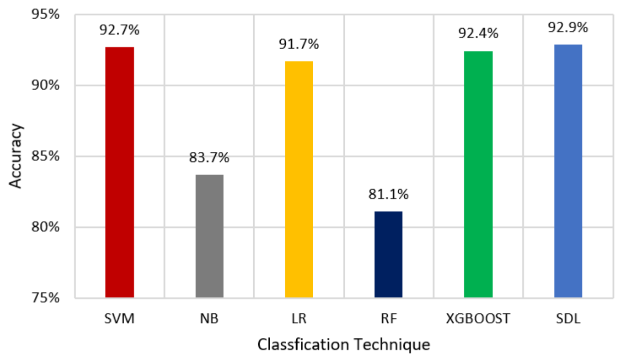

FIGURE 5. Base classifier selection.

accuracy compared with the other tested classification algorithms. However, these classifiers are vulnerable to the concept drift and their performance degrades when tested with newer versions of malware variants.

To demonstrate the effect of the concept drifts on the classification accuracy, the aforementioned classification algorithms that were trained in the previous experiments were tested using a newer version of malware namely the DS2 to DS10. Figure 6 presents the performance of the tested classifiers which were trained on old data when they were used to classify new samples. As shown in Figure (6) the performance of all trained classifiers drops as new malware data is introduced to the classifier. This because the classification algorithm assumes a static relationship between input features and class labels while it is not correct for the ever-evolving malware variants. However, when the malware author creates malware variants, the distribution of the extracted features varies compared to that of the features extracted from the original malware version causing performance degradation.

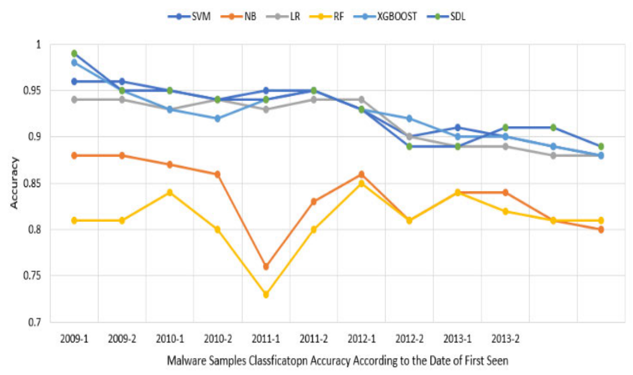

FIGURE 6. Degrade of the performance due to the concept drift problem.

To show the improvement gained by the proposed AIBL-MVD model, a comparison between different incremental learning strategies was conducted. The results were compared with the fixed batch incremental learning technique. Figures (7a), (7b), (7c), and (7d) illustrate the performance of the proposed AIBL-MVD model. The figures also show the results of the comparison between two types of incremental learning strategies, the fixed batch size incremental batch learning model (IBL), and the proposed adaptive batch size incremental model with concept drift detection (AIBL-MVD). Different fixed batch sizes have been experimented with 64, 128, 256, and 512. The proposed AIBL-MVD experimented with sliding windows size of 64 and 128 for drift detection, respectively. In the fixed batch window size, the model is updated using incremental learning when the number of misclassifications becomes equal to the fixed batch size. Meanwhile, in the proposed adaptive batch incremental learning AIBL-MVD, the model is incrementally learned when the drift is detected and the classification error rate deviates much from the mean error rate of the previous time when the model is incrementally learned.

As shown in Figure (7a), the average accuracy performance of both adaptive batch sizes with statistical processing Control based Concept Drift (SPC) with sliding window size SPC WS=128 and SPC WS=64 outperformed the implemented fixed batch buffer size IBL (BS=64, BS=128, BS=256, and BS=512). Similarly, the average error rate using these windows is lower than that of fixed batch size as shown in Figure (7b). The average updating time is varies based on the batch size as shown in Figure (7c). In terms of the updating frequency, the incremental learning with a fixed batch size has an inverse relationship with window size as shown in Figure (7d). Meanwhile, the updating frequency of the adaptive window size is varies based on the number of times the concept is drifted and the type of the drift e.g., sudden, incremental, etc. Thus, the updating frequency of the adaptive batch window size less than that of the fixed size. This is because the model is not incremented regularly as the classification error rate reaches the batch size, it is incremented only if the drift is detected.

Table 3 lists the detailed results of the proposed model AIBL-MVD (SPS WS=128) tested on the datasets that were listed in Table 2. The performance of the proposed AIBL-MVD (SPS WS=128) model in terms of the accuracy (ACC), false-positive rate (FPR) and detection rate (DR), and F measure (F1) is shown in Table 3.

TABLE 3. Detailed results of the proposed model AIBL-MVD (SPS WS=128).

Dataset	ACC	FPR	DR	F1
DS1	99.37%	0.94%	99.59%	99.33%
DS2	99.36%	0.92%	99.55%	99.32%
DS3	99.35%	0.92%	99.54%	99.31%
DS4	99.35%	0.91%	99.54%	99.32%
DS5	99.37%	0.90%	99.58%	99.34%
DS6	99.38%	0.89%	99.59%	99.35%
DS7	99.37%	0.89%	99.59%	99.35%
DS8	99.37%	0.90%	99.58%	99.34%
DS9	99.36%	0.89%	99.56%	99.33%
DS10	99.39%	0.89%	99.63%	99.40%
DS11	99.43%	0.84%	99.62%	99.44%
Average	**99.37%**	**0.90%**	**99.58%**	**99.35%**

As shown from Table 3, the overall performance in terms of F1 is stable higher than 99% with all tested datasets. Meanwhile, the false positive rate is under 1% and getting decreased when the number of updates increases. The reason for the gained improvement in the detection performance is that the proposed adaptive batch incremental learning model is based on the concept drift detection while the fixed batch blindly increments the model learning when the batch buffer is full. This makes the model biased to the new malware variants and forgets the older versions. Thus, the accuracy of

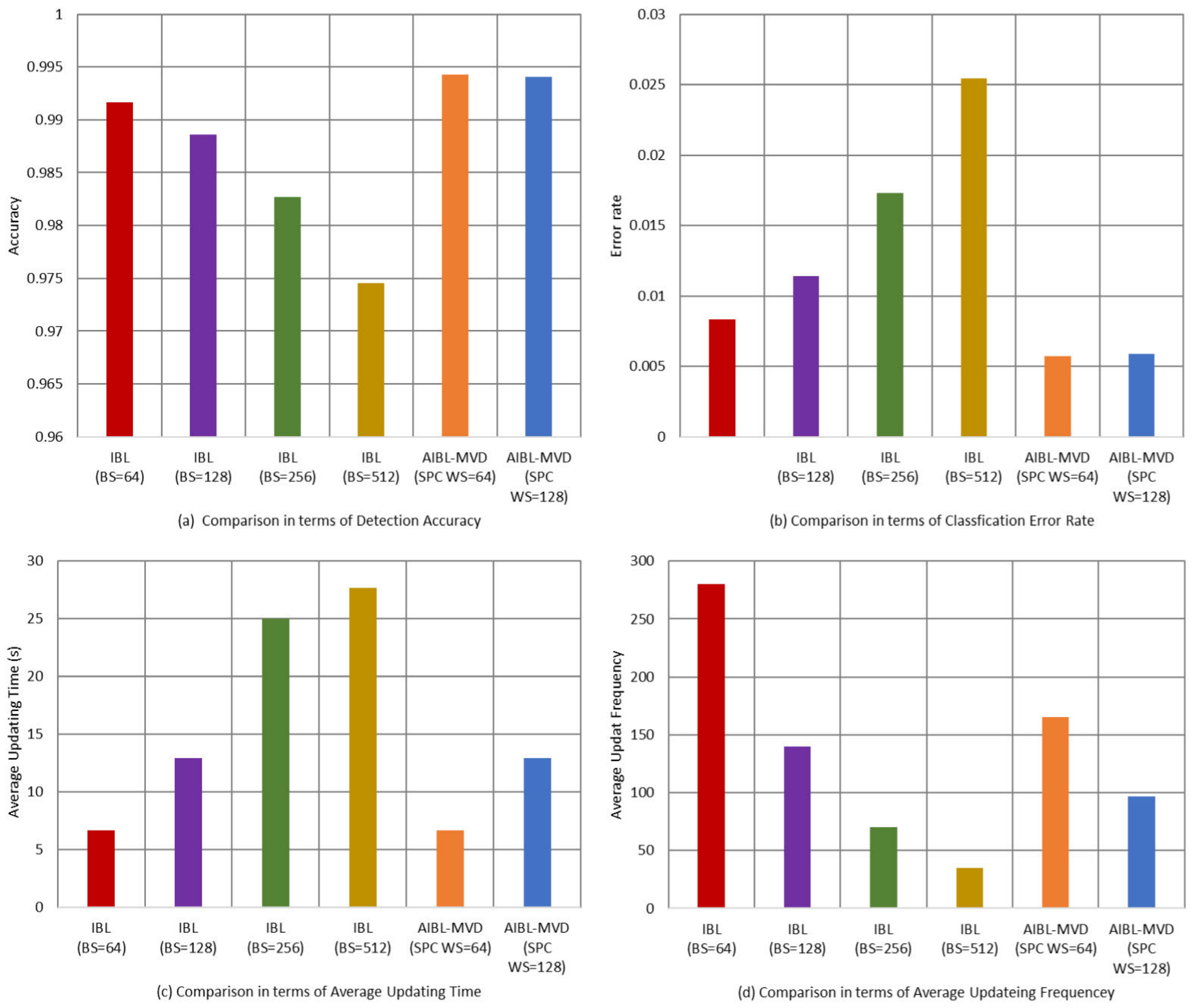

FIGURE 7. Results of comparison between the proposed adaptive batch size (AIBL-MVD) with the fixed batch size (IBL).

the model degrades with time (see Figure 6). In the case of the concept drift, the proposed model maintains the accuracy at its highest level. More particularly, the accuracy of the model will be slightly degraded during the warning level and will be boosted after updated when the batch window is full or the out-of-control state occurs. Figure (8) and Figure (9) show the detailed results of the performance of both fixed batch window size and adaptive batch window size in terms of detection accuracy in Figure (8) and classification error in Figure (9). As can be observed, in both figures the proposed concept-drift-based adaptive incremental batch performs better than the other fixed batch size incremental learning strategies.

As shown in Figure (8), the proposed ABIL-MVD with both sliding window sizes of 46 and 128 samples outperforms the fixed incremental batch with all tested incremental batch. It can also be noted from Figure (8) that as the batch size decreases the accuracy is improved while the corresponding error rate decreases as shown in Figure (9). Although this property is desirable, it is noted that the small fixed-size incremental batch increases the update frequency and disrupts the normal operation of malware detection. Compared with the adaptive batch size, the number of updates depends on the number of times concept drift is detected. That is if concept drift is detected, the model is incrementally updated otherwise no update is performed. Accordingly, the proposed ABIL-MVD is more efficient and more practical than the fixed batch size. Table 4 shows the computational complexity in terms of the times the model spends for learning (the average time needed to complete the online incremental learning). It also shows the average monthly update frequency of the model in terms of the number of updates (learning time) per month.

As shown in Table 4, the proposed AIBL-MVD, (SPC WS=128) model needs around 12.9 seconds for each

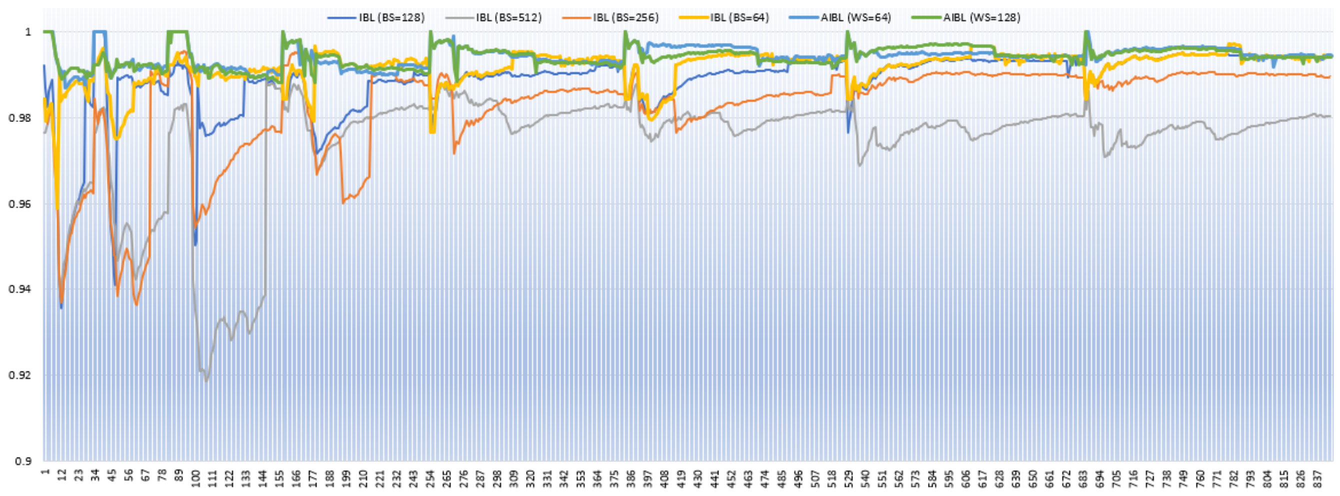

FIGURE 8. Detailed accuracy results of comparison between the proposed adaptive batch size (AIBL-MVD) with the fixed batch size (IBL).

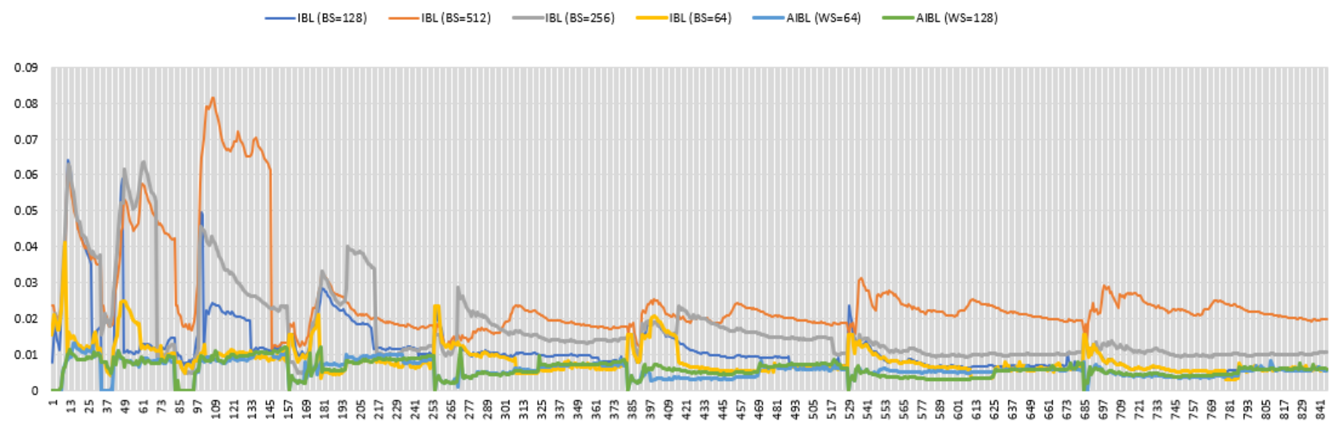

FIGURE 9. Detailed error rate results of comparison between the proposed adaptive batch size (AIBL-MVD) with the fixed batch size (IBL).

TABLE 4. Average learning time and frequency.

On Average	Learning Time (Second)	Updating Frequency (Per Month)	An update Every (x Day/s)
IBL (BS=64)	6.68	3.89	7.71
IBL (BS=128)	12.90	1.94	15.43
IBL (BS=256)	24.98	0.97	30.86
IBL (BS=512)	27.64	0.49	61.71
AIBL-MVD (SPC WS=64)	6.68	2.29	13.09
AIBL-MVD (SPC WS=128)	**12.90**	**1.35**	**22.27**

update. Compared with the other tested incremental models AIBL-MVD (SPC WS=64) achieved the lowest updating time. However, AIBL-MVD (SPC WS=64) needs 2.29 updates every month (one update every 13.09 days). That is, on average, dynamic analysis by malware analysis experts needed to be conducted every 13.09 days approximately. Meanwhile, in the case of the proposed AIBL-MVD (SPC WS=128) model, on average it needs only a single update every 22.27 days. Although the fixed batch size with a small buffer size e.g., IBL (BS=64) and IBL (BS=128) achieve similar update time with the proposed AIBL-MVD (SPC WS=64) and AIBL-MVD (SPC WS=128), the updating frequency strongly depends on the size of the batch in case of fixed batch size and the size of concept drift detection window (WS) for the proposed model. For example, on average IBL (BS=64) and IBL (BS=128) need to be updated every 7.71 days (approximately every week) and 15.43 days (every two weeks), respectively.

To demonstrate the dynamic features change due to malware variants, Figure 10 illustrates the problem of feature fading over time due to the evolved malware variants. It contains the most important features in terms of API calls sequence that were selected using the information gain feature selection technique. An API calls sequence contains the names of the API functions that have been called during the malware run time. The figures show how the weights of some features fade over time and some have fluctuant behavior.

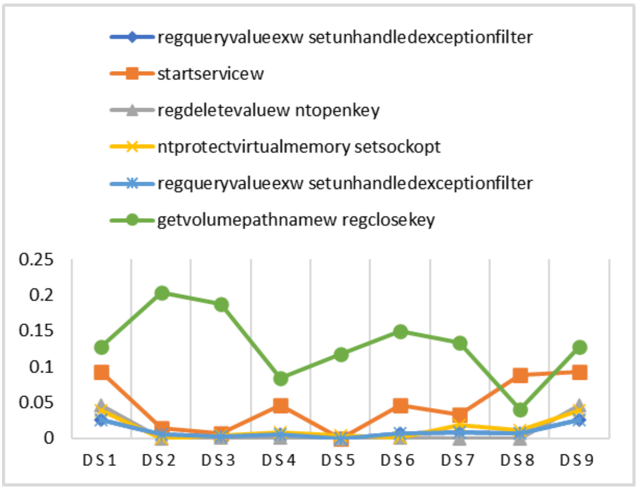

FIGURE 10. **Feature fading problem due to the use of obfuscations techniques by malware's authors.**

It can be seen from Figure 10 that the most important feature in the dataset DS1 is "GetVolumePathNameW RegCloseKey" (f55 in DS1). DS1 contains malware samples that are first seen in 2009. Meanwhile, in the dataset DS8 which contains malware shambles that was firstly shown in 2014, this feature is no longer the most important in the database. We observed that most of the top important features selected were extracted using the n-gram technique with n = 2. We have noticed that the long sequences feature which is extracted using n-gram with n > 2 is less effective than when the n = 2. This may imply that the malware authors tend to heavily change the follow of API calls to obfuscate the malware. However, a deep investigation should be carried on to study how the API call sequence change over time based on each malware family. Such a study will be the subject of our next publications. As shown in Figure 10, the feature weights are changing continuously with time. Some features have fluctuant behavior which requires that the learning model should remember the old malware patterns because they may occur again. However, traditional incremental learning gradually evades the patterns of old malware and replaces them with the new malware patterns which cause the degradation of the model detection accuracy. To avoid such a problem, the proposed model mixes the new malware samples with older ones to evade this issue.

Figures 11, 12, and 13 show the most important features which are selected using the XGboost feature selection technique. As shown in Figures 11 and 12, which use samples created in 2009 and 2010 respectively, the most important features are the feature denoted by f55 and f29 respectively, while in Figure 13 which contains samples created in 2013, the most important features were f29 and f55. The sets of features extracted in each dataset vary based on the time when the malware was first shown. This variation in the features that represent the malware cause degradation of the detection performance. This implies the importance of the proposed adaptive incremental batch model to monitor the performance and continuously adapt to this change.

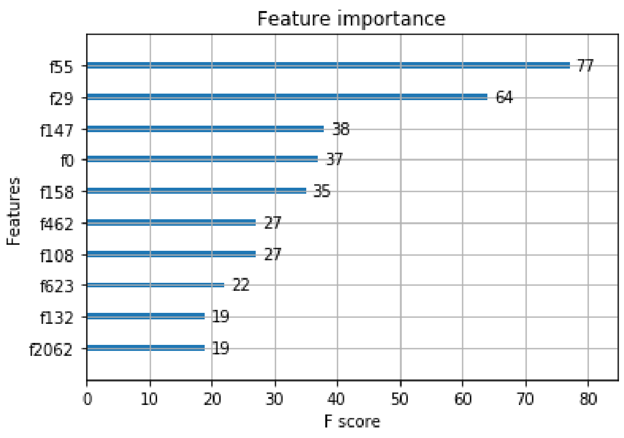

FIGURE 11. **Features extracted from DS1 (2009).**

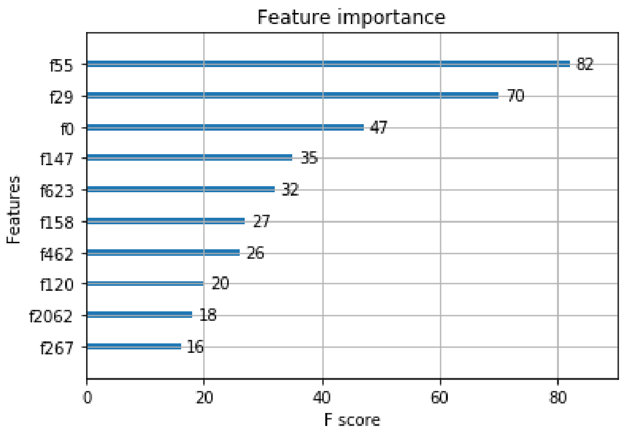

FIGURE 12. **Features extracted from DS2 (2010).**

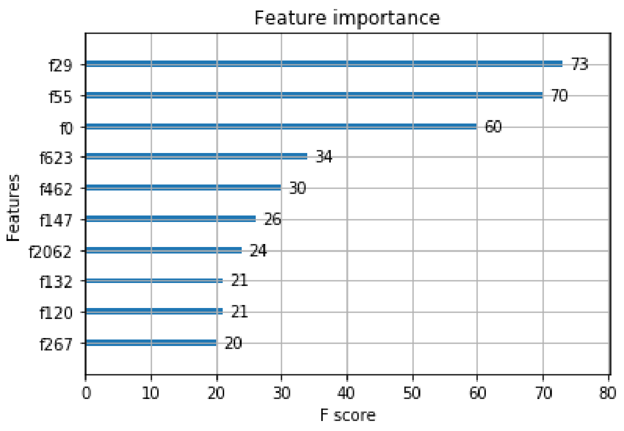

FIGURE 13. **Features extracted from DS8 (2013).**

VI. LIMITATIONS AND FUTURE WORK

This study focused on malware detection and not analysis. That is the study assumes availability of the representative malware variants feature of malware in the detection stage. Moreover, the study focusses on the malware variants that resulted from obfuscation against static analysis and does not have evasive behavior in which malware do not execute their malicious payload due to the presence of controlled environments (such as virtual machine and sandbox) or time delay (time-based evasive) or the need of user inputs. More particularly, it focuses on detecting malware variants that their behaviors do not deviate much from the original malware. Evasive malware may not execute its malicious payload if some conditions are satisfied. For instance, if an evasive malware detects any indication that it is under analysis, then it does not execute its malicious payload. Thus, the extracted API sequence for the sample will not be representative. This problem affects the detection of a zero-day attack and may have less effect on the detection of malware variants. The idea behind that the behavior of malware variant has similarities with the behavior of its origin. In addition, as the executable malware cannot hide its malicious behavior once it is in the attack mode, a malware variant can be detected using real-time malware detection such that proposed in [43], [48], [49].

Although the study tried to minimize the impact of the evasive behavior by naturalizing the operating system and make it looks like a real user machine and shade the virtual environment, such detection evasive malware behavior needs extensive research study. To analyze the evasive malware behavior and time-based malware, in future research, researchers may need to extract new features related to the malware interactions with the analysis environment. Another limitation of the proposed model is that labeled malware samples must be available right before the model incremental learning. One potential solution is to use real-time detection with unsupervised predictive analysis techniques such as the deep incremental and deep clustering algorithm [50], [51] to improve the effectiveness and the efficiency of the malware detection model.

VII. CONCLUSION

In this study, the concept drift problem case by malware variants is addressed by presenting an adaptive batch incremental deep learning model for improving the accuracy of malware variants detection. The one batch training approach used by the existing work has been replaced by an adaptive batch for online learning. The API calls were extracted through behavioral analysis and the n-gram model was used to extract the short API sequence. These extracted features were represented using the term frequency/ inverse term frequency technique. The most important features then were selected for classification to reduce the training time and model complexity. The base classifier was trained based on sequential deep learning using API call sequences extracted from an old set of malware files. Then, an adaptive batch size incremental learning technique was developed using statistical process control to detect the concept drift and trigger incremental learning. The catastrophic forgetting problem was solved by mixing the older malware samples with the new emerging versions to prevent the model from underestimating the previous knowledge learned from older malware samples. Extensive experiments were conducted to evaluate and validate the proposed model including procedures such as features selection, base model classifier selection, and incremental strategy investigation. The results showed that the adaptive batch size incremental learning using the statistical process control method achieved the best tradeoff between accuracy and efficiency in terms of updating time and frequency. The proposed adaptive batch window size-based incremental model could be applied in different domains. One limitation of the proposed model is that labeled malware samples must be available right before updating the model. Because the focus of the study was to detect malware variants due to obfuscations against static analysis, the model does not distinguish between benign software and the evasive malware behavior in which the malware stops executing its malicious payload. However, once the malicious payload is executed, the proposed model can then detect it. Currently, we are studying addressing such issues and the use of real-time unsupervised predictive analysis such as deep learning and incremental clustering algorithms to further improve the effectiveness and the efficiency of the malware variant detection model.

REFERENCES

[1] G. Xiao, J. Li, Y. Chen, and K. Li, "MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks," *J. Parallel Distrib. Comput.*, vol. 141, pp. 49–58, Jul. 2020, doi: 10.1016/j.jpdc.2020.03.012.

[2] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *J. Netw. Comput. Appl.*, vol. 153, Mar. 2020, Art. no. 102526, doi: 10.1016/j.jnca.2019.102526.

[3] X. Liu, Y. Lin, H. Li, and J. Zhang, "A novel method for malware detection on ML-based visualization technique," *Comput. Secur.*, vol. 89, Feb. 2020, Art. no. 101682, doi: 10.1016/j.cose.2019.101682.

[4] B. A. S. Al-Rimy, M. A. Maarof, and S. Z. M. Shaid, "Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions," *Comput. Secur.*, vol. 74, pp. 144–166, May 2018, doi: 10.1016/j.cose.2018.01.001.

[5] J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity," *J. Comput. Syst. Sci.*, vol. 80, no. 5, pp. 973–993, Aug. 2014, doi: 10.1016/j.jcss.2014.02.005.

[6] S. S. Chakkaravarthy, D. Sangeetha, and V. Vaidehi, "A survey on malware analysis and mitigation techniques," *Comput. Sci. Rev.*, vol. 32, pp. 1–23, May 2019, doi: 10.1016/j.cosrev.2019.01.002.

[7] AV-TEST. (2020). *Malware Statistics and Trends Report*. The Independent IT-Security Institute. Accessed: Dec. 27, 2020. [Online]. Available: https://www.av-test.org/en/statistics/malware/#:~:text=Every%20day%2C%20the%20AV%2DTEST,potentially%20unwanted%20applications%20(PUA)

[8] X. Liu, X. Du, Q. Lei, and K. Liu, "Multifamily classification of Android malware with a fuzzy strategy to resist polymorphic familial variants," *IEEE Access*, vol. 8, pp. 156900–156914, 2020.

[9] N. Kumar, S. Mukhopadhyay, M. Gupta, A. Handa, and S. K. Shukla, "Malware classification using early stage behavioral analysis," in *Proc. 14th Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, Aug. 2019, pp. 16–23, doi: 10.1109/AsiaJCIS.2019.00-10.

[10] D. Du, Y. Sun, Y. Ma, and F. Xiao, "A novel approach to detect malware variants based on classified behaviors," *IEEE Access*, vol. 7, pp. 81770–81782, 2019.

[11] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3187–3196, Jul. 2018.

[12] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Comput. Secur.*, vol. 81, pp. 123–147, Mar. 2019, doi: 10.1016/j.cose.2018.11.001.

[13] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: A user-oriented behavior-based malware variants detection system for Android," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1103–1112, May 2017.

[14] J. Zhang, Z. Qin, H. Yin, L. Ou, and K. Zhang, "A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding," *Comput. Secur.*, vol. 84, pp. 376–392, Jul. 2019, doi: 10.1016/j.cose.2019.04.005.

[15] Y. A. Ahmed, B. Koçer, S. Huda, B. A. S. Al-Rimy, and M. M. Hassan, "A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection," *J. Netw. Comput. Appl.*, vol. 167, Oct. 2020, Art. no. 102753, doi: 10.1016/j.jnca.2020.102753.

[16] Z. Salehi, A. Sami, and M. Ghiasi, "MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values," *Eng. Appl. Artif. Intell.*, vol. 59, pp. 93–102, Mar. 2017, doi: 10.1016/j.engappai.2016.12.016.

[17] D. Konopiský, "Malware detection in applications based on presence of computer generated strings," U.S. Patent 0 285 565 A1, Oct. 4, 2018.

[18] M. H. Nguyen, D. L. Nguyen, X. M. Nguyen, and T. T. Quan, "Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning," *Comput. Secur.*, vol. 76, pp. 128–155, Jul. 2018.

[19] M. Tang and Q. Qian, "Dynamic API call sequence visualisation for malware classification," *IET Inf. Secur.*, vol. 13, no. 4, pp. 367–377, Jul. 2019.

[20] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.

[21] Z. Chen, Q. Yan, H. Han, S. Wang, L. Peng, L. Wang, and B. Yang, "Machine learning based mobile malware detection using highly imbalanced network traffic," *Inf. Sci.*, vols. 433–434, pp. 346–364, Apr. 2018, doi: 10.1016/j.ins.2017.04.044.

[22] S. Parkinson, "Use of access control to minimise ransomware impact," *Netw. Secur.*, vol. 2017, no. 7, pp. 5–8, Jul. 2017.

[23] R. Sihwail, K. Omar, and K. A. Z. Ariffin, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 8, nos. 4–2, p. 1662, Sep. 2018.

[24] M. Ghiasi, A. Sami, and Z. Salehi, "Dynamic malware detection using registers values set analysis," in *Proc. 9th Int. ISC Conf. Inf. Secur. Cryptol.*, Sep. 2012, pp. 54–59.

[25] A. Tajoddin and M. Abadi, "RAMD: Registry-based anomaly malware detection using one-class ensemble classifiers," *Int. J. Speech Technol.*, vol. 49, no. 7, pp. 2641–2658, Jul. 2019.

[26] I. Lee, H. Roh, and W. Lee, "Poster abstract: Encrypted malware traffic detection using incremental learning," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 1348–1349, doi: 10.1109/INFOCOMWKSHPS50562.2020.9162971.

[27] A. Singh, A. Walenstein, and A. Lakhotia, "Tracking concept drift in malware families," in *Proc. 5th ACM Workshop Secur. Artif. Intell.*, 2012, pp. 81–92.

[28] Z. Wang, M. Tian, J. Wang, and C. Jia, "An ensemble learning system to mitigate malware concept drift attacks (Short Paper)," in *Proc. Int. Conf. Inf. Secur. Pract. Exper.* Cham, Switzerland: Springer, 2017, pp. 747–758.

[29] A. Yan, Z. Chen, R. Spolaor, S. Tan, C. Zhao, L. Peng, and B. Yang, "Network-based malware detection with a two-tier architecture for online incremental update," in *Proc. IEEE/ACM 28th Int. Symp. Qual. Service (IWQoS)*, Jun. 2020, pp. 1–10.

[30] Y. Dai, H. Li, Y. Qian, R. Yang, and M. Zheng, "SMASH: A malware detection method based on multi-feature ensemble learning," *IEEE Access*, vol. 7, pp. 112588–112597, 2019.

[31] D. Hu *et al.*, "The concept drift problem in Android malware detection and its solution," *Secur. Commun. Netw.*, vol. 2017, doi: 10.1155/2017/4956386.

[32] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.

[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.

[34] E. Fitkov-Norris and S. O. Folorunso, "Impact of sampling on neural network classification performance in the context of repeat movie viewing," in *Engineering Applications of Neural Networks*. Berlin, Germany: Springer, 2013, pp. 213–222.

[35] J. Demšar and Z. Bosnić, "Detecting concept drift in data streams using model explanation," *Expert Syst. Appl.*, vol. 92, pp. 546–559, Feb. 2018, doi: 10.1016/j.eswa.2017.10.003.

[36] B. A. S. Al-Rimy, M. A. Maarof, M. Alazab, S. Z. M. Shaid, F. A. Ghaleb, A. Almalawi, A. M. Ali, and T. Al-Hadhrami, "Redundancy coefficient gradual up-weighting-based mutual information feature selection technique for crypto-ransomware early detection," *Future Gener. Comput. Syst.*, vol. 115, pp. 641–658, Feb. 2021, doi: 10.1016/j.future.2020.10.002.

[37] B. A. S. Al-Rimy, M. A. Maarof, M. Alazab, F. Alsolami, S. Z. M. Shaid, F. A. Ghaleb, T. Al-Hadhrami, and A. M. Ali, "A pseudo feedback-based annotated TF-IDF technique for dynamic crypto-ransomware pre-encryption boundary delineation and features extraction," *IEEE Access*, vol. 8, pp. 140586–140598, 2020, doi: 10.1109/ACCESS.2020.3012674.

[38] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–40, Oct. 2017, doi: 10.1145/3073559.

[39] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. V. Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 65–79, doi: 10.1109/SP.2012.14.

[40] A. Sami, B. Yadegari, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2010, pp. 1020–1025.

[41] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. San Francisco, CA, USA: No Starch Press, 2012.

[42] A. G. Kakisim, M. Nar, and I. Sogukpinar, "Metamorphic malware identification using engine-specific patterns based on co-opcode graphs," *Comput. Standards Interfaces*, vol. 71, Aug. 2020, Art. no. 103443, doi: 10.1016/j.csi.2020.103443.

[43] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, "Uncovering the face of Android ransomware: Characterization and real-time detection," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1286–1300, May 2018, doi: 10.1109/TIFS.2017.2787905.

[44] M. Wadkar, F. D. Troia, and M. Stamp, "Detecting malware evolution using support vector machines," *Expert Syst. Appl.*, vol. 143, Apr. 2020, Art. no. 113022, doi: 10.1016/j.eswa.2019.113022.

[45] S. Jain and Y. K. Meena, "Byte level *n*-gram analysis for malware detection," in *Computer Networks and Intelligent Computing*. Berlin, Germany: Springer, 2011, pp. 51–59.

[46] A. Khalilian, A. Nourazar, M. Vahidi-Asl, and H. Haghighi, "G3MD: Mining frequent opcode sub-graphs for metamorphic malware detection of existing families," *Expert Syst. Appl.*, vol. 112, pp. 15–33, Dec. 2018, doi: 10.1016/j.eswa.2018.06.012.

[47] M. Rhode, L. Tuson, P. Burnap, and K. Jones, "LAB to SOC: Robust features for dynamic malware detection," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.-Ind. Track*, Jun. 2019, pp. 13–16, doi: 10.1109/DSN-Industry.2019.00010.

[48] S. Yang, S. Li, W. Chen, and Y. Liu, "A real-time and adaptive-learning malware detection method based on API-pair graph," *IEEE Access*, vol. 8, pp. 208120–208135, 2020, doi: 10.1109/ACCESS.2020.3038453.

[49] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 2, pp. 289–302, Feb. 2016, doi: 10.1109/TIFS.2015.2491300.

[50] M. Jabi, M. Pedersoli, A. Mitiche, and I. B. Ayed, "Deep clustering: On the link between discriminative models and K-Means," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 6, pp. 1887–1896, Jun. 2021, doi: 10.1109/TPAMI.2019.2962683.

[51] X. Zhan, J. Xie, Z. Liu, Y.-S. Ong, and C. C. Loy, "Online deep clustering for unsupervised representation learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 6688–6697.

ABDULBASIT A. DAREM (Member, IEEE) received the Ph.D. degree in computer science from the University of Mysore, India, in 2014. He is currently an Assistant Professor with the Department of Computer Science, Northern Border University, Saudi Arabia. He has more than 20 years of experience in the IT filed. He published more than 19 research articles in reputed international journals and conferences. His research interests include cybersecurity, web engineering, HCI, usability, e-government, and cloud computing.

FUAD A. GHALEB received the B.Sc. degree in computer engineering from the Faculty of Engineering, Sana'a University, Yemen, in 2003, and the M.Sc. and Ph.D. degrees in computer science (information security) from the Faculty of Engineering, School of Computing, Universiti Teknologi Malaysia (UTM), Johor, Malaysia, in 2014 and 2018, respectively. He is currently an Assistant Professor with the Faculty of Engineering, School of Computing, UTM. His research interests include vehicular network security, cyber security, intrusion detection, data science, data mining, and artificial intelligence. He was a recipient of many awards and recognitions, such as the Postdoctoral Fellowship Award, the Best Postgraduate Student Award, the Excellence Awards, and the Best Presenter Award from the School of Computing, Faculty of Engineering, UTM, as well as best paper awards from many international conferences.

ASMA A. AL-HASHMI received the Ph.D. degree in computer science from the University of Mysore, India, in 2015. She is currently an Assistant Professor with the Department of Computer Science, Northern Border University, Saudi Arabia. She has more than ten years of experience in the IT filed. She published more than 17 research articles in reputed international journals and conferences. Her research interests include, cybersecurity, software engineering, e-government, and cloud computing.

JEMAL H. ABAWAJY (Senior Member, IEEE) received the B.S.E., M.Sc., Ph.D., and D.Sc. degrees. He is currently a Full Professor with the Faculty of Science, Engineering and Built Environment, Deakin University, Australia. His leadership is extensive spanning industrial, academic and professional areas, such as the IEEE Technical Committee on Scalable Computing, Academic Board, Faculty Board, and Research Integrity Advisory Group. He has delivered numerous keynote addresses, invited seminars, and media briefings, such as Voice of America's English Radio. He is a Senior Member of the IEEE Society, the IEEE Technical Committee on Scalable Computing (TCSC), the IEEE Technical Committee on Dependable Computing and Fault Tolerance, and the IEEE Communication Society. He has been actively involved in the organization of more than 400 national and international conferences in various capacity, including chair, general co-chair, vice-chair, best paper award chair, publication chair, session chair, and program committee. He has served on the editorial-board of numerous international journals. He is also serving as an Associate Editor for the IEEE TRANSACTIONS ON CLOUD COMPUTING, *International Journal of Big Data Intelligence*, and *International Journal of Parallel, Emergent and Distributed Systems*. He has also guest edited many special issue journals. More details can be found at: http://www.deakin.edu.au/~jemal.

SULTAN M. ALANAZI received the master's degree in IT and the Ph.D. degree in computer science from the University of Nottingham, U.K. He worked as a Teaching Assistant with the University of Nottingham. He is currently an Assistant Professor with the Department of Computer Science, Northern Border University, Saudi Arabia. He has more than ten years of experience in the IT field. He published several research articles in reputed international journals and conferences. His research interests include cybersecurity, machine learning, NLP, social network analysis (mining), user-modeling, and recommender systems.

AFRAH Y. AL-REZAMI received the Ph.D. degree in statistics from Al-Mustansiriya University, Iraq, in 2004. She is currently an Assistant Professor with the Mathematics Department, Prince Sattam Bin Abdulaziz University, Saudi Arabia. She focuses mainly on studying behavioral aspects. She published many research articles in reputed international journals and conferences. Her research interests include applied statistics and data analysis.