Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 19 No. 6 (2025) | OPEN ACCESS

https://doi.org/10.3991/ijim.v19i06.53885

PAPER

Methods for Detecting Android Malware: Employing Mobile Devices to Improve Procedures for Inquiry-Based Learning

Sandeep Yelisetti[1], P. Suresh[2], Suresh Kumar Jha[3](✉), Neha Arora[4], Ketan Anand[5], Vedala Naga Sailaja[6]

[1]Information Technology, Velagapudi Ramakrishna Siddhartha Engineering College, Deemed to be University, Vijayawada, Andhra Pradesh, India

[2]Department of Computer Science and Engineering, Saveetha School of Engineering, SIMATS, Thandalam, Chennai, Tamil Nadu, India

[3]Manipal University, Jaipur, Rajasthan, India

[4]School of Information Science, Presidency University, Bengaluru, Karnataka, India

[5]Freelance Trainer, Hyderabad, Telangana, India

[6]Dept of MBA, KLEF Deemed to be University, Green Fields, Vaddeswaram, Andhra Pradesh, India

suresh.jha@jaipur.manipal.edu

ABSTRACT

The operating system (OS) of a computer controls both its hardware and software. It handles necessary functions including input and output processing, file and memory management, and peripheral device management, including disc drives and printers. Programs created for particular purposes are referred to as application software. These programs, which are frequently open source and freely accessible, contribute to the growing number of downloads. This paper discusses the basics of Android malware, its evolution, and malware analysis tools and techniques. Providing the research gaps and giving a review of the literature on Android malware detection using machine learning and deep learning are its main objectives. It offers the knowledge gathered from the literature as well as potential avenues for future research, which may aid in the development of reliable and precise methods for classifying Android malware. This paper conducts a systematic and comprehensive assessment of the methods and tools utilized for the analysis, classification, and detection of malicious Android apps. Several research gaps are indicated based on the thorough literature evaluation. Additionally, the report offers insights on future research paths that may aid academics in developing novel and reliable methods for identifying and categorizing Android malware.

KEYWORDS

Android malware, operating system (OS), hardware and software, academics, deep learning

1 INTRODUCTION

Android has emerged as the most widely used operating system (OS) for smart mobile devices since its 2008 introduction. Approximately 86.6% of smartphones sold worldwide in 2019 were Android-based. More than 2.8 million apps were available on Google Play, the official Android app store, by the end of April 2020. The integrity of Android apps is gravely threatened by the numerous security attack

Yelisetti, S., Suresh, P., Jha, S.K., Arora, N., Anand, K., Sailaja, V.N. (2025). Methods for Detecting Android Malware: Employing Mobile Devices to Improve Procedures for Inquiry-Based Learning. *International Journal of Interactive Mobile Technologies (iJIM)*, 19(6), pp. 103–114. https://doi.org/10.3991/ijim.v19i06.53885

Article submitted 2024-11-16. Revision uploaded 2025-01-05. Final acceptance 2025-01-08.

surfaces that result from several variables, including the available ecology mode of Android programs, their coarse-grained permission control, and their capacity to invoke third-party code. More than 3.25 million Android apps were found to be infected with malware in 2016 alone, according to statistics. This indicates that a new Android malware app was detected approximately every ten seconds.

Malware identification, vulnerability detection, developer reviews, and application reinforcement are some of the solutions that have been put out to guarantee the security of the Android ecosystem [1]. Among the different security choices, Android malware detection is a popular security protection technique that can stop malware from being installed and deployed or pushed into the Android application marketplace. Three types of Android malware detection techniques can be distinguished based on prior research: hybrid, dynamic, and static detection. People now carry mobile phones with them everywhere they go. Six billion people presently have cellphones, and in the upcoming years, that figure is predicted to rise by hundreds of millions. Smartphones have made it feasible to perform daily duties from the palm of your hand, including sending messages, making calls, and managing bank accounts. This ability has presented several difficulties. Because they always carry their smartphones with them, users are used to storing both sensitive and non-sensitive data on them. The idea that everything is safe since the smartphone is always with the owner has led to a false sense of security. Users are unaware that smartphones are currently vulnerable to a variety of assaults.

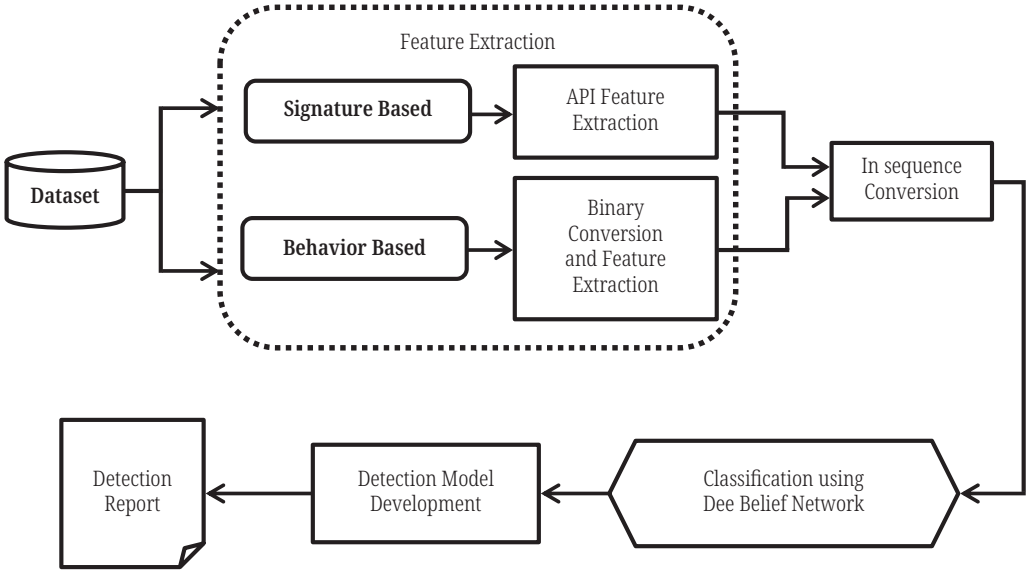

Fig. 1. Methods for classifying and detecting malware

There are several apps accessible right now for people to download and use for their benefit. The goal of many of the harmful apps that are now on the market is to identify and steal user data from smartphones without the user's awareness. Unbeknownst to them, users install these programs, which are malware with malicious intent that targets private information. The OS that is open-source among these two is Android [2]. The main factor for the great contribution of community developers has been found to be their adaptability and transparency. Because of the healthy ecosystem this has produced, Android has thrived in terms of stability

and functionality. This openness, however, has also played a significant role in luring developers to create mobile malware or dangerous apps.

First, the feature extraction procedure receives the benign and malicious data from the CICAndMal2017 datasets. The data used in feature extraction are divided into two categories [3]: behavior-based data and signature-based data.

First, the behavior-based data is transformed into binary data for feature extraction, and then the API feature extraction is carried out in the signature-based data. Following feature extraction, the features are transformed into sequence data and used in the classification stage.

In order to distinguish between harmful and benign data, the DBN model is also given the transformed sequence data. The identified malicious data is classified and then turned into a detection report for additional processing. The general process of the MAD-NET technique is shown in Figure 1.

The remainder of the paper follows this format: Section 2 discusses related work. Section 3 outlines the research approach. There are more sub-sections in this part. The paper quality is evaluated, the exclusion criteria are listed in Section 4, the data synthesis process is used to extract data and formulate the results, and Section 5 thoroughly examines the presented answers to research questions and validity threats before presenting the conclusion.

2 RELATED WORKS

Static analysis, dynamic analysis, and cloud-based solutions are the three main kinds of methods suggested for Android malware detection. Finding dangerous features or problematic code segments in an application without running them is the goal of the rapid and low-cost technique known as static analysis. They began by looking for dangerous APIs and keywords linked to known malware's unusual behavior [4]. Following the extraction of permission data from the decompressed app files, the apps were examined for the presence of those dangerous APIs and keywords. The sole goal of their detection was to identify the apps' root exploits.

Behavior-based Android malware intrusion detection techniques and signature-based Android malware detection techniques served as the foundation for the development of numerous solutions. The former controls the low level of false positives for invasions and is a straightforward detection technique. The latter, which uses AI algorithms to identify harmful attacks, is based on anomaly detection. Using machine learning and deep learning techniques, such as the DT and deep learning methodologies [5], several research articles addressed the detection and classification of Android malware and attacks. It was demonstrated that conventional machine learning was effective in identifying malware in an Android environment by utilizing the generative adversarial networks approach.

Since the original version of the application is completely safe and devoid of malicious code, such self-updating malware cannot be identified using standard static techniques. Static analysis of the host application is essentially meaningless when dynamic code loading is used to turn a benign application into a malicious one. It is easy to circumvent detection by dynamic analysis tools by deploying the malicious payload in a time-delayed or filtered manner. An attacker might, for instance, offer a benign update on days 1–5, change to a malicious one on day [6], or offer the update to a specific geolocation, filter by carrier, device type, OS version, etc.

Efficient detection mechanisms are necessary to identify zero-day malware, as seen by the sharp rise in malware targeting Android devices. Android lets users download apps from third-party shops [7], which are different from other mobile operating systems in that many of them lack tools or methods to scan submitted software for malware. The Google Play Store filters submitted apps using a program called Bouncer. It has been previously shown, although, that certain basic anti-emulator methods can be used to get around the Bouncer dynamic analysis procedure.

Some writers give summaries of the security model of the Android operating system. The permission-based security mechanism is among Android devices' most crucial security features. Every application outlines which device resources must be accessed, and the user either approves or disapproves the installation of the application based on the required permissions. Permission security model analysis and enforcement for Android has been suggested by a number of writers [8]. The proliferation of genuine malware has shown that users readily trust any application request and install it on their phones, even if they are cautioned about the dangers of accepting dubious permissions.

We provide a dynamic analysis-based method for detecting malware that takes into account system call sequences that are more likely to be found in malicious software than in legitimate applications. The following is an explanation of the reasoning behind our decision. Malware evolution frequently consists primarily of changes made to already-existing malware. Program authors tend to mix or enhance payloads, obfuscation strategies, or infection methods that have already been used in earlier programs [9]. This actually clarifies the reason malware is categorized into families, or harmful applications that have similar traits, behaviors, and methods of deployment.

Only predetermined signatures can be identified using the signature-based detection method, which makes it unsuitable for small mobile devices like Android and degrades the detection rate of new infections because it requires a high-capacity repository to manage signatures. The behavior-based detection approach uses machine learning classifiers to identify attack patterns during malware execution and process behavior in typical scenarios that arise on devices that have malware detected by examining system calls [10], file system logs, and event modules from an Android kernel standpoint.

3 METHODS AND MATERIALS

3.1 Detection of Mobile Malware Using SDN

The OpenFlow protocol-based malware detection methods we employ are described in this section after first describing the planning of our mobile malware detection system.

Design of the system. Inspired by the SDN architecture's adaptability and the finding that the majority of mobile malware necessitates Internet access, we developed a system that detects mobile malware using the SDN framework and real-time traffic analysis. We assume that all mobile devices are connected to a single wireless access point for simplicity's sake [11]. We presume that the techniques outlined can be used to filter out the mobile traffic.

In this system, an OpenFlow controller governs the access point, which is basically a switch with OpenFlow support. The OpenFlow controller sends mobile traffic

to the access point, which then uses a secure channel to receive and install flow entries from the controller. An OpenFlow controller module called the malware detection system is capable of extracting the traffic data that we are looking for. The controller will be instructed to control traffic by configuring flow entries to the switch based on the malware detection module's analysis findings.

This logically centralizes network control and allows for real-time enforcement of security policies from the network layer.

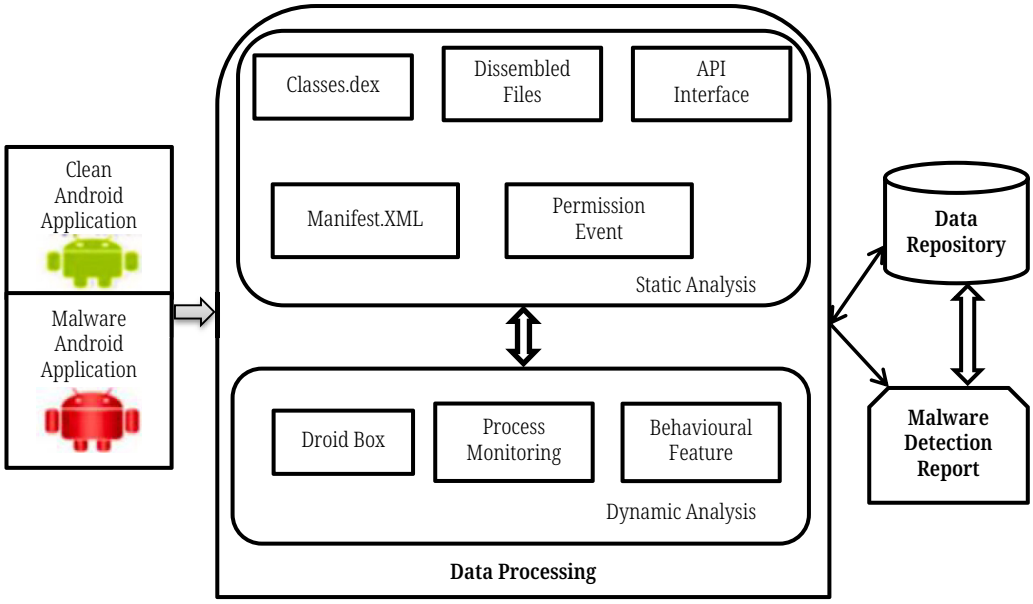

Fig. 2. Extraction and processing of Android features

Apps are distributed and installed on Android devices by the Android Operating Environment through the use of the Android Package Kit (APK) file format.

An APK bundle contains everything a mobile application requires to install and function correctly. Malicious and useful files downloaded from "CICAndmal2017" are decompiled using the APK TOOL to extract the relevant data. Permissions and intents can be extracted from the AndroidManifest.xml file using the AAPT2 tool. The framework for successful feature extraction is shown in Figure 2 [12]. The primary objective of the classification model is to forecast a class label from a range of potential outcomes. Binary classifiers, which identify the simplest lessons to classify, and multiclass classifications, which use the class version to predict a few lessons, are the two main categories of class issues. From a system studies perspective, Android malware detection could be seen as a binary class problem. Our aim in this study is to determine if an Android application is safe to use or dangerous based on static features by using binary classification.

Detection methods. In conventional networks, implementing a real-time traffic inspection strategy necessitates complex cooperation and additional devices. Designing detection algorithms that are easily linked with the OpenFlow controller under SDN architecture allows for logically centralized traffic control. This section describes the algorithms that we used.

- **IP block list:** A simple method of network security is to keep a blacklist of dangerous IP addresses, which can be compiled from publicly accessible sources or historical data, and to instantly block any network traffic involving an IP address

on the blacklist. It is easy to construct a blacklist-based protection mechanism in the OpenFlow protocol in the manner described below. In the event that a connection request packet is received that does not match any flow entry in the OpenFlow switch, the OpenFlow controller can quickly determine the IP address of the connection's destination when a new connection is being established. The IP address will be blocked if it is on the blacklist. If the connection request packet is not received, it will be transmitted to the target host.

- **Ratio of connection success:** We created a connection success ratio-based malware detection technique, which draws inspiration from the finding that the likelihood of an effective link attempt should be significantly higher for a benign host than for a malevolent one. The program seeks to identify scanning worms, which attempt to find and infect compromised devices within a network.

  The controller keeps track of all the pending connections for each host that have not yet received a response in order to apply the OpenFlow protocol in this algorithm [13]. The more pending connections that originate from a host, the more likely it is that the host has been infected. Once the connection request reply is received, the related pending relationship will be removed from the list. The two-way flow entries are subsequently installed by our algorithm to regulate the flow's following packets.
- **Connection throttling:** The study found that whereas an infected machine will attempt to connect to as many different devices as possible during viral transmission, an uninfected machine will act differently: Because the virus is likely to reconnect to recently accessed machines, there is a slower rate of link building and local correlation. A similar trend is anticipated on mobile devices. Every time a new connection request is received, an algorithm that keeps track of each host's Recently Accessed Hosts (RAH) is checked in order to restrict the frequency of connections to fresh hosts and determine whether clients have been affected.
- **Analyzing aggregation:** Malware rarely infects only one person. Therefore, it is reasonable to assume that, particularly in a big network, several other hosts may become infected with malware when one host becomes infected. Furthermore, we anticipate that the infected client will exhibit similar behavioral traits to benign clients in their network activities. Our system, which draws inspiration from this, finds aggregates of "similar" communications to identify the infected hosts.

4 IMPLEMENTATION AND EXPERIMENTAL RESULTS

In order to assess our method's accuracy and effectiveness on actual mobile devices, we chose different distinct Android smartphones. Four common standard devices are among these.

Table 1. Mobitive approach comparison with existing methods

Feature Type	Extraction Time (s)	Classification Method	Accuracy
Opcode Seq	6.066	Deep Learning (CNN)	96.8%
API call Seq	4.516	Deep (CNN)	97.26%
iCFG	298.921	Representation Learning	91.99%
Manifest API Call Opcode Seq	6.893	Deep Learning (CNN)	97.88%
Manifest API Call	1.0516	Deep learning (GRU)	97.76%

Table 2. Multi-class dataset detection outcome

Name	Type	Total Data	Test Data
admogo	Adware	2919	293
adwo	Adware	4883	499
airpush	Trojan	7369	738
anydown	Adware	2344	235
baiduprotect	Adware	4027	303
cnzz	File-Infector	1071	97
commplat	File-Infector	2443	144
domob	File-Infector	6697	570
dowgin	Adware	4224	322
feiwo	Adware	2695	169
fictus	Adware	2436	144
gappusin	Adware	10379	938
igexin	Spyware	4912	391
jiagu	Riskware	3663	266
kuguo	Adware	5032	403
kyview	Adware	2434	143
leadbolt	Adware	6930	593
mecor	Riskware	934	593
plankton	Trojan	3572	83
revmob	Trojan	8518	752
Scamapp	Trojan	969	87
Overall		81,251	8124

Tables 1 and 2 detection result with multi-class data settable illustrates this. In order to verify the impact of our recently added features, we discover that CNN's accuracy rose by 5% and RNNs' accuracy improved by more than 1%. In the meantime [14, 15], we discover that

- Two feature types combined input have a significantly superior result than a single feature type when comparing the results on various feature categories and deep neural networks;
- Generally speaking, RNN models outperform CNN models. On our dataset, GRU models outperform LSTM models in terms of accuracy. Furthermore, MobiTive outperforms the other four methods available with a significant increase in detection accuracy. We find that MobiTive can effectively manage the majority of malware families (i.e., 18/21 get an accuracy larger than 92%) in order to validate the influence on multi-class categorization.

Table 3. Comparison of MobiTive's run-time performance with a dynamic Android analysis tool

CPU Usage	Memory Usage (MB)	Energy Usage	Extraction Time (S)
P_{40}	70	Q_{MX} 0.47	0.47
$P_{10\%}$ X n	140 X n	Q_{LX} t_X n	∞

On common spec devices, we may observe that the detection time costs less than 1% of the overall time. Therefore, cutting down on feature preparation time will significantly boost our detecting system's performance. It serves as a powerful motivator for us as well. Furthermore, the Android OS will always unzip the downloaded Android APK as a result of the installation process. Therefore, the identical step—extracting the same raw data from the target APK—will be involved in both our method and the installation processes. The time spent unzipping each stage of our method will be reduced if we can implement it directly on the Android OS framework. It is a fresh area of study for this discipline going forward. The results of Table 3 indicate that Inspeckage's CPU and energy consumption appear to be superior to MobiTive's if a single application is successfully detected in the same constrained time frame.

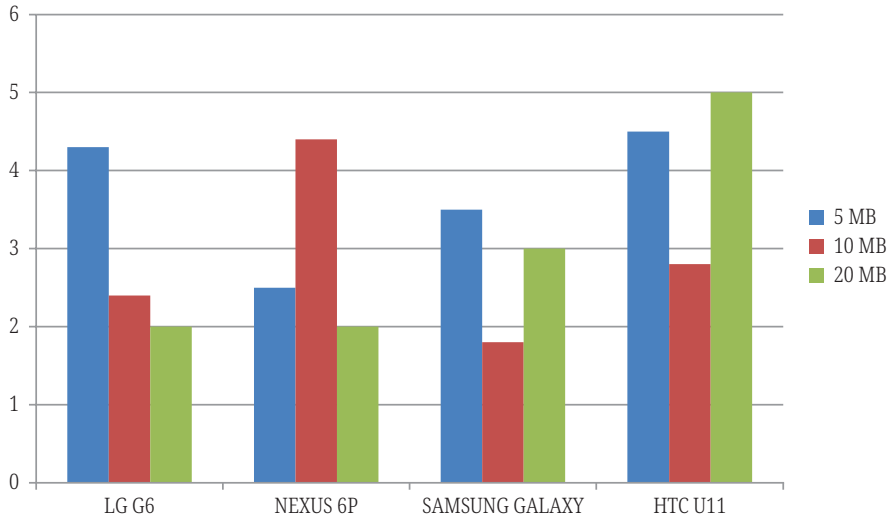

Fig. 3. Time spent extracting on various mobile devices

Figure 3 shows extraction time across various mobile devices with similar specifications. It displays the average feature extraction time of 50 MB apps located between 6.216 and 2.089 seconds.

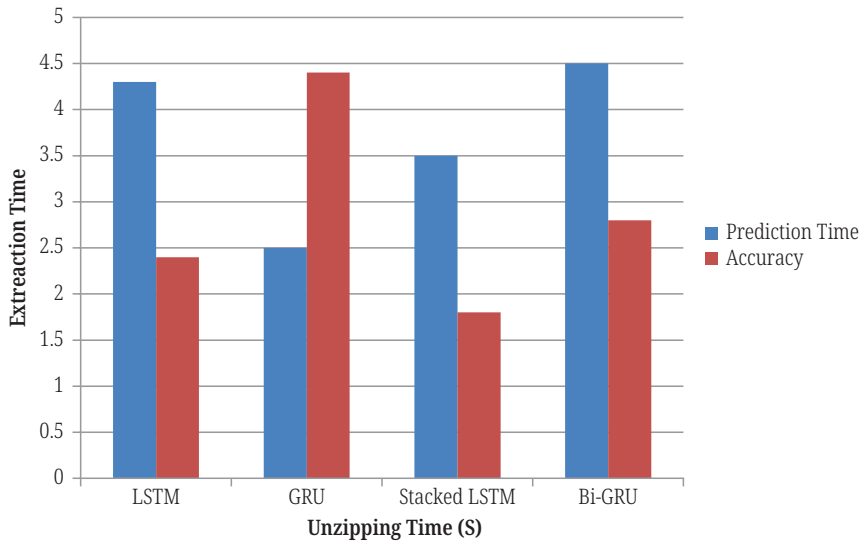

Fig. 4. The RNN models' forecast time and accuracy

Figure 4 shows the Huawei P30's RNN models' accuracy and prediction time. We can see that the pre-trained model with GRU outperforms the other RNN models by comparing their prediction times, which are shown in the histogram. However, the grey accuracy line in this image shows that, with a very slight difference, it has the second-highest accuracy of all of them when compared to the accuracy of bidirectional GRU.

Table 4. Mobitive's accuracy and performance on real mobile devices

Devices	Unzipping Time	Extraction Time
Nexus 6P	0.99	0.75
LG G6	1.05	1.29
Samsung Galaxy J7 Pro	1.53	2.47
HTC UII	0.56	2.24
Huawei Mate 10	0.46	2.76
Huawei P30	0.25	0.25

When comparing the feature preparation and prediction time in typical cases as presented in Table 4, we can see that the detection time only costs less than 1% of the whole time on common spec devices. Therefore, cutting down on feature preparation time will significantly increase our detection performance system. It serves as a powerful motivator for us as well. Furthermore, the Android OS will always unzip the downloaded Android APK as a result of the installation process. Therefore, the identical step—extracting the same raw data from the target APK—will be involved in both our method and the installation processes. The time spent unzipping each stage of our method will be reduced if we can implement it directly on the Android OS framework. It is a fresh area of study for this discipline going forward.

5 CONCLUSION

The purpose of this study was to find out what factors influence the adoption of AI-based chatbots in higher education and to find out how instructors and students perceive these tools. The study's objective was to develop and verify a model that could account for the intricate network of linkages affecting the employment of these AI tools in academic contexts using a PLS-SEM technique. The study was supported by a strict methodology, which started with a thorough literature analysis that guided the creation of a foundational framework, an initial hierarchical model, and an extensive questionnaire.

Android virus detection software comes preinstalled on mobile devices. Depending on the effectiveness of specific characteristics and the efficiency of collecting them, MobiTive can directly provide a reliable detection accuracy and fast response detection service on mobile devices. We test MobiTive on six actual mobile devices in order to confirm its effectiveness and dependability. We also conduct a thorough examination of the presentation trend on more than 200 mobile phones in order to shed further light on this work.

6 REFERENCES

[1] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124579–124607, 2020. https://doi.org/10.1109/ACCESS.2020.3006143

[2] S. Poornima and R. Mahalakshmi, "Automated malware detection using machine learning and deep learning approaches for android applications," *Measurement: Sensors,* vol. 32, p. 100955, 2024. https://doi.org/10.1016/j.measen.2023.100955

[3] A. Alzubaidi, "Detecting android malware using deep learning algorithms: A survey," *Computers and Electrical Engineering*, vol. 119, p. 109544, 2024. https://doi.org/10.1016/j.compeleceng.2024.109544

[4] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in android based mobile devices," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies,* Oxford, UK, 2014, pp. 66–71. https://doi.org/10.1109/NGMAST.2014.57

[5] H. Alkahtani and T. H. Aldhyani, "Artificial intelligence algorithms for malware detection in android-operated mobile devices," *Sensors*, vol. 22, no. 6, p. 2268, 2022. https://doi.org/10.3390/s22062268

[6] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1–18, 2014. https://doi.org/10.1016/j.cose.2014.02.009

[7] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Emulator vs real phone: Android malware detection using machine learning," in *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytic,* 2017, pp. 65–72. https://doi.org/10.1145/3041008.3041010

[8] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 15–26. https://doi.org/10.1145/2046614.2046619

[9] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting android malware using sequences of system calls," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, 2015, pp. 13–20. https://doi.org/10.1145/2804345.2804349

[10] H. S. Ham and M. J. Choi, "Analysis of android malware detection performance using machine learning classifiers," in *2013 International Conference on ICT Convergence (ICTC),* Jeju, 2013, pp. 490–495. https://doi.org/10.1109/ICTC.2013.6675404

[11] R. Feng, S. Chen, X. Xie, G. Meng, S. W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2021. https://doi.org/10.1109/TIFS.2020.3025436

[12] R. Feng, S. Chen, X. Xie, G. Meng, S. W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security,* vol. 16, pp. 1563–1578, 2021. https://doi.org/10.1109/TIFS.2020.3025436

[13] M. Soliman, R. A. Ali, J. Khalid, I. Mahmud, and W. B. Ali, "Modelling continuous intention to use generative artificial intelligence as an educational tool among university students: Findings from PLS-SEM and ANN," *J. Comput. Edu.* 2024. https://doi.org/10.1007/s40692-024-00333-y

[14] A. wael Al-khatib, A. S. Moh'd Anwer, and M. Khattab, "How can generative artificial intelligence improve digital supply chain performance in manufacturing firms? Analyzing the mediating role of innovation ambidexterity using hybrid analysis through CB-SEM and PLS-SEM," *Technology in Society*, vol. 78, p. 102676, 2024. https://doi.org/10.1016/j.techsoc.2024.102676

[15] A. Saihi, M. Ben-Daya, M. Hariga, and R. As' ad, "A structural equation modelling analysis of generative AI chatbots adoption among students and educators in higher education," *Computers and Education: Artificial Intelligence,* vol. 7, p. 100274, 2024. https://doi.org/10.1016/j.caeai.2024.100274

7 AUTHORS

Sandeep Yelisetti is a Senior Assistant Professor in the Department of Information Technology at Siddhartha University, Vijayawada, Andhra Pradesh. With 14 years of teaching experience, he specializes in research on efficient aspects and emotion prediction in sentiment analysis using advanced deep learning models. He earned his Ph.D. in Computer Science and Technology from Sri Krishnadevaraya University, Anantapuramu, Andhra Pradesh, in 2024, and completed his M.Tech in Computer Science and Engineering in 2009. Sandeep has published numerous research papers in SCI and Scopus-indexed journals and has actively participated in several workshops, as well as national and international conferences. He possesses strong skills in data analysis and visualization. He teaches a range of subjects, including Data Mining, Data Visualization, Semantic Web and Social Networks, Business Intelligence, Big Data, Business Analytics, Database Management Systems, and Financial Analytics. His research interests include Data Mining, Data Science, Machine Learning, and Artificial Intelligence (E-mail: y.sandeepit@vrsiddhartha.ac.in).

Dr. P. Suresh, Ph.D., is a Professor in the Department of Computer Science and Engineering at Saveetha School of Engineering, SIMATS, Chennai. He was associated with KPR Institute of Engineering and Technology, Coimbatore, and Vel Tech Multi Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Chennai. He obtained his B.Tech (IT) from Vel Tech Engineering College (Anna University, Chennai) and M.E (CSE) from Vel Tech Multi Tech Dr. Rangarajan Dr. Sakunthala Engineering College (Anna University, Chennai). He did his PhD at Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai. He has been in the teaching profession for the past 15 years and has handled both UG and PG courses. His area of research is the Internet of Things (IoT), Machine Learning and Deep Learning. He has published three books, 35 research articles in International Journals and has presented 15 papers in International and National Conferences. He has attended various Training Program, Workshop and Faculty Development Program sponsored by AICTE related to his interest area. Recently, the students won the Project Innovation Awards for the IoT project under his guidance in various contests and have applied for a patent for the same. He is a lifetime member of IAENG and ISTE (E-mail: sureshpadmanaban.sse@saveetha.com).

Dr. Suresh Kumar Jha is working as an Assistant Professor (Sr. Scale) in the department of Computer and Communication Engineering at Manipal University, Jaipur. He received his B.E (CSE) from Government Engineering College, Kota, and his M.Tech (CSE) from RTU, Kota. He has completed his Ph.D. from MBM University, Jodhpur, Rajasthan. His areas of research are multi-agent network, Consensus theory, graph theory, and distributed algorithms. He has 9 years of academics as

well as 3 years of IT industry experience as a technology trainer and technical lead. He teaches courses such as Algorithms, Data Structures, Operating Systems, Theory of Computation etc. (E-mail: suresh.jha@jaipur.manipal.edu).

Neha Arora, Assistant Professor, is an accomplished academician and industry expert with over 13 years of experience in teaching and corporate environments. A Gold Medalist in M.Tech (CSE) from Uttarakhand Technical University and a Bronze Medalist in MCA from Graphic Era University, she excels in machine learning, natural language processing, and big data analytics. With a strong portfolio of research publications, patents, industry consultancy, and government-sponsored projects, Neha combines academic rigor with practical expertise. Her dedication to innovation and problem-solving positions her as a leader in advancing AI and data science solutions (E-mail: neha.arora@presidencyuniversity.in).

Ketan Anand, Freelance Trainer, is a seasoned, forward-looking academician, having a total of 10 years of experience teaching and training the students for UG and PG in competitive programming and the sophistications of AI. He did B.Tech (IT) M.Tech (Computer Science major in AI and ML) from West Bengal University of Technology and Central University in 2012 and 2015, respectively. Currently pursuing a PhD in Computer Science Engineering from the National Institute of Technology Patna, Patna. His area of research is NLP using AI. His area of interest lies with Mathematics, Data Structures and Algorithm, Natural Language Processing and Artificial Intelligence. I am married and living with my wife and daughter in Hyderabad, India (E-mail: ketan@anandketan.in).

Dr. Vedala Naga Sailaja, Associate Professor, is a faculty of management with a distinguished career in academia and research. With 24 years of teaching experience, she has honed her expertise and made significant contributions to the field of management. In 2011, she earned her Doctoral Degree from Acharya Nagarjuna University and proved her academic excellence by qualifying for the AP State Eligibility Test (SET). Dr. Sailaja is an esteemed author; she has authored two books, “Security Analysis and Portfolio Management” and “Auditing,” and also contributed chapters for many books. As a prolific researcher, she has left a significant impact on her field with a total of 60 published articles. Among these, 25 have been recognized and indexed by Scopus, showcasing the depth and quality of her research. Her other works have found recognition in esteemed journals indexed by ABDC and multiple indexing databases. Her exceptional academic contributions have garnered several well-deserved awards, including the Academic Excellence Award in 2017, the Best Senior Faculty Award in 2018, and the Best Teacher Award from KL (Deemed to be) University in 2018. Driven by a passion for continuous learning, she has enriched her knowledge through certificate courses from prestigious institutions like E&Y, NPTEL etc. She has presented 56 papers in national and international seminars, where peers and scholars appreciate her insights and research findings (E-mail: drvedalasailaja@kluniversity.in).