Wireless Networks (2024) 30:6177–6198
https://doi.org/10.1007/s11276-023-03414-5

NMal-Droid: network-based android malware detection system using transfer learning and CNN-BiGRU ensemble

Farhan Ullah[1] · Shamsher Ullah[2] · Gautam Srivastava[3,5,6] · Jerry Chun-Wei Lin[4] · Yue Zhao[1]

Accepted: 24 May 2023 / Published online: 27 June 2023
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract
Currently, malware activities pose a substantial risk to the security of Android applications. These risks are capable of stealing important information and causing chaos in the economy, social structure, and financial sector. Malicious network traffic targets Android applications due to their constant connectivity. This study develops the NMal-Droid approach for network-based Android malware detection and classification. First, we designed a packet parser algorithm that filters the combination of HTTP traces and TCP flows from PCAPs (Packet Capturing) files. Second, the fine-tune embedding approach is developed that uses a word2vec pre-trained model to analyze features' embeddings in three different ways, i.e., random, static, and dynamic. It is used to learn and extract feature-matrix matrices with related meanings. Third, The Convolutional Neural Network (CNN) is used to extract effective features from embedded information. Fourth, the Bi-directional Gated Recurrent Unit (Bi-GRU) neural network is designed to compute gradient computation in the context of time-forward and time-reversed. Finally, a multi-head ensemble of CNN-BiGRU is developed for accurate malware classification and detection. The proposed approach is evaluated on five different activation functions with 100 filters and a range of 1–5 kernel sizes for in-depth investigation. An explainable AI-based experiment is conducted to interpret and validate the proposed approach. The proposed method is tested using two big Android malware datasets, CIC-AAGM2017 and CICMalDroid 2020, which comprise a total of 10.2k malware and 3.2K benign samples. It is shown that the proposed approach outperforms as compared to the state-of-the-art methods.

Keywords Network traffic · Malware classification · Transfer learning · Explainable AI · Cybersecurity

1 Introduction

Android is currently the most widely used mobile Operating System (OS) globally. Because of its widespread use, it has unfortunately become a favorite of hackers who use Android to distribute millions of malware attacks. Mobile applications are no longer limited to telecom services as they formerly were. They have evolved into being capable of making online payments, communicating with peers, and playing games [1]. In the Google Play Store[1] compared to December 2009, there were more than twice as many applications (apps) available for download in December 2022. In December 2022, there were more than 3.553 million applications (apps) accessible through the Google Play Store, up from a little over 1 million in July 2013. Meanwhile, cloud solutions are a major contributor to the exponential growth of data generated by mobile networks. When it comes to mobile OSs, Android dominates the marketplace. These statistics indicate the wide acceptance of the Android OS. The explosive growth of Android has produced a thriving ecosystem of Android apps. Numerous Android app stores account for billions of direct downloads. The prevalence of cyberattacks increases in tandem with the proliferation of mobile devices such as smartphones and tablets [2]. Malicious programs are becoming more ruthless and difficult to combat. As a result, we must now deal with threats ranging from simple phishing emails to sophisticated network-based malicious attacks that can wipe sensitive information. In addition, malicious actors are getting better at developing malware that can escape traditional sandboxes [3]. Despite Android's security features and mobile antivirus, sophisticated mobile malware

Extended author information available on the last page of the article

[1] https://www.statista.com/statistics/266210/number-of-available-applications-in-thegoogle-play-store.

can still infiltrate mobile systems. Mobile devices are frequently linked to individual properties and sensitive data. The need for a reliable network-based Android malware detection system is critical. Faruki et al. [4] classified the mobile malware detection approach into static, dynamic, and traffic-based categories. Several earlier studies used a static approach to find security breaches and malware in Android apps. This approach is difficult owing to code polymorphism and obfuscation [5, 6]. The goal of dynamic analysis methods is to alter the OS of a device to monitor and transfer confidential data. These approaches work, but they require a lot of execution to cover all app activity patterns [7]. Numerous malware detection techniques emphasize network traffic caused by connected apps. Malware is identified by suspicious network behavioral patterns. Because the overwhelming majority of Android malware performs destructive operations through network traffic, this type of malware detection system is extremely useful [8]. Malware must communicate with a remote host via the Internet in order to perform malicious actions. These footprints allow specific malware to be tracked and identified. The development and implementation of network-based malware detection systems is also simpler than those of static or dynamic analysis techniques. For instance, network-based malware detection may be implemented at an access point or gateway. These solutions are solely based on user-generated network traffic data, ensuring that users continue to have access to their mobile apps. Furthermore, these methods require no user interaction other than authorizing licenses for the identifying service. [9]. The purpose of network traffic-based techniques is to find distinguishing characteristics of harmful data that may be utilized to precisely classify it. However, selecting effective features is a difficult task.

In this study, the NMal-Droid method is proposed to classify and detect Android malware by utilizing features from both HTTP and TCP traffic. HTTP requests are used since they are the most commonly used protocol for mobile apps. Network traffic can be classified using the vast quantity of data that is sent in HTTP requests. Jiang et al. [10] have recently used HTTP headers to derive a mobile app's network profile and categorize it. The HTTP request header can effectively detect Android malware. However, it is difficult to retrieve useful information from mobile apps due to HTTP encryption. To address this issue, we mined TCP flow to improve malware detection. In this case, TCP is being studied because it is a popular transport layer protocol. [11]. A session in TCP flow is a grouping of messages having the same five-tuple (source IP, source port, destination IP, destination port, protocol type). Our findings demonstrate that TCP flows can be utilized to classify and detect network-based malware. We discovered that combining these diverse types of network features can result in an improved detection rate for mobile malware.

The following are the main contributions of this paper:

1. Detect and classify Android malware, we developed the NMal-Droid approach, which employs fine-tune embedding and the multi-head ensemble of CNN-BiGRU. Experimental results show that the proposed accurately classifies Android malware using the combined features of HTTP traces and TCP flows.
2. For extensive analysis, the word embedding mechanism is fine-tuned with random, static, and dynamic weights using a pre-trained word2vec model. Further, we used 5 activation functions to analyze the comparative analysis of the proposed method. It is shown that the proposed approach obtains the highest performance on the dynamic-based word2vec model.
3. Extract deep and broad features, a multi-headed ensemble neural network known as CNN-BiGRU is designed for accurate malware classification and detection. Thus, it reduces the challenges associated with feature selection in network-based malware detection approaches.
4. The effectiveness of the proposed method is intended to be interpreted and validated by means of an explainable AI strategy.

The following is a breakdown of the remaining content: Section 2 describes the related work, Section 3 thoroughly describes the proposed methodology, and Section 4 presents the experimental results. Section 5 discusses validating the performance, while Section 6 provides the conclusion.

2 Related work

The Android OS defends effected target machines by utilizing a range of safety features, such as authorization procedures, as shown by numerous research findings [12, 13]. However, those who wish to safeguard their administrative privileges should be familiar enough with security concerns to warrant that protection. When you put too much faith in your customers, you open the door for malicious software like Android malware to infiltrate their devices and spread to others. The majority of these tools evaluate apps for suspicious behavior by checking for things like extra permissions and advertisements. These anti-virus scanners provide some protection from harmful programs. Nevertheless, the number and variety of malicious apps is constantly growing and evolving. These anti-virus utilities protect the device from malicious software. However, malicious software is constantly adapting and expanding its range of attack. Therefore, improvements to

malware detection infrastructure are required. Several anti-malware tools can now analyze APK files for malicious code even before they are opened. A static-based method that reliably categorized malware was established by Sanz et al. [14]. This method worked by recording the permissions and log files of users. The suggested research achieved 86.41% classification accuracy. The same method was utilized by Puerta et al. [15] in to identify malicious files by utilizing the Drebin dataset, and they observed an accuracy of 96.05%. A technique of detecting malware that consists of two stages was presented by Liu et al. [16]. The first step is to examine the Manifest.xml file of the app, which lists the privileges that are being requested. The second step consists of preprocessing the APK file by utilizing the APK toolkits to get the smali code. There is a possibility that Smali code contains details regarding asserted privileges, such as API calls, which can be utilized to identify inappropriate actions. With the proposed method, detection rates are at 98.6%.

According to Shanshan et al. [17], a malware classification system was proposed for analyzing suspicious networks by combining HTTP and TCP files. The data transfer that occurs over the phone app is also carried out by the wireless device. The cloud is used for the handling of all data and the identification of malware. This enables mobile apps to use the fewest resources possible without negatively impacting the user engagement. A detection accuracy of 97.89% is achieved for Android malware using a combination of network flows and machine learning algorithms. Aresu et al. [18] investigate the HTTP-based transmitted data produced by mobile apps when communicating with distant malicious servers. Signatures from various families of malware are generated using a clustering technique. Later, these fingerprints are employed to spot suspected attacks. Traffic flow was suggested as a conceptual model by Shanshan et al. [19]. for the tracking of Android malware. Natural language processing (NLP) methods are applied to the HTTP text file in order to conduct a sentiment analysis. When analyzing suspicious data traffic, these HTTP-based characteristics are used for investigation. The next step is to identify malicious software by analyzing the textual characteristics of data collected from the network. The proposed method classifies malware at 99%. TextDroid was introduced by Wang et al. [20]. It parses an HTTP stream for unique characters and generates n-gram trends to analyze the distribution of the extracted features. Furthermore, it compiled data sequences for the purpose of feeding it into a machine learning model to identify malware. Consequently, this text-based approach obtained an accuracy rate of 76.99%. TrafficAV collects information from both TCP and HTTP traffic attributes and then uses Decision Tree (DT) classifier to evaluate the classification accuracy. Moreover, the classification model may not take advantage of both TCP and HTTP flows, which limits the usefulness of the approach. Using HTTP traffic, it can identify malicious code at a rate of 98.16% [21]. The WebEye framework proposed by Johann et al. [22] effectively creates credible HTTP traffic. It enriches collected traffic with additional information, and categorizes files as unwanted or legitimate utilizing predictive modelling with an average performance of 89.52%.

Numerous studies utilizing deep learning to categorize malicious files have yielded improved outcomes [23–25]. It was proposed by Chen et al. [26] that the CNN method be used to categorize mobile apps according to their HTTP flows. The utilization of CNN speeds up the process of feature selection, which ultimately leads to more accurate findings related to network traffic. The success rate of the proposed method increased to 98%. The DeepSign approach, developed by David et al. [27], makes use of deep belief networks. It is able to produce unchangeable and succinct interpretations of the actions that malware engages in, which enables it to effectively differentiate nearly all existing malware families with a success rate of 98.6%. Malware can be detected through HTTP requests to the Android app, as shown by Shanshan et al. [28]. Next, a multi-view neural net is used to identify potentially malicious activities at varying depths of penetration. This procedure can be used to concentrate on particular characteristics of input variables by allocating adequate attention to features. The best and worst prediction accuracy for this method are 98.81% and 89.35%, respectively.

This study proposed the NMal-Droid approach for detecting network-based Android malware. Network analysis collects HTTP and TCP flows because these are the protocols most commonly used by mobile apps. For improved detection results, the network embedding process is fine-tuned with three alternative strategies: random, static, and dynamic. The multi-headed ensemble-based CNN-BiGRU model is designed to detect and classify malware accurately. An explainable AI and t-SNE experiment is carried out for model interpretation and validation. The novel method can generate distinguishable data, making it easier to select features for network-based malware detection systems.

3 Proposed scheme: NMal-Droid

The architectural framework for the proposed NMal-Droid approach is given in Fig. 1. The combined network flows for HTTP and TCP are mined from PCAPs using the packet parser technique. Next, the fine-tune embedding approach uses word2vec-based transfer learning to analyze embeddings in three ways. It is employed to discover and

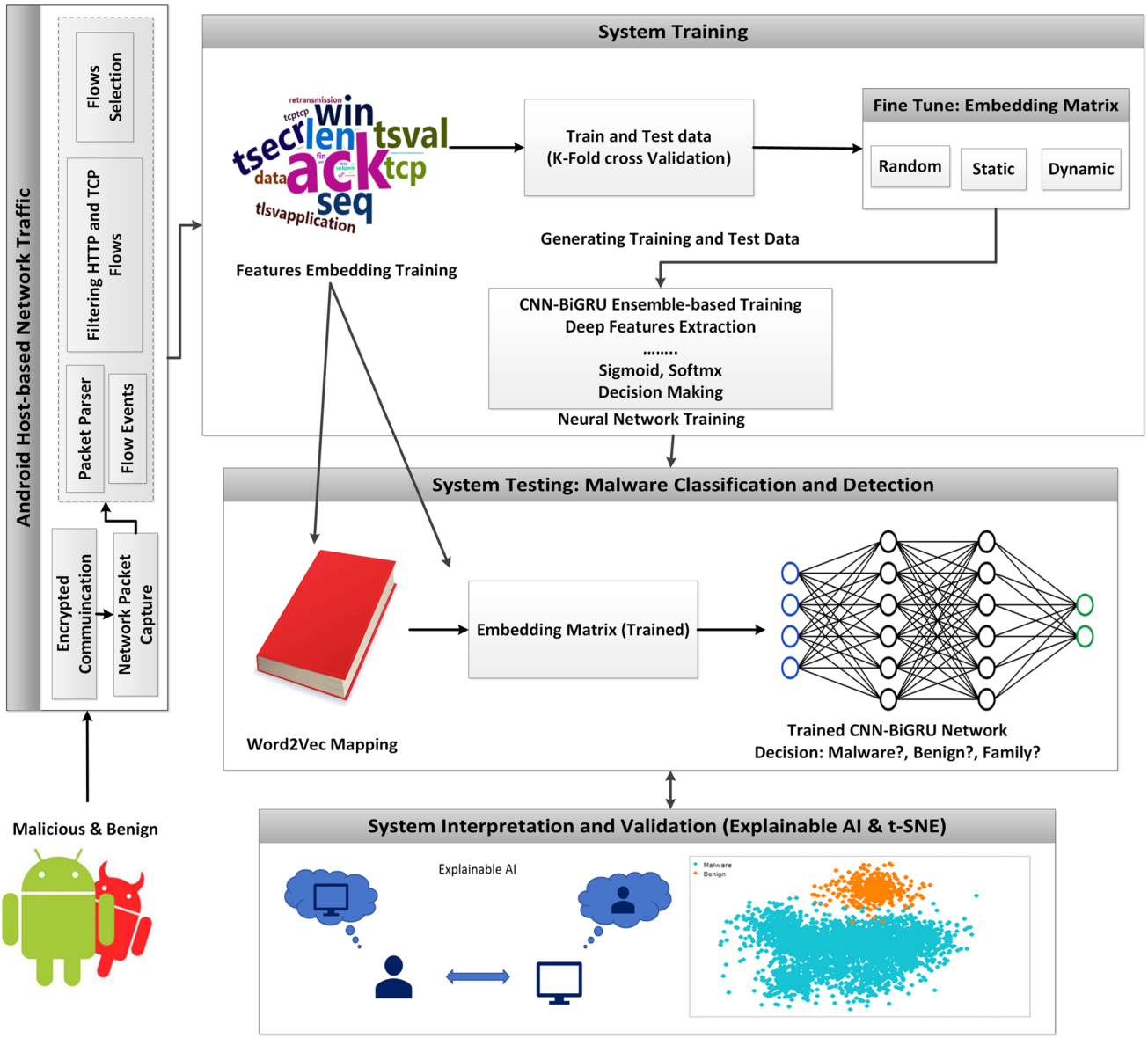

Fig. 1 Architectural framework for the proposed work

extract semantic vectors with associated meanings. Finally, a CNN-BiGRU ensemble with multiple heads is designed to accurately classify and detect malware.

3.1 Preparing network traffic

We employ HTTP requests because it is the most prevalent protocol for data transmission. HTTP headers contain information that can be utilized to categorize malicious activity. Since most app-to-app communication occurs over HTTP, it may not be possible to glean any useful information from mobile apps. To address this, we obtain HTTP and TCP flows from PCAP files. Because of its pervasiveness, TCP is the most appropriate transport layer protocol for analysis [11]. The PCAP file includes an immense quantity of network data, some of which may be characterized as noisy for the suggested technique. Algorithm 1 demonstrates the development of a packet parser technique that can decipher encoded network traffic and mine HTTP/TCP traffic simultaneously.

Algorithm 1 Packet Parser Algorithm

Require: Packet Capturing Files
Ensure: TCP, and HTTP as output files
Set $P = \{p_1, p_2, \ldots, p_n\}$, where P is collection of Packets
Parse $P^{'} \leftarrow P$
while Computes **do**
  if PCAP **then**
    Compute $P^{'}$ from PCAP where $P^{'} = (IP, TCP, HTTP, \ldots, n)$
  else
    Filter $HTTP + TCP$ from $P^{'}$
  end if
end while
Finish

The HTTP flows include the following information: host address, source information, source IP, destination IP, destination port, number of bytes, packet length, frame length, and TTL. The source info section also includes a great deal of additional information, such as GET and POST commands, as well as URLs, such as "www.yahoo.com." TCP flows also involve 3 handshake additional data, including bytes uploaded and downloaded and total packet numbers during various sessions. In addition, it includes the source information formatted as SYN, ACK, and FIN. Table 1 displays a portion of the source data that has been extracted from the traffic data. These tangled datasets used to classify malware are useless and may contain significant levels of noise. It may cause the proposed method to become overburdened, lowering malware classification accuracy. These textual details must be processed and transformed into smooth HTTP and TCP flow attributes. To address this issue, we developed a semantic tokenizer tool that can filter away extraneous data while maintaining the intended meaning of the input. For instance, choosing a specific character from the Uniform Resource Locator (URL) "www.yahoo.com" is pointless. This can only be used to demonstrate that a whole domain name is recognized as an entity. Any URL is made up of fundamental parts, each of which represents a different piece of data. We delete non-malware-related words from the data received because not all feature extraction phrases can be used to identify harmful software. The purpose of configuring such filtering rules is to produce a clean set of features. The primary procedures of data pre-processing are:

- Duplicate features that appear in a row must be removed from input sequences to avoid data redundancy.
- Due to the possibility that short segments do not contain sufficient details to determine the appropriate network activity, there are omitted from the dataset.
- Since deep learning algorithms become confused when presented with sequences of varying lengths, standardizing sequence length is essential for malware classification. A fixed sequence length, L, is used in this method. Sequences with length greater than L retain their initial length L names, while sequences with length less than L are combined via zero padding.

Table 1 A chunk of source information from network traffic

Network traffic	Source information
HTTP	GET /getAds.php?pn=com.elm &ver_p=3 &imei=352584066095073
	POST/socket.io/?EIO=3 &transport=polling &t=1472474822453-26139 &sid=QzhXKVYFWHT6NaBtAAAe HTTP/1.1 (text/plain)
	GET/socket.io/?EIO=3 &transport=polling &t=1472474848518-26142 &sid=QzhXKVYFWHT6NaBtAAAe HTTP/1.1
	push.eventmobi.com, api.fanjie.com:8001, sygicfcd.cloudapp.net
TCP	48406 ¿ 8001[SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=6373438 TSecr=0 WS=64
	41455 ¿ 8001 [ACK] Seq=1 Ack=2 Win=1386 Len=0 TSval=6368051 TSecr=2070961621
	8001 ¿ 48406 [PSH, ACK] Seq=1 Ack=321 Win=15616 Len=237 TSval=2071016625 TSecr=6373505 [TCP segment of a reassembled PDU]
	[TCP Retransmission] 37497>8001 [FIN, ACK] Seq=1 Ack=1 Win=1386 Len=0 TSval=6379665 TSecr=2070841606

3.2 Fine-tune embedding

Vectors are fed into the neural network. Therefore, we describe network traffic as vectors with a fixed size (L). We may utilize a one-shot vector. But its size is limited by the number of unique features in our dataset. As a result, such a strategy is unsuitable for large-scale training. Thus, we need a condensed vector that also has a meaningful value. To meet these needs, we chose the word embedding approach, specifically word2vec [29]. Our major aim is to generate a dense vector for each network feature that tracks its contexts throughout a huge dataset of Android apps. Additionally, we can employ geometric approaches on the network vectors to determine the semantic relationship between their functionality, i.e., attackers frequently use the same URL or TCP session for the same target. Figure 2 shows the visualization of embedding words using word2vec and TensorFlow. In our case, we train these vectors using word2vec from a dataset of benign and malicious apps. The embedding word model output is a matrix KxA, where K is the embedding vector size and A is the number of unique network features. Our word embedding is tuned concurrently with the neural network for a certain task, i.e., detection. The embedded word vector can be learned independently of the malware classification and detection task [30, 31]. The embedding values are trainable parameters rather than manually specified. These are commonly 8-dimensional for small datasets and 1024-dimensional for large datasets. We set 300-dimensional for the combined features of HTTP and TCP. For finer word associations, higher dimensional embeddings require more data to learn. The embeddings are fine-tuned as random, static, and dynamic for extensive network analysis. We used word2vec to fine-tune the features during static and dynamic data analysis for better classification results.

(a) **Random:** The model trains with a network embedding layer that randomly initializes feature vectors. Backpropagation is used to gradually adjust them during a training session. Once trained, learned network embeddings can effectively contain feature similarities for network data analysis. In this approach, the weights are allocated at random to each feature, which may carry erroneous or chaotic data for the built neural network.

(b) **Static:** The model uses the 300-dimensional word2vec pre-trained word embedding model. Throughout the training procedure, the vectors are maintained in a static position. By applying that function, all of the features are integrated into a single vector. Consequently, according to this mapping function, the same feature cannot have multiple interpretations.

(c) **Dynamic:** It also uses a pre-trained word2vec model with 300-dimensional vectors. The vectors may stay dynamic throughout the training operation. This approach distributes comparable word types across many vectors. As a result, this mapping function is versatile enough to allow for many interpretations of the same feature across time. This strategy is superior because it allows us to extract multiple meanings for the same word, which neural networks can easily comprehend.

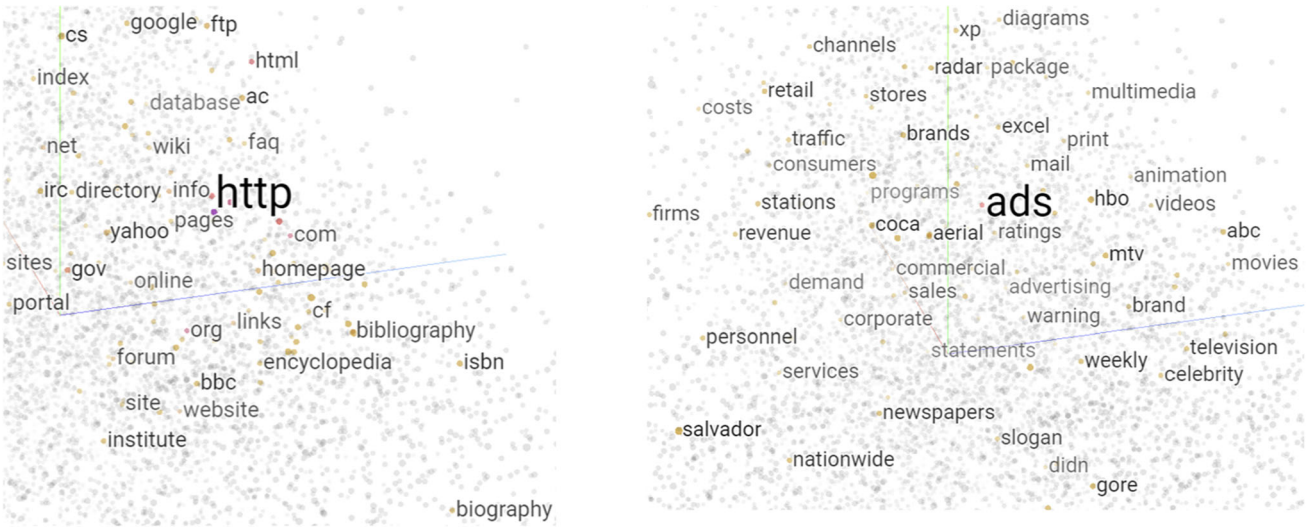

Fig. 2 Semantic visualization of embedding words (http, ads) using word2vec

Algorithm 2 The process of fine-tune embedding

Require: Features
Ensure: Network embedding Trained Features
Set $F = f_1, f_2, \ldots, f_n$, and $TF = tf_1, tf_2, \ldots, tf_n$, where F is a Features & TF is a Trained Feature:
Select T and F:
if Selects: Randomly, Statically, and Dynamically (T and F) **then**
    RT & RF
    ST & SF
    DT & DF
end if
while Computes **do**
    if Training **then**
        Compute RT & RF as $\sim w2vec$
        Compute ST & SF as w2vec
        Compute DT & DF as $w2vec$
    else[Apply cross-validation]
        Compute T_r as a Training data
        Compute T_e as a Testing data
    end if
end while
if $T_e = i \geq 10$ **then** Features extraction of T_e
end if
Finish

3.3 Features extraction

For deep feature extraction, the embedded features are fed into the CNN network as input vectors. This strategy utilized a 1-D CNN network that included dropout layers, fully connected layers, pooling layers, and convolutional layers. There are several studies [32, 33] that proposed the use of the CNN model to classify Android malware. The convolution layer receives the relevant data via the input layer and convolutes it using the kernel of the convolution layer as shown in Eq. 1.

$$x_j^l = f\left(\sum (x_i^{(l-1)}.\omega_{ij}^l\right) \tag{1}$$

where x_j^l is the outcome of j channel with respect to layer l. $x^(l-1)_i$ is the input of the i channel for the recent layer l. ω_{ij}^l is the computed weighted matrix of the same layer and b_j^l is the related bias terms for the given function f. Semantic features are iteratively spun through the convolution-based filter to extract the optimal features. A feature map is a new set of features that are generated by each filter. Hyperparameter optimizations are used to find the best number of filters. Layers of convolution and pooling are linked together. In order to prevent overfitting, the pooling layer filters data and reduces the number of parameters used to compute the network as shown in Eq. 2.

$$x_j^l = f\left(\beta_j^l down\left(x_j^{l-1}.M^l\right) + b_j^l\right) \tag{2}$$

where β_j^l is the weight matrix of j, $down()$ represents the pooling function, and M^l represents the size of the pool window. The max-pooling layer narrows the focus of the feature space while decreasing the required number of computations. This layer also creates a feature map from the most salient features in the previous feature set. The proposed CNN network makes use of the softmax function in conjunction with dropout layers to combat the overfitting problem. Equation 3 shows the mathematical form of the probability distribution computed by softmax for classes k.

$$O = \begin{bmatrix} p(y=1 \mid x:\omega_1 b_1) \\ p(y=1 \mid x:\omega_2 b_2) \\ \cdots \\ p(y=K \mid x:\omega_k b_k) \end{bmatrix} = \frac{1}{\sum_j^k exp(\omega_j x + b_j)} \begin{bmatrix} exp(\omega_1 x + b_1) \\ exp(\omega_2 x + b_2) \\ \cdots \\ exp(\omega_k x + b_k) \end{bmatrix} \tag{3}$$

Equation 4 is used to represent the outcome of the one-dimensional CNN network.

$$o_k^1 = f\left(c_k^1 + \sum_{(i=1)}^{N_{(l-1)}} Con1D\left(X_{ik}^{(l-1)}, t_i^{(l-1)}\right)\right) \quad (4)$$

where c_k^1 is the parameter bias of the kth neuron in the first layer, $t_i^{(l-1)}$ is the outcome of the ith neuron in layer $l-1$, $X_{ik}^{(l-1)}$ is the kernel strength from the ith neuron in layer l-1 to the kth neurons in layer l, and "f()" is the activation function.

3.4 Gradient computation

Chung et al. [34] proposed the GRU neural network. GRU is an excellent solution to the gradient vanishing issue that happens during the training phase of recurrent neural networks (RNNs). In comparison to LSTM, GRU is straightforward. GRU is composed of only two gates, i.e., the update gate, and the reset gate. It reduces the network's parameters and hence speeds up model training. The update gate (z) stipulates that pertinent information is preserved until the next state, while the reset gate (r) describes how the prior and new knowledge is united. When series data is used, GRU is capable of effectively learning long-term dependencies, which are determined in Eqs. 5–8.

$$z_t = \sigma(W_z * [x(t), h(t-1)]) \quad (5)$$

$$r_t = \sigma(W_r * [x(t), h(t-1)] \quad (6)$$

$$\hat{h}(t) = tanh(W_h * [x(t), (r_t * h(t-1))] \quad (7)$$

$$h_t = (1 - z_t) * h(t-1) + z_t * \hat{h}(t) \quad (8)$$

$x(t)$ shows the input data, $\sigma()$, and $tanh()$ are activation functions. The weighted matrices for the update, reset, and output gates are described by W_z, W_r, and W_h, respectively. The $h(t-1)$ shows the output data of the recent state. BiGRU is made up of two standards GRUs [35], each of which analyses the embedded vector in one direction (forth and time-reversed), and after, it integrates the corresponding outputs. As a result, BiGRU can catch feature representations that the traditional GRU may skip, resulting in better performance. Figure 3 shows the architecture of the Bi-GRU neural network.

3.5 Multi-head ensemble of CNN-BiGRU

A multi-head CNN-BiGRU ensemble model is proposed for malware classification using network embedding features as shown in Fig. 4. The ensemble is a term that refers to the aggregation of different predictions made by a model from multiple source datasets to obtain more accurate predictions. To train quickly, embedding vectors are employed as input to the specified ensemble model. Good models that rely on independent training of different embedding features, perform well, which may be because they acquire effective feature knowledge from their input data. That is, they anticipate viewing the word2vec embedding vectors from various angles to get real knowledge. To produce a more accurate outcome, the prediction outputs from both models can be merged, and the most straightforward integration approach is the weighted average of each prediction result. This not only permits the acquisition of deeper feature learning from various datasets but also permits the dominance of the finest dataset. The ensemble model is divided into three distinct steps.

- CNN-BiGRU takes the training set from each malware dataset in turn and uses it to incrementally learn the feature data within the dataset. The trained models are preserved for use in subsequent calls.
- When a trained model is fed the validation set for a dataset, it generates a probability distribution of predicted labels using a softmax layer. The accumulated probability and sample label estimation are obtained by assigning weights to each output. When the probability distributions are weighted, the ensemble model reliably

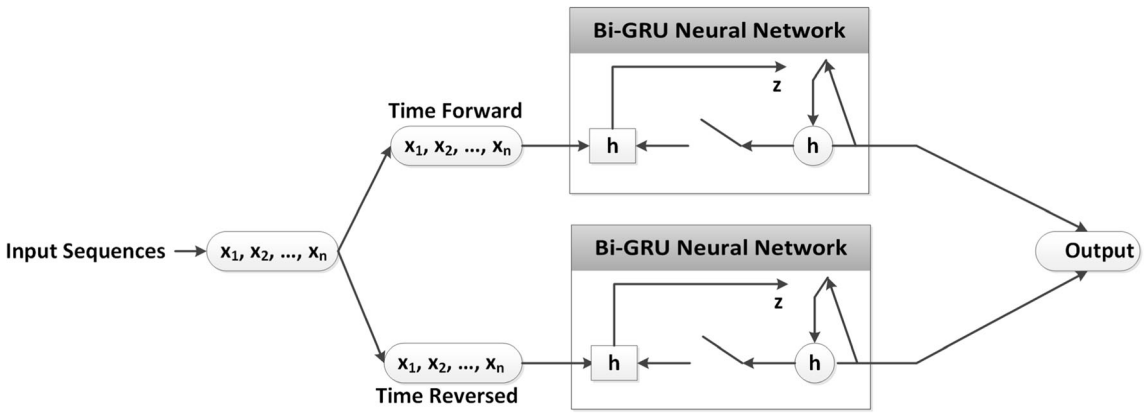

Fig. 3 Architecture of Bi-GRU neural network

predicts labels (0 for malicious, 1 for benign). Likewise, the seperate models may increase the risk of misclassification, whereas the ensemble model significantly enhances the performance.

- The trained model is fed the test set for each dataset separately. Ultimately, the predictions from all of the models are combined to form the final classification.

The complete process is given in Algorithm 3.

Algorithm 3 Malware classification via multi-headed ensemble-based CNN-BiGRU

Require: T_r, T_e
Ensure: Malware, Benign
Set $T_r = t_{r1}, t_{r2}, \ldots, t_{rn}$, and $T_e = T_{e1}, T_{e2}, \ldots, T_{en}$, where T_r and T_e is a Training & Testing Data
Apply w2vec as a Transfer learning:
if $T_r^{'} = w2vec(T_r)$ && $T_e^{'} = w2vec(T_e)$ **then**
    Train CNN
else
    Train BIGRU
end if
while $T_e^{'} = w2vec(T_e)$ **do**
    if Training **then**
        $CNN(T_r^{'})$
        $CNN(T_e^{'})$
    else$[CNN(T_r^{'})$ && $BIGRU(T_r^{'})]$
        $BIGRU(T_r^{'})$
        $BIGRU(T_e^{'})$
    end if
end while
if Tests==Model **then**
    Model=Malware
else
    Bengin
end if
Finish

4 Experimental results

4.1 Dataset collection

The proposed method is thoroughly examined using two datasets obtained from the Canadian Institute for Cybersecurity[2] [36] which is gathered semi-automatically by installing Android apps on authorized mobiles. The dataset is generated using 1900 apps and is separated into three classes: adware, general malware, and benign. The adware contains 250 malicious apps, including Airpush, Dowgin, kemoge, mobidash, and shuanet. The General Malware consists of 150 malicious apps, including AVpass, fakeAV, fakeflash, GGtracker, and penetho. A total of 1500 apps are included in the Benign. Table 2 contains a detailed description of the dataset. The second dataset CICMalDroid 2020 [37, 38] collected over 17,341 Android samples from different sources, including the VirusTotal service, the Contagio security blog, AMD, and MalDozer. From December 2017 to December 2018, the samples were taken. In addition to benign, this dataset includes malicious examples of Adware, Banking, Riskware, and SMS. There are 1253 instances of adware, 2100 instances of banking malware, 2546 instances of riskware, 3904 instances of benign software, and 1795 instances of benign software. Tables 2 and 3 provide a comprehensive breakdown of each app.

4.2 Performance indicators

To reduce bias, the training and testing data are created using k-fold cross-validation. We set k to 10 to generate random samples from both the training and test sets. We

[2] https://www.unb.ca/cic/datasets/index.html. The first dataset, the Canadian Institute of Cybersecurity Android Adware and General Malware (CICAAGM2017)

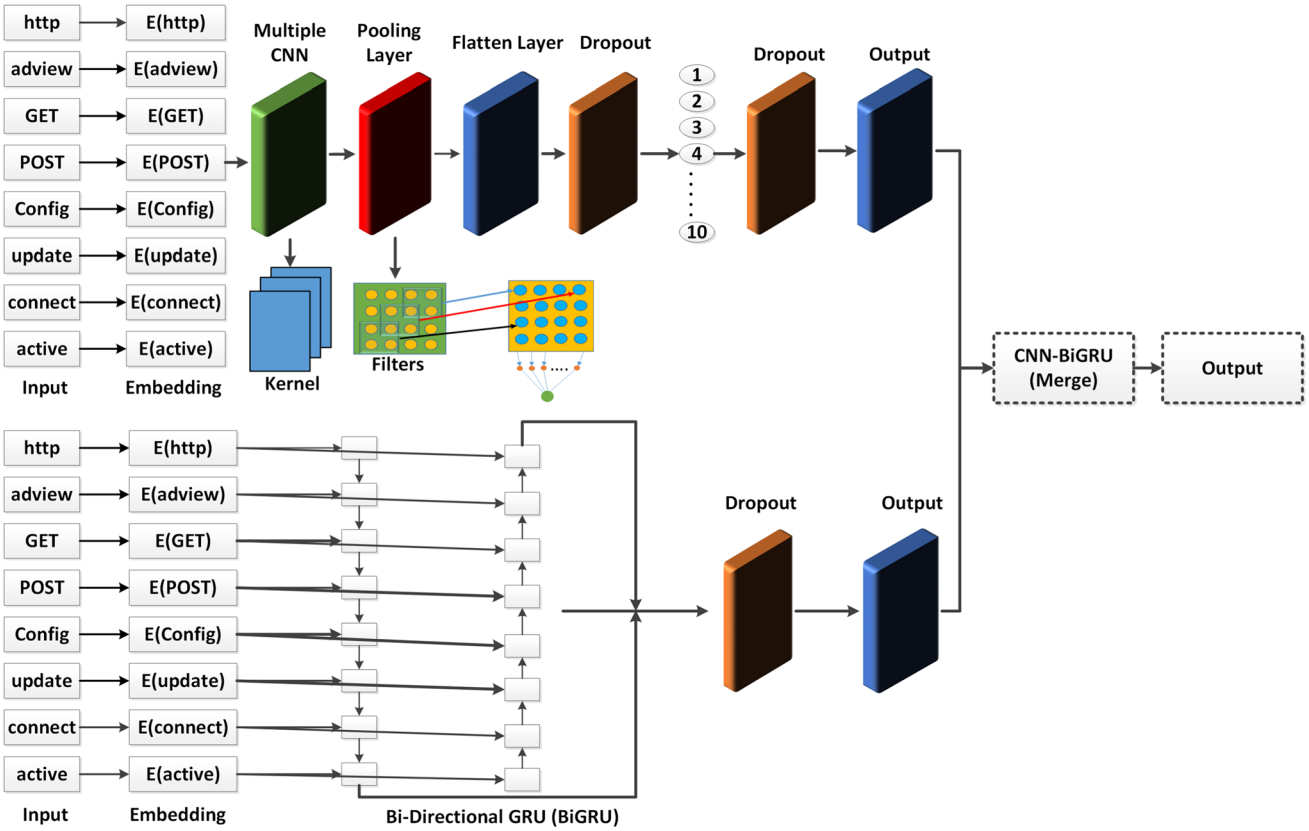

Fig. 4 Multi headed-based ensemble of CNN-BiGRU neural network

then use these random samples to generate our final training and test sets. This cycle is repeated, with the model being trained on the training set and compared to the test set. This data partition approach can help to reduce overfitting. The desired output is achieved by employing a mechanism that terminates the process prematurely once a predetermined threshold is reached. It is possible to shorten the process by stopping the model before the specified number of epochs have passed such as early stopping. We used four kinds of performance indicators such as Precision, Recall, F1-score, and Accuracy. The abbreviations used are False Positive (FP), True Positive (TP), False Negative (FN), and True Negative (TN). A TP occurs when the model correctly predicts the benign class. A TN is when the model predicts the malware class correctly. When a model wrongly concludes that a hypothesis is correct when it is not, this is an example of FP. An FN occurs when something is incorrectly thrown out. An accuracy metric is used to evaluate general classification performance. The performance indicators are shown in Eqs. 9–12.

$$Recall = \frac{FP}{(FP + TN)} * 100 \tag{9}$$

$$Precision = \frac{TP}{(TP + FP)} * 100 \tag{10}$$

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} * 100 \tag{11}$$

$$F1 - score = \frac{(2 * Precision * Recall)}{(Precision + Recall)} * 100 \tag{12}$$

4.3 Results analysis and performance comparisons

We use static, dynamic, and random embeddings to show that our proposed method works. For extensive comparisons, the proposed approach is tested with five different activation functions such as *relu*, *tanh*, *elu*, *para relu*, and *leaky relu*. Every time we used an activation function for one of the five kernels, we got the performance output for that function. The performance comparisons of five activation functions using random embeddings are shown in Table 4. Accuracy is denoted by the word acc. During each fold of 10-fold cross-validation, accuracy is extracted which is in the range of 1 to 10. We investigate two hyperparameters by extending the kernel sizes from 1 to 5 using 100 filters and analyzing the model's effects. The last

Table 2 Android adware and general malware dataset (CIC-AAGM2017)

Apps	No. of Apps	Families	Description
Adware	250	Airpush	It spreads malicious advertisements to compromise systems
		Dowgin	As a data-gathering advertising platform
		Kemoge	It hijacks the Android device
		Mobidash	Built to spread commercials and provide unauthorized access
		Shuanet	It has the ability to take control of the device
General Malware	150	AVpass	A utility program that masquerades as a clock
		FakeAV	Trickery for downloading unlocked software via e-mail
		FakeFlash	This fake Flash player leads to a suspicious domain
		Ggtracker	Utilized to acquire data via SMS fraud
		Penetho	Hacking tool that pretends to recover WiFi credentials
Benign	1500	Benign	Clean apps (Not malicious)

Table 3 CICMalDroid 2020 dataset

Apps	Families	No. of Apps	Description
Malware	Adware	1,253	Malicious software may contain advertisements that are not immediately obvious
	Banking	2100	Direct access is made to the user's online account
	Riskware	2546	It is possible to cause harm through the misuse of any legal software
	SMS	3904	It uses text messages as a means of attack
Benign	Benign	1795	Clean apps (Not malicious)

column shows how well each of the 10 folds worked together. It can be shown that *tanh* has the best classification accuracy, i.e., 97.1%, with a kernel size of 1 and *relu* has the worst, i.e., 87.1, with a kernel size of 2. The second and third highest performances, i.e., 94.6%, 94.4%, are *para_relu*, and *relu* with kernel sizes 4, and 4, respectively.

Figure 5 shows the overall average accuracy comparisons of five activations functions using fine-tune embeddings of random, static, and dynamic. Word2vec has been used for a static and dynamic approach. For both datasets, we conducted experiments for malware classification and detection. It can be observed that dynamic word embeddings provide the best classification results for both datasets, whereas random is the worst. For instance, random embedding has the maximum classification accuracy of 96.71% for the *relu* activation function using the CICMalDroid dataset, whereas dynamic embedding has the highest classification accuracy of 99.95% for the same dataset and activation function. Similarly, the random strategy has a maximum detection rate of 94.61% for CICMalDroid and *tanh* activation function, whereas dynamic has a detection rate of 99.89% for the same dataset and elu activation function. The static performs in between random and dynamic embeddings. For instance, static embedding has the maximum 99.78% classification rate for CIC-AAGM2017 with *relu* activation function while 99.83% detection rate for the CICMalDroid dataset with elu activation function. Overall, dynamic embedding outperforms random and static embedding because it mines embedding properties that change over time. As a result, it is more versatile to compute a range of semantically similar features. The accuracy and loss curves for training and testing data are obtained to examine the running behavior against each epoch. Figure 6 shows the accuracy and loss curves for random embedding with the most efficient activation function utilizing 1-fold cross-validation. Parts (a, b, c, and d) show the training and testing loss curves for malware classification using the CIC-AAGM2017 and CICMalDroid datasets. *Tanh* and *relu* are the most efficient activation functions for both datasets. The blue and red colors represent the training and testing data, respectively. We made use of 100 epochs. Part (a) begins with the training accuracy at 30% and the test at 40%. When both curves reach 80% at the 15th epoch, they drop and then rise to 97%. Following then, both curves become more or less constant. Both curves behave between 30% and 97.2%. In part (b), the green and yellow colors represent the training and testing loss. Initially, both training and testing loss are greater than 100%, but they eventually drop to 4% after some epochs. In the 90th epoch, there is a slight rise in

Table 4 Random embedding: performance comparisons of activation functions using 10-fold

Activation	Filters	acc1	acc2	acc3	acc4	acc5	acc6	acc7	acc8	acc9	acc10	Avg
tanh	1	98.8	98.8	100	100	100	100	87.8	100	100	86.6	97.2
para_relu	4	100	98.8	86.6	84.1	100	100	85.4	91.5	100	100	94.6
relu	4	100	88	100	86.6	98.8	81.7	100	100	100	89	94.4
tanh	3	100	97.6	100	87.8	100	100	85.4	82.9	91.5	82.9	92.8
leaky_relu	2	100	86.7	84.1	87.8	100	82.9	91.5	100	98.8	95.1	92.7
para_relu	5	84.3	100	86.6	100	81.7	100	89	100	100	84.1	92.6
elu	5	89	100	89.8	99.8	95.8	83.9	82.5	88.6	95.2	100	92.5
elu	2	88	100	87.8	98.8	98.8	82.9	80.5	86.6	100	100	92.3
leaky_relu	4	88	100	84.1	100	87.8	89	84.1	87.8	100	100	92.1
relu	5	97.6	100	89	100	79.3	82.9	100	91.5	100	80.5	92.1
elu	1	100	84.3	86.6	87.8	100	85.4	100	100	91.5	82.9	91.8
elu	4	84.3	88	85.4	84.1	100	100	100	100	90.2	85.4	91.7
elu	3	100	88	86.6	100	84.1	87.8	81.7	100	100	86.6	91.5
tanh	2	100	100	98.8	86.6	76.8	85.4	84.1	100	98.8	82.9	91.3
leaky_relu	1	78.3	85.5	100	92.7	89	100	80.5	98.8	86.6	100	91.1
relu	3	100	100	82.9	85.4	100	86.6	84.1	100	84.1	86.6	91
leaky_relu	5	92.8	85.5	100	80.5	92.7	85.4	98.8	89	82.9	100	90.8
leaky_relu	3	88	100	100	89	100	85.4	86.6	81.7	90.2	86.6	90.7
para_relu	1	100	100	85.4	81.7	86.6	100	79.3	84.1	98.8	87.8	90.4
relu	1	89.2	89.2	80.5	100	87.8	100	100	86.6	84.1	81.7	89.9
para_relu	2	90.4	100	92.7	84.1	80.5	84.1	81.7	100	85.4	100	89.9
tanh	5	100	85.5	89	80.5	81.7	87.8	100	85.4	84.1	100	89.4
tanh	4	78.3	84.3	85.4	100	86.6	87.8	100	92.7	85.4	90.2	89.1
para_relu	3	79.5	100	85.4	87.8	86.6	90.2	87.8	84.1	84.1	87.8	87.3
relu	2	81.9	100	86.6	89	86.6	81.7	86.6	82.9	87.8	87.8	87.1

training loss, but both curves remain relatively steady. Similarly, the training and testing loss accuracy curves behave in the 10% to 96.5%. On the 80th epoch, there is a slight drop in the test curve, but after that, both curves behave more or less consistently. The model accuracy and loss curves for malware detection utilizing CICMalDroid are shown in parts (e, f). *Tanh* is the most efficient activation function, as can be observed. The training and testing accuracy curves behave steadily between 20% and 94.2%.

Figure 7 illustrates the model training and loss curves for malware detection and classification using static embedding. Using both datasets, the result shows the most efficient activation function. Using the CIC-AAGM2017 and CICMalDroid datasets, the *relu* activation function is the most efficient for malware classification. However, utilizing the CICMalDroid dataset, the elu activation function is the most effective for malware detection. In part (a), the training and testing curves for the model accuracy behave between 10% and 99.80 for the malware classification using CIC-AAGM2017. Similarly, the training and test accuracy curves behave between 5% and 99.85% for malware classification using the CICMalDroid dataset. Parts (e, f) show the model accuracy and model for malware detection using the CICMalDroid dataset. The training and testing curves for model accuracy behave between 18% and 99.83%. Figure 8 shows the model accuracy and model loss curves for both datasets using dynamic embedding. The *relu* is the most efficient activation function for malware classification while the elu is the most activation function for malware detection using dynamic embedding. The training curve starts from 85% while the test starts from 50%. They both gradually increase up to 99.9% in the 40th epoch. There is a drop-in training accuracy of up to 70% and then again increases up to maximum. Again, there is a slight drop up to 80% in test accuracy but after that, both curves behave more or less constantly. Collectively, both curves behave between 50% and 99.88%. Similarly, part (c) shows the model accuracy for training and testing curves. These curves behave between 65% and 99.95% for malware classification. Parts (e, f) show the model accuracy and loss curves for malware detection using the CICMalDroid dataset. The training and testing accuracy curves between 50% and 99.90%. It can be seen that dynamic embedding outperforms as compared to the random and static embedding approaches. Similarly,

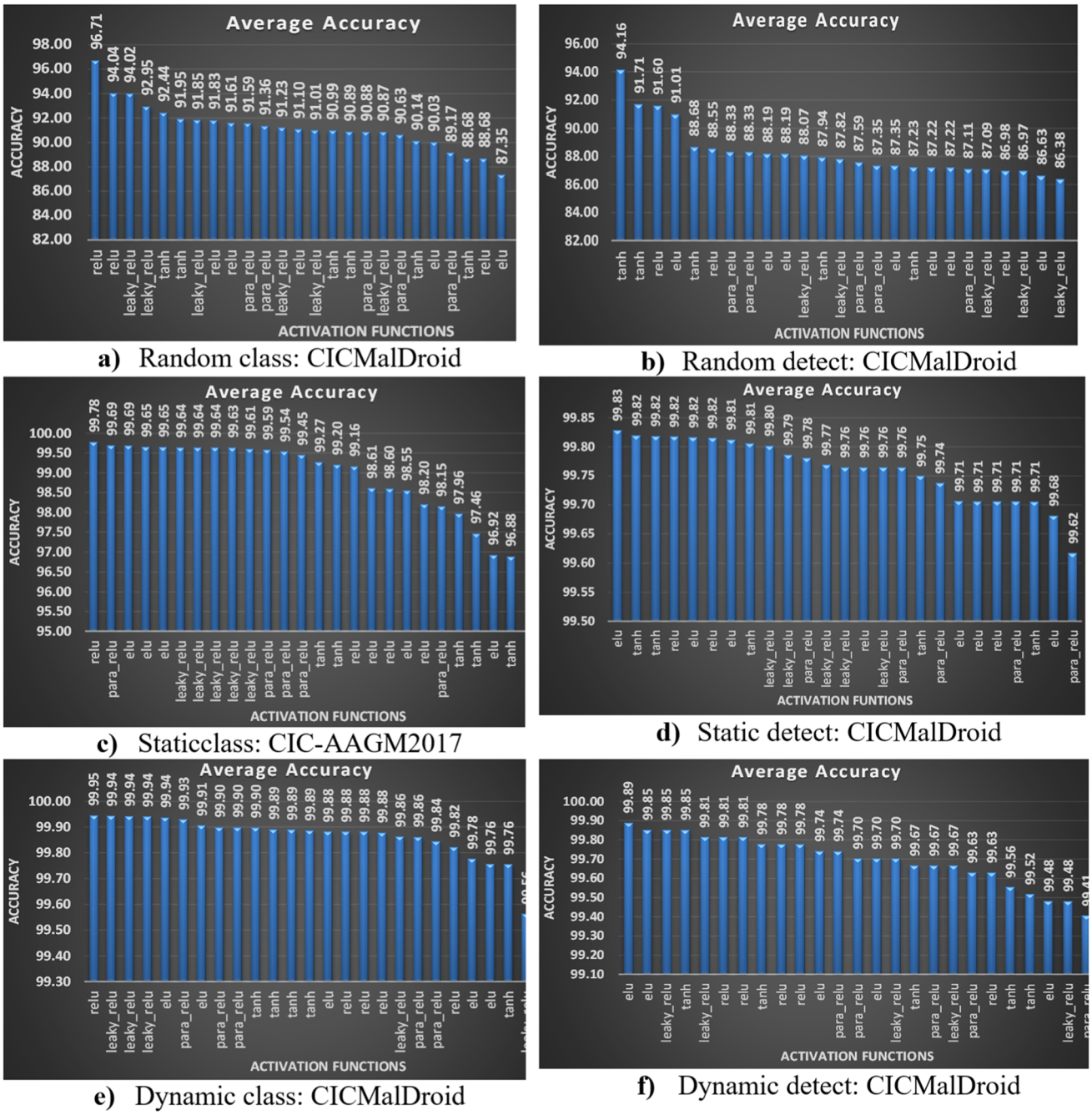

Fig. 5 Average accuracy comparisons with different activations functions using fine-tune embedding using 10-fold

the random performs the worst as compared to the other two approaches.

The experiments are conducted to demonstrate performance indicators for malware classification and detection using both datasets. We used five performance indicators, such as Precision, Recall, F1-score, and Accuracy for three different types of embeddings: random, static, and dynamic. Experiments are carried out for five activation functions, each with a distinct set of hyperparameters for the kernel and filters, and each with 10-fold cross-validation. Table 5 depicts the classification performance of the top three activation functions for random embedding. For the CIC-AAGM2017 dataset, Precision, Recall, F1-score, and Accuracy for the malware classification are 97.03%, 97.14%, 97.08%, and 97.19%, respectively, for the *tanh* activation function. The lowest classification performance for the same dataset is 94.2%, 93.92%, 94.05%, and 94.40%, respectively. The top three activation functions for malware classification using the same dataset are *tanh*, *para relu*, and *relu*. The performance measures

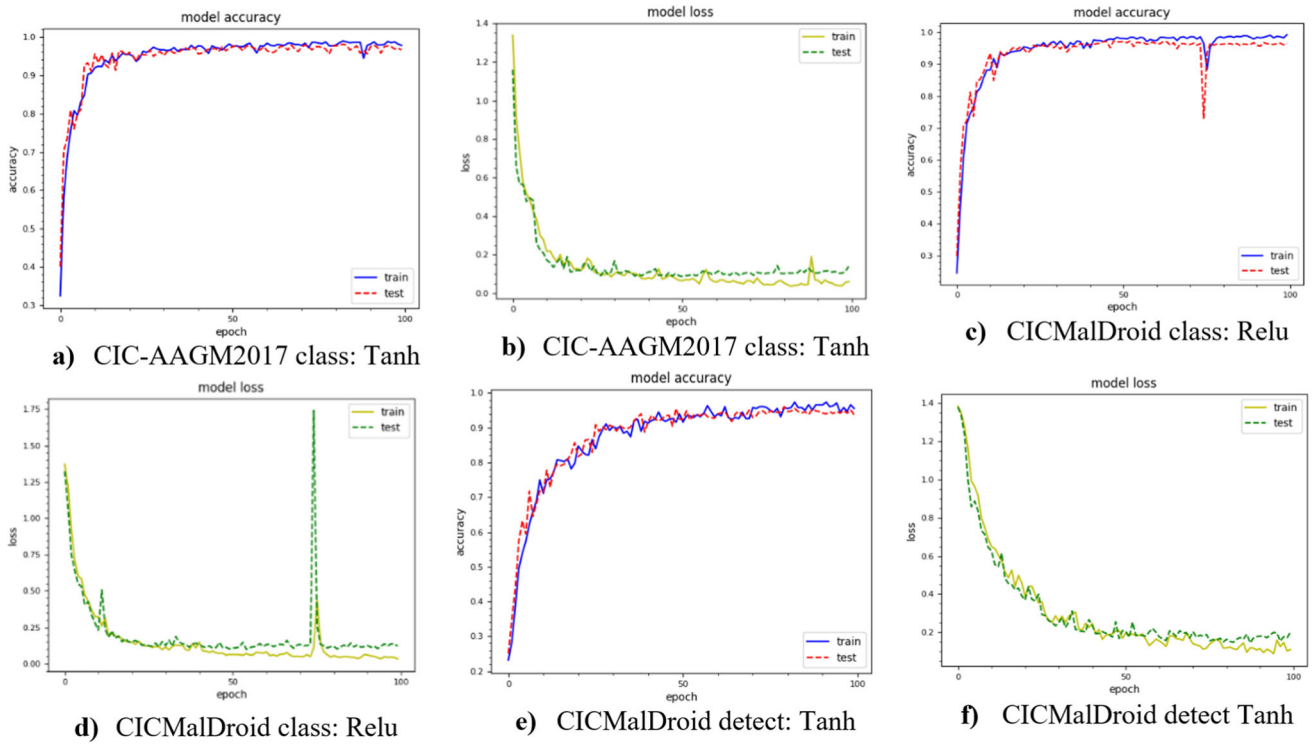

Fig. 6 Random embedding: epoch curves for top activation function using 1-fold

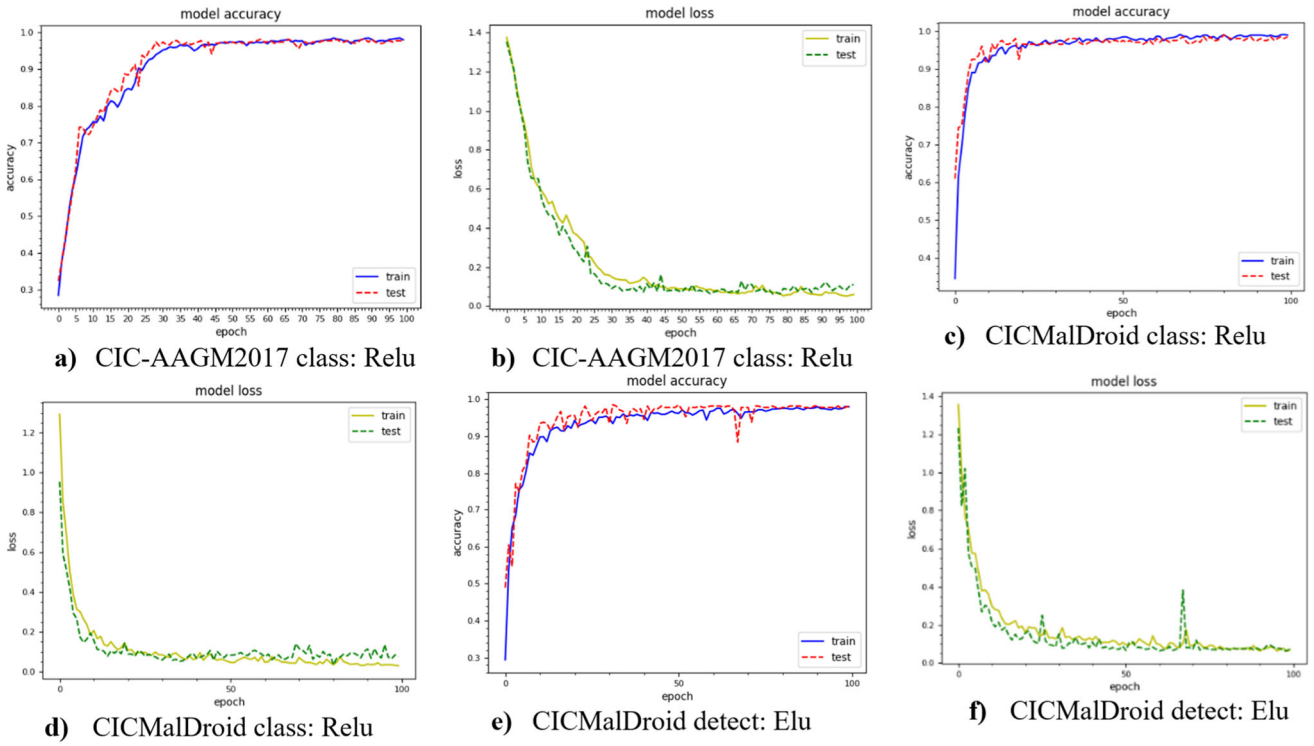

Fig. 7 Static embedding: epoch curves for top activation function using 1-fold

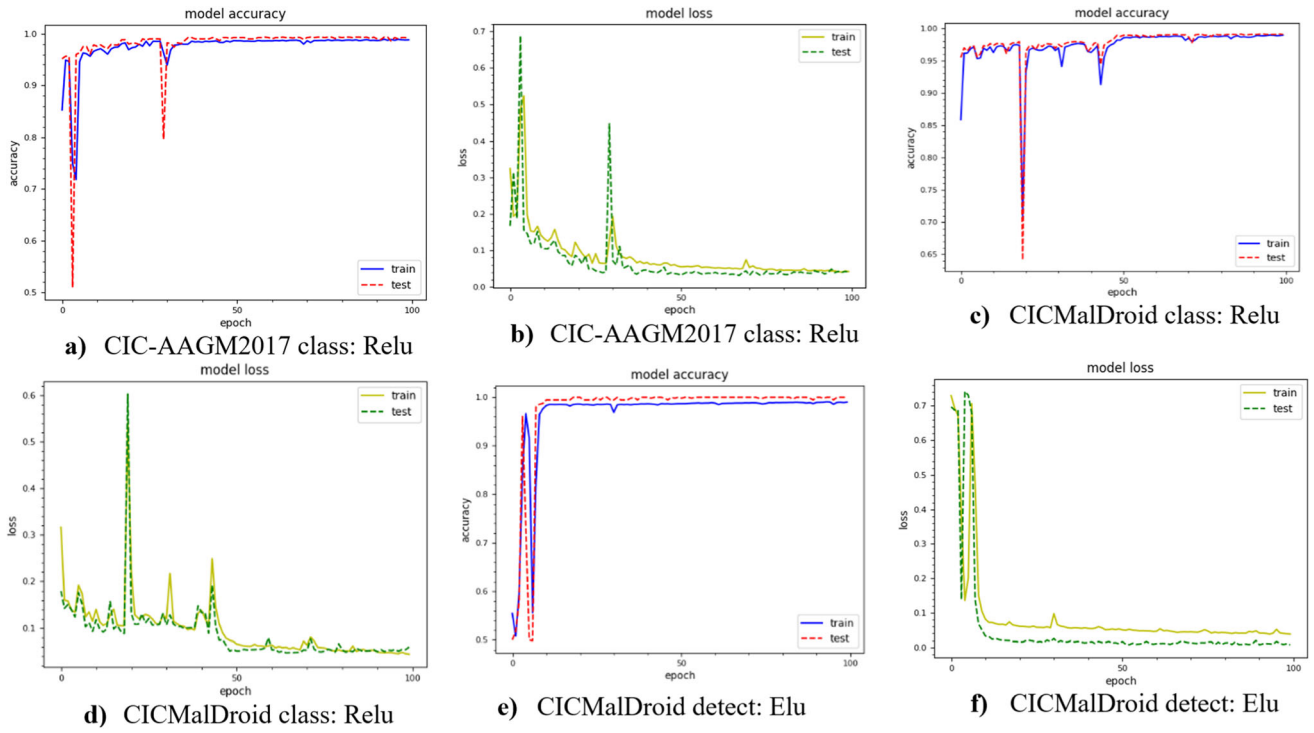

Fig. 8 Dynamic embedding: epoch curves for top activation function using 1-fold

for malware classification and detection for the CICMalDroid dataset are 96.68%, 96.62%, 96.64%, 97.71%, and 91.56%, 91.38%, 91.46%, and 91.6%, respectively. For the CICMalDroid dataset, the top three activation functions for malware classification and detection are (*relu*, *relu*, *leaky_relu*), (*tanh*, *tanh*, *relu*). Table 6 shows the performance indicators for static embedding with the top 3 activation function functions. For the CIC-AAGM2017 dataset, Precision, Recall, F1-score, and Accuracy are 99.64%, 99.44%, 99.53%, and 99.78%, respectively using the *relu* activation function. For malware detection using these indicators 99.78%, 99.7%, 99.74%, and 99.83%, respectively using the CICMalDroid dataset. The top activation functions for malware classification and detection for both datasets are (*relu*, *para_relu*, *elu*), (*relu*, *elu*, *elu*), (*elu*, *tanh*, *tanh*), respectively. Table 7 shows the same performance indicators for the dynamic embedding using both datasets and 10-fold cross-validation. It can be seen that; the dynamic embedding performs better with all five performance indicators as compared to the random and static approach. Thus, it is proved that the fine-tuned embedding with a dynamic approach provides the highest performance for both malware classification and detection.

The proposed method is compared to other published research in Table 8. When classifying Android malware, these studies focused primarily on using data collected from network traffic. Aresu et al. [18] demonstrate how to categorize Android malware by assessing HTTP packets. To achieve this, it evaluates HTTP incoming traffic to cluster malicious files. Additionally, signatures for detecting new clustered malicious files can be extracted using this method with a detection rate of 98.66%. A Droid Classifier was introduced by Li et al. [20]. This classifier instantly constructs various models based on a collection of compiled malicious apps. A set of common identifiers extracted from network traffic is used to construct each model. To accurately portray a wide variety of malware characteristics, a system with automatic threshold configurations was developed, and it achieved a 94.66% success rate. The concept of classifying malicious programs based on their URLs was presented by Shanshan et al. [25]. Multi-view deep learning models provide a broader and deeper perspective for malware analysis. Additionally, it creates and disseminates soft attention-weighting components for use with particular data. URL-based malware classification has a 95.74% success rate. Shyong et al. [39] categorize Android apps by combining static permission with changing network tracking. Hazardous network flows are used to acquire features in the dynamic evaluation step, and Random Forest is employed to recognize malicious files. Malware detection rates on Android average 98.86%. A method for identifying malicious Android apps through their URLs was introduced by Shanshan et al. [28]. Malware detection models that prioritize feature depth are built using multi-view neural networks. The feature weights are distributed so that they can be used with specific inputs.

Table 5 Random embedding: Performance indicators for top 3 activations function using 10-fold

Classification CIC-AAGM2017	Activation	Filters	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
	Tanh	1	97.03	97.14	97.08	97.19
	Para_relu	4	94.27	93.88	94.07	94.63
	Relu	4	94.2	93.92	94.05	94.40
CICMalDroid						
Classification	Relu	5	96.68	96.62	96.64	96.71
	Relu	4	93.84	93.72	93.77	94.04
	Leaky_relu	1	93.82	93.7	93.75	94.02
Detection	Tanh	1	94.09	93.99	94.03	94.16
	Tanh	3	91.68	91.52	91.57	91.71
	Relu	4	91.56	91.38	91.46	91.6

Table 6 Static embedding: Performance indicators for top 3 activation functions using 10-fold

Classification CIC-AAGM2017	Activation	Filters	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
	Relu	5	99.64	99.44	99.53	99.78
	Para_relu	2	99.45	99.18	99.31	99.69
	Elu	4	99.24	99.08	99.15	99.69
CICMalDroid						
Classification	Relu	1	99.82	99.78	99.8	
	Elu	3	99.76	99.72	99.74	99.77
	Elu	4	99.72	99.68	99.7	99.74
Detection	Elu	3	99.78	99.7	99.74	99.83
	Tanh	3	99.75	99.65	99.69	99.82
	Tanh	5	99.74	99.62	99.67	99.82

Table 7 Dynamic embedding: Performance indicators for top 3 activation functions using 10-fold

Classification CIC-AAGM2017	Activation	Filters	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
	Relu	1	99.84	99.78	99.81	99.88
	Tanh	3	99.84	99.76	99.8	99.88
	Tanh	2	99.76	99.66	99.7	99.85
CICMalDroid						
Classification	Relu	1	99.92	99.92	99.92	99.95
	Leaky_relu	4	99.9	99.88	99.89	99.94
	Leaky_relu	2	99.9	99.84	99.87	99.94
Detection	Elu	5	99.82	99.81	99.81	99.89
	Elu	3	99.78	99.72	99.74	99.85
	Leaky_relu	2	99.76	99.72	99.74	99.85

Ficco et al. [40] combine API calls and ensemble detectors to efficiently incorporate generic and specialized features. It works to improve performance for unknown malware families with an accuracy of 98% throughout the analysis process and raises the volatility of the detection technique. Ullah et al. [41] classified Android malware using a hybrid approach combining Control Flow Graphs (CFGs) and picture attributes. API Call Graphs (ACGs) are extracted from CFGs to examine the dynamic behavior of harmful operations and provide 99.27% accuracy. Ullah et al. [42] proposed an Intrusion Detection System for Imbalanced Network Traffic (IDS-INT) that employs a combination of transfer learning CNN-LSTM. IDS-INT analyzes feature associations in both network feature representation and imbalanced data to obtain 99.21% accuracy. The proposed method is highly accurate, at 98%. With a detection rate of 99.75%, our method easily wins out over the competition. Table 9 shows the comparison of

the proposed approach with state-of-the-art methods. It can be seen that the proposed approach provides the best classification results. Algorithm execution time and storage space depend on computational complexity. Table 10 shows the computational complexities for each algorithm. The costly terms of the proposed scheme are defined in Algorithms 1, 2, and 3 respectively.

5 Performance validation with explainable AI and t-SNE

We derived a subset of the most crucial features from the embedded matrix in order to better understand and verify the proposed method. A visual representation of the significance of the features among the 25 features is shown in Fig. 9. It is clear that the "F17" feature is the most efficient, contributing the most to the detection of malware classes. In contrast, the "F8" function is the least efficient and may deliver the worst results for the proposed method. In terms of efficiency, the "F10" function is the next best feature. Consequently, we can easily rank the features from most significant to least.

We used the Local Interpretable Model-agnostic Explanation (LIME) and SHAPSHapley Additive exPlanations (SHAP) libraries to explain how each feature affected model output [43]. Figure 10 depicts the influence of each attribute on the model output, both individually and combined. Parts (a, and b) represent the combined and individual effects, respectively. In part (a), the color ranges show the contribution of each feature. For instance, the red color implies a significant contribution, but the blue color denotes a low contribution. Part (b) shows that the red color shows the contribution of each feature towards the malware class, while the green color shows the contribution towards the benign class. It can be seen that the combined effect of the "F17" feature is significant, and to validate this, the same feature adds significantly to the malware class in part (b). The cumulative effect of feature "F25" is the lowest, and to validate this, the same feature contributes the least to the malware class. The color of the feature "F16" is red in part (b), which indicates that this feature has the highest impact and contribution. To check this, the same feature has a better contribution in the benign class. We can now easily explain the impact of each feature for a given category, such as malicious or benign [44]. This study demonstrates how each feature influences the output by assessing its efficacy. The objective of the t-distributed Stochastic Neighbor Embedding (t-SNE) method of visual representation is to determine whether the features retain extensive or limited knowledge. The t-SNE technique is also meant to assess how well the proposed method works. High-dimensional data can be visualized using the t-SNE method, which was proposed by Maaten et al. [45]. The ratio of local to global scores for semantic and syntactic features is depicted in Fig. 11. In the first experiment, we try to determine the threshold of complexity beyond which it becomes impossible to distinguish between benign and

Table 8 Comparisons with previously published works

Work	Year	Methods	Accuracy (%)
Aresu et al. [18]	2015	Signature-based clustering	96.66
Li et al. [20]	2016	Droid classifier	94.66
Shanshan et al. [25]	2018	Skip-gram with neural network	95.74
Shanshan et al. [17]	2019	C4.5 decision tree	97.89
Shyong et al. [39]	2020	Random forest	98.86
Shanshan et al. [28]	2020	Multi-view neural network	98.81
Ficco et al. [40]	2021	API-calls with ensemble	98.00
Ullah et al. [41]	2022	CFG with stack ensemble	99.27
Ullah et al. [42]	2023	transfer learning with CNN-LSTM	99.21
NMal-Droid	–	Fine-tune embedding with CNN-BiGRU	99.75

Table 9 Comparisons with state-of-the-art methods

Methods	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
Random Forest	94.21	93.84	93.91	94.08
CNN-1D	95.32	95.48	94.66	95.58
LSTM	95.84	94.92	96.14	96.11
RNN	96.82	96.94	96.40	96.89
NMal-Droid	99.81	99.73	99.70	99.75

malware. The second experiment determined that the malware clusters with the highest perplexity scores are the most effective ones for Android. Parts (a) and (c) have the lowest perplexity values, whereas parts (b) and (d) have the highest. As a method for classifying samples, t-SNE relies on repeated calculations. To illustrate the various classes of malware and benign files, we ran 500 iterations for each perplexity factor. Classification outcomes are drastically affected by the dataset's density. In most cases, higher densities result in better accuracy since more qualitative data is used during the training process. When using the optimal perplexity settings, the visual clusters produced by t-SNE are more clearly divided. An appropriate perplexity value can be used to partition a dataset, and then useful hyperparameters can be used to classify the partitions. This process proved the method's efficacy by extracting conceptual aspects and classifying them as malware or benign.

Table 10 Computational complexity analysis

Complexity terms	Algorithm 1	Algorithm 2	Algorithm 3
P, F, TF	$\|P\|$	$n\|f\|, n\|tf\|$	$n\|f\|, n\|tf\|$
$P^{'}, T_r, T_e$	$n\|P\|, \|n\|$	$\|T_r\|, \|T_e\|$	$\|T_r\|, \|T_e\|$
$PCAP, T_r^{'}$	$n\|P\|$	–	$\|T_r\| + \|T_e\|$
$HTTP, TCP, T_e^{'}$	$n\|P\|$	–	$\|T_r\| + \|T_e\|$

6 Conclusion

Malware activities presently cause significant harm to the integrity of Android apps. These hazards have the potential to steal sensitive data and cause havoc in the commerce, social system, and financial industry. Because of their continual network access, Android apps are easily attacked by network activity. This study presents the NMal-Droid malware detection approach, which employs fine-tuned embedding and an ensemble neural network. The packet parsing technique is used to filter network traffic for combined HTTP and TCP flow events. A fine-tune embedding technique is developed that investigates feature embedding using a pre-trained word2vec model. The embedded features are extracted using three different ways such as random, static, and dynamic. It's a tool for learning and extracting feature-matrix matrices with similar meanings. The CNN model is designed to extract the deep features and then Bi-GRU is used to compute the gradient information for each feature. To take the benefits from both models, an ensemble of multi-head CNN-BiGRU is developed for accurate malware classification and detection. For a more detailed analysis, the proposed method is evaluated on five separate activation functions with 100 filters and a range of 1-5 kernel sizes. An explainable AI-based experiment is conducted to evaluate and validate the proposed method. According to the results, the proposed

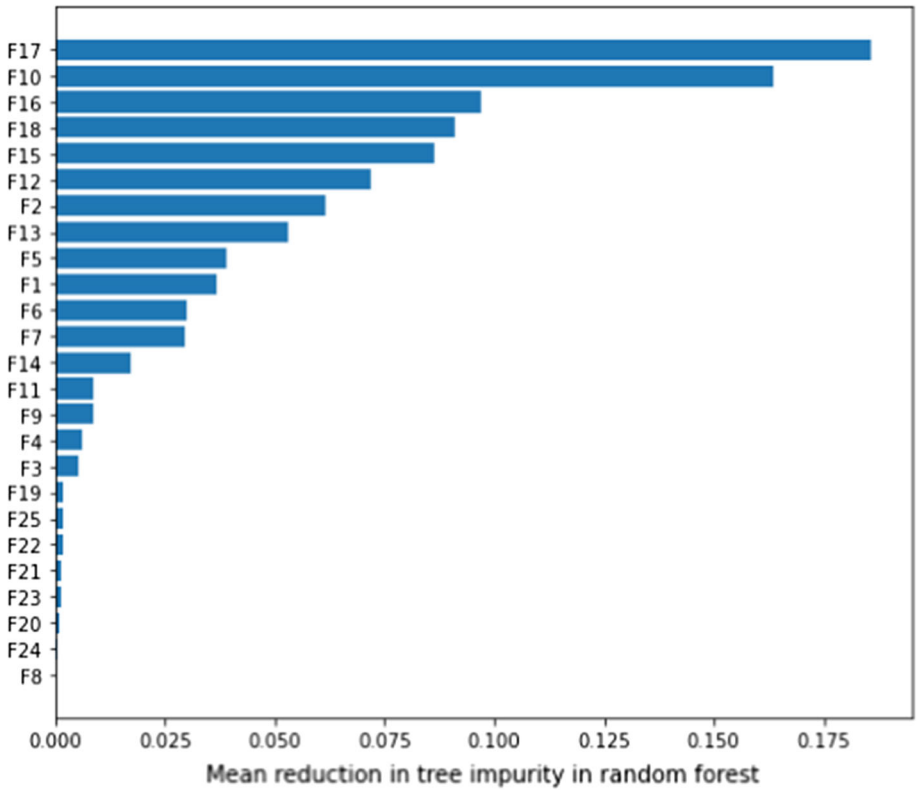

Fig. 9 Important features extraction

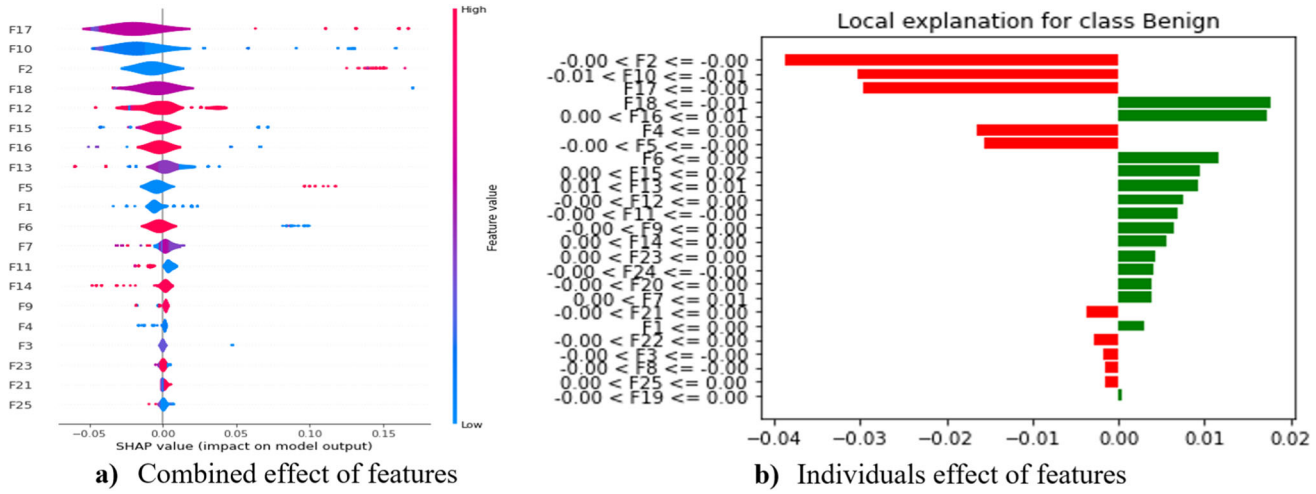

Fig. 10 Combined and individual effects of features on model output

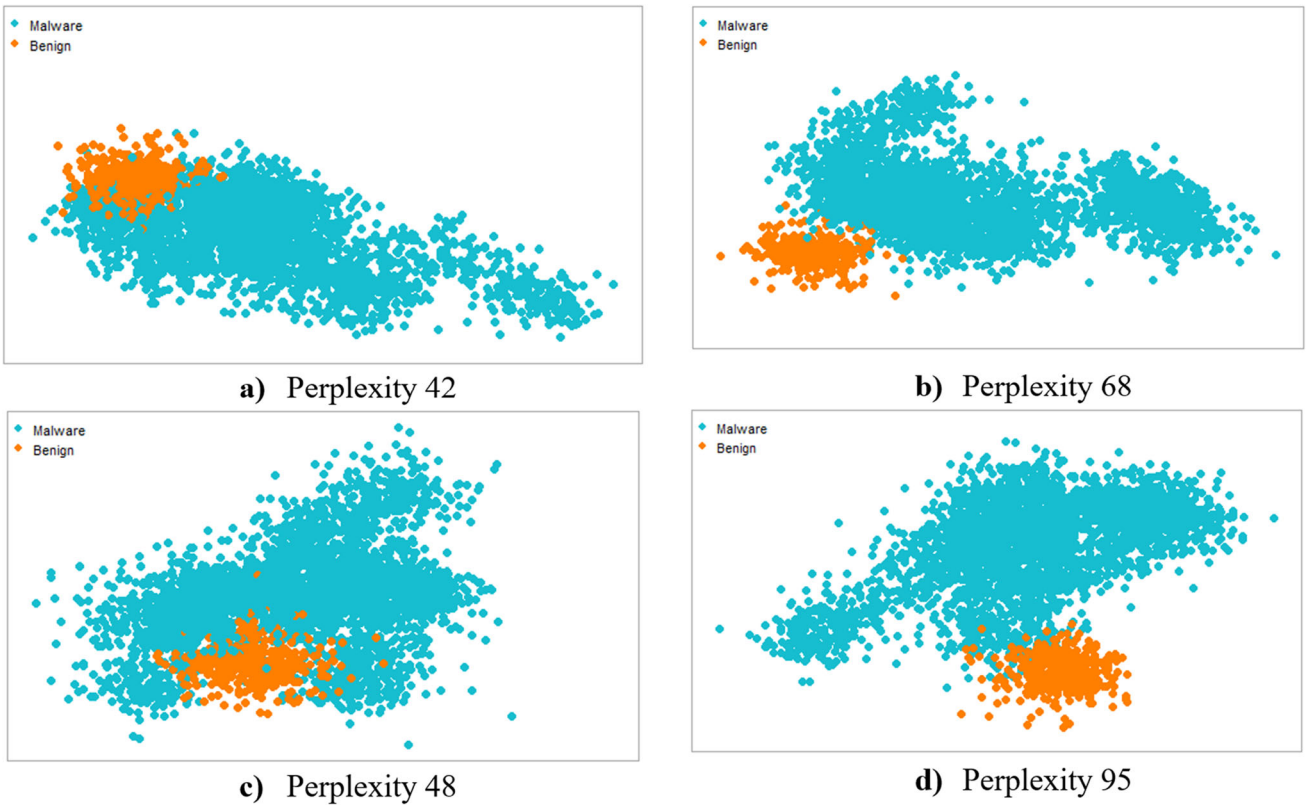

Fig. 11 t-SNE visualization for fused features using minimum (42, and 68) and optimal (48, and 95) perplexity values

strategy beats current methods with a precision of 99.81%, recall of 99.73%, f1-score of 99.7%, and accuracy of 99.75%.

In the future, pre-trained models such as GloVe, BERT, and FastText can be used to improve the fine-tuned embedding. Experiment with the two hyperparameters by varying the kernel size from 1 to 10 and adding or removing filters to see how it affects model performance.

Author Contributions FU proposed the study, simulated it, and wrote the manuscript. SU helped in writing algorithms and formatting. YZ reviewed. and made writing suggestions. GS and JC-WL reviewed

and analyzed the proposed research. All authors have read and agreed to the published version of the manuscript

Funding Funding was provided by Natural Sciences and Engineering Research Council of Canada (Grant no. RGPIN-2020-05363).

Data availability The data that support the findings of this study are openly available in Canadian Institute for Cybersecurity-CIC-AAGM2017 and CICMalDroid2020 at https://www.unb.ca/cic/datasets/android-adware.html, and https://www.unb.ca/cic/datasets/maldroid-2020.html, respectively.

Declarations

Conflict of interest The authors declare no conflict of interest.

Ethical approval This article does not contain any studies with human participants performed by any of the authors.

References

1. Arshad, S., Shah, M. A., Khan, A., & Ahmed, M. (2016). Android malware detection & protection: A survey. *International Journal of Advanced Computer Science and Applications, 7*(2), 463–475.
2. Felt, A.P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011). A survey of mobile malware in the wild. In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*.
3. Berman, D. S., Buczak, A. L., Chavis, J. S., & Corbett, C. L. (2019). A survey of deep learning methods for cyber security. *Information, 10*(4), 122.
4. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2014). Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys & Tutorials, 17*(2), 998–1022.
5. Zhu, H.-J., You, Z. H., Zhu, Z. X., Shi, W. L., Chen, X., & Cheng, L. (2018). DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing, 272*, 638–646.
6. Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., & Xiang, Y. (2020). A survey of android malware detection with deep neural models. *ACM Computing Surveys (CSUR), 53*(6), 1–36.
7. Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2008). A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR), 44*(2), 1–42.
8. Zhou, Y. & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. In: 2012 IEEE symposium on security and privacy. IEEE.
9. Yang, L., Han, Z., Huang, Z., & Ma, J. (2018). A remotely keyed file encryption scheme under mobile cloud computing. *Journal of Network and Computer Applications, 106*, 90–99.
10. Jiang, J., Yin, Q., Shi, Z., & Li, M. (2018). Comprehensive behavior profiling model for malware classification. In: 2018 IEEE Symposium on Computers and Communications (ISCC). IEEE.
11. Chen, Z., Peng, L., Gao, C., Yang, B., Chen, Y., & Li, J. (2017). Flexible neural trees based early stage identification for IP traffic. *Soft Computing, 21*(8), 2035–2046.
12. Talha, K. A., Alper, D. I., & Aydin, C. (2015). APK Auditor: Permission-based Android malware detection system. *Digital Investigation, 13*, 1–14.
13. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., & Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security, 9*(11), 1869–1882.
14. Ullah, F., Srivastava, G., & Ullah, S. J. J. O. C. C. (2022). *A malware detection system using a hybrid approach of multi-heads attention-based control flow traces and image visualization., 11*(1), 1–21.
15. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., & Alvarez, G. (2013). Puma: Permission usage to detect malware in android. *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions.* Springer.
16. de la Puerta, J.G., Sanz, B., Santos Grueiro, I., & Bringas, P. G. (2015). The evolution of permission as feature for Android malware detection. In: Computational Intelligence in Security for Information Systems Conference. Springer.
17. Liu, X. & Liu, J. (2014). A two-layered permission-based android malware detection scheme. In: 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering. IEEE.
18. Wang, S., Chen, Z., Yan, Q., Yang, B., Peng, L., & Jia, Z. (2019). A mobile malware detection method using behavior features in network traffic. *Journal of Network and Computer Applications, 133*, 15–25.
19. Aresu, M., Ariu, D., Ahmadi, M., Maiorca, D., & Giacinto, G. (2015). Clustering android malware families by http traffic. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). IEEE.
20. Wang, S., Yan, Q., Chen, Z., Yang, B., Zhao, C., & Conti, M. (2017). Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security, 13*(5), 1096–1109.
21. Li, Z., Sun, L., Yan, Q., Srisa-an, W., & Chen, Z. (2016). Droidclassifier: Efficient adaptive mining of application-layer header for classifying android malware. in International Conference on Security and Privacy in Communication Systems. Springer.
22. Wang, S., Chen, Z., Zhang, L., Yan, Q., Yang, B., Peng, L., & Jia, Z. (2016). Trafficav: An effective and explainable detection of mobile malware behavior using network traffic. In: 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS). IEEE.
23. Vierthaler, J., Kruszelnicki, R. & Schutte, J. (2018). Webeye-automated collection of malicious http traffic. arXiv preprint arXiv:1802.06012.
24. Naeem, H., Ullah, F., Naeem, M. R., Khalid, S., Vasan, D., Jabbar, S., & Saeed, S. (2020). Malware detection in industrial Internet of Things based on hybrid image visualization and deep learning model. *Ad Hoc Networks, 105*, 102154.
25. Xu, P., Eckert, C., & Zarras, A. (2021). Falcon: Malware detection and categorization with network traffic images. In: International Conference on Artificial Neural Networks. Springer.
26. Wang, S., Chen, Z., Yan, Q., Ji, K., Wang, L., Yang, B., & Conti, M. (2018). Deep and broad learning based detection of android malware via network traffic. In: 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS). IEEE.
27. Chen, Z., Yu, B., Zhang, Y., Zhang, J., & Xu, J. (2016). Automatic mobile application traffic identification by convolutional neural networks. In: 2016 IEEE Trustcom/BigDataSE/ISPA. IEEE.
28. David, O. E. & Netanyahu, N. S. (2015). Deepsign: Deep learning for automatic malware signature generation and classification. In: 2015 International Joint Conference on Neural Networks (IJCNN). IEEE.
29. Wang, S., Chen, Z., Yan, Q., Ji, K., Peng, L., Yang, B., & Conti, M. (2020). Deep and broad URL feature mining for android malware detection. *Information Sciences, 513*, 600–613.

30. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems,* 26.
31. Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2018). MalDozer: Automatic framework for android malware detection using deep learning. *Digital Investigation, 24*, S48–S59.
32. Qiao, Y., Zhang, W., Du, X., & Guizani, M. (2021). Malware classification based on multilayer perception and word2Vec for IoT security. *ACM Transactions on Internet Technology (TOIT), 22*(1), 1–22.
33. Lee, W. Y., Saxe, J., & Harang, R. (2019). SeqDroid: Obfuscated Android malware detection using stacked convolutional and recurrent neural networks, In: Deep learning applications for cyber security. Springer. pp. 197–210.
34. Vasan, D., Alazab, M., Wassan, S., Safaei, B., & Zheng, Q. (2020). Image-based malware classification using ensemble of CNN architectures (IMCEC). *Computers & Security, 92*, 101748.
35. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555
36. Bai, H., Liu, G., Liu, W., Quan, Y., & Huang, S. (2021). N-gram, semantic-based neural network for mobile malware network traffic detection. *Security and Communication Networks,* 2021.
37. Lashkari, A.H., Kadir, A. F. A., Gonzalez, H., Mbah, K. F., & Ghorbani, A. A. (2017). Towards a network-based framework for android malware detection and characterization. In: 2017 15th Annual conference on privacy, security and trust (PST). IEEE.
38. Mahdavifar, S., Kadir, A. F. A., Fatemi, R., Alhadidi, D., & Ghorbani, A. A. (2020). Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE.
39. Shyong, Y.-C., Jeng, T.-H., & Chen, Y.-M. (2020). Combining static permissions and dynamic packet analysis to improve android malware detection. In: 2020 2nd International Conference on Computer Communication and the Internet (ICCCI). IEEE.
40. Ficco, M. (2021). Malware analysis by combining multiple detectors and observation windows. *IEEE Transactions on Computers, 71*(6), 1276–1290.
41. Ullah, F., Srivastava, G., & Ullah, S. (2022). A malware detection system using a hybrid approach of multi-heads attention-based control flow traces and image visualization. *Journal of Cloud Computing, 11*(1), 1–21.
42. Ullah, F., Ullah, S., Srivastava, G., & Lin, J. C. W. (2023). IDS-INT: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic. *Digital Communications and Networks.*
43. Mathews, S.M. (2019). Explainable artificial intelligence applications in NLP, biomedical, and malware classification: A literature review. In: Intelligent computing-proceedings of the computing conference. Springer.
44. Ullah, F., Alsirhani, A., Alshahrani, M. M., Alomari, A., Naeem, H., & Shah, S. A. (2022). Explainable malware detection system using transformers-based transfer learning and multi-model visual representation. *Sensors, 22*(18), 6766.
45. Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research, 9*(11).

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Farhan Ullah is an Associate Professor at the School of Software, Northwestern Polytechnical University (NPU), Xi'an Shaanxi, P.R. China. He received Ph.D. Computer Science degree in 2020 from College of Computer Science, Sichuan University Chengdu, P.R. China. He was awarded a full-time Chinese Government Scholarship (CGS) for his Ph.D. He performed as a Co-PI of the science and technology project of Taicang (2020), Jiangsu, China. He served as a Special Issue Lead Guest Editor for Security and Communication Networks Journal. He also served as a Guest Editor (GE) for a special issue of Future Internet Journal, MDPI. He received research landmark achievement award from School of Software, NPU, Xian, China. He also received Research Productivity Award (RPA) from COMSATS Institute of Information Technology (CIIT), Sahiwal, Pakistan in 2016. His research work is published in various renowned journals of IEEE, Springer, Elsevier, Wiley, MDPI, and Hindawi.

Shamsher Ullah postdoctoral fellow at the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. Dr. Shamsher received Postdoctrate certificate from the School of Software, Northwestern Polytechnical University (NPU), Taicang Campus, Jiangsu, P.R. China. Dr. Shamsher received his Ph.D. degree from the School of Computer Science and Technology at the University of Science and Technology of China (USTC), Hefei, Anhui, P.R. China. He received Master Degree at the Department of Information Technology, Hazara University Mansehra, KPK, Pakistan in 2015. His research work is published in reputed journals and conferences. His research interest includes Cryptography, Information Security, Privacy, Data Trading, and E-commerce.

Gautam Srivastava was awarded his B.Sc. degree from Briar Cliff University in U.S.A.in the year 2004, followed by his M.Sc. and Ph.D. degrees from the University of Victoria in Victoria, British Columbia, Canada in the years 2006 and 2012, respectively. He was promoted to the rank of Professor in January 2023 at Brandon University, Canada. In his academic career, he has published a total of 400 papers in high impact conferences in many countries and in high-status journals (SCI, SCIE) and has also delivered invited guest lectures on Big Data, Cloud Computing, Internet of Things, and Cryptography. He is an Editor of several international scientific research journals. His research program is funded federally by the Natural Sciences and Engineering Research Council of Canada (NSERC) and MITACS.

Jerry Chun-Wei Lin received his Ph.D. from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan in 2010. He is currently a full Professor with the Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway. He has published more than 300 research articles in refereed journals (IEEE TKDE, IEEE TCYB, ACM TKDD, ACM TDS) and international conferences (IEEE ICDE, IEEE ICDM, PKDD, PAKDD). His research interests include data mining, soft computing, artificial intelligence and machine learning, and privacy-preserving and security technologies. He is also the project co-leader of the well-known SPMF: An Open-Source Data Mining Library, which is a toolkit offering multiple types of data mining algorithms. He also serves as the Editor-in-Chief of the International Journal of Data Science and Pattern Recognition. He is the IET Fellow, senior member for both IEEE and ACM.

Yue Zhao received the B.S. degree in computer science and technology and the M.S. degree in computer application from the School of Computer Science and Technology, Jilin University, Jilin, China, in 2006 and 2009, respectively, the Ph.D. degree in computer science from the University of Arkansas—Little Rock, Arkansas, USA, in 2015. He is currently an Associate Professor with the School of Software, Northwestern Polytechnical University, Xi'an, China. His research interests include big data analytics, artificial intelligence, machine learning, bioinformatics.

Authors and Affiliations

Farhan Ullah[1] · Shamsher Ullah[2] · Gautam Srivastava[3,5,6] · Jerry Chun-Wei Lin[4] · Yue Zhao[1]

✉ Gautam Srivastava
srivastavag@brandonu.ca

Farhan Ullah
farhan@nwpu.edu.cn

Shamsher Ullah
shamsher.ullah@skt.umt.edu.pk

Jerry Chun-Wei Lin
jerrylin@ieee.org

Yue Zhao
yxzhao01@nwpu.edu.cn

[1] School of Software, Northwestern Polytechnical University, Xi'an 710072, Shanxi, People's Republic of China

[2] School of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518000, People's Republic of China

[3] Department of Mathematics and Computer Science, Brandon University, Brandon, MB R7A 6A9, Canada

[4] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, 5063 Bergen, Norway

[5] Research Centre for Interneural Computing, China Medical University, Taichung 40402, Taiwan

[6] Department of Computer Science and Math, Lebanese American University, 1102, Beirut, Lebanon