# Harnessing Deep Learning for Proactive Detection of Security Threats in Android OS

MSc Research Project
Data Analytics

Rajat Nagaraj Murdeshwar
Student ID: x22150927

School of Computing
National College of Ireland

Supervisor: Prof. Vladimir Milosavljevic

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Rajat Nagaraj Murdeshwar |
| **Student ID:** | x22150927 |
| **Programme:** | Data Analytics |
| **Year:** | 2023 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Prof. Vladimir Milosavljevic |
| **Submission Due Date:** | 14/12/2023 |
| **Project Title:** | Harnessing Deep Learning for Proactive Detection of Security Threats in Android OS |
| **Word Count:** | 4978 |
| **Page Count:** | 19 |

| | |
|---|---|
| **Signature:** | |
| **Date:** | 14th December 2023 |

# Harnessing Deep Learning for Proactive Detection of Security Threats in Android OS

Rajat Nagaraj Murdeshwar
x22150927

**Abstract**

In this rapidly evolving domain of mobile technologies, ensuring the security of Android systems is a critical challenge. This research project addresses the urgent need for proactive vulnerability detection in Android OS by developing an automated system that integrates with the National Vulnerability Database (NVD) and Common Weakness Enumeration (CWE). I have designed functions to update its dataset regularly, enabling the immediate detection of newly recorded vulnerabilities. The core of this research project involved the applications of machine learning models and deep learning models: Graph Neural Networks (GNN) using Graph Convolutional Network (GCN), Support Vector Machines (SVM), and Random Forests. These models analyse CVE descriptions, processed through TF-IDF and Word2Vec, to predict vulnerabilities with high accuracy and precision-recall values. The effectiveness of these models demonstrates the project's contribution to enhancing Android security. Random Forests model with Word2Vec performed well in precision recall with a high accuracy of 98%. A key limitation was the lack of demonstrative code or bad code for all CWE vulnerabilities, restricting the training data's comprehensiveness.

## 1 Introduction

In the third quarter of 2023, Android continued to dominate the global mobile operating system market, maintaining a 70.5 percent share[1]. As an open-source platform, Android is widely used across various mobile ecosystems, including personal, corporate, and governmental spheres.

The security of these systems is not just a technical challenge, but also a critical aspect of global infrastructure. With the number of mobile phone users projected to double over the next five years, enhancing vulnerability detection systems is imperative to prepare for potential cyber attacks. Each year, numerous cyber attack incidents are reported, significantly impacting many individuals and resulting in substantial financial losses. Although the Android Play Store and Samsung Galaxy Store have built-in security checks, these measures may not be entirely foolproof against malware attacks. A recent incident, where 19 malware infected apps were downloadable from the Play Store[2], potentially affected millions of users. This incident highlights the fact that the Google Play Store is not

[1] 1:`https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/`
[2] 2:`https://www.gizchina.com/2023/04/25/delete-these-19-malicious-android-apps-now/`

completely secure. There are two types of vulnerability detection: static and dynamic approaches. This research presents a comprehensive study on the effectiveness of advanced machine learning techniques, specifically Graph Neural Networks (GNN), Random Forest, and Support Vector Machines (SVM), in detecting vulnerabilities within Android systems. My analysis employed a static approach, further enriched by continuously monitoring and updating the risk through the Common Weakness Enumeration (CWE) and Common Vulnerabilities and Exposures (CVE) standards. This ensures that the research is grounded in industry recognized vulnerability database. While previous research has utilis CVE data to detect vulnerabilities or focused solely on CWE data, it has not fully integrated CVE with CWE. By doing so, the detection of potential vulnerabilities can be more accurate, providing in-depth knowledge of the vulnerability and its associated risks. In my approach, I have proactively detected vulnerabilities in the Android System. This study was worked on previous efforts and introduces a unique approach by employing neural network models to identify vulnerabilities, focusing on the descriptions provided in CVE datasets and mapping them to their related CWE entries for enhanced accuracy and comprehensiveness. This research not only addresses the immediate need for robust security in the ever expanding Android ecosystem but also contributes to the broader field of cybersecurity. By utilising cutting-edge machine learning techniques, we aim to set a new benchmark in vulnerability detection, one that is adaptable, efficient, and more importantly, ahead of the evolving threats. Furthermore, the integration of CVE and CWE data in this study is not just a technical achievement, but also a strategic move to create a more holistic and informed approach to cybersecurity. This methodology could serve as a model for future research and development in the field. The insights gained from this study could inform the development of more secure software, contribute to the enhancement of existing security protocols, and inspire innovative approaches to tackling cybersecurity challenges. Lastly, the research underscores the importance of continuous learning and adaptation in the field of cybersecurity. As new vulnerabilities emerge and threat actors evolve their tactics, our methods and strategies must also advance. This study represents a step forward in that ongoing journey, showcasing the potential of machine learning in staying one step ahead in the ever-changing landscape of cyber threats. The structure of the report is described in the following paragraph.

Section 2 discusses the previous studies in the field by credible researchers. Section 3 provides a detailed synopsis of the research carried out using the KDD methodology. Section 4 discusses the fundamental technological design and the innovative solutions proposed in this project. The implementation of the proposed solution is thoroughly explained in Section 5. Section 6 evaluates all the experiments conducted in this project and compares their findings.

## 2 Related Work

In this section on related work, I have included two distinct types of research that are relevant to my study. The first type focuses solely on using Machine Learning (ML) and Deep Learning (DL) models, while the second type involves a combination of Deep Learning models with Natural Language Processing (NLP) techniques. These are detailed in subsections 2.1 and 2.2, respectively.

## 2.1 Machine Learning (ML) and Neural Network

Gencer and Başçiftçi (2021) conducted a study that employed time series modelling techniques to forecast security vulnerabilities. They utilised data from the National Vulnerability Database (NVD) and Common Vulnerabilities and Exposures (CVE) and experimented with various models, including time series multilayer perceptron (MLP), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), ConvLSTM, and CNN-LSTM-based models. Their primary objective was to identify the most accurate forecasting model based on error rates. The results showed that the LSTM model had an error rate of 26.830, while the Autoregressive Integrated Moving Average (ARIMA) model outperformed with an error rate of 18.449. However, it is essential to note that Gencer and Başçiftçi's study solely focused on forecasting vulnerabilities and did not extensively leverage the available data for other purposes.

In conclusion, research in the field of forecasting security vulnerabilities has made substantial strides, with studies employing a variety of techniques, including time series modelling and machine learning. However, there was a need for more comprehensive approaches that harness the full potential of available data sources and address the limitations observed in the existing models. This review sets the stage for the present study, which seeks to further advance our understanding of security vulnerability forecasting.

In this research (Senanayake et al.; 2023) addresses the critical issue of vulnerability in source code of C/C++ applications, which poses significant reliability and security challenges. The study emphasises the importance of incorporating software security principles early in the development lifecycle. To achieve this, the research proposes a Machine Learning (ML)-based method to detect vulnerabilities in source code. The unique aspect of this study was the application of Explainable Artificial Intelligence (XAI). XAI is crucial because it aids developers in identifying which parts of the source code are vulnerable and understanding the reasons behind these vulnerabilities. This approach is not only about detecting issues but also providing insights that can guide developers in making informed decisions and corrections. The research tested several ML classifiers to achieve these results. Notably, the Random Forest (RF) algorithm performed well in binary classification, while Extreme Gradient Boosting (XGB) excelled in the multi-class approach. They didn't really make use of deep learning models.

In this research paper (Liu et al.; 2020) they are trying to do a review on current approaches in malware detection in android. This has a lot of information related to ML models that are used and the steps that can be followed and building a good android malware detection. This paper also gives a lot of information on the different ML models and their advantages and disadvantages for detecting the vulnerabilities. Over the years how ML models have performed and which different approaches like static analysis, dynamic and hybrid analysis have been done. I have taken some insights from this paper.This paper presents a thorough survey of machine learning-based approaches to Android malware detection, highlighting its importance given the rapid evolution of both Android applications and associated malware threats. It aims to fill gaps in existing literature and guide future research in this critical area of cybersecurity

In this paper (Malik et al.; 2019) introduces an innovative method focusing on anomaly detection in system calls of both benign and malicious Android applications. The study hypothesised that the type, frequency, and sequence of system calls can reliably indicate the presence of vulnerabilities. By monitoring this aspects, the research aims to differentiate between normal and malicious process behaviours. Utilises ML techniques like kNN, LSTM, and GA-LSTM to detect vulnerabilities based on patterns of system calls, with a primary focus on ML. The research opens avenues for further exploration, particularly in refining machine learning models for even more accurate detection, exploring the impact of different types of system calls, and potentially extending the methodology to other operating systems or application environments.This maybe not a viable option while gathering the system calls and create a dataset according to it.

In this paper (Arslan; 2021) Android Malicious Software Detection Based on Deep Learning This paper is similar to other paper which uses static analysis and the network calls and using the APK and extracting the code to perform the malware detection. This whole Androianalyzer is using the DNN model to detect the malware. It is not that effective in finding the latest vulnerabilities and malware

In this paper Amin et al. (2019) presents a significant advancement in the field of mobile application security. Its hybrid approach for automated vulnerability detection, particularly for Android applications, addresses a critical need in the rapidly evolving landscape of mobile app development. By combining static and dynamic analyses, the model provides a comprehensive solution for identifying a wide range of security vulnerabilities. The development of a user-friendly platform and the decision to make the code publicly available further enhance the impact and relevance of this research in both academic and practical realms.The model's effectiveness is demonstrated through evaluations against various applications with different security vulnerabilitie. A notable aspect of this model is its ability to detect a wide range of flaws, including information leaks and insecure network requests, which are often overlooked by other detection platforms. This capability enhances the model's utility in safeguarding user privacy

## 2.2 Neural Network with Natural Language Processing (NLP)

In this research Wartschinski et al. (2022) Introduces VUDENC, a tool that uses a word2vec model (an NLP technique) along with LSTM cells (an ML technique) for detecting vulnerabilities in software code. This VUDENC is mostly implemented for python code base By utilising LSTM models, VUDENC suggests several avenues for future research and improvements. Enhancing the understandability and actionability of Vudenc's results is critical for its practical application in bug or vulnerability prediction tools. Other propsed enhancements include refining data labelling, integrating Vudenc with other methods for better results, and using commit context for actionable fix recommendations. VUDENC represents a significant advancement in the field of automated vulnerability detection. Its use of deep learning on natural source code, combined with its promising results and potential for future enhancements, positions it as a valuable tool for developers and researchers in cybersecurity. The study sets a precedent for the innovative use of machine learning in software security and opens new pathways for research and development in this domain.

In this paper Renjith and Aji (2022) Discusses vulnerability detection in Android OS using graph neural networks and Graph Embedding Mechanisms. While primarily ML-focused, the Graph2Vec algorithm can be seen as bridging ML and NLP, as it deals with embedding and can be related to techniques used in NLP. The study identifies a gap in the existing methods for vulnerability detection in the Android operating system. Most current approaches do not adequately address the complexities and scale of the Android OS, focusing instead on more constrained environments like individual apps or framework changes. A key contribution of this research is the design and implementation of a vulnerability detection mechanism based on Graph Neural Networks (GNNs). GNNs are particularly suitable for this task as they can effectively process the graph-structured data, which in this case represents the source code's structure and relationships. In conclusion, this research represents a pivtal step forward in the field of cybersecurity, particularly in the context of the Android operating system. I have worked on this and improved the gap that I found in the vulnerability detection steps.

In this research Garg and Baliyan (2020) introduces M2VMapper, a novel deep learning (DL) framework designed to address the critical challenge of mapping malware to potential vulnerabilities in the Android mobile platform. Recognizing that over 90% of mobile malware targets Android, the research highlights the importance of understanding the complex many-to-many relationships between malware and vulnerabilities. While primarily ML-focused, the context of their work may imply the use of NLP techniques, especially if the analysis involves textual data like code or system logs.The study's robustness is evident in its extensive data collection, encompassing 150 malware families from diverse datasets (AMD, CICInvesAndMal2019, Androzoo) with a total of 48,907 malware samples and 9 types of vulnerabilities affecting Android. This comprehensive dataset enables a thorough analysis and enhances the reliability of the results.M2VMapper has achieved remarkable results, with an accuracy of 99.81% when combining XLNET with TextCNN, and precision and F1-scores above 95% using other DL models.The study not only fills a crucial gap in the current literature but also sets a new standard for the application of deep learning and text processing in malware analysis. The results of this research offer significant insights and tools for enhancing the security of Android applications, marking a notable advancement in the field of cybersecurity.

In this paper Yuan et al. (2020) addresses the critical issue of malware detection in Android applications, a problem exacerbated by the proliferation of third-party Android app markets. The lack of regulation in these markets has led to an increase in malicious apps, posing a significant security threat. The dynamic nature of malware evolution and improvements in the Android system make it challenging to design an effective and efficient long-term detection method. Additionally, incorporating more features into detection models increases their complexity and computational costs.In conclusion, this paper presents a novel and effective approach to detecting malware in Android apps using a combination of TF-IDF and machine learning. By focusing on app permissions and applying static analysis, the method addresses the challenges of dynamic malware evolution and system complexity. The high accuracy and quick processing time of the proposed method signify a substantial advancement in the field of Android security, providing a promising direction for future research and application development in malware detection.

In this paper Raghav et al. (2021) represents a significant advancement in the field of Android malware detection. By integrating static analysis with sophisticated NLP techniques, it addresses a critical limitation of existing methods. The use of document embeddings to capture semantic information opens new avenues for more accurate and effective malware detection. This research not only contributes to the technical domain of cybersecurity but also has practical implications for safeguarding the increasing number of Android users against evolving malware threats.The research identifies that existing machine learning and deep learning approaches for Android malware detection primarily rely on frequency-based vectors derived from various files in the Android application package (APK). However, a significant limitation of these methods is their failure to capture the semantic information inherent in these files, which is crucial for a more nuanced and effective detection of malware. By leveraging document embeddings, the proposed method captures a richer, more contextually informed representation of the data contained in APK files.The use of binary classifiers in this context is particularly relevant for practical applications, where quick and clear decisions are crucial.

## 2.3 Discussion

The field of Android vulnerability detection is marked by rapid advancements and continual adaptation to emerging threats. The advancement of increasingly powerful and efficient detection strategies is largely dependent on the combination of ML, DL, and NLP techniques. As the subject develops further, the emphasis will probably move to producing more precise, scalable, and adaptable solutions to protect against the always shifting cyberthreat environment. The comprehensive understanding gained from previous investigations serves as a guide for my investigation, prioritizing creativity, flexibility, and a multifaceted strategy for identifying vulnerabilities.

# 3 Methodology

In this section, I describe the step-by-step process employed in detecting vulnerabilities, Included a comprehensive flowchart in Figure 1. Each step of the workflow, along with its specific functions and methodologies, is detailed in subsections 3.1 and 3.2 below.

## 3.1 Dataset

The datasets was divided into two sections. The first part is accessible in JSON format from the National Vulnerability Database (NVD) *National Vulnerability Database* (n.d.), encompassing data from 2013 to 2023, which includes various Common Vulnerabilities and Exposures (CVEs). The second part consists of Common Weakness Enumeration (CWE) data from *Common Vulnerabilities and Exposures* (n.d.). The dataset includes approximately 8300 records detailing vulnerabilities specific to Android, while the CWE section contains around 600 entries, each describing different weaknesses. This dual dataset consist of more information about the vulnerabilities and it will really helped in accurately detecting the vulnerabilities.

## 3.2 KDD Steps

The research made use of the Knowledge Discovery in Databases (KDD) techniques, which is described in more detail in the subsections that follow and is shown in Figure 1.

### 3.2.1 Data Collection

In this step, The data collected from multiple source first is NVD website APIs and this step was crucial for accessing a broad range of data points, specifically focusing on extracting columns relevant to our analysis and second was CWE data csv file.

### 3.2.2 Data Pre-processing

In this step, data collected from the both the sources were mapped. I mapped both the CWE and CVE datasets, The 'CWE-ID' serves as a common link, allowing me to correlate specific vulnerabilities listed in the CVE dataset with their corresponding weakness types in the CWE dataset. By mapping the datasets in this manner, it became possible to create a comprehensive view where each CVE entry is associated with its respective weakness type as classified in the CWE framework.

### 3.2.3 Data Transformation

In this step, the merged dataset underwent several transformation processes. Initially, column names were modified for clarity and consistency, ensuring they accurately represented the data they contained and finding the data type and changing the data type as required per model. The next phase involved a thorough examination for missing values, identified as 'NA' or null entries. This step was critical to ensure data integrity and reliability. Columns that contained a significant number of missing values or those that were deemed irrelevant to the objectives of the analysis were dropped from the dataset.Consequently, this rigorous transformation process streamlined the dataset and Tranformed from 24 columns to just 8 columns datasets.

### 3.2.4 Data Mining

In this process, I tried to collect the demonstrative code for each of the CWE, so it could help in building the model and detecting the bad code. I tried to find like 4000 demonstrative example which was not sufficient for the ML models. I used the word embedding for extracting the data from description of the vulnerability

### 3.2.5 Evaluation

In this evaluation phase, the cleaned and transformed data was utilized to train the proposed models. Three distinct models were employed: Graph Neural Networks (GNN), Support Vector Machines (SVM), and Random Forest. Each model incorporated word embedding techniques using Word2Vec and the TF-IDF (Term Frequency-Inverse Document Frequency) process. For each model, line graphs were generated to visually represent their performance. Key metrics such as Accuracy, F1 Score, Precision, Recall, and Confusion Matrix were evaluated. Additionally, various plots were created to facilitate a comprehensive comparison of the models' performances.

### 3.2.6 Interpretation

In this interpretation phase, we analyze the efficacy of three models: Graph Neural Networks (GNN), Support Vector Machines (SVM), and Random Forest, each enhanced with Word2Vec and TF-IDF word embedding. The evaluation through Accuracy, F1 Score, Precision, Recall, and Confusion Matrix reveals distinct strengths and weaknesses of each model. GNN, for instance, may excel in complex pattern recognition, crucial in cybersecurity data analysis. The visual plots, including line graphs, facilitate a comparative understanding of the models. This analysis not only highlights each model's technical effectiveness but also provides insights into cybersecurity vulnerabilities, guiding future research and methodology improvements.

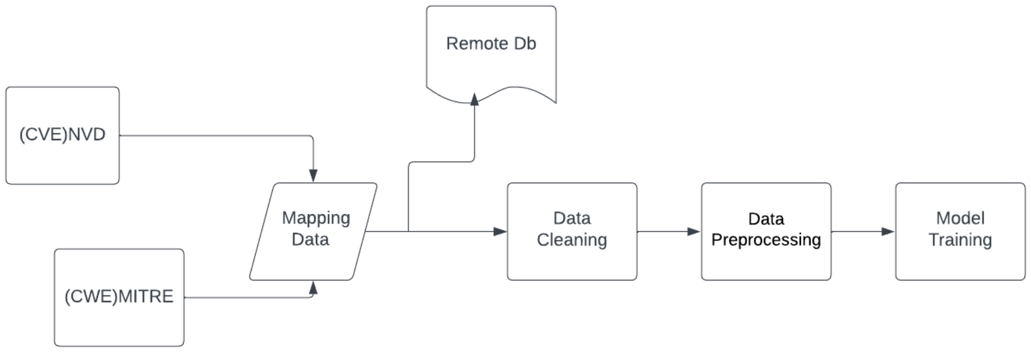

Figure 1: Flow Chart

## 4 Design Specification

This research project mainly focused on two techniques TF-IDF and Word2Vec to develop a machine learning system where the efficacy primarily hinges on two key feature extraction techniques. The approach I used is static analysis utilizing CVE descriptions and severity levels.The Severity_HIGH column serves as a binary target variable for classification.

To develop a system for detecting vulnerabilities in Android using three models: GNN, SVM, and Random Forest, with a focus on analyzing CVE descriptions and severity levels. CVE descriptions are preprocessed using TF-IDF and Word2Vec to create meaningful numerical features. The Severity_HIGH column serves as a binary target variable for classification. GNN model leverages a graph structure with CVE IDs and CWE relationships, enhanced with text features from CVE descriptions. SVM and Random Forest models use processed CVE description features to predict the Severity_HIGH label.

### 4.1 TF-IDF

This method will be used to convert the text of CVE descriptions into a numerical format, highlighting the importance of certain words based on their frequency in a specific document and their inverse frequency across all documents. It helps in identifying the most relevant words in each CVE description. I have added the idf equation [3]

$$idf\,(t, D) = log\,\left( \frac{N}{count\,(d \in D{:}t \in d)} \right)$$

$$tf\,idf\,(t, d, D) = tf\,(t, d)\;.\;idf\,(t, D)$$

[3]:https://monkeylearn.com/blog/what-is-tf-idf/

## 4.2 Word2Vec

This approach involves creating word embeddings for the CVE descriptions. Word2Vec maps words into a high-dimensional vector space where the position of each word is determined based on the context in which it appears in the CVE descriptions. This is useful for capturing the semantic meaning of words as shown in Figure 2 [4] it's working flow.

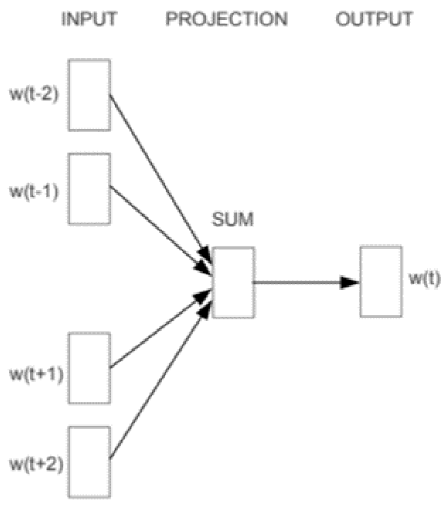

Figure 2: Word2Vec

## 4.3 Model Implementation

The implementation of the three models — Graph Neural Network (GNN) with Graph Convolutional Network , Support Vector Machine (SVM), and Random Forest — was carried out with a focus on processing CVE descriptions for Android vulnerability detection. Each model was uniquely tailored to leverage the strengths of the respective algorithms. Hyper-parameter tuning was an essential part of the implementation process, aimed at optimizing the performance of each model.Throughout the tuning process for GNN, RNN and SVM, a balance was sought between model complexity and performance, ensuring robust and reliable predictions while avoiding over fitting. The hyper-parameter tuning not only enhanced the accuracy but also improved the models' efficiency and interpretability.

# 5 Implementation

The implementation of this project involved the development of three distinct machine learning models: Graph Neural Network (GNN) (see Figure 3[5]), Support Vector Machine

[4] 2:`https://arxiv.org/pdf/1301.3781.pdf`

[5] `https://vitalflux.com/graph-neural-networks-explained-with-examples/`

(SVM) (see Figure 5),[6] and Random Forest (see Figure 4).[7] These models were designed to analyze and predict Android vulnerabilities based on CVE descriptions, processed using TF-IDF and Word2Vec, with the Severity_HIGH column serving as a binary target.

## 5.1 Common Vulnerability Enumeration

In my project the Common Vulnerabilities and Exposures (CVE) database provided a extensive details about various vulnerabilities, including their severity levels. Each columns includes a 'last modified date', indicating the most recent update, and the date when the vulnerability was first reported. The database encompasses the current status of each vulnerability along with a comprehensive descriptions content. Key elements of the CVE entries include Base Score, Base Severity, Exploitability Score, Impact Score, and the associated Common Weakness Enumeration (CWE) Name. Severity levels are categorized as Low, Medium, High, and Critical. Additionally, the database contains fields for the vulnerability status and encoded values for CVE Base Severity.

## 5.2 Development Environment

The instruments and software platforms listed below were used to conduct the research's experiments:

- **OS**: macOS Ventura 13.5
- **Language**: Python
- **Development Tool**: Jupyter Notebook
- **Package Manager**: Conda Environment
- **Cloud Resources**: Google Collab & Google Drive

## 5.3 Important Libraries used in the Project

- **Gensim**: Natural Language Processing Library
- **sklearn**: For Random Forests and SVM machine learning models
- **PyTorch**: For Graph neural network model

## 5.4 Data Pre-processing

This part, I mainly focused on the initial stage of preparing data for subsequent analysis and modeling. In the context of Android vulnerability detection, data pre-processing involves several key steps:

[6] Reproduced from *Enhancing Surface Fault Detection Using Machine Learning for 3D Printed Products*`https://www.researchgate.net/figure/Architecture-of-SVM-algorithm_fig3_351590398`

[7] Reproduced from *Intrusion Detection Systems Based on Machine Learning Algorithms* `https://www.researchgate.net/figure/Random-forest-Architecture_fig3_353906529`

### 5.4.1 Data Cleaning and Transformation

I have identified and rectifed any inconsistencies, missing values, or errors in my data. This might include normalizing text data, handling missing values in the dataset, and filtering out irrelevant information.

### 5.4.2 Feature Extraction

I applyed TF-IDF and Word2Vec to transform the content CVE descriptions into numerical features. TF-IDF highlights the importance of words in relation to their frequency across documents, while Word2Vec captures the contextual relationships between words.

### 5.4.3 Data Normalization

The scaling of data to a standard range or format, ensuring that different data types are compatible for modeling.

## 5.5 NLP based Strategy

In this part, I delved into the application of Natural Language Processing (NLP) techniques for analyzing and interpreting the textual data related to vulnerabilities:

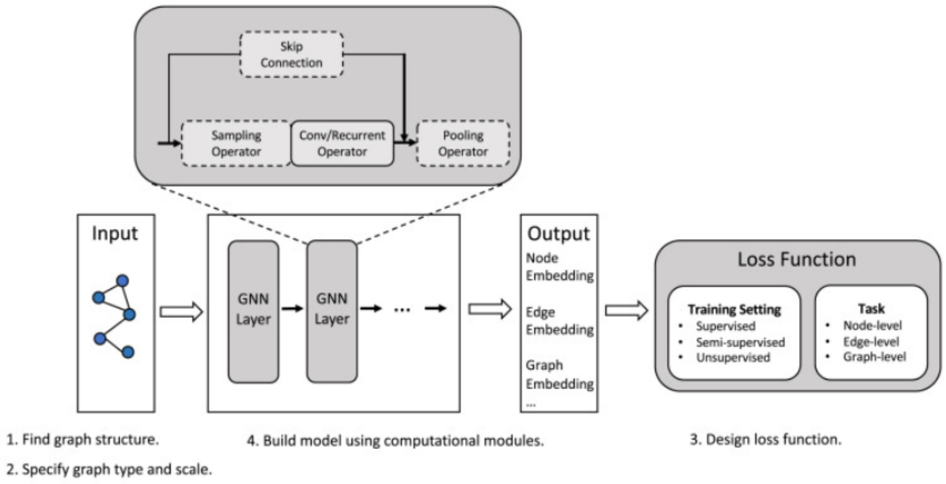

Figure 3: GNN Architecture

### 5.5.1 Textual Data Analysis

Utilizing NLP to process and analyze textual data from CVE descriptions. This includes parsing the text, extracting relevant information, and understanding the contextual meaning of words and phrases.

### 5.5.2 Pattern Recognition

Employing algorithms to identify common patterns or indicators of vulnerabilities within the textual data.

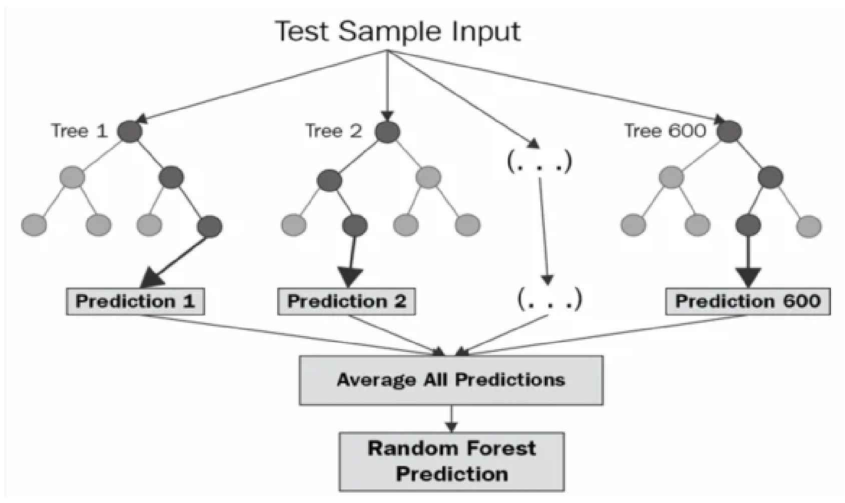

Figure 4: Random Forest Architecture

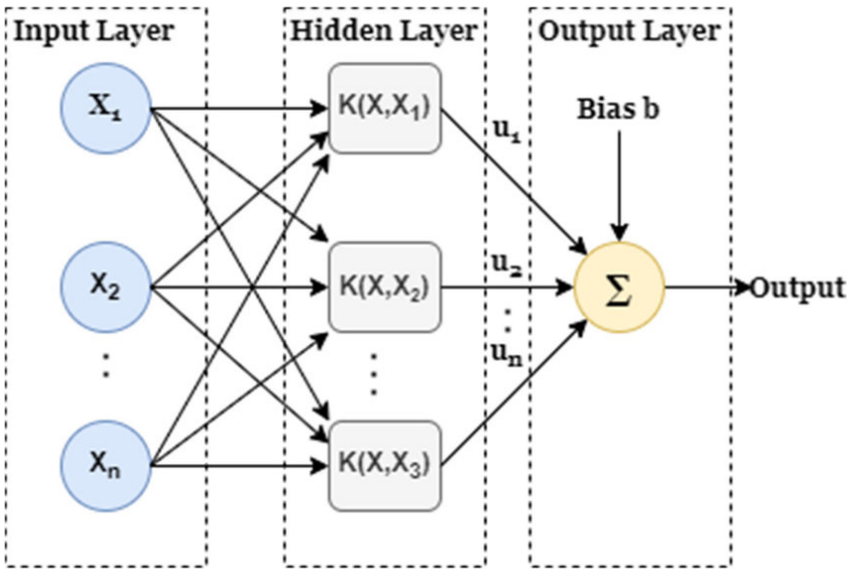

Figure 5: SVM Architecture

### 5.5.3 Semantic Analysis

Applying advanced NLP techniques like sentiment analysis or named entity recognition to glean deeper insights into the nature of the vulnerabilities.

### 5.5.4 Integration with Machine Learning

Combining NLP-processed data with machine learning models (GNN, SVM, Random Forest) to enhance the detection and prediction of vulnerabilities. For GNN, Each node represents a CVE and carries attributes like CVE score, severity, and associated CWE ID. The CWE ID is converted to an integer if it's numeric building a graph that maps the intricate relationships between different CVEs and CWEs as shown in Figure 6.I used **Graph Convolutional Network** (GCN) model type and is configured structured with a 16-dimensional hidden space and **Adam optimizer** is used with a learning rate of 0.01. The loss function is binary cross-entropy with logits (BCEWithLogitsLoss), which combines a sigmoid layer and the BCE loss in one single class.

In Random Forest model, it is set with **100 trees** balancing computational efficiency with the ability to capture diverse patterns in the data. In SVM kernel is set to **Linear** because it aligns with nature of our data, ensuring optimal separation and accuracy.

## 6 Evaluation

In this evaluation part, I have compared the machine learning and deep learning models using NLP techniques on various metrics. Before testing the model, I trained it by splitting the dataset into train-test splits with 80% and 20%. The different evaluation metrics used are accuracy, F1, preccison recall and ROC/AUC curve. The following subsection 6.1 and 6.2

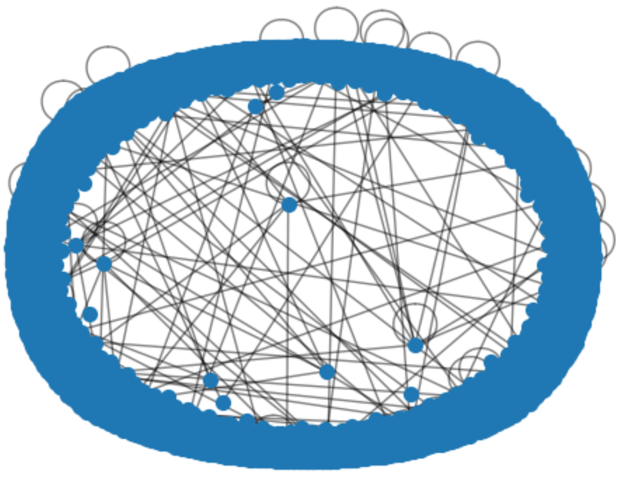

Figure 6: GNN Graph

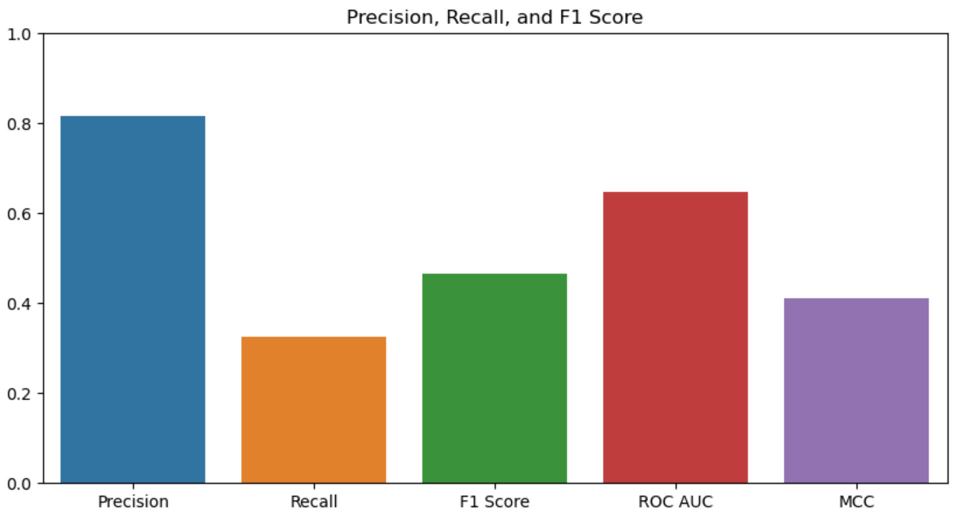

Figure 7: GNN results in Bar Chart

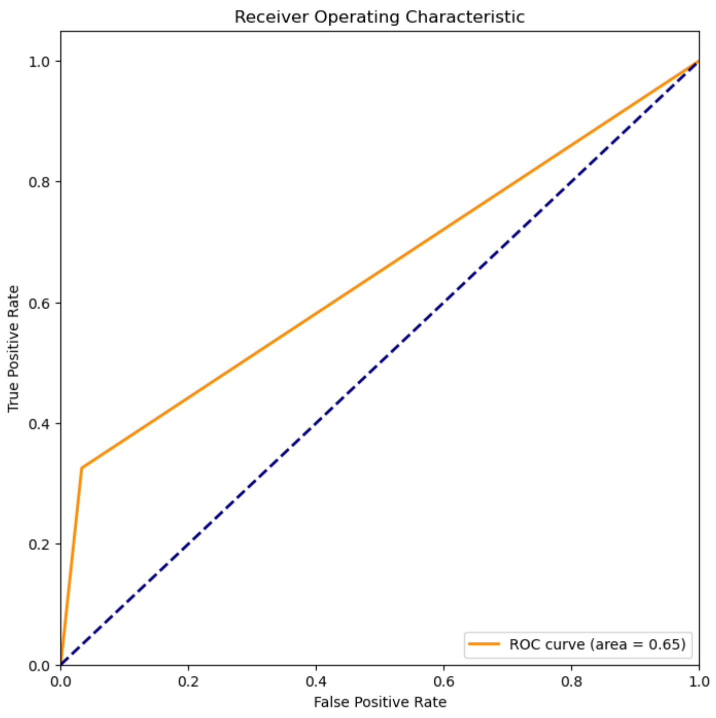

Figure 8: GNN ROC Curve for Word2Vec

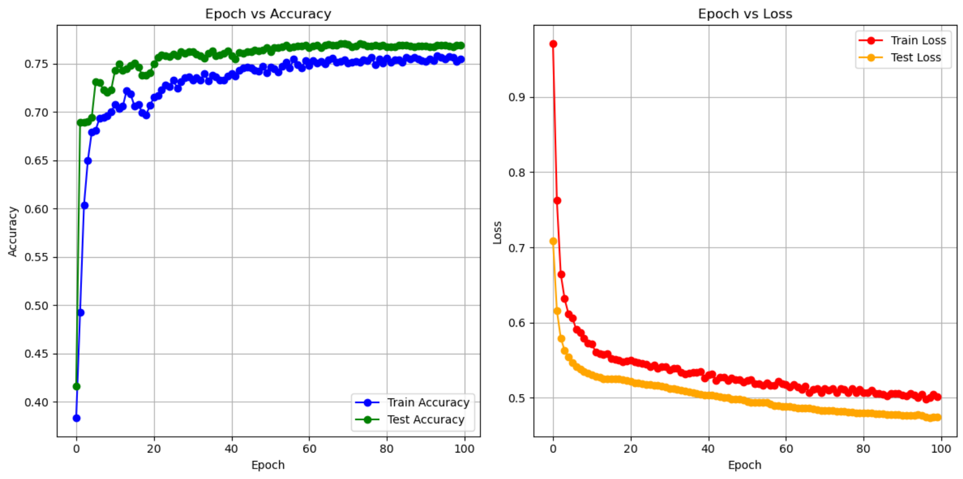

Figure 9: GNN Epoch vs Accuracy on Left and Epoch vs Loss on Right

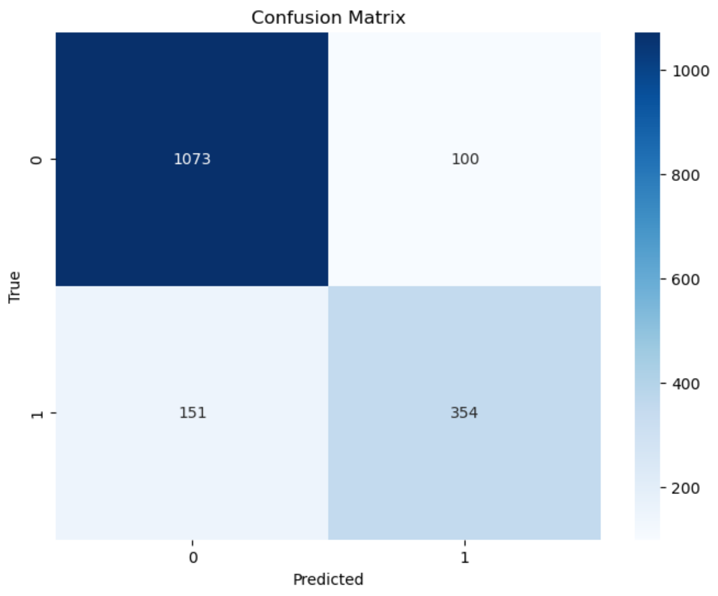

Figure 10: GNN Confusion Matrix

## 6.1 Performance Comparison of ML & DL models used in NLP

All the different models are compared based on the NLP techniques employed and their performance is evaluated using various metrics and scores.

### 6.1.1 Evaluation Metrics Comparison

In this, all the model scores are compared. Table 1 presents a comparison of the scores from different models, The best performing **Random Forests with Word2Vec** model not only accurately classifies the most instances correctly (high accuracy) but also maintains a high balance between precision and recall (as indicated by the F1 score) as shown in this bar chart Figure 12. Furthermore, its high AUC/ROC score suggests excellent capability in distinguishing between classes as shown in confuion matrix Figure 11, which is crucial for effective vulnerability detection.

In contrast, while the Random Forest with TF-IDF also performs well, especially in accuracy and precision, its F1 score and AUC/ROC are slightly lower than the Word2Vec variant, indicating that the Word2Vec version may be better at handling a balanced performance across different types of data. The GNN model did well in mapping the CVE and CWE with nodes and edges. Here is the results for GNN with GCN model in Figure 7 where precission was good but in other scores didn't perform well and Figure 8 where the curve is not that ideal, Figure 6 it is Epoch vs Accuracy and Epoch vs Loss, Figure 10 here is the detailed confusion matrix.

Therefore, for a comprehensive measure of performance across all these metrics, the Random Forest with Word2Vec stands out as the most effective model in this context

Table 1: Comparison of the scores from different models

| **Models** | **Accuracy** | **Precision** | **Recall** | **F1** | **AUC/ROC** |
|---|---|---|---|---|---|
| GNN + TF IDF | 0.85 | 0.77 | 0.70 | 0.73 | 0.80 |
| GNN + Word2Vec | 0.76 | 0.81 | 0.325 | 0.46 | 0.64 |
| Random Forest + TF IDF | 0.96 | 0.96 | 0.96 | 0.96 | 0.92 |
| Random Forest + Word2Vec | 0.98 | 0.98 | 0.96 | 0.97 | 0.97 |
| SVM + TF IDF | 0.81 | 0.75 | 0.78 | 0.56 | 0.74 |
| SVM + Word2Vec | 0.78 | 0.66 | 0.56 | 0.60 | 0.71 |

## 6.2 Discussion

In this research, I employed various models such as Graph Neural Networks (GNN) with Graph Convolutional Network (GCN), Support Vector Machines (SVM), and Random Forests to detect vulnerabilities in Android. Utilizing data from Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE), the study delved deeply into identifying different vulnerabilities, uncovering approximately 8500 distinct vulnerabilities reported by NVD.

With the available data, the CVE descriptions, CVE-related weakness IDs, and CWE data were leveraged to ascertain the severity of issues in the code. The models were trained on CVE descriptions to identify high-severity vulnerabilities and subsequently tested on similar data. This methodology was partly inspired by the study conducted by Renjith and Aji (2022), which provided insights into understanding the codebase.

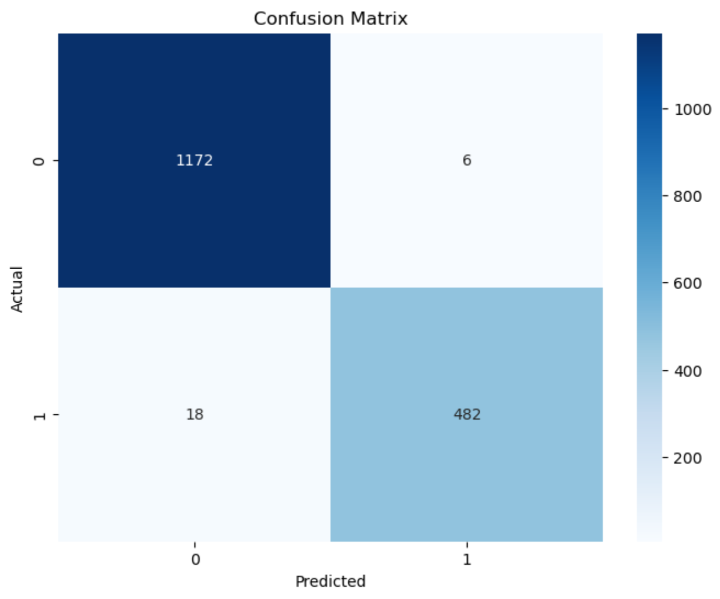

Figure 11: Random Forest Confusion Matrix

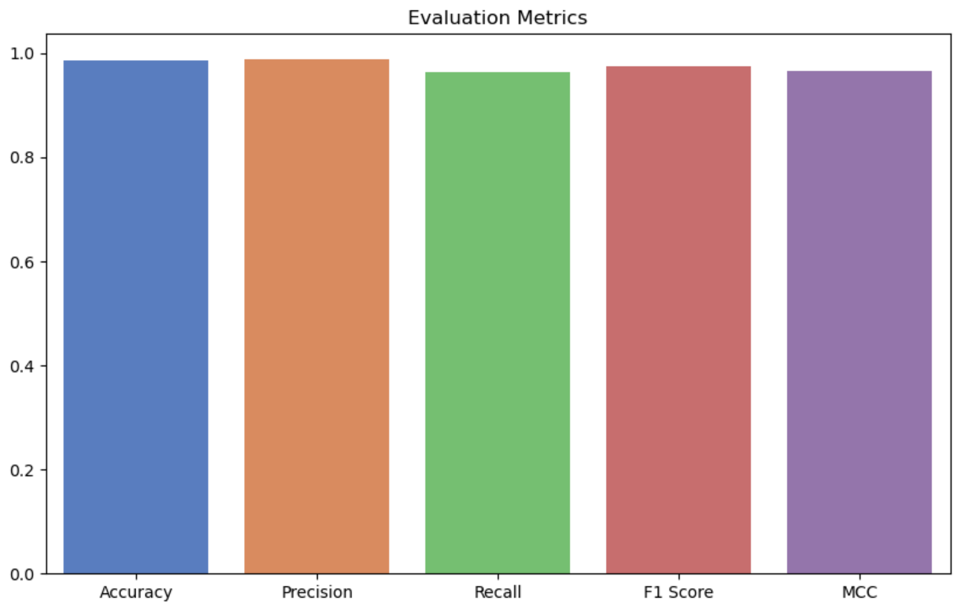

Figure 12: Random Forest results in Bar Chart

One of the primary challenges encountered was the limited availability of demonstrative CWE code. Access to more extensive CWE code examples would significantly enhance the capability to detect vulnerabilities in Android. This additional data would allow for more comprehensive training of the models, potentially unlocking the full potential of the system in identifying and addressing security vulnerabilities.

## 7 Conclusion and Future Work

In this study, I successfully addressed the research question concerning proactive vulnerability detection and security concern mitigation. By regularly updating the dataset and automating the process, the system can detect vulnerabilities as soon as they are added to the NVD database or when their weakness enumeration is linked with problematic code. This approach has proven effective in both detecting and mitigating vulnerabilities.

A key finding of this research is the ability to detect vulnerabilities using the descriptions provided in the CVE entries, which yielded high accuracy and precision-recall values. However, one limitation encountered was the lack of demonstrative code for all CWE vulnerabilities. Access to such code samples for all vulnerabilities would enable more precise training, enhancing the capability to scan codebases or APKs and pinpoint vulnerabilities or malware in the Android OS.

Looking ahead, leveraging comprehensive code examples for training could greatly improve the detection capabilities, especially as the coding landscape diversifies. With the increasing use of Java, Kotlin, and hybrid app development platforms like React Native, Flutter, and Ionic in Android development, there is a growing need for a more robust vulnerability detection system. Future work will focus on incorporating these aspects to enhance the ability to identify vulnerabilities in a more varied and evolving code environment.

## References

Amin, A., Eldessouki, A., Magdy, M. T., Abdeen, N., Hindy, H. and Hegazy, I. (2019). Androshield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach, *Information* **10**(10): 326.

Arslan, R. S. (2021). Androanalyzer: android malicious software detection based on deep learning, *PeerJ Computer Science* **7**(34084934): e533.

*Common Vulnerabilities and Exposures* (n.d.). `https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=android`. Accessed: Dec 08, 2023.

Garg, S. and Baliyan, N. (2020). Machine learning based android vulnerability detection: A roadmap, *in* S. Kanhere, V. T. Patil, S. Sural and M. S. Gaur (eds), *Information Systems Security*, Springer International Publishing, Cham, pp. 87–93.

Gencer, K. and Başçiftçi, F. (2021). Time series forecast modeling of vulnerabilities in the android operating system using arima and deep learning methods, *Sustainable Computing: Informatics and Systems* **30**: 100515.
**URL:** *https://www.sciencedirect.com/science/article/pii/S2210537921000081*

Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D. and Liu, H. (2020). A review of android malware detection approaches based on machine learning, *IEEE Access* **8**: 124579–124607.

Malik, Y., Campos, C. R. S. and Jaafar, F. (2019). Detecting android security vulnerabilities using machine learning and system calls analysis, *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 109–113.

*National Vulnerability Database* (n.d.). `https://nvd.nist.gov/vuln/search?results_type=overview&query=android+os&search_type=all&form_type=Basic&isCpeNameSearch=false`. Accessed: Dec 08, 2023.

Raghav, U., Martinez-Marroquin, E. and Ma, W. (2021). Static analysis for android malware detection with document vectors, *2021 International Conference on Data Mining Workshops (ICDMW)*, pp. 805–812.

Renjith, G. and Aji, S. (2022). Unveiling the security vulnerabilities in android operating system, *in* S. Shakya, K.-L. Du and W. Haoxiang (eds), *Proceedings of Second International Conference on Sustainable Expert Systems*, Springer Nature Singapore, Singapore, pp. 89–100.

Senanayake, J., Kalutarage, H., Al-Kadri, M. O., Petrovski, A. and Piras, L. (2023). Android source code vulnerability detection: A systematic literature review, *ACM Comput. Surv.* **55**(9).
**URL:** *https://doi.org/10.1145/3556974*

Wartschinski, L., Noller, Y., Vogel, T., Kehrer, T. and Grunske, L. (2022). Vudenc: Vulnerability detection with deep learning on a natural codebase for python, *Information and Software Technology* **144**: 106809.
**URL:** *https://www.sciencedirect.com/science/article/pii/S0950584921002421*

Yuan, H., Tang, Y., Sun, W. and Liu, L. (2020). A detection method for android application security based on tf-idf and machine learning, *PLoS One* **15**(9): e0238694.
**URL:** *https://pubmed.ncbi.nlm.nih.gov/32915836/*