Deep Learning-Based Android Malware Detection with CNN-GRU Model

MSc Research Project
MSc in Cybersecurity

Harisankar Kalathil Salim
Student ID: x23151552

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Harisankar Kalathil Salim		
Student ID:	x23151552		
Programme:	MSc in Cybersecurity	**Year:**	2023-2024
Module:	MSc Research Project		
Supervisor:	Vikas Sahni		
Submission Due Date:	12-08-2024		
Project Title:	Deep Learning-based Android Malware Detection with CNN-GRU model		
Word Count:	7811	**Page Count**	22

Signature: [illegible]

Date: 12-08-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Deep Learning-based Android Malware Detection with CNN-GRU Model

Harisankar Kalathil Salim
x23151552

Abstract
Android malware presents substantial security hazards to mobile users worldwide, jeopardizing personal, financial, and device data. Given the exponential increase in these dangers, it is imperative to implement effective detection systems to safeguard consumers and preserve the integrity of the smartphone ecosystem. Conventional approaches frequently lack strong security measures because of the ever-changing characteristics of malicious software. Therefore, advanced techniques based on deep learning have emerged as promising approaches to enhance detection accuracy and efficiency, this study developed a sophisticated hybrid detection model using Convolutional Neural Networks (CNN) and Gated Recurrent Units (GRU) to classify Android applications as either benign or malicious, thereby contributing to the ongoing effort against mobile malware threats. The research involved preprocessing the dataset in four distinct ways, including SMOTE and Chi-square to address imbalance and optimize feature selection. The result demonstrated that the developed CNN-GRU model achieved superior performance with the highest accuracy of 99% accuracy surpassing several existing models

Keywords: CNN-GRU Model, SMOTE, Chi-square

1 Introduction

The installation of malicious applications can cause major problems for people who use Android smartphones, the applications that are installed can be used to gather data for targeting advertising, commit fraud, secretly collect personal information about the user, retrieve data, such as capturing passwords, harmful activities and enabling malicious activities (e.g.: - turning a device into parts of botnets) (Tully and Mohanraj,2017).
Android maintains a dominant position in the global smartphone market with a 70.5% share. Among Android devices, Samsung leads with a market share of 35.30% followed by Xiaomi at 14.70%, Oppo at 10.00%, vivo at 9.50%, Huawei at 6.30%, RealMe at 4.00%, Motorola at 3.30% and others collectively holding 16.90% [1]
By the end of 2025, there will be 3.9 billion Active users spread across 190 countries. This shows the domination of Android in the Global market[2], global mobile spending is projected to exceed $270 billion. Notably, 32% of users aged 11-20 regularly use mobile apps daily, and 45% have numerous applications. For the 31-40 age group, 17% use mobile applications daily, with 38% having a reasonable application. In the 41-50 age range, 16% use applications daily, with 24% having a limited number of applications. Among those aged 51-60, 8% use applications regularly with 22% having a smaller selection of applications [3]

[1] https://www.enterpriseappstoday.com/stats/android-phones-statistics.html

[2] https://www.bankmycell.com/blog/how-many-android-users-are-there

[3] https://www.enterpriseappstoday.com/stats/mobile-app-industry-statistics.html

The global mobile application market is valued at USD 206.73 billion in 2022, and is predicted to grow at a compound Annual Growth Rate (CAGR) of 8.83% from 2022 to 2027 by the end of 2027 revenue generated by the mobile application industry will be US$673.80. billion. Key growth factors include China and India. Additionally, the evolution of the online retail sector and rising mobile gaming popularity play a significant to this growth. The COVID-19 pandemic also increased the downloads of games, social media, and entertainment applications[4]
Google Play Store remains the leading app store, over 113 billion apps and games were downloaded from Google Play store in 2023. Currently, around 2.61 billion apps and games are available in Google Play store[5]. Other major app stores include Xiaomi Apps Store, Galaxy Store, Vivo Appstore, RealMe Store, Huawei AppGallery, Amazon App Store, and Tencent App Store. Where Huawei AppGallery had over 580 million users by the end of 2022[6] . While the Amazon App Store had 528,164 apps 80.81% of apps are free and 19.19% are paid apps[7]. Tencent App Store has 64,194 apps, all of which are free apps for use. [8]
Xiaomi's App has 200 billion downloads all around the world and 564 million Global monthly active users[9]. Samsung's Phone is available in over 180 countries, having hundreds of millions of active Galaxy Device users who download billions of apps from the Galaxy store.[10]
In the Chinese market Appstore Market, Huawei AppGallery having the largest share of 22.03% market share with 284.35 million monthly active users (MAU) and 37.82 million daily active users (DAU). Tencent Appstore comes next with a 13.72% share having 177.07 million MAU and 40.60 million DAU. Oppo Software store has a monthly active users around 162,281,900 and daily active user count of 44,236,300 giving it a market share of 12.58%. ,The Xiaomi Market on other hand maintains a market share of 12.14% having 156,671,100 MAU and 56,278,100 DAU. The Vivo App Store holds a market share of 10.27% with 132,491,900 MAU and 24,765,900 DAU.The Samsung App Store holds a
A market share of 4.89% with a Monthly active user (MAU) count of 63,09,100 and 5,562,00 Daily active users respectively, following closely is the 360 Mobile Assistant with a market share of 4.13% and a Monthly active user (MAU) count of 53,264,800 and 9,89,200 Daily active users (DAU), following up Baidu Mobile Assistant having 3.64% share hold with 47,406,300 Monthly active users (DAU) and 3,827,600 Daily Active Users (DAU) lastly Lenovo Le Store with a share of 0.44% with 5,723,700 Monthly Active Users (MAU) and 355,400 Daily Active Users (DAU)[11]
Given the widespread usage and market penetration of Android devices and Android apps, addressing Android malware threats is crucial for ensuring user security and maintaining

[4] https://www.enterpriseappstoday.com/stats/mobile-application-revenue-statistics.html

[5] https://www.businessofapps.com/data/google-play-statistics/

[6] https://www.enterpriseappstoday.com/stats/huawei-statistics.html

[7] https://42matters.com/amazon-appstore-statistics-and-trends

[8] https://42matters.com/tencent-appstore-statistics-and-trends

[9] https://global.developer.mi.com/document?doc=quickStart.aboutGetApps

[10] https://developer.samsung.com/galaxy-store/discover-galaxy-store.html

[11] https://appinchina.co/market/app-stores/

trust. With such a vast user base and expensive app ecosystem, robust measures to combat malware are essential for protecting personal data, ensuring safe app downloads, and sustaining the overall health of the Android Platform Globally.

Research Question 1. Does the performance of the CNN-GRU model built in the study with the Drebin dataset surpass that of other existing deep learning studies using the same Derbin dataset in terms of accuracy?
Research Question2. Can employing SMOTE and Chi2 feature selection enhance the model's performance in detecting Android malware?
Research Question3. Does cross-k validation technique normal training making any significant impact on the performance of models?

Section 1 of the report comes first; Section 2 reviews the literature with reference to traditional approaches, hybrid models, and algorithms under discussion. Section 3 includes the specifics of the approach used for developing the Android malware detection model suggested in this work. The information related to the design criteria of the system suggested for the research is found in Section 4 of the report. Section 5 comprises the specifics related to the application of the Android malware detection technique. Section 6 covers the specifics related to the outcomes of the research and the evaluation of the performance of the model developed in this work. Section 7 offers the study's findings as well as potential improvements the work can get.

2 Related Work

The review of the different literature associated with the detection of Android malware using machine learning and deep learning methods are presented in this section

2.1 Traditional Android Malware Detection Methods

Growing concerns about dangerous software on Android systems have spurred a lot of study on malware detection techniques. (Pardhi, Jitendra Kumar Rout and Niranjan Kumar Ray, 2021) Use static code analysis in a signature-based method to find known malware. Their solution tells users of possible security threats by classifying applications according to the permissions they want. Including safe browsing and app lockdown, it demonstrates efficiency with low system resource needs. It depends on a thorough and constantly updated malware signature database, though, which might be a drawback when new malware types develop. After that, (Alam et al., 2017) introduce DroidNative, a fresh method aiming at native code to fill in voids left by tools mostly analysing Java bytecode. With a 93.57% detection rate and strong resistance to obfuscations, DroidNative uses intermediate language MAIL to detect malware efficiently using signature generating methods. The complexity and diversity of native code as well as very high computing needs for real-time detection could restrict the method even with its great speed.Using feature extraction and classification approaches, (Apvrille and Apvrille, 2015) presented the SherlockDroid framework shifts emphasis to spotting unidentified malware. SherlockDroid improves detection accuracy by combining several algorithms and marketplace crawlers, therefore focussing on fresh dangers. Comprehensive testing confirms its effectiveness, however depending on proper feature extraction and the possibility of false positives still present difficulties.
Finally, (Liang et al., 2022) presents a structured approach to behavior-based detection by means of a formal method using process algebra to describe and evaluate Android application behaviours. Extending the π-calculus theory, this approach provides theoretical foundation for precisely characterising app behaviour, hence increasing detection accuracy. Still, it might

be challenging to precisely replicate behaviours and guarantee minimal false positive rates. Although conventional approaches show different degrees of success in malware detection, they can have restrictions including a reliance on updated signature databases, computational complexity, and difficulties in modeling app behaviour.

2.2 Existing Machine Learning and Deep Learning Models in Malware Detection

Existing Android malware detection methods are now applied using ML and DL, therefore displaying a high degree of accuracy, strength, and relevance. (Kouliaridis and Kambourakis, 2021). examined conventional ML models and discovered Naive Bayes (NB) with 99.8% detection accuracy to be the best (Gupta et al., 2020). On the Malgenome dataset, Random Forests (RF) performed best with a TPR of 0.99 and an FPR of 0.014, however their shortcomings in dynamic behaviour identification were recognised. While (Feng et al., 2021) created MobiTive, a real-time DL-based malware detection system with 96.75% accuracy, subject to adversarial assaults, (Almarshad et al., 2023) employed a Siamese neural network with one-shot learning and achieved 98.9% accuracy. (Alkahtani and Aldhyani 2022) investigated several ML and DL models; SVM scored 100% on the CICAnd Mal2017 dataset while LSTM scored 99.40% on the Drebin dataset. With severe gradient boosting, (Hadiprakoso, Kabetta and Buana, 2020) suggested a hybrid static-dynamic analysis method attaining 99.36% accuracy. Targeting zero-day assaults using static analysis mixed with ML and DL models, (Sara and Hossain 2023) obtained a 96% F1 score on the Drebin dataset. With static APK analysis, trained models Logistic Regression, KNN, and Decision Tree, attaining 97.8%,98.6% and 97.6% accuracy, (Vanusha D et al., 2024) Combining Grammatical Evaluation with XGBoost, (Jundi and Hasanen Alyasiri, 2023) obtained up to 99.28% accuracy over many datasets. Reaching 97% accuracy, (Tian et al., 2024) combined static and dynamic ML analysis for privacy leakage detection. Reaching 98.36% accuracy, (Kirubavathi G and Nithish S, 2024) presented a dynamic ensemble learning system with explainable AI. Showing great promise for efficient distributed security systems, (Ullah et al., 2024) utilised a semantic-based FL approach with transformer-based transfer learning to achieve 99.38% detection accuracy on CIC-And Mal2017 and 99.14% on CIC MalDroid2020 datasets.

.(Ganesh et al., 2017) demonstrated a CNN model with a 93% accuracy in identifying malicious apps through permission patterns analysis. Building on this, (Brahami Menaouer et al., 2023). reported a remarkable 98.50% accuracy by integrating stacked AutoEncoders for dimensionality reduction with CNNs for classification on the Drebin-2015 dataset. (Muhammad Aamir et al., 2024) further improved results with the AMDDL model, achieving an extraordinary 99.92% accuracy, alongside high precision (98.61%), recall (99.16%), and F1-score (98.88%) on the Drebin dataset. These models' efficacy extends beyond a single dataset. (Lachtar, Ibdah and Bacha, 2020) evaluated various CNN architectures, including LeNet, AlexNet, and InceptionV3, attaining a 99.7% detection accuracy with LeNet on a balanced dataset and addressing performance on imbalanced datasets. (Marwa Ben Jabra et al., 2023) employed seven pre-trained CNN models and a custom CNN model across Drebin and Malimg datasets, achieving an average accuracy of 98.26% and a peak accuracy of 100%. They effectively utilized regularization techniques such as L1 and L2 regularization, dropout, and data augmentation to mitigate overfitting and enhance generalization. Practical considerations like energy efficiency were also addressed, with LeNet found to be the most efficient, with a runtime of 342 ms and energy consumption of 1 J for classification.Despite these advancements, challenges remain. Studies acknowledge potential false positives and the dynamic nature of malware threats (Ganesh et al., 2017; Brahami Menaouer et al., 2023).

(Muhammad Aamir et al., 2024) pointed out issues related to model interpretability and scalability. Future research directions include exploring different CNN architectures and combining CNN models with other machine learning techniques to enhance performance further (Marwa Ben Jabra et al., 2023).Hybrid models have shown superior performance over traditional and single deep learning approaches.

(Lu et al., 2020) introduced a hybrid deep learning model combining Deep Belief Network (DBN) and Gated Recurrent Unit (GRU), achieving 97.79% detection accuracy. . (Dong, Shu and Nie, 2024) presented a hybrid CNN and Deep Neural Network (DNN) model, achieving 96.80% accuracy on the Drebin and Google Play Store datasets by integrating permission features and API call graphs for comprehensive feature representation. (Xu et al., 2020) proposed a hybrid CNN and Long Short-Term Memory (LSTM) model for detecting malicious behaviour in power systems, demonstrating superior accuracy and efficiency. Both studies highlight the effectiveness of combining CNNs with other deep learning frameworks, offering substantial benefits for malware detection. These hybrid models leverage the strengths of individual components to provide a deeper understanding and more comprehensive feature representation, enhancing detection accuracy, robustness, and resistance to sophisticated malware attacks. The success of these models in their respective fields underscores the potential of hybrid architectures for a wide range of applications, including malware detection, where combining CNNs with models like GRU can enhance accuracy and efficiency in cybersecurity solutions

2.3 The Critical Role of SMOTE and Chi-square Feature

Using RFECV for optimal feature selection, the research by (Mehedi Hasan Shakil and Md. Mynul Hasan, 2023) showed an ensemble deep learning strategy combining Bi-LSTM, Bi-GRU, and 1D CNN, with a startling accuracy of 98.99% on the Drebin dataset. Likewise, (Liu, Zhang and Long, 2022) presented an enhanced CNN model, BIR-CNN, which combines Batch Normalisation and Inception-Residual networks, obtaining an accuracy of 0.97 and AUC of 0.99 on the CICAnd Mal2017 dataset, thereby proving the potency of advanced feature learning approaches. Using Chi-Square feature selection mixed with NLP approaches, (Areeg Fahad Rasheed, M. Zarkoosh and Sana Sabah Al-Azzowitz, 2023) concentrated on the IoT ecosystem and achieved an amazing accuracy of 99.93% using SVM on the IoTPot dataset. Achieving 98.02% accuracy in Android malware detection, (Dhalaria and Gandotra, 2020) also used Chi-Square feature selection in conjunction with an ensemble learning technique. In order to maximise Random Forest classifiers, (Habib and Hafsa Binte Kibria, 2024) used several feature selection strategies like Chi-Square and Mutual Information alongside sampling techniques like SMote-NC, so obtaining 98.5% accuracy and stressing explainable AI for model transparency. (Eom et al., 2018) highlighted how well feature selection enhanced Random Forest's performance on the Malware Genome Project data. By means of Gain Ratio and ReliefF, (K., Chakravarty and Varma P., 2020) obtained 94.47% accuracy with a smaller feature set. (Ahsan, Gomes and Denton, 2018) showed how well SMote balanced datasets by mproved efficiency of model detection from 89.87% to 97.17% using oversampling techniques, greatly raising XGBoost phishing detection accuracy. Using static properties of Windows executables, (Aslam et al., 2020) underlined how better Random Forest performs in malware categorisation. Finally, (Khoda et al., 2020) outperformed conventional methods by addressing the unbalanced data problem in mobile malware detection using a new synthetic oversampling technique. These investigations taken together highlight the need of Chi-Square feature selection, SMote for data balancing, and cross-valuation for accuracy increase in building strong and effective malware detection systems.

Table 1: summary of the literature

Study	Methods	Dataset	Result	Limitation
(Pardhi, Jitendra Kumar Rout and Niranjan Kumar Ray, 2021)	Signature-based approach	Norton database(collection of malware signatures)	Implemented a malware scanner	Limited to signature-based detection
(Alam et al., 2017)	Static analysis using Malware Analysis Intermediate Language (MAIL)	5490 Android applications (1758 malware)	Detects 93.57% of malware with a 2.7% false positive rate	Limited to static analysis, struggles with excessive control flow obfuscations and packed malware
Brahmi Menaouer et al., (2023)	Stacked AutoEncoders (SAE), Convolutional Neural Networks	Drebin-215 dataset	Achieved accuracy of 98.50%	no feature selection methods considered
Lu et al., (2020)	Deep Belief Network , Gate Recurrent Unit	Mixed datasets from Google Play, APKpure, VirusShare, PRAGuard	The hybrid DBN-GRU model attained 97.79% accuracy in detecting Android malware, showcasing robust performance across varied datasets.	The computational complexity associated with deep learning methods posed challenges in practical deployment.
(Muhammad Aamir et al., 2024)	CNN model	Drebin dataset	The model achieved an outstanding accuracy of 99.92%	critical issues in interpretability and scalability of the model for practical deployment
(Almarshad et al., 2023)	Siamese neural network with one-shot learning	Drebin dataset	achieved 98.9% accuracy	The model it may still struggle with completely novel malware
(Alkahtani and Aldhyani, 2022	SVM,KNN,LDA,LSTM,CNN-LSTM models	CICAndMal2017 - Drebin	SVM achieved 100% accuracy with CICAndMal2017 dataset - LSTM achieved 99.40% accuracy with Drebin dataset	LDA performed poorly,autoencoder less effective compared CNN-LSTM models
(Hadiprakoso, Kabetta and Buana, 2020).	SVM,K-NN,MLP,Random Forest,Decision Tree, Naïve Bayee, GB	Drebin dataset,	GB achieved 96% , random forest 95.91%.MLP achieved 96%,Naïve Bayes achieved 80%, Decision Tree 92%, SVM 94%, K-NN 80%	Significant Training time for SVM. The accuracy for Naïve bayes and K-NN Is not particularly high
(Sara and Hossain,	Random Forest ,LR,SVM, multi model	Drebin dataset	Random forest 93%,LR 83%,SVM	Requires extensive feature extraction

2023)	opcode+permission+ API (CNN)		89%, multi model opcode+permission+ API (CNN) 96%	and combination performance depends on feature selection and tuning
(Dong, Shu and Nie, 2024)	CNN-DNN model	Drebin dataset 5560 malware samples and benign samples 5560 from google play store and other app stores	Achieved accuracy 96.80%	10-cross-fold validation chosen and Requires more training time
(Vanusha D et al., 2024)	Decision Tree, Logistic Regression, KNN	Drebin dataset	Decision Tree Achieved accuracy 97.6%, Logistic regression achieved 97.8 and KNN achieved 98.6	Even with the use of GridSearchCV hyperparameter tuning. Might still overfit the training data
(Eom et al., 2018).	Pre- and post-feature selection procedures	API data	Almost 100% accuracy in detection	High computational cost due to extensive feature selection
(Ahsan, Gomes and Denton, 2018)	Oversampling methods	dataset from UCI Machine Learning Repository	Improved efficiency of model detection from 89.87% to 97.17% using oversampling techniques	Overfitting potential due to oversampling

3 Research Methodology

Given the Current Scenario, the Usage of Android mobile phones is increasing rapidly .broad criteria of people using Android phones not just teenagers but also older adults and those who are less technologically savvy. As a result, they may unknowingly install applications without understanding the potential dangers Compared with IOS Android remains more affordable in the market which makes Android users more prone to malware attacks.
This situation shows the need for advancement in Android malware detection. One method that has been used effectively for the detection of malicious smartphone applications is machine learning (Chowdhury et al.., 2023). With the rapid increase in Android malware, it is crucial to continuously update research on machine learning for malware detection. Additionally, new machine-learning techniques should be explored to enhance the detection of Android malware. Focusing on static features of mobile applications allow us to identify malicious apps before they are run or installed. This paved the idea to develop a deep learning hybrid model using the Android malware detection AI model

3.1 Selection of Dataset

Two main criteria dominated the process of choosing a dataset: the availability of trustworthy data on malicious Android application static features and the inclusion of static features from Android applications. First under consideration were a number of datasets, including the

Android Malware Genome Project and the AndroZoo collection. These datasets also included dynamic elements, which did not fit the particular criteria of the research, though. In the end, the Drebin dataset was chosen since it is well-known and emphasises only static features, which is perfect for Android malware detection. Peer-reviewed research have made great use of the Drebin dataset, therefore guaranteeing its relevance and dependability. Furthermore, employing this dataset enables performance comparison with other studies, thereby opening the path for the creation of a unique Hybrid deep model for Android detection grounded on this dataset**.**

3.2 Model Selection

Examining the current research on model selection for Android malware detection shows that many recent studies especially those involving deep learning hybrid models those including CNNs are aiming higher on the Drebin dataset (Muhammad Aamir et al., 2024). CNN models are adept at spotting trends in feature sets that is, particular combinations of permission that can point to fraudulent activity, including access to contacts, SMS, and the internet. Though the Drebin dataset is not naturally time-series data, the sequence and connections between stationary features nonetheless provide important information. In this sense, GRU models are useful since they can learn these dependencies and help the model to realize how the presence or absence of some features, when paired with others, can indicate malware. For example, a GRU can evaluate the sequence of events if permissions or API calls follow a certain order, which might be absolutely vital for spotting malware trends. Combining CNN and GRU helps a hybrid model to maximize the advantages of both methods: While GRUs capture the sequence and dependencies inside these features, which is crucial for efficient malware identification, CNNs are skilled in extracting significant local patterns and interactions between static features.

3.3 Considering feature selection and data balancing

It was found that during preprocessing imbalances between malicious and non malicious samples were found in the dataset. SMOTE, an oversampling method creating synthetic samples for the minority class, was introduced experimentally to solve this problem even if the imbalance was not significant.

Apart from SMOTE, for feature selection Chi-square algorithms were considered. With 215 features in the dataset, the Chi-square method helps to find the most pertinent features having a statistically significant correlation with the target variable. This method not only lowers the dimensionality of the data but also improves the capacity of the model to concentrate on the most important indicators of malicious behavior.

3.4 Experimental Analysis of Model Performance

Four different approaches of preprocessing the Drebin dataset produced four independent datasets for study. The first dataset was preprocessed which involved removing any null values and undesired columns were deleted. Using feature selection where 60 features were chosen from the initial 215 for the second dataset While (Brahmi Menaouer et al. 2023) obtained 98.5% without feature selection, previous research include (Almarshad et al. (2023) selected 40 features with an accuracy of 98.9%. Experiments involving 60 features were carried out to investigate this more. To solve class imbalance, the third dataset was handled with both feature selection and SMOTE. The fourth dataset underwent Smote alone during preprocessing.

Training on these four datasets and comparing their performance will help to find the ideal performing model. This approach aims to assess the performance of the AI model and the influence of every preprocessing step feature selection, SMOTE, or a mix of both. Each of the four datasets was trained using a CNN-GRU model both with a normal training of dataset and with k-fold cross-validation, therefore guaranteeing a strong comparability. By means of this study, can ascertain whether training the CNN-GRU model on various preprocessed datasets results in any appreciable performance variations.

.

3.5 Evaluation and Comparison

The performance of the four models was evaluated based on several metrics, including F1-score, accuracy, precision, recall, as well as testing and training time. Additionally, the accuracy of the four models was compared between those trained using the standard method and those trained with k-fold cross-validation. Finally, the optimal model was selected, and its performance was compared to existing models.

4 Design Specification

4.1 CNN-GRU model Architecture

This model uses CNN's feature extraction strengths as well as GRUs for sequence learning. CNNs are good at spotting unique patterns, like odd combinations of permissions or frequent API calls pointing to malware. Conversely, GRUs excel in catching temporal dependencies which are essential for comprehending the flow of events normal for malware. The model can identify suspicious patterns of API requests, for example, accessing sensitive data and then network communication. Particularly on datasets such as Drebin, this combined approach of CNN and GRU model can be a useful tool for malware detection since its capacity to manage high-dimensional data and learn complex feature interactions improves its detection accuracy, so reducing false positives and negatives. The arrangement of this model made use of the following libraries. **Pandas** for data manipulation; **scikit-learn** for model selection and preprocessing; pickle for serialisation; **tensorflow** and **keras** for deep learning model building; matplotlib and seaborn for data visualization and **numpy** for numerical calculations. Conv1D layers of the model are meant to progressively extract and refine sequential data characteristics. To capture first patterns, the first Conv1D layer employs 256 filters with a 3-pixel kernel size ReLU activation. Maintaining a kernel size of 3, the second Conv1D layer decreases the number of filters to 128 and keeps processing the features via ReLU activation, hence enabling more complicated representations. Focussing on higher-level features and maintaining the kernel size and ReLU activation, the third Conv1D layer further downsplays the filters to 64. At last, the fourth Conv1D layer with 32 filters, a kernel size of 3, and ReLU activation polishes the feature maps even more, so guaranteeing the model catches the most abstract and high-level patterns required for the next layers to operate effective categorisation. Following every Conv1D layer, batch normalisation is used to stabilise and speed up training; dropout at a rate of 0.3 helps to prevent overfitting. Following the second, third, and fourth Conv1D layers to lower dimensionality, Max Pooling 1D with a 2 pool size then capture temporal dependencies with two GRU layers; the first GRU layer has 128 units and return sequences enabled; the second GRU layer has 64 units. Dropout and recurrent dropout set to 0.3 are used both at GRU layers. ReLu activation and dropout abound in the dense layer; the final output dense layer comprises one unit with sigmoid activation for binary classification. The Adam optimiser, binary cross-entropy loss function, and accuracy measure build the model to assess performa

5 Implementation

This research study focuses on detecting Android malware using a CNN-GRU model. The Drebin dataset was utilized, and eight experimental studies were conducted. The dataset underwent four distinct preprocessing approaches: the first included data processing with SMOTE, the second combined feature selection with SMOTE, the third involved only feature selection, and the fourth applied basic preprocessing. During the preprocessing stage, label encoding and feature selection processes were saved. These four newly preprocessed datasets were then trained using the CNN-GRU model, both with k-fold cross-validation and with a train-test split, resulting in a total of eight experiments. The performance of the models was subsequently evaluated, and the models were saved for integration into the GUI application

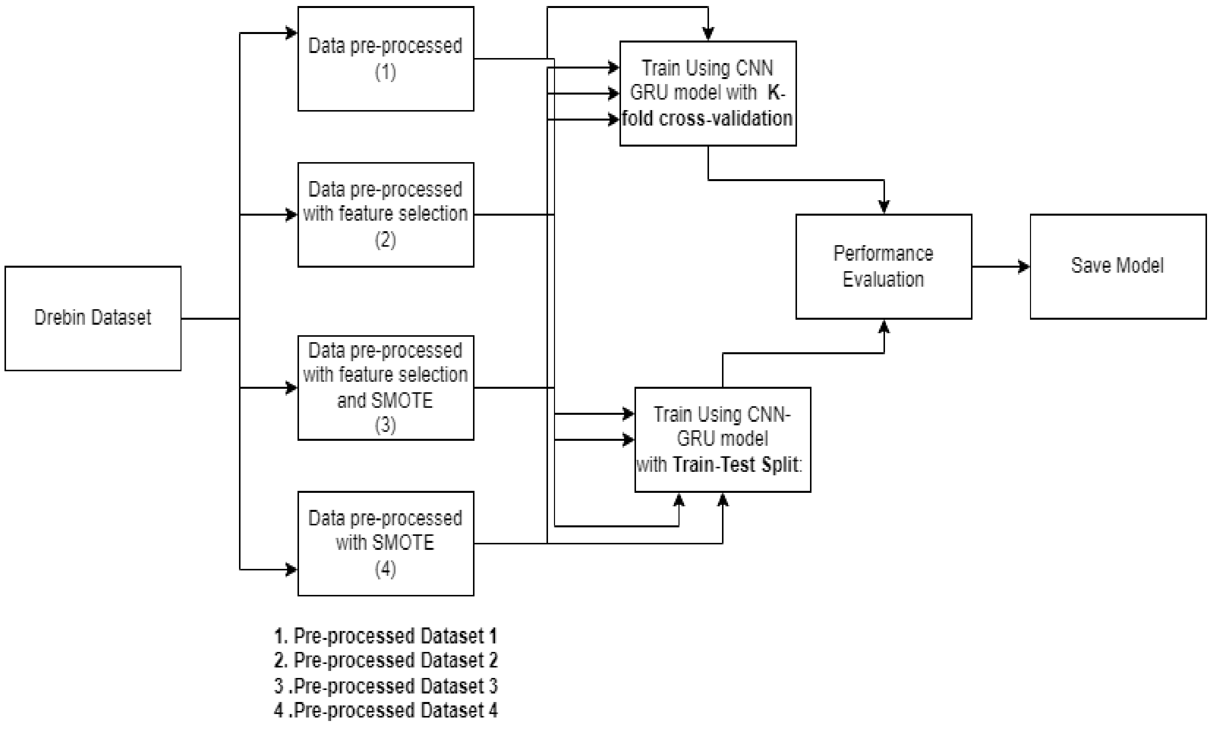
Drebin Dataset
Data pre-processed (1)
Data pre-processed with feature selection (2)
Data pre-processed with feature selection and SMOTE (3)
Data pre-processed with SMOTE (4)
Train Using CNN GRU model with K-fold cross-validation
Train Using CNN-GRU model with Train-Test Split
Performance Evaluation
Save Model
1. Pre-processed Dataset 1
2. Pre-processed Dataset 2
3 .Pre-processed Dataset 3
4 .Pre-processed Dataset 4

Figure 1: Workflow of CNN-GRU-Based Android Malware Detection

5.1 Dataset

The Drebin dataset [12] is an extensive collection specifically created to support research in Android malware detection. The dataset comprises 129,013 Android applications, out of which 5,560 have been classified as malware samples belonging to 179 distinct malware families crawled from August 2010 to October 2012. Each application in the dataset is represented by a feature vector encompassing several key attributes (Arp et al., 2014):

[12] https://www.kaggle.com/datasets/likhithadurusoju/drebindataset

- **Hardware Components**: This includes access to hardware features like the camera and GPS, which can be indicators of malicious intent.
- **Requested Permissions**: Permissions listed in the manifest file, often signal potentially harmful behavior
- **App Components**: The types and names of components such as activities, services, content providers, and broadcast receivers, which may help identify malware. **Filtered Intents**: Intents used for inter-component communication, often exploited by malware for actions like Boot completed.
- **Restricted API Calls**: Critical API calls that require specific permissions, which may reveal malicious activities.
- **Used Permissions**: Permissions utilized by the application, providing an overview of its behavior.
- **Suspicious API Calls**: Calls accessing sensitive data, network operations, SMS, external commands, and employing obfuscation techniques.
- **Network Addresses**: IP addresses, hostnames, and URLs used for malicious network connections.

5.2 Dataset Preprocessing

Four alternative approaches of preprocessing the dataset helped to produce separate pre-processed datasets. **Pre-processed Dataset 3** loaded necessary libraries including pandas, SelectKBest, chi2, LabelEncoder, and SMOTE. Drebin.csv was the source of the dataset; the last row was deleted and null value columns were noted. The target variable "class" was evaluated for class imbalance; columns with the "?" character were found; the pointless "TelephonyManager.getSimCountryIso" column was deleted. LabelEncoder encoded the "class" column; features (X) and the target variable (y) were split; SelectKBest with a chi-squared statistic chose the top 60 features. Oversampling the minority class using SMOTE helped to balance the dataset, producing a balanced dataset preserved as "Drebin_balanced_dataset".Though without using SMOTE, the approach for **Pre-processed Dataset 2** was same to Pre-processed Dataset 3. Importing fundamental libraries including pandas, SelectKBest, chi2, and LabelEncoder, the same preprocessing techniques were followed. The dataset stayed unbalanced, nonetheless, and was saved as "Drebin_unbalanced1_dataset." **Pre-processed Dataset 1** loaded necessary libraries like pandas and LabelEncoder. The dataset was read; the last row was deleted; null value columns were noted; and columns with the '?' character were found. The "class" field was encoded with LabelEncoder while the "TelephonyManager.getSimCountryIso" column was deleted. The cleaned dataset was stored as "Drebin_cleaned_dataset," after separating features (X) and the target variable (y). **Pre-processed Dataset 4** loaded necessary libraries including pandas, LabelEncoder, and SMOTE. Following the same procedures as in Dataset 3, SMOTE was used to oversampling the minority class so balancing the dataset. This produced a cleaned-up and balanced dataset saved under "Drebin_cleaned_balanced_dataset".

5.3 Feature Selection

In the context of a dataset, feature selection is the act of choosing, from the initial collection of features, a subset of pertinent features (Bahassine et al., 2020).. Reducing overfitting, raising accuracy, and accelerating training time will help to enhance the machine learning model's performance. In order to improve model performance by keeping the most pertinent features and therefore lowering the dimensionality of the dataset, feature selection was applied in this work. This procedure consists on choosing the top attributes most likely to influence the target variable. In this study, the Chi-squared approach is applied in feature selection. The features start off apart from the target variable. The Chi-squared test is then used to assess the relevance of every feature in respect to the target variable. Selected are the top sixty features according to their Chi-squared values. After that, the chosen characteristics together with their p-values and scores are assembled into a DataFrame for examination. This guarantees that, by excluding unnecessary or less significant data, only the most powerful elements are used for training the model, therefore enhancing its accuracy and efficiency. Dealing with related features is one difficulty using feature selection. Highly correlated features might cause redundancy—that is, where the chosen features offer overlapping information. This may affect model performance and lower the efficiency of the feature choosing process. Establishing the ideal feature count for this research is difficult.

5.4 Data Balancing

Data balancing is the adjusting of the class distribution in a dataset to mitigate biases caused by unequal class frequencies, enhancing model performance in handling minority and majority class predictions(Ahsan, Gomes, and Denton, 2018). Data balancing is used in this study to address class imbalances between malware and benign samples, which can bias the model towards the majority class and reduce its accuracy. To implement data balancing, the Synthetic Minority Over-sampling Technique (SMOTE) is used. SMOTE generates synthetic samples for the minority class (malware) to create a balanced dataset. This is achieved by fitting the SMOTE algorithm The Synthetic Minority Over-sampling Technique (SMOTE) is a method used to solve class imbalance in datasets. It generates synthetic samples for the minority class depending on its present cases (Elreedy, Atiya, and Kamalov, 2023). This method generates new synthetic instances resembling the current minority class samples but not exact replicas, therefore helping to equalize the class distribution. By offering more representative samples for the minority class, SMOTE helps models educated on imbalanced datasets perform betterto the selected features and the target variable, resulting in a dataset with an equal distribution of classes. The balanced dataset is then saved for further analysis, ensuring the model receives balanced input data during training, which improves its ability to accurately detect malware. Dimensionality issues occurred during development.

5.5 Training

The Four dataset is trained using the CNN-GRU model with two different methods k-cross-validation and normal performance evaluation.

- K-Fold cross-validation

K-fold Cross-validation divides the dataset into 5 parts called (“fold) for each 5 fold, the model is trained using 4 of the folds and tested on the remaining 1 fold. This process is repeated 5 times. For each fold ,the model is saved if it performs the best on validation data. After completing all the folds it reports the average accuracy and loss, This makes K-fold Cross-Validation particularly valuable in making the reported F1 score, Precision, and recall reflective of the model's true performance. But the drawback is amount of resources and training time is significant

5.6 Performance Evaluation

A total four experiments which trained with cross-k validation and train-split method will be evaluated by measuring the performance metrics accuracy, precision, recall, and f1 score. The results from each experiment will be compared to determine which one performs the best. Also, the performance of the best model is compared to the existing mode

5.7 GUI application

the GUI application for the Android malware detection system lets users to enter APK file features with unambiguous buttons for operations like loading features and executing predictions. Text boxes provide openness by displaying preprocessing actions, underlining particular traits, and showing prediction results. The graphical user interface leads consumers through the process and offers instantaneous comments on whether an APK qualifies as benign or malicious

6 Evaluation

A total of 4 experiments were conducted. The four experiments were trained using k-fold cross-validation and normal training Experiment 1 utilized pre-processed dataset 1 Experiment 2 used pre-processed dataset 2 used following Experiment 3 employed pre-processed dataset 3 , and finally experiment 4 utilized pre-processed dataset 4 used

6.1 Experiment 1: Pre-processed Dataset 1 Including Cross-validation Accuracy

The training time taken by the model was 4110.6 seconds, and the testing time was 1.38 seconds. The accuracy per fold was as follows:

Table 2: accuracy per fold

Fold	Loss	Testing Accuracy
Fold 1	0.070	98.41%
Fold 2	0.121	97.31%
Fold 3	0.115	97.51%
Fold 4	0.085	97.93%
Fold 5	0.079	98.06%

Table 3: Average scores for all Folds

Average Testing Accuracy	97.84%
Average Loss	0.094

Final Accuracy: 98.48% loss: 0.08

Table 4: Classification Report

	Precision	Recall	F1-score	Support
0	0.97	0.96	0.97	342
1	0.99	0.99	0.99	1108
Accuracy			0.98	1450
Macro avg	0.98	0.98	0.98	1450
Weighted avg	0.98	0.98	0.98	1450

6.2 Experiment 2: Pre-processed dataset (with feature selection, K-Fold Cross-Validation)

The training time taken by the model was 1252.53 seconds, and the testing time was 0.95 seconds.

Table 5: Accuracy per fold

Fold	Loss	Accuracy
Fold 1	0.12	96.82%

Fold 2	0.17	95.72%
Fold 3	0.19	95.72%
Fold 4	0.14	96.89%
Fold 5	0.15	96.68%

Table 6: Average scores for all Folds

Average Testing Accuracy	96.37%
Average Loss	0.157

Final Testing Accuracy: 96.82% loss: 0.11

Table 7: Classification Report

	Precision	Recall	F1-score	Support
0	0.94	0.93	0.93	342
1	0.98	0.98	0.98	1108
Accuracy			0.97	1450
Macro avg	0.96	0.95	0.96	1450
Weighted avg	0.97	0.97	0.97	1450

6.3 Experiment 3: Pre-processed dataset with feature selection and SMOTE applied, with K-Fold Cross-Validation

The training time taken by the model was 1483.04 seconds, and the testing time was 1.21 seconds.

Table 8: Accuracy per folds

Fold	Loss	Testing Accuracy
Fold 1	0.10	97.07%
Fold 2	0.14	96.67%
Fold 3	0.10	97.52%
Fold 4	0.10	97.12%
Fold 5	0.09	97.39%

Table 9: Average scores for all Folds

Average Testing Accuracy	97.15%
Average Loss	0.110

Final Testing accuracy: - 97.53% loss 0.09%

Table 10: Classification Report

	Precision	Recall	F1-score	Support
0	0.98	0.97	0.98	1136
1	0.97	0.98	0.97	1088
Accuracy			0.98	2224
Macro avg	0.98	0.98	0.98	2224
Weighted avg	0.98	0.98	0.98	2224

6.4 Experiment 4: Pre-processed dataset with SMOTE applied, K-Fold Cross-Validation

The training time taken by the model was 4582.42 seconds, and the testing time was 1.96 seconds.

Table 11:accuracy per fold

Fold	Loss	Testing Accuracy
Fold 1	0.06	98.33%
Fold 2	0.07	98.33%
Fold 3	0.04	98.87%
Fold 4	0.03	98.92%
Fold 5	0.07	98.56%

Table 12: Average scores for all Folds

Average Testing Accuracy	98.60%
Average Loss	0.06

Final accuracy: 98.83% loss:0.04

Table 13: Classification Report

	Precision	Recall	F1-score	Support
0	0.99	0.99	0.99	1136
1	0.99	0.99	0.99	1088
Accuracy			0.99	2224
Macro avg	0.99	0.99	0.99	2224
Weighted avg	0.99	0.99	0.99	2224

6.5 The four experiments were evaluated using both k-fold cross-validation and normal validation for comparison

Table 14: experiment comparison

	k-cross-validation Testing Accuracy	Standard Testing Accuracy	Training time (k-cross-validation accuracy)	Testing time (k-cross-validation accuracy	Testing time (standard accuracy)	Training time (standard accuracy)
`Experiment1	98.48%	98.62%	4110.63 sec	1.38sec	1.71 sec	4383.9 sec
experiment2	96.82%	96.83%	1252.53 sec	0.95 sec	1.17 sec	1283.0 sec
Experiment3	97.53%	97.57%	1483.04 sec	1.21 sec	1.10 sec	1377.1 sec
Experiment4	98.83%	98.52%	4582.4 sec	1.96 sec	1.92 sec	4509.9 sec

6.6 Comparison of Existing models with the proposed best model

Table 15: Model comparison

Study	Model	Accuracy
(Hadiprakoso, Kabetta and Buana, 2020).	GB	96%
	Naïve Bayes	80%
	Decision Tree	92%
	SVM	94%
		80%
(Brahmi Menaouer et al., 2023)	Stacked AutoEncoders (SAE), Convolutional Neural Networks	98.50%
(Dong, Shu and Nie, 2024)	CNN-DNN model	96.80%
(Vanusha D et al., 2024)	Logistic regression	97.8%
Proposed best model	CNN-GRU	98.83%

6.7 Discussion

Feature selection and impact on Accuracy

Experiment 2 and experiment 2 without k-cross validation shows a small decrease in accuracy but improved testing time. In experiment 3 and experiment 3 with normal validation the combination of feature selection and smote observed a balance in accuracy and testing time improvement

Impact of Smote

Experiments using SMOTE (experiment 3, experiment 4, experiment 3 with normal validation, and experiment 4 with normal validation)

Improved Accuracy: Experiments using SMOTE shows higher accuracy with experiment 4 achieving the highest.

Increase Training and Testing: SMOTE increases training and testing times due to the larger, more complex datasets. Experiment 4 which applied SMOTE without feature selection, had the highest training time

Impact of k-cross-validation and normal validation

Both k-fold cross-validation and normal validation results got comparable accuracy with minor differences. experiment 1's accuracy difference between k-fold and train-test is 0.14% in favour of the train-test split. In experiment 4 the k-fold cross-validation showed a slight advantage (0.3) over standard triaining

The training time is generally slightly higher compared to the normal validation, but the testing time is relatively similar.

Impact of SMOTE on Imbalanced vs Balanced Datasets.

Experiment 3 and Experiment 4 both used SMOTE to balance the datasets. This resulted in high precision, recall, and f1-scores for both classes, indicating that the model was able to correctly identify instances from both the minority and majority classes more effectively

Experiment 4 (balanced dataset with SMOTE) achieved the highest testing accuracy) with the lowest loss of 0.04 demonstrating that the model trained on the balanced dataset with SMOTE had the most reliable predictions with the least error. However, When implementing the trained model in a Python Gui, the chosen model was experiment 3 trained with the train-split method. This choice is due to its balanced accuracy and lower testing time, which is crucial for real-time detection systems.

7 Conclusion and Future Work

As the increase of Android-based smart devices and applications are mostly targetable by malware attacks. The hybrid CNN-GRU model developed has successfully outperformed numerous existing models, and also conducted detailed experiment runs, to find the about the overall optimal model exploring the different combinations of data balancing and feature selection algorithms,. The best results obtained were obtained by experiment 4 having an accuracy 99%, precision of 99%, recall of 99%, and f1 score of 99% respectively

While the Drebin dataset is well-known for its use in Android malware detection, the dataset is relatively old consisting data collected during 2010 to 2012, latest datasets like CICMalDroid 2020, CCCS-CIC-AndMal-2020 , should be considered for further research. Moreover, combining these datasets for training with the CNN-GRU model could provide valuable insights. Additionally, exploring an ensemble model approach that integrates the CNN-GRU model with other models like Random Forests and XGBoost could be a promising direction for future work.

References

Tully, S. and Mohanraj, Y., 2017. 'Mobile security,' in *Elsevier eBooks*, pp. 5–55. Available at: https://doi.org/10.1016/b978-0-12-804629-6.00002-x [Accessed 4 July 2024].

Chowdhury, M.N.-U.-R., Islam, T., Zaman, F., Hasan, M.M., Ahmed, F., & Rahman, M.A. (2023). Android malware detection using machine learning: A review. *TechRxiv*. https://doi.org/10.36227/techrxiv.22580881.v1

Jafari, S. and Byun, Y.-C., 2023. A CNN-GRU Approach to the Accurate Prediction of Batteries' Remaining Useful Life from Charging Profiles. *Computers*, 12(11), pp.219–219. doi:https://doi.org/10.3390/computers12110219.

Pardhi, P.R., Rout, J.K. and Ray, N.K., 2021. Implementation of a malware scanner using signature-based approach for Android applications. *OCIT 2021: Proceedings of the International Conference on Optical and Computational Image Processing Technologies*. DOI: https://doi.org/10.1109/ocit53463.2021.00015.

Alam, S., Qu, Z., Riley, R., Chen, Y. and Rastogi, V. (2017). DroidNative: Automating and optimizing detection of Android native code malware variants. *Computers & Security*, [online] 65, pp.230–246. doi:https://doi.org/10.1016/j.cose.2016.11.011.

Jafari, S. and Byun, Y.-C. (2023). A CNN-GRU Approach to the Accurate Prediction of Batteries' Remaining Useful Life from Charging Profiles. *Computers*, 12(11), pp.219–219. doi:https://doi.org/10.3390/computers12110219.

Apvrille, L. and Apvrille, A. (2015). *Identifying Unknown Android Malware with Feature Extractions and Classification Techniques*. [online] IEEE Xplore. doi:https://doi.org/10.1109/Trustcom.2015.373.

Liang, D., Shen, L., Chen, Z., Ma, C. and Feng, J. (2022). A Formal Method for Description and Decision of Android Apps Behavior Based on Process Algebra. *IEEE Access*, 10, pp.108668–108683. doi:https://doi.org/10.1109/access.2022.3210386.

Kouliaridis, V., & Kambourakis, G. (2021). A comprehensive survey on machine learning techniques for Android malware detection. *Information, 12*(5), 185. https://doi.org/10.3390/info12050185

Ganesh, M., Pednekar, P., Prabhuswamy, P., Nair, D.S., Park, Y., & Jeon, H. (2017). CNN-based Android malware detection. In *2017 International Conference on Software Security and Assurance (ICSSA)* (pp. 58-63). IEEE. https://doi.org/10.1109/icssa.2017.18

Gupta, A., Maurya, S., Kapil, D., Mehra, N., & Negi, H. (2020). Android malware detection using machine learning. *International Journal of Recent Technology and Engineering, 8*(2S12), 65-70. https://doi.org/10.35940/ijrte.b1011.0982s1219

Brahami, M., Menaouer, I., & Nada, M. (2023). Android malware detection approach using stacked autoencoder and convolutional neural networks. *International Journal of Intelligent Information Technologies, 19*(1), 1-22. https://doi.org/10.4018/ijiit.329956

Lu, T., Du, Y., Ouyang, L., Chen, Q., & Wang, X. (2020). Android malware detection based on a hybrid deep learning model. *Security and Communication Networks, 2020*, 1-11. https://doi.org/10.1155/2020/8863617

Muhammad, A., Iqbal, M.W., Nosheen, M., Ashraf, M.U., Shaf, A., Almarhabi, K.A., Alghamdi, A.M. and Bahaddad, A.A., 2024. AMDDLmodel: Android smartphones malware detection using deep learning model. *PLOS ONE*, 19(1), pp.e0296722. doi: https://doi.org/10.1371/journal.pone.0296722.

Almarshad, F.A., Zakariah, M., Ghada Abdalaziz Gashgari, Eman Abdullah Aldakheel and Abdullah (2023). Detection of Android Malware Using Machine Learning and Siamese Shot Learning Technique for Security. *IEEE Access*, 11, pp.127697–127714. doi:https://doi.org/10.1109/access.2023.3331739.

Feng, R., Chen, S., Xie, X., Meng, G., Lin, S.-W. and Liu, Y. (2021). A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices. IEEE Transactions on Information Forensics and Security, 16, pp.1563–1578. doi:https://doi.org/10.1109/tifs.2020.3025436.

Alkahtani, H. and Aldhyani, T.H.H. (2022). Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices. *Sensors*, 22(6), p.2268. doi:https://doi.org/10.3390/s22062268.

Hadiprakoso, R.B., Kabetta, H. and Buana, I.K.S. (2020). *Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICIMCIS51567.2020.9354315.

Sara, J.J. and Hossain, S. (2023). *Static Analysis Based Malware Detection for Zero-Day Attacks in Android Applications*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICICT4SD59951.2023.10303336.

Dong, S., Shu, L. and Nie, S. (2024). Android Malware Detection Method Based on CNN and DNN Bybrid Mechanism. *IEEE transactions on industrial informatics*, pp.1–10. doi:https://doi.org/10.1109/tii.2024.3363016.

Vanusha D, Singh, S., Abhijeet Kumar Jha and Delsi Robinsha S (2024). SecuDroid : Android Malware Detection using ML classifier on Static Features. doi:https://doi.org/10.1109/icnwc60771.2024.10537417.

Jundi, Z.Z. and Hasanen Alyasiri (2023). Android Malware Detection Based on Grammatical Evaluation Algorithm and XGBoost. doi:https://doi.org/10.1109/aiccit57614.2023.10217965.

Tian, Y., Dai, X., Li, Z., Guo, H., Mao, X. and Li, Y. (2024). Research on Personal Privacy Security Detection Techniques for Android Applications. doi:https://doi.org/10.1109/icetis61828.2024.10593754.

Kirubavathi G and Nithish S (2024). Dynamic Ensemble Learning Framework Enhanced with XAI To Detect Android Malware. doi:https://doi.org/10.1109/iscs61804.2024.10581314.

Ullah, F., mostarda, L., Diletta Cacciagrano, Chen, C.-M. and Kumari, S. (2024). Semantic-based Federated Defense for Distributed Malicious Attacks. *IEEE Consumer Electronics Magazine*, pp.1–9. doi:https://doi.org/10.1109/mce.2024.3431792.

Habib, M. and Hafsa Binte Kibria (2024). Feature Selection-Based Machine Learning Approaches for Detecting Android Malware with Explainable AI. doi:https://doi.org/10.1109/icaeee62219.2024.10561751.
Eom, T., Kim, H., An, S., Jong Sou Park and Dong Seong Kim (2018). Android Malware Detection Using Feature Selections and Random Forest. doi:https://doi.org/10.1109/icssa45270.2018.00023.
K., S.J., Chakravarty, S. and Varma P., R.K. (2020). *Feature Selection and Evaluation of Permission-based Android Malware Detection*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICOEI48184.2020.9142929.
Ahsan, M., Gomes, R. and Denton, A. (2018). *SMOTE Implementation on Phishing Data to Enhance Cybersecurity*. [online] IEEE Xplore. doi:https://doi.org/10.1109/EIT.2018.8500086.
Aslam, W., Muhammad Moazam Fraz, Rizvi, S.K. and Saleem, S. (2020). Cross-validation of machine learning algorithms for malware detection using static features of Windows portable executables: A Comparative Study. *2020 IEEE 17th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*. doi:https://doi.org/10.1109/honet50430.2020.9322809.
Khoda, M.E., Joarder Kamruzzaman, Gondal, I., Imam, T. and Rahman, A. (2020). Mobile Malware Detection with Imbalanced Data using a Novel Synthetic Oversampling Strategy and Deep Learning. doi:https://doi.org/10.1109/wimob50308.2020.9253433.
Dhalaria, M. and Gandotra, E. (2020). *Android Malware Detection using Chi-Square Feature Selection and Ensemble Learning Method*. [online] IEEE Xplore. doi:https://doi.org/10.1109/PDGC50313.2020.9315818.
Areeg Fahad Rasheed, M. Zarkoosh and Sana Sabah Al-Azzawi (2023). The Impact of Feature Selection on Malware Classification Using Chi-Square and Machine Learning. doi:https://doi.org/10.1109/iccce58854.2023.10246084.
Liu, T., Zhang, H. and Long, H. (2022). Malicious Software Detection Based on Improved Convolution Neural Network. *2022 2nd International Conference on Frontiers of Electronics, Information and Computation Technologies (ICFEICT)*. doi:https://doi.org/10.1109/icfeict57213.2022.00065.
Bahassine, S., Madani, A., Al-Sarem, M. and Kissi, M. (2020). Feature selection using an improved Chi-square for Arabic text classification. *Journal of King Saud University - Computer and Information Sciences*, 32(2), pp.225–231. doi:https://doi.org/10.1016/j.jksuci.2018.05.010.
Lachtar, N., Ibdah, D. and Bacha, A. (2020). Towards Mobile Malware Detection Through Convolutional Neural Networks. *IEEE Embedded Systems Letters*, pp.1–1. doi:https://doi.org/10.1109/les.2020.3035875.Jaiswal, R. and Singh, B. (2022). *A Hybrid Convolutional Recurrent (CNN-GRU) Model for Stock Price Prediction*. [online] IEEE Xplore. doi:https://doi.org/10.1109/CSNT54456.2022.9787651.
Bhargavi Suvarnam and Viswanadha Sarma Ch (2019). Combination of CNN-GRU Model to Recognize Characters of a License Plate number without Segmentation. *International Conference on Advanced Computing*. doi:https://doi.org/10.1109/icaccs.2019.8728509.
Mehedi Hasan Shakil and Md. Mynul Hasan (2023). Empowering Android Malware Detection: A Deep Learning Ensemble with Optimal Features. doi:https://doi.org/10.1109/iccit60459.2023.10441465.
Xu, A., Chen, L., Kuang, X., Huahui Lv, Yang, H., Jiang, Y. and Li, B. (2020). A Hybrid Deep Learning Model for Malicious Behavior Detection. doi:https://doi.org/10.1109/bigdatasecurity-hpsc-ids49724.2020.00021.

Lachtar, N., Ibdah, D. and Bacha, A. (2020). Towards Mobile Malware Detection Through Convolutional Neural Networks. *IEEE Embedded Systems Letters*, pp.1–1. doi:https://doi.org/10.1109/les.2020.3035875.
Marwa Ben Jabra, Cheikhrouhou, O., Nesrine Atitallah, Anouar Ben Amor and Habib Hamam (2023). Malware Detection Using Deep Learning and CNN Models. doi:https://doi.org/10.1109/cw58918.2023.00073.
Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H. and Rieck, K. (2014). Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. Proceedings 2014 Network and Distributed System Security Symposium. [online] doi:https://doi.org/10.14722/ndss.2014.23247.
Dey, S.K., Uddin, K.M.M., Babu, H.Md.H., Rahman, Md.M., Howlader, A. and Uddin, K.M.A. (2022). Chi2-MI: A hybrid feature selection based machine learning approach in diagnosis of chronic kidney disease. *Intelligent Systems with Applications*, 16, p.200144. doi:https://doi.org/10.1016/j.iswa.2022.200144.
Elreedy, D., Atiya, A.F. and Kamalov, F. (2023). A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning. *Machine Learning*. doi:https://doi.org/10.1007/s10994-022-06296-4.