Betalogger: Smartphone Sensor-based Side-channel Attack Detection and Text Inference Using Language Modeling and Dense MultiLayer Neural Network

ABDUL REHMAN JAVED and SAIF UR REHMAN, Air University, Pakistan
MOHIB ULLAH KHAN, University of Wah, Pakistan
MAMOUN ALAZAB, Charles Darwin University, Australia
HABIB ULLAH KHAN, Qatar University, Qatar

With the recent advancement of smartphone technology in the past few years, smartphone usage has increased on a tremendous scale due to its portability and ability to perform many daily life tasks. As a result, smartphones have become one of the most valuable targets for hackers to perform cyberattacks, since the smartphone can contain individuals' sensitive data. Smartphones are embedded with highly accurate sensors. This article proposes *BetaLogger*, an Android-based application that highlights the issue of leaking smartphone users' privacy using smartphone hardware sensors (accelerometer, magnetometer, and gyroscope). *BetaLogger* efficiently infers the typed text (long or short) on a smartphone keyboard using Language Modeling and a Dense Multi-layer Neural Network (DMNN). *BetaLogger* is composed of two major phases: In the first phase, Text Inference Vector is given as input to the DMNN model to predict the target labels comprising the alphabet, and in the second phase, sequence generator module generate the output sequence in the shape of a continuous sentence. The outcomes demonstrate that *BetaLogger* generates highly accurate short and long sentences, and it effectively enhances the inference rate in comparison with conventional machine learning algorithms and state-of-the-art studies.

CCS Concepts: • **Security and privacy → Side-channel analysis and countermeasures;**

Additional Key Words and Phrases: Natural language processing (NLP), language modeling, text inference, cyber security, dense neural network, keylogger, side-channel attack

ACM Reference format:
Abdul Rehman Javed, Saif ur Rehman, Mohib Ullah Khan, Mamoun Alazab, and Habib Ullah Khan. 2021. Betalogger: Smartphone Sensor-based Side-channel Attack Detection and Text Inference Using Language Modeling and Dense MultiLayer Neural Network. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* 20, 5, Article 87 (June 2021), 17 pages.
https://doi.org/10.1145/3460392

Authors' addresses: A. R. Javed, Department of Cyber Security, Air University, Islamabad, Pakistan; email: abdulrehman.cs@au.edu.pk; S. ur Rehman, Faculty of Computing and AI, Air University, Islamabad, Pakistan; email: 181065@students.au.edu.pk; M. U. Khan, Faculty of Computing and Data Science, University of Wah, Islamabad, Pakistan; email: mohibullah.khan@uow.edu.pk; M. Alazab, College of Engineering, IT and Environment, Charles Darwin University, Australia; email: alazab.m@ieee.org; H. U. Khan, Department of Accounting and Information Systems, College of Business and Economics, Qatar University, Qatar; email: habib.khan@qu.edu.qa.

2375-4699/2021/06-ART87 $15.00
https://doi.org/10.1145/3460392

1 INTRODUCTION

A smartphone is a device that consolidates robust hardware sensors and a versatile mobile operating system into one unit [5, 8, 23]. Smartphones contain various sensors: magnetometer, accelerometer, a gyroscope that their software can utilize [33], and remote connectivity conventions, for example, Bluetooth, Wi-Fi, and satellite navigation [7, 11]. With the aggressive ascent in smartphones' usage, people became more dependent on these technologies, which led them to store more of their data on smart devices for convenient use [2, 17]. However, this aroused a threat to individuals' data as cyberattacks are getting more common these days [32].

The Android architecture and privacy policy permit applications to take information from multiple hardware embedded in smartphones, while iOS keeps a stricter term about allowing applications to use sensors. This iOS policy only allows a few third-party applications to access hardware sensors [15]. However, such escalated features and permissions make smartphones vulnerable to side-channel attacks [12]. A side-channel cyberattack is executed by obtaining data from the execution of a system rather than vulnerability in the executable algorithm. The periodic flow of information, system power utilization, electromagnetic data leakage, and audio waves can provide an additional cause of information exploitation [1, 3, 30]. Figure 1 graphically represents some common attacks caused by the side-channel attack. These attacks include timing attacks in which an attacker monitors the timing of different events regarding hardware, such as computations of execution timing. Another major side-channel attack includes a cache-attack based on the monitoring of cache accesses done by the victim.

Differential fault analysis is a type of side-channel attack in which anomalies are found by introducing faults in the computation. Another major side-channel attack is a Power-monitoring attack in which the attacker monitors the hardware power consumption frequency and other configurations. The electromagnetic attack is a type of side-channel attack in which a hacker can analyze electromagnetic radiations from hardware that can retrieve actual plain text and other relevant information such as cryptographic keys. Similarly, Acoustic cryptanalysis is a type of side-channel attack that exploits sensitive information through sound frequency, amplitude, and other attributes emitted from the computer hardware.

1.1 Problem Statement

Smartphones are the most valuable targets for hackers to perform cyberattacks. Hackers commonly observe to compromise an individual's sensitive data through the smartphone screen's keystrokes. We demonstrate the feasibility of inferring smartphone keystrokes accurately from smartphone sensor readings and predicting the individual's personalized writing pattern.

This article is an extension of our previous paper [15]. This article focuses on inferring short and long sentences typed on the soft keyboard of a smartphone. We extend the previously collected data by adding a space key to separate words in the sentences. This article uses a deep learning model to infer and generate sentences accurately.

The main contributions of this article are as follows:

(1) Introduce a novel approach in the domain of side-channel attacks detection named *BetaLogger* that uses a dense using **Language Modeling (LM)** and **Dense Multi-layer Neural Network (DMNN)** to predict and generate the long or short sentences typed on a smartphone keyboard.
(2) Enhances the dataset collected in our previous study [15] by adding blank spaces to separate words in the sentences.
(3) *BetaLogger* is evaluated by using DMNN and traditional machine learning algorithms: **Logistic Regression (LR)**, **Gradient Boosting (GB)**, **K-Nearest Neighbors Classifier (KNN)**, **Support Vector Machine (SVM)**, **Naive Bayes (NB)**.

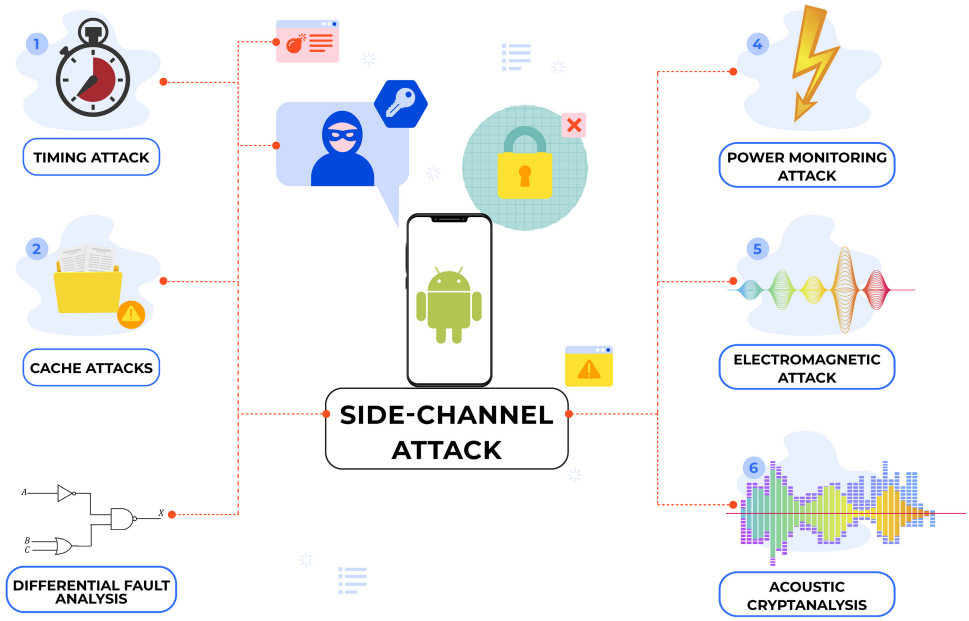


Fig. 1. Graphical representation of side-channel attacks through Android smartphone.

(4) Result shows that *BetaLogger* efficiently infers the typed text (long or short) on a smartphone keyboard and achieves the highest accuracy in comparison with traditional algorithms and state-of-the-art studies.

The article is divided into five major sections. Section 2 provides related work in the context of technical review and recent development in side-channel attacks. Section 3 presents the detailed description of the proposed *Betalogger*. The experimental evaluation and results are discussed in Section 4. Section 5 provides highlights of the article and briefly describes future work.

2 RELATED WORK

The literature includes the most recent researches, which are empowered by the robustness of software and hardware technologies. Authors of Reference [15] proposed and developed an application developed on the Android platform that runs in the background of user interactive application and gathers information at a frequency of 10 Hz/section This frequency is recorded through embedded physical sensors, which include an accelerometer, gyroscope, and magnetometer. These sensors are embedded in the smartphone to precisely analyze the keystrokes keyed on the soft keyboard. Their model attains an accuracy of 90.2% while detecting the motion-based side-channel attack. Authors of Reference [14] proposed a novel approach based on blind Zero watermarking. It detects the cyber-attacks on the software of the device by analyzing the watermark. This watermark is added by the proposed model logically into the code by using the characteristic attributes of software code and returns an output of a constructive solution. The extraction part of the system algorithm is coded to eliminate watermark using that key and distinguish tampering. On a certain tempering level, the original code is restored to defend software against attacks. Their deep learning classifier method achieves an accuracy of 93% while determining the activities.

Authors of Reference [6] performed the smartphone susceptibility assessment based on the Android operating system using machine learning algorithms for intelligent IoT applications. They

conducted an experiential analysis for over 1,406 Android applications to assess the security risk level. They applied six Machine learning techniques in which the Random Forest classification algorithm outperforms all other algorithms. Authors of Reference [22] proposed a framework named AdDroid, which acts as an **intrusion detection system (IDS)** by examining and analyzing Android applications' behaviors specific rules. This framework monitors application activities such as internet connectivity and usage, files accessing and uploading to a specific server, and installing any other application. Their framework consists of an ensemble-based classification technique in which they combine traditional Machine Learning classifiers with Adaboost classifier. AdDroid achieves an overall accuracy of 99.11% in detecting malicious applications.

Authors of Reference [20] presented an inclusive review of Android malware detection approaches using machine learning classifiers. The authors study and describe the basic Android application and operating system functionality and security policies. Then they examined and evaluated different machine learning approaches such that how they encounter the threats and vulnerabilities in Android systems. Similarly, authors of Reference [13] proposed a deep learning approach to detect Android malware at static and binary layers. Similarly, authors of Reference [31] proposed a deep learning approach to detect DDOS attacks in Android. The authors of Reference [28] proposed two endways Android malware detection approaches based on deep learning models, which include **Convolutional Neural Network (CNN)** and **Recurrent Neural Network (RNN)**. Their presented approach manipulates the classes.dex files of Android application by resampling the raw bytecodes. They train their model over 8,000 samples and test out on 8,000 different samples. Their model achieves an average accuracy of 94.6% while detecting malware. Authors of Reference [16] used Multistage CNN to detect the anomalies in the vehicle software. Similarly, authors of Reference [27] proposed an approach for detecting intrusion attacks on a controller area network using CNN. Authors of Reference [4] proposed an approach to supervised learning for classifying malicious Android applications. They apply reverse engineering to the Android applications to investigate the code and its structure and then apply classification and clustering to information to identify the malicious applications.

Authors of Reference [21] analyzed 260,000 Android applications declared as malware or benign by at least one Antivirus, in which 80,000 applications are tagged as malware. Their study yielded 41 different malware categories. The researchers also did a comparative analysis between antivirus frameworks that were used to detect harmful applications. The Machine Learning algorithm and Graph Community Algorithms application resulted in antivirus aid to classify unknown apps as adware or threat with an F1-score of 0.84. Authors of Reference [24] proposed a machine learning approach composed of multiple feature vectors to create a machine learning model. This model chooses optimal features for the classification of malware in an application. Their study states that their model is capable of detecting malware at different API levels. The authors of Reference [34] proposed a machine learning approach subjected to stateful event generation and performed a comparative analysis of their capabilities with a random-based Monkey approach. One of the presented research is associated with Droidbot, and the other one is a hybrid approach that syndicates the state-based and random-based methods. They examine three separate input techniques in the context of their capability to record functional behavioral features and the influence on several machine learning algorithms that apply the behavioral features to detect malware.

The authors of Reference [29] proposed a method to prevent the cyber-attack from bypassing the security system by falsifying the time-domain numerical features of original signals. For this purpose, they implement the MFCC algorithm in the feature extraction segment. With the combination of machine learning classifiers, their model achieves an accuracy of 96.71% in detecting attacks. The authors of Reference [25] Examine and conclude that attackers can increase the severity of the attack by using deep learning techniques. The researchers propose a noise injection

Table 1. Comparison with Existing Works

Reference	Purpose	Limitation (s)
[26]	Smartphone keystroke inference	Low accuracy and limited to recognize keystrokes within the applications
[22]	Intrusion detection system (IDS)	Does not focus on side channel attack
[6]	Smartphone susceptibility assessment	Does not focus on side channel attack
[20]	Review of Android malware detection approaches	Does not focus on side channel attack
[12]	Survey on smartphones vulnerability assessment to keylogger side-channel attacks	Does not provide detection mechanism
[15]	Smartphone keystroke inference	Low accuracy and does not predict personalized pattern of writing

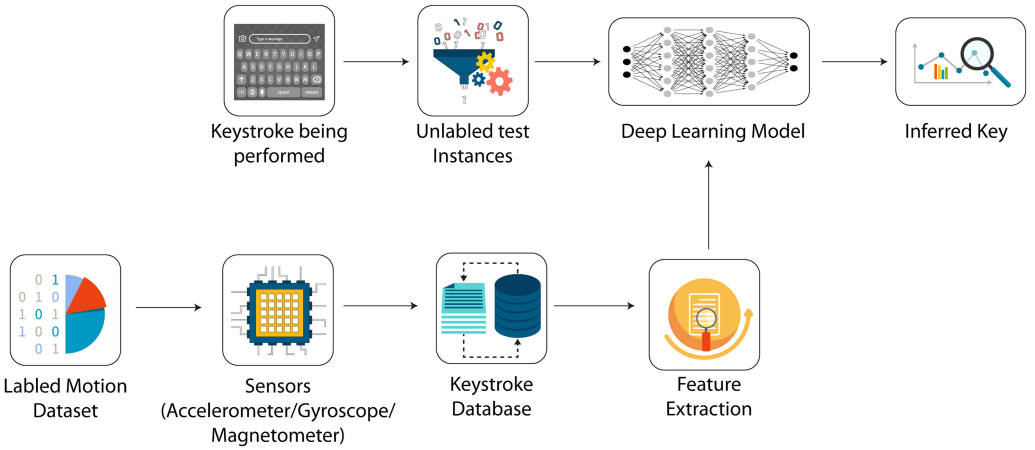


Fig. 2. Workflow figure of proposed approach.

scheme that can efficiently decrease the App sniffing accuracy and intelligent enough to ignore benign apps to mitigate this.

Table 1 demonstrates the limitation of existing works. The most relevant work to this article is proposed in Reference [26], which lacks in providing accurate results for classifying keystrokes. Similarly, another baseline work [15] achieved low accuracy. In this article, we address these limitations.

3 BETALOGGER

In this section, we describe our proposed research that is an efficient model for Text Inference Detection. We mold our research problem into a **Sequence 2 Label Sequence (S2LS)** task. In this task, each sequence is represented by a vector know as TIV that succeeds by a Deep learning model to predict the pressed alphabet. Figure 2 depicts the proposed deep learning model for keystroke detection.

3.1 Keystroke Data Collection and Pre-processing

An application is developed to gather data from Android smartphones. Ten different individuals and Five different smartphones are monitored to obtain the data, including Samsung Grand Prime,

Oppo F1, Huawei Honor, Oppo F3, and Samsung J7. The motivation behind collecting this dataset is to produce side-channel attacks. The data is collected from the user in those postures, including little movement and noise (i.e., when an individual is standing and sitting). The test candidate holds the smartphone in portrait orientation while holding the device held in their hands and typing with both hand's thumbs. The parameters while the data collection process is such that what data is required to be collected and how often it is required. The data is gathered at a constant instance frequency of 30 instances per second as prescribed by References [18, 19]. The duration of the data collection process lasts approximately 2–3 min to record all signals. The dataset is composed of sensor readings that are generated when the smartphone keyboard is used for typing. The sensor availability varies through smartphones. Different models are trained to encounter that limitation, which includes a model trained with a dataset composed of raw acceleration values, and other models are based on values in collection with other sensors (magnetometer and gyroscope). The readings are stored in a comma-delimited CSV structured with timestamps to arrange a 27 keystrokes files dataset.

3.2 Feature Extraction

The raw data is converted into a sensor event window. A window with 500 samples is selected from each file of each participant. The selected window is different enough to acquire all the readings to input in the classification method. We assign the labels as the alphabets have been known as a ground truth. This configuration enables the model to accurately address and record the sensor readings and alphabets being used. A feature matrix is generated based on 130,000 raw values from sensors in which each reading is composed of a 3-axis of all three sensors.

3.3 Dense Multilayer Neural Network (DMNN)

This research is based on predicting the given alphabets as labels using TTIV and then using those alphabets labels to generate output sequences. We propose the methodology of S2LS. The architecture used for the S2LS problem is based on making better decisions and can tune its respective weights more efficiently, and the S2L model can learn the input features more accurately.

This research solves the Text Inference classification problem by predicting the target labels and then use those predicted labels and pass them to the sequence generator module to generate the output sequence. This type of method allows us to apply the DMNN for the text inference classification task. This research aims to generate the output sequence using a text inference vector to predict the alphabet's characters sequence.

3.3.1 Input Layer. Input features length is equal to 9. Thus, there are 9 features to form one sequence that pass it as input. The number of neurons in the first (input) layer is 100 neurons.

3.3.2 Hidden Layer. Model architecture using three hidden layers and one dense layer. The first hidden layer consists of 200 neurons, and the second hidden layer has 120 neurons, and the dense layer contains 100 neurons. The purpose of using three layers with a dense layer is to extract more significant features. Each hidden layer applies various functions on the input feature vectors.

3.3.3 Activation Function. Activation functions are also known as squashing functions. These functions are normally applied to the output result from the hidden layers of the model. However, every hidden layer of the model is a linear transform accompanied by a squashing function nonlinearity. The activation function accepts one input reading and applies mathematical procedures over it. For the activation function, we apply **Rectified Linear Unit (RELU)**:

$$g(x) = \sigma(W_i^T X + bias). \tag{1}$$

3.3.4 Rectified Linear Unit (RELU). RELU is an activation function that returns zero when the value is less than zero ($x < 0$), and then linear with slope 1 when value is greater than zero ($x > 0$). Its mathematical form is

$$f(x) = max(0, x). \tag{2}$$

RELU output is thresholds at zero value like less than any value converted into zero. RELU is always prioritized due to two major reasons. The first reason is RELU works on low computational requirements than other activation functions, which include sigmoid and tanh functions, since it does not require complex computations such as exponential. The second reason is that it is highly accelerating the intersection point, i.e., **Stochastic-Gradient-Decent (SGD)** convergence compared to tanh and sigmoid.

3.3.5 Objective/Cost Function. The objective/cost function calculates the difference between the true labels y and the predicted labels y' of the classifier. It is a mechanism that is used for evaluating how effective the model is during the training. Cost function and error are directly proportional. The error in the system will be higher if the cost function returns a greater value. This model use softmax as a cost function. In LR, we have binary labels that are: y ∈ 0, 1. However, the softmax function can work with more than two classes where $y' \in 1, \ldots,$ k , where k is the total target classes:

$$Z^{[l]} = W^{[l]} a^{[l]} + bias^{[l]}, \tag{3}$$

$$a^{[l]} = \frac{e^{z^{[l]}}}{\Sigma_{i=1} ti}. \tag{4}$$

3.3.6 Regularization. Regularization prevents **Neural Network (NN)** from over-fitting, and it acts upon when NN properly fits the data for training but unable to execute on the data for testing. To prevent over-fitting, L2-regularization is used as it penalizes the squared magnitude of all the NN parameters except bias-inputs and then embeds it in the objective function. L2-regularization gives more diffuse weights that are near to zero. λ (regularization controlling factor) is used to predict the quantity of penalization of weights:

$$R(W_i) = \frac{\lambda}{2m} \mid\mid W \mid\mid^2 . \tag{5}$$

3.3.7 Weight Initialization. Weight initialization in NN contributes a significant part to the model training process and convergence conduct. It is a recommended procedure to utilize RELU units. Equation (5) represents the initialization of the weights. This kind of weight initialization maintains the variance across all the layers in the DMNN. We used a Gaussian distribution method to randomly initialize the weights with a standard deviation of $\sqrt{(2/m)}$, where m represents the inputs:

$$W_i = random(m) * \sqrt{\frac{2.0}{m}}. \tag{6}$$

3.3.8 Optimization. Optimization methods act as a major driving force to minimizing the cost functions. Adam optimizer is a stochastic base method. It is the combination of momentum and RMSprop. This method is computationally efficient and has very fewer memory requirements. Adam's optimizer convergence rate as compared to other optimization algorithms is so rapid. The values of $\epsilon = 10 - 8$, $\beta 1 = 0.9$, and $\beta 2 = 0.999$.

At first, we give our generated TIV data to the DMNN model, which returns predicted labels in alphabet characters. DMNN comprises two parts: (1) using TIV as an input feature and predict the

target labels, which are composed of alphabet characters; (2) used these target alphabet characters to generate output sequence using sequence generator module.

3.4 Machine Learning Models

This section presents the traditional machine learning algorithm's details to evaluate and compare the models' performance.

3.4.1 Logistic Regression (LR). LR is a statistical model that uses a logistic function. The logistic model is used to model the probability of a certain target class or existing event. LR is used to predict the class category of individuals based on one or multiple predictor variables. LR does not return the class of observations directly but depends on the estimated probability of class membership. The probability will range between 0 and 1 as shown in Equation (7):

$$\ell(y', y) = -\frac{1}{m}\left[\sum_{i=1}(ylog(y') + (1-y)log(1-y')\right], \tag{7}$$

where y' shows the predicted value, which should be small, y represents the actual target value. Equation (7) represents the cost function using a sigmoid function (σ) to predict the target label.

3.4.2 Gradient Boosting (GB). GB is a machine learning boosting algorithm based on the intuition that the next best possible model is combined with the next model with previous models. This way minimizes the overall prediction error. The idea is to set the target outcomes for the next model to minimize error. GB is a sort of greedy algorithm [10] as it takes to benefit from regularization methods that punish various parts of the algorithm for improving the model's performance by reducing over-fitting. Where the key idea is to train a model (F) to predict target values x, F (x) = y' by minimizing the **mean-squared error (MSE)**:

$$h_n(x) = y - F_n(x). \tag{8}$$

N shows the number of stages in the algorithm, y represents the actual values, and n represents each stage, h_n represents the new estimator in GB.

3.4.3 K-Nearest Neighbors Classifier (KNN). KNN is a lazy learning and non-parametric algorithm. The key idea behind KNN is to explore the euclidean distances for a test sample x and all the instances in the training data and choose the specified number of K instances closest to the x, then select the label that is having the highest weighted score concerning distance [9], where f' represent the predicted label for text sample x. As shown in Equations (9), (10), and (11):

$$f'(x_t) \leftarrow \sum_{i=1} w_i f(x_(i)), \tag{9}$$

$$d(x_i, x_t) = \sqrt{\sum_{i=1}((x_i) - (x_t))^2}, \tag{10}$$

$$w_i = 1/d(x_i, x_t)^2. \tag{11}$$

3.4.4 Support Vector Machine (SVM). To separate the data points, different possible hyperplanes could be chosen. The main objective is to find that specific hyper-planes with the maximum-margin, in-between data points of two classes. Increasing the distance up to max offers reinforcement such that the test data points can be separated and identified with better confidence. Hyper-planes are decision boundaries that are used to classify the data points. Data points lie on either side of the hyperplane can be part of different classes. The hyper-plane dimension depends

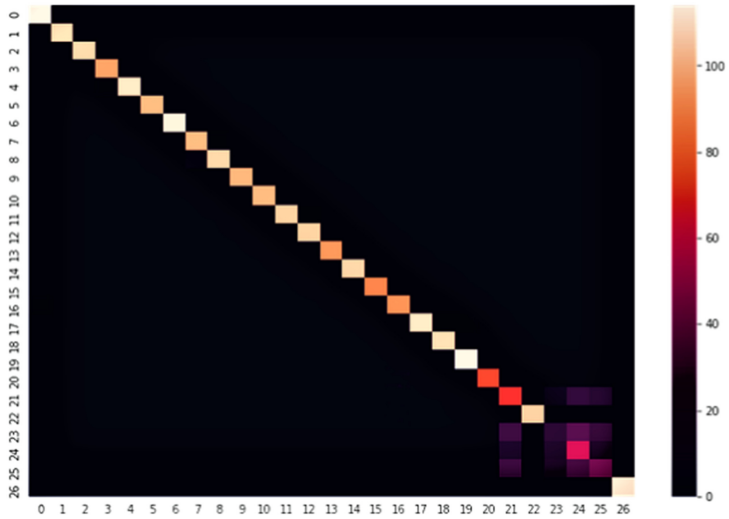


Fig. 3. Confusion matrix of DMNN.

upon the number of features in a dataset. The data points near the hyperplane that affect the hyperplane's orientation and position are support vectors. Using support vectors, we maximize the classifier margin. As shown in the following equations,

$$|| \ x \ ||= \sqrt{(x_1)^2 + (x_2)^2 + (x_3)^2}, \tag{12}$$

where the length of vector $x(x_1, x_2, x_3)$ can be calculated using Equation (12),

$$x \bullet y =| \ x \ || \ y \ | \ cos(\theta), \tag{13}$$

where x and y are 2 vectors and there dot product are calculated for support vectors Equation (13),

$$y = a * x + b, \tag{14}$$

$$a * x + b - y = 0, \tag{15}$$

$$W \bullet X + b = 0, \tag{16}$$

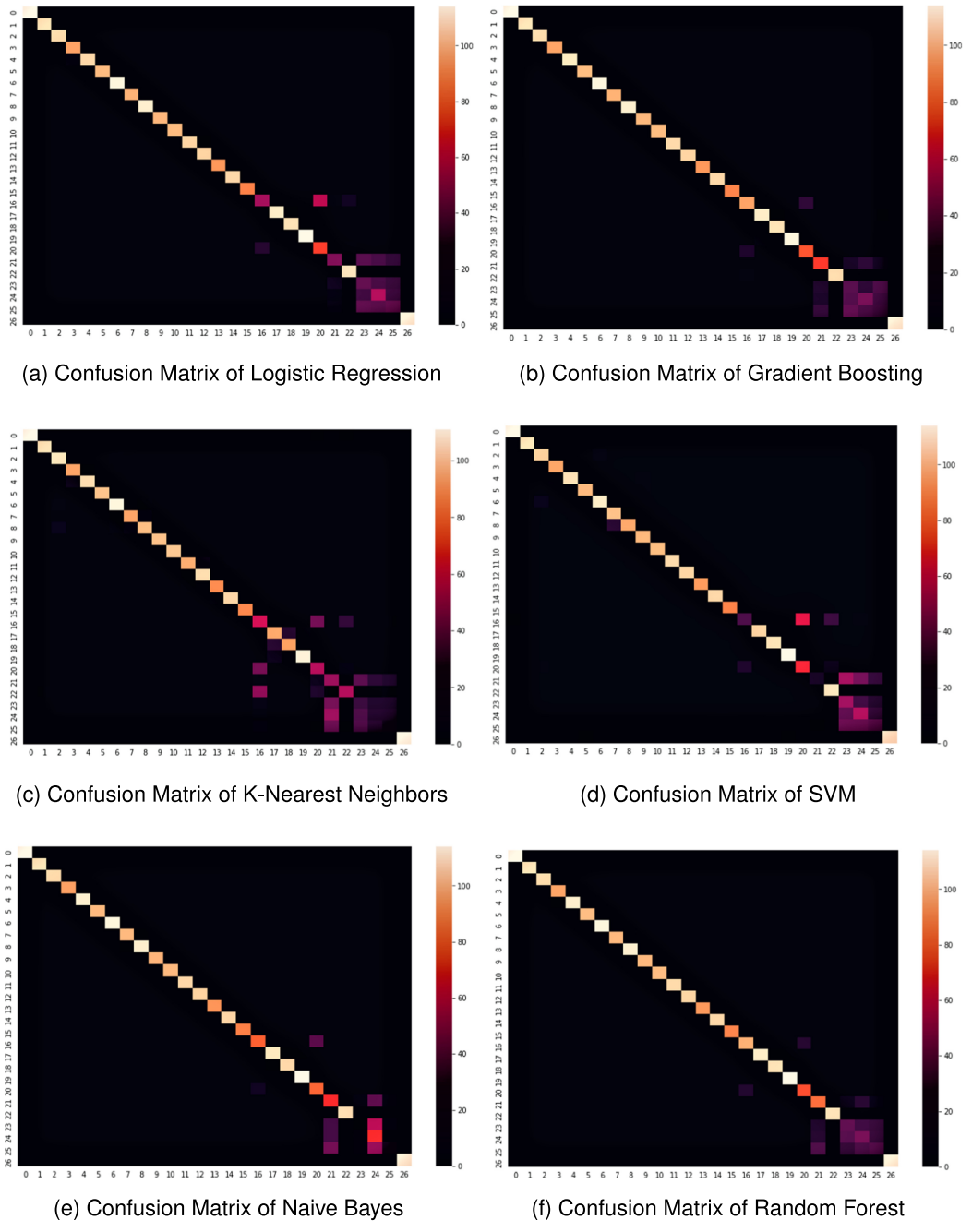
where X represents the vector (x, y) and $W(a, -1)$ form a hyper-plane Equations (14), (15), and (16).

3.4.5 Naive Bayes (NB). NB is a probabilistic machine learning algorithm that is used for classification problems. The crux of the NB is based on the Bayes theorem. Bayes theorem is used to calculate the probability of X happening, given that Y has occurred. Where X is the hypothesis, Y represents the evidence. The assumption is that the feature predictors are independent. The calculation for Bayes Theorem can be calculated using the following Equation (17):

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)}. \tag{17}$$

4 EVALUATION

To evaluate the proposed model's performance, we build a sequence generation model that regenerates the sequence using the predicted alphabets. First, it converts the text sequence into a TIV



(a) Confusion Matrix of Logistic Regression

(b) Confusion Matrix of Gradient Boosting

(c) Confusion Matrix of K-Nearest Neighbors

(d) Confusion Matrix of SVM

(e) Confusion Matrix of Naive Bayes

(f) Confusion Matrix of Random Forest

Fig. 4. Graphical representation of confusion matrix of machine learning classifiers.

and feeds it as input to the DMNN model. DMNN model produces the output vectors containing the target labels and passes them to the sequence generation module, where it uses the target labels dictionary to convert these target labels into characters and then combine these characters into an output sequence sentence.

Table 2. Model Testing on Short Sentences Taken from Previous Study [26]

No	Model	Text
1	Typed text	"well that was probably one of the hardest thing i have ever had to do in my life"
	Generated text	"well thst was probablz one of the hardest thing i haxe ever had to do in mz life"
2	Original text	"going through our pictures made me realize hoe stuoid i was to ruin everything we had"
	Generated text	"going through our pictures made me realixe hoe stuoid i was to ruin everzthing we had"
3	Original Text	"happy birthday to my favorite person in the world when i would be there celebrating when you"
	Generated text	"happz birthdaz to mz faxorite person in the world when i would be there celebrating when zou"
4	Original text	"A Quick Brown Fox jump over the lazy Dog"
	Generated text	"a quick brown foy jump over the laxz dog"

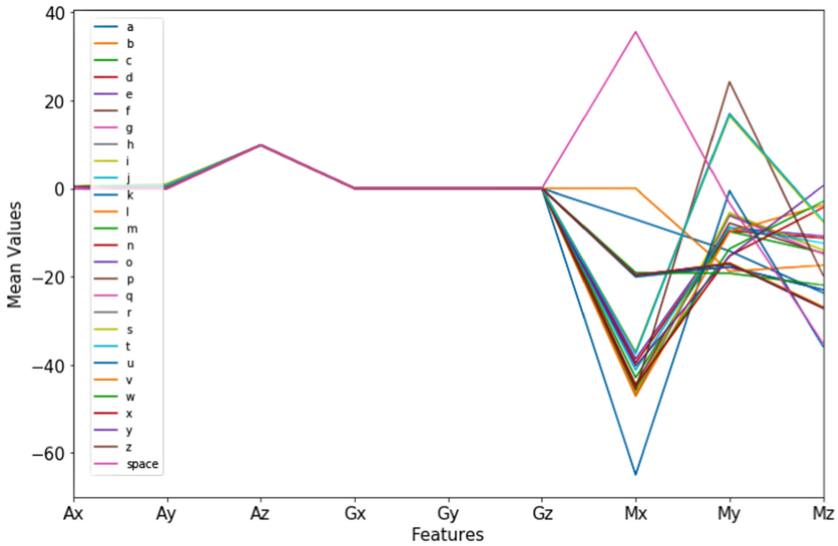


Fig. 5. Feature plot of mean feature values.

Table 2 presents the model's test samples where it can be noticed that most alphabets are predicted accurately, whereas letters v, x, y, and z are not predicted accurately. Table 3 presents the model testing on long sentences. The first row presents the user typed text and second row presents the model generated text. It can be noticed that most alphabets are predicted accurately.

4.1 Results and Discussion

Table 4 presents the deep learning approach *DMNN* in comparison with machine learning techniques, including LR, GB, kNN, SVM, and NB. The performance metrics of evaluation include accuracy, F1-score, precision, and Recall. *DMNN* achieves a proficient accuracy of 91.14%, F1-score of 90.16%, precision of 90.29%, and Recall of 90.03% while detecting the keystrokes and forming

Table 3. Model Testing Long Sentences

No	Model	Text
1	Typed text	"How its using AI in automotive Beginning as Googles exploration of self driving vehicles Waymo is now its own company creating driverless vehicles that can safely deliver people from points A to B With over eight million autonomous miles driven to date Waymos 360 degree perception technology detects pedestrians other vehicles cyclists road work and other obstacles from up to 300 yards away Industry impact Waymo is already providing test rides in the Phoenix metro area If youre local and want to experience it for yourself you can apply to try it out"
	Generated text	"how its using ai in automotixe beginning as googles eyploration of self drixing vehicles wazmo is now its own companz creating drixerless vehicles that can safelz delixer people from points a to b with oxer eight million autonomous miles drixen to date, wazmos 360 degree perception technologz detects pedestrians other vehicles, czclists road work and other obstacles from up to 300 zards awaz industrz impact wazmo is alreadz proxiding test rides in the phoeniy metro area if zoure local and want to eyperience it for zourself zou can applz to trz it out"

Table 4. Evaluation Results of *Betalogger* for Alphabet Prediction

Model	Accuracy	F1-Score	Precision	Recall
DMNN	91.14%	90.16%	90.29%	90.03%
LR	87.92%	88.35%	88.78%	87.92%
GB	90.0%	89.99%	89.98%	90.0%
KNN	79.63%	79.75%	79.87%	79.62%
SVM	85.28%	86.32%	87.38%	85.29%
NB	89.77%	89.72%	89.71%	89.74%

the whole sentence. The proposed approach *DMNN* efficiently performs with an accuracy gain of 1.14%, gain in F1-score of 0.17%, gain in precision of 0.39%, and gain in the Recall of 0.03%. The model KNN stands out to be the weakest among other classifying algorithms. GB turns out to be the most proficient among the machine learning algorithms while having an accuracy of 90.0%, F1-score of 89.99%, the precision of 89.98%, and Recall of 90.0%. The rest of the models, NB, LR, SVM, and KNN, show an F1-score of 89.72%, 87.92%, 86.32%, 79.75%, respectively.

Figure 3 represents the confusion matrix of the proposed *DMNN* model. Class 0 in the confusion matrix represents the alphabet "a." Similarly, the rest of the classes represent corresponding alphabets in ascending order, i.e., "b" is represented by class 1, "c" is represented by class 2, and so on up to "z" is represented by class 25 and 26 represents white space. It shows proficient results regarding predictions. There are 26 target classes as shown in Figure 3, of which 22 classes are predicted with greater accuracy, while classes 21, 23, 24, and 25 are not predicted as accurately as the other classes.

Figures 4(a)–4(f) depict the confusion matrix of LR, GB, KNN, SVM, NB, and random forest, respectively.

Figure 5 shows the mean feature values against the target labels, where each test sample consists of 9 feature sequence against a target label. It shows target class "g" has the highest mean values

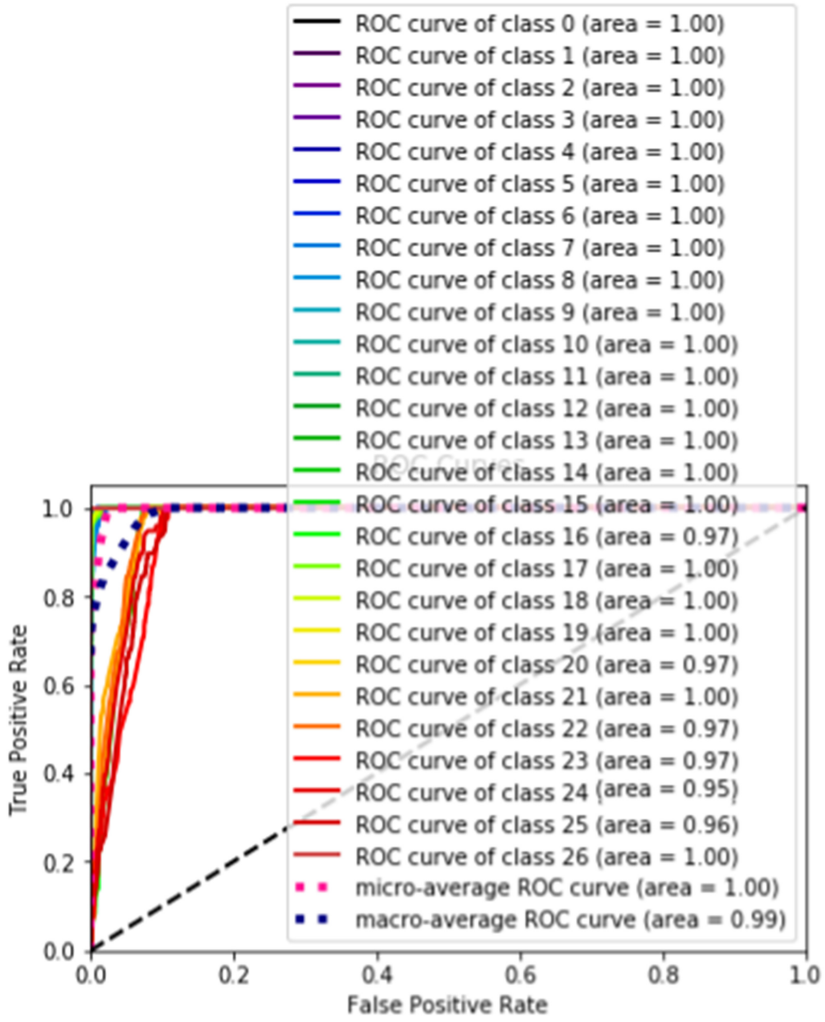


Fig. 6. Graphical representation of receiver operating characteristic curve.

in feature Mx, and class "u" shows the lowest mean values. However, the lowest mean values of most of the target classes lie in the Mx feature.

Figure 6 represents a **Receiver Operating Characteristic (ROC)** curve. It shows the performance of a model at all classification thresholds. As you can see DMNN ROC curve shows significantly good results for most classes, but in classes 24 and 25, their scores are relatively lower than those of other classes.

As depicted in Figure 7, the accuracy plot DMNN model performs well in training as the epoch's number increased accuracy from 0.912 with a gradual varying behavior and attain the highest point of 0.9138. In the testing phase, accuracy increased gradually as the epoch increased and attained a maximum 0.914% score at 30 epochs. Figure 8 shows the loss of our model where it starts from 0.200 starts decreasing as the number of epochs increased, and the lowest value achieved is 0.198 in a train and the test 0.188.

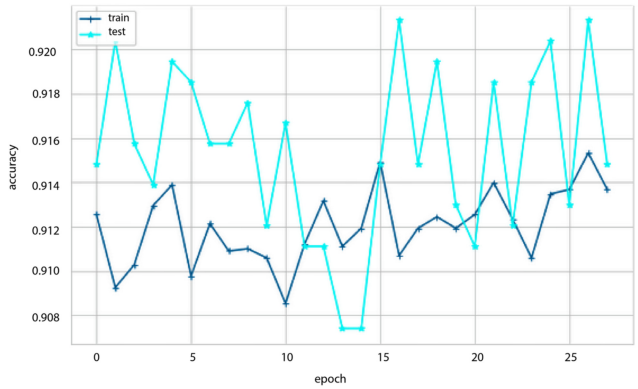


Fig. 7. Accuracy plot of DMNN during training and testing phase.

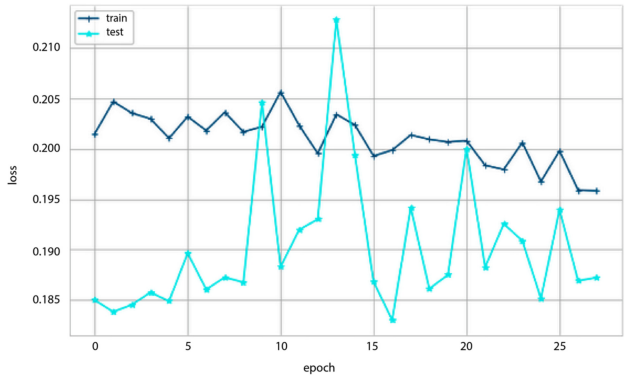


Fig. 8. Loss plot of DMNN during training and testing phase.

Table 5. Performance Comparison of *DMNN* with Baseline Approach [15]

Reference	Accuracy	F1-Score	Precision	Recall
[15]	0.9020	0.9020	0.9020	0.9020
DMNN	**0.9114**	0.9016	**0.9029**	0.9003

In Table 5, the performance metrics of the *DMNN* are compared with the recent state-of-the-art baseline study [15]. *DMNN* shows an accuracy gain of 0.94% from the baseline approach and 0.09% gain in precision due to its good training and proficient neural network.

Figure 9 depicts the accuracy comparison of *Betalogger* with baseline approaches [15]. *Betalogger* performs efficiently well with higher accuracy of 91.14% while Alphalogger performs with an accuracy of 90.02% and Textlogger performs with an accuracy of 32.10%. Thus, *DMNN* stands out to be more proficient than other typical techniques.

5 CONCLUSION AND FUTURE SCOPE

This article presented a study on side-channel cyber-attacks that hackers commonly observe to compromise an individual's sensitive data through the smartphone screen's keystrokes. We proposed an efficient in-depth learning approach named *Betalogger* that uses DMNN, based on S2LS

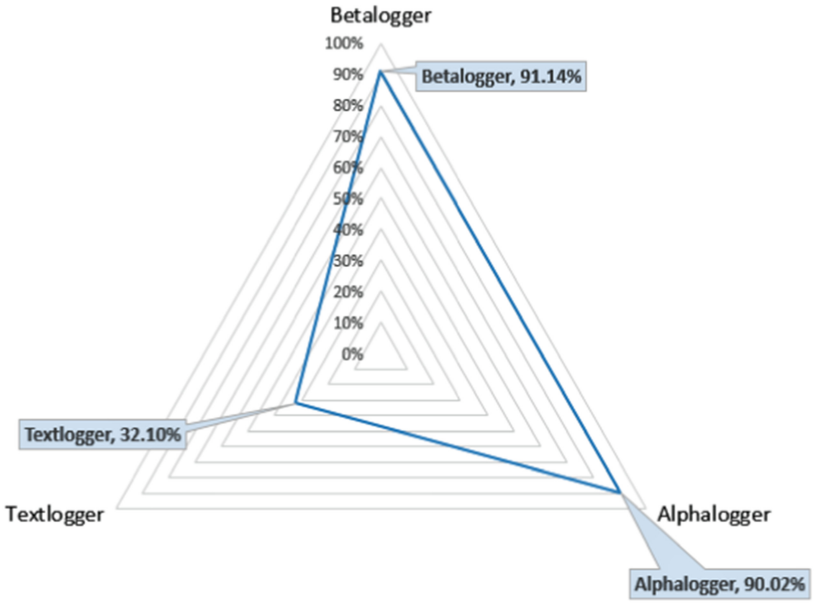


Fig. 9. Graphical depiction of accuracy comparison of betalogger with baseline approaches.

architecture. Comparative analysis of the proposed approach, *DMNN*, is also done with machine learning approaches, which include LR, gradient boost, k nearest neighbors, SVM, and NB, in which *DMNN* stands out to be more proficient than these machine learning algorithms with an accuracy of 91.14%, F1-score 90.16%, precision of 90.29%, and Recall of 90.03%. In the future, we plan to enhance our model with further advancements of deep learning, such that the model will make predictions to the sentence level. The detection of side-channel attacks at the earliest stage can be considered an open challenge in the future.

REFERENCES

[1] Sara Afzal, Muhammad Asim, Abdul Rehman Javed, Mirza Omer Beg, and Thar Baker. 2021. URLdeepDetect: A deep learning approach for detecting malicious URLs using semantic vector models. *J. Netw. Syst. Manage.* 29, 3 (2021), 1–27.

[2] Mamoun Alazab, Suleman Khan, Somayaji Siva Rama Krishnan, Quoc-Viet Pham, M. Praveen Kumar Reddy, and Thippa Reddy Gadekallu. 2020. A multidirectional LSTM model for predicting the stability of a smart grid. *IEEE Access* 8 (2020), 85454–85463.

[3] Mamoun Alazab, Robert Layton, Roderic Broadhurst, and Brigitte Bouhours. 2013. Malicious spam emails developments and authorship attribution. In *Proceedings of the 4th Cybercrime and Trustworthy Computing Workshop*. IEEE, 58–68.

[4] Giuseppina Andresini, Annalisa Appice, and Donato Malerba. 2020. Dealing with class imbalance in android malware detection by cascading clustering and classification. In *Complex Pattern Mining*. Springer, Switzerland, 173–187.

[5] Muhammad Zubair Asghar, Fazli Subhan, Hussain Ahmad, Wazir Zada Khan, Saqib Hakak, Thippa Reddy Gadekallu, and Mamoun Alazab. 2021. Senti-eSystem: A sentiment-based eSystem-using hybridized fuzzy and deep neural network for measuring customer satisfaction. *Software: Pract. Exper.* 51, 3 (2021), 571–594.

[6] Jianfeng Cui, Lixin Wang, Xin Zhao, and Hongyi Zhang. 2020. Towards predictive analysis of android vulnerability using statistical codes and machine learning for IoT applications. *Comput. Commun.* 155 (2020), 125–131.

[7] Sougata Deb, Youheng Ou Yang, Matthew Chin Heng Chua, and Jing Tian. 2020. Gait identification using a new time-warped similarity metric based on smartphone inertial signals. *J. Ambient Intell. Human. Comput.* 11, 10 (2020), 4041–4053.

[8] Natarajan Deepa, Quoc-Viet Pham, Dinh C. Nguyen, Sweta Bhattacharya, Thippa Reddy Gadekallu, Praveen Kumar Reddy Maddikunta, Fang Fang, Pubudu N. Pathirana et al. 2020. A survey on blockchain for big data: Approaches, opportunities, and future directions. Retrieved from https://arXiv:2009.00858.

[9] Sahibsingh A. Dudani. 1976. The distance-weighted k-nearest-neighbor rule. *IEEE Trans. Syst. Man, Cybernet.* SMC-6, 4 (1976), 325–327.

[10] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* 29, 5 (2001), 1189–1232.

[11] Thippa Reddy Gadekallu, Dharmendra Singh Rajput, M. Praveen Kumar Reddy, Kuruva Lakshmanna, Sweta Bhattacharya, Saurabh Singh, Alireza Jolfaei, and Mamoun Alazab. 2020. A novel PCA–whale optimization-based deep neural network model for classification of tomato plant diseases using GPU. *J. Real-Time Image Process.* 0, 0 (2020), 1–14.

[12] Muzammil Hussain, Ahmed Al-Haiqi, A. A. Zaidan, B. B. Zaidan, M. L. Mat Kiah, Nor Badrul Anuar, and Mohamed Abdulnabi. 2016. The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks. *Pervas. Mobile Comput.* 25 (2016), 1–25.

[13] Syed Ibrahim Imtiaz, Saif ur Rehman, Abdul Rehman Javed, Zunera Jalil, Xuan Liu, and Waleed S. Alnumay. 2021. DeepAMD: Detection and identification of android malware using high-efficient deep artificial neural network. *Future Gen. Comput. Syst.* 115 (2021), 844–856.

[14] Celestine Iwendi, Zunera Jalil, Abdul Rehman Javed, Thippa Reddy, Rajesh Kaluri, Gautam Srivastava, and Ohyun Jo. 2020. KeySplitWatermark: Zero watermarking algorithm for software protection against cyber-attacks. *IEEE Access* 8 (2020), 72650–72660.

[15] Abdul Rehman Javed, Mirza Omer Beg, Muhammad Asim, Thar Baker, and Ali Hilal Al-Bayatti. 2020. AlphaLogger: Detecting motion-based side-channel attack using smartphone keystrokes. *J. Ambient Intell. Human. Comput.* 0, 0 (2020), 1–14.

[16] Abdul Rehman Javed, Muhammad Usman, Saif Ur Rehman, Mohib Ullah Khan, and Mohammad Sayad Haghighi. 2020. Anomaly detection in automated vehicles using multistage attention-based convolutional neural network. *IEEE Trans. Intell. Transport. Syst.* 0, 0 (2020), 1–10.

[17] Jae Woong Joo, Seo Yeon Moon, Saurabh Singh, and Jong Hyuk Park. 2017. S-Detector: An enhanced security model for detecting smishing attack for mobile computing. *Telecommun. Syst.* 66, 1 (2017), 29–38.

[18] A. Krause, M. Ihmig, E. Rankin, D. Leong, Smriti Gupta, D. Siewiorek, A. Smailagic, M. Deisher, and U. Sengupta. 2005. Trading off prediction accuracy and power consumption for context-aware wearable computing. In *Proceedings of the 9th IEEE International Symposium on Wearable Computers (ISWC'05)*. IEEE, 20–26.

[19] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. 2011. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.* 12, 2 (Mar. 2011), 74–82. https://doi.org/10.1145/1964897.1964918

[20] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. 2020. A review of android malware detection approaches based on machine learning. *IEEE Access* 8 (2020), 124579–124607.

[21] Ignacio Martín, José Alberto Hernández, and Sergio de los Santos. 2019. Machine-learning based analysis and classification of android malware signatures. *Future Gen. Comput. Syst.* 97 (2019), 295–305.

[22] Anam Mehtab, Waleed Bin Shahid, Tahreem Yaqoob, Muhammad Faisal Amjad, Haider Abbas, Hammad Afzal, and Malik Najmus Saqib. 2020. AdDroid: Rule-based machine learning framework for android malware analysis. *Mobile Netw. Appl.* 25, 1 (2020), 180–192.

[23] Sparsh Mittal and Venkat Mattela. 2019. A survey of techniques for improving efficiency of mobile web browsing. *Concurr. Comput.: Pract. Exper.* 31, 15 (2019), e5126.

[24] Long Nguyen-Vu, Jinung Ahn, and Souhwan Jung. 2019. Android fragmentation in malware detection. *Comput. Secur.* 87 (2019), 101573.

[25] Rui Ning, Cong Wang, ChunSheng Xin, Jiang Li, and Hongyi Wu. 2018. Deepmag: Sniffing mobile apps in magnetic field through deep convolutional neural networks. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom'18)*. IEEE, Athens, Greece, 1–10.

[26] Dan Ping, Xin Sun, and Bing Mao. 2015. Textlogger: Inferring longer inputs on touch screen using motion sensors. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, New York, 1–12.

[27] Abdul Rehman, Saif Ur Rehman, Mohibullah Khan, Mamoun Alazab, and Thippa Reddy. 2021. CANintelliIDS: Detecting in-vehicle intrusion attacks on a controller area network using CNN and attention-based GRU. *IEEE Trans. Netw. Sci. Eng.* 0, 0 (2021), 1–10.

[28] Zhongru Ren, Haomin Wu, Qian Ning, Iftikhar Hussain, and Bingcai Chen. 2020. End-to-end malware detection for android IoT devices using deep learning. *Ad Hoc Netw.* 101 (2020), 102098.

[29] Congcong Shi, Rui Song, Xinyu Qi, Yubo Song, Bin Xiao, and Sanglu Lu. 2020. ClickGuard: Exposing hidden click fraud via mobile sensor side-channel analysis. In *Proceedings of the IEEE International Conference on Communications (ICC'20)*. IEEE, 1–6.

[30] Khoi-Nguyen Tran, Mamoun Alazab, Roderic Broadhurst et al. 2014. Towards a feature rich model for predicting spam emails containing malicious attachments and URLs. In *ANU Research Publications*. Australian Computer Society, Australia, 1–10.

[31] Saif ur Rehman, Mubashir Khaliq, Syed Ibrahim Imtiaz, Aamir Rasool, Muhammad Shafiq, Abdul Rehman Javed, Zunera Jalil, and Ali Kashif Bashir. 2021. DIDDOS: An approach for detection and identification of distributed denial of service (DDoS) cyberattacks using gated recurrent units (GRU). *Future Gen. Comput. Syst.* 118 (2021), 453–466.

[32] R. Vinayakumar, Mamoun Alazab, Sriram Srinivasan, Quoc-Viet Pham, Soman Kotti Padannayil, and K. Simran. 2020. A visualized botnet detection system-based deep learning for the internet of things networks of smart cities. *IEEE Trans. Industry Appl.* 56, 4 (2020), 4436–4456.

[33] Robert-Andrei Voicu, Ciprian Dobre, Lidia Bajenaru, and Radu-Ioan Ciobanu. 2019. Human physical activity recognition using smartphone sensors. *Sensors* 19, 3 (2019), 458.

[34] Suleiman Y. Yerima, Mohammed K. Alzaylaee, and Sakir Sezer. 2019. Machine learning-based dynamic analysis of android apps with improved code coverage. *EURASIP J. Info. Secur.* 2019, 1 (2019), 4.

Received May 2020; revised April 2021; accepted April 2021