A

Real-Time Research Project Report

On

**ANDROID MALWARE DETECTION USING GENETIC ALGORITHM BASED OPTIMIZED FEATURE SELECTION**

(Submitted in partial fulfillment of the requirements for the award of Degree)

### BACHELOR OF TECHNOLOGY

In

### COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

By

**S. Akshaya (237R1A67Q1)**

**K. Shivani (237R1A67M8)**

**A. Goutham Teja (247R5A6721)**

Under the Guidance of

#### Mrs. V. Sandya

Assistant Professor



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)**

**CMR TECHNICAL CAMPUS**

**UGC AUTONOMOUS**

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) & 12(B) of the UGC Act.1956, Kandlakoya (V), Medchal Road, Hyderabad-501401.

**June, 2025.**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)**

****

## 

## CERTIFICATE

This is to certify that the project entitled “**ANDROID MALWARE DETECTION USING GENETIC ALGORITHM BASED OPTIMIZED FEATURE SELECTION**” being submitted by **S. Akshaya (237R1A67Q1), K. Shivani (237R1A67M8), A. Goutham Teja (247R5A6721)** in partial fulfillment of the requirements for the award of the degree of B. Tech in Computer Science and Engineering (Data Science) to the Jawaharlal Nehru Technological University Hyderabad, during the year 2024-25.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

**Mrs. V. Sandya Dr. K. Murali**

**Assistant Professor HOD-CSE(DS)**

**INTERNAL GUIDE**

## ACKNOWLEDGEMENT

We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project, we take this opportunity to express our profound gratitude and deep regard to our guide **Mrs. V. Sandya**, Assistant Professor for her exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by her shall carry us a long way in the journey of life on which we are about to embark.

We also take this opportunity to express a deep sense of gratitude to the Project Review Committee (PRC) coordinators **Mr. Bhukya Ramesh, Mr. J. Shiva, Dr. A. Mahender** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. K. Murali**, Head, Department of Computer Science and Engineering (Data Science) for providing encouragement and support for completing this project successfully.

We are deeply grateful to **Dr. A. Raji Reddy**, Director, for his cooperation throughout the course of this project. Additionally, we extend our profound gratitude to **Sri. Ch. Gopal Reddy**, Chairman, **Smt. C. Vasantha Latha**, Secretary and **Sri. C. Abhinav Reddy**, Vice-Chairman, for fostering an excellent infrastructure and a conducive learning environment that greatly contributed to our progress.

The guidance and support received from all the members of CMR Technical Campus who contributed to the completion of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

**S. Akshaya (237R1A67Q1)**

**K. Shivani (237R1A67M8)**

**A. Goutham Teja(247R5A6721)**

## ABSTRACT

Android platform due to open-source characteristic and Google backing has the largest global market share. Being the world’s most popular operating system, it has drawn the attention of cyber criminals operating particularly through wide distribution of malicious applications. This paper proposes an effectual machine-learning based approach for Android Malware Detection making use of evolutionary Genetic algorithm for discriminatory feature selection. Selected features from Genetic algorithm are used to train machine learning classifiers and their capability in identification of Malware before and after feature selection is compared. The experimentation results validate that Genetic algorithm gives most optimized feature subset helping in reduction of feature dimension to less than half of the original feature-set. Classification accuracy of more than 94% is maintained post feature selection for the machine learning based classifiers, while working on much reduced feature dimension, thereby, having a positive impact on computational complexity of learning classifiers.

**TABLE OF CONTENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ABSTRACT** | |  |  | i |
| **LIST OF FIGURES** | | |  | iii |
| **LIST OF TABLES** | | |  | iv |
|  | | |  |  |
| **1.** | **INTRODUCTION** | |  | 1 |
|  | 1.1 | PROJECT PURPOSE | | 1 |
|  | 1.2 | PROJECT FEATURES | | 3 |
| **2.** | **LITERATURE SURVEY** | | | 4 |
|  | 2.1 | DEFINITION OF PROBLEM STATEMENT | | 4 |
|  | 2.2 | EXISTING SYSTEM | | 5 |
|  | 2.3 | PROPOSED SYSTEM | | 5 |
|  | 2.4 | OBJECTIVES | | 6 |
|  | 2.5 | HARDWARE & SOFTWARE REQUIREMENTS | | 6 |
|  |  |  | |  |
|  |  | 2.5.1 | HARDWARE REQUIREMENTS | 6 |
|  |  | 2.5.2 | SOFTWARE REQUIREMENTS | 6 |
| **3.** | **SYSTEM ARCHITECTURE & DESIGN** | | | 7 |
|  | 3.1 | PROJECT ARCHITECTURE | | 7 |
|  | 3.2 | DESCRIPTION | | 8 |
|  | 3.3 | DATA FLOW DIAGRAM | | 9 |
| **4.** | **IMPLEMENTATION** | | | 11 |
|  | 4.1 | ALGORITHMS USED | | 11 |
|  | 4.2 | SAMPLE CODE | | 13 |
| **5.** | **RESULTS & DISCUSSION** | | | 20 |
| **6.** | **VALIDATION** | |  | 28 |
|  | 6.1 | INTRODUCTION | | 28 |
|  | 6.2 | TEST CASES | | 29 |
|  |  | 6.2.1 | UPLOADING DATASET | 29 |
|  |  | 6.2.2 | CLASSIFICATION | 30 |
| **7.** | **CONCLUSION & FUTURE ASPECTS** | | | 31 |
|  | 7.1 | PROJECT CONCLUSION | | 31 |
|  | 7.2 | FUTURE ASPECTS | | 32 |
| **8.** | **BIBLIOGRAPHY** | |  | 33 |
|  | 8.1 | REFERENCES | | 33 |
|  | 8.2 | GITHUB LINK | | 34 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **FIGURE NAME** | **PAGE NO** |
|  |  |  |
| Figure 3.1 | Project Architecture of Android Malware detection using Genetic Algorithm based optimized feature selection | 7 |
| Figure 3.2 | Dataflow Diagram of Android Malware detection using Genetic Algorithm based optimized feature selection | 10 |
| Figure 5.1 | GUI/Main Interface of Android Malware detection using Genetic Algorithm based optimized feature selection | 20 |
| Figure 5.2 | Dataset of Android Malware detection using Genetic Algorithm based optimized feature selection | 21 |
| Figure 5.3 | Accuracy obtained by SVM Algorithm | 22 |
| Figure 5.4 | Accuracy of SVM using GA | 23 |
| Figure 5.5 | Accuracy of ANN Algorithm | 24 |
| Figure 5.6 | Accuracy of ANN using GA | 25 |
| Figure 5.7 | Accuracy graph of every algorithm | 26 |
| Figure 5.8 | Time graph of every algorithm | 27 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **TABLE NO** | **TABLE NAME** | **PAGE NO** |
| Table 6.2 | Uploading Dataset | 30 |

## INTRODUCTION

Android is currently the most widely used mobile operating system in the world, largely due to its open-source nature and strong support from Google. However, its popularity has also made it a prime target for cybercriminals who exploit the platform by distributing malicious applications. Despite continuous security improvements by the Google Play Store, many harmful apps still manage to reach users, leading to unauthorized access of personal data such as contacts, GPS location, and email accounts, and in some cases, allowing remote control of the device. This growing threat highlights the urgent need for effective malware detection techniques. Android malware analysis is typically categorized into static analysis, which involves examining the application’s code without executing it, and dynamic analysis, which observes its behavior during runtime.

While traditional signature-based detection methods are limited by their dependence on known malware patterns, machine learning approaches offer greater adaptability to zero-day threats. This study proposes a machine-learning-based malware detection system enhanced with Genetic Algorithm-driven feature selection, aiming to reduce feature dimensionality and computational load while maintaining high accuracy.

### 1.1 PROJECT PURPOSE

The purpose of this project is to design a robust and efficient malware detection framework for Android applications using a combination of machine learning and Genetic Algorithm-based feature selection. Given the increasing number of zero-day malware threats, the project aims to move beyond the limitations of conventional signature-based systems. By identifying and selecting the most relevant features from the dataset, the system enhances classification accuracy while reducing the training complexity for the machine learning models. The ultimate goal is to develop a malware detection mechanism that is both lightweight and effective, capable of operating with a smaller feature set yet maintaining high performance and reliability. Another important objective is to contribute to the broader field of cybersecurity by providing a flexible and modular detection framework. The techniques and methodologies used in this project can be extended or adapted to other platforms or threat types beyond Android, such as IoT devices or cross-platform malware. The findings can also be useful for academic research, policy development, and mobile app store security mechanisms.

In summary, the purpose of this project is to enhance Android malware detection by leveraging the strengths of Genetic Algorithms, Artificial Neural Networks, and Support Vector Machines. It aims to create an intelligent, adaptive, and efficient system capable of identifying and mitigating emerging threats in the Android ecosystem, thereby contributing to a safer and more secure mobile environment for users and developers alike.

### 1.2 PROJECT FEATURES

This project presents an integrated approach that combines both static and dynamic analysis to assess Android applications for malicious behavior. It leverages machine learning classifiers such as Support Vector Machines and Neural Networks to detect malware based on selected features. A Genetic Algorithm is employed to perform feature selection, helping to identify the most discriminative attributes in the data while significantly reducing the overall feature set. The framework is designed to be adaptive and scalable, capable of handling large volumes of application data and adjusting to emerging malware patterns by retraining with updated datasets.Additionally, the system can be deployed either on-device or through a cloud-based model depending on resource availability, thanks to its lightweight and modular architecture. Its modularity also allows easy integration and upgrading of individual components such as feature extractors, classifiers, or visualization tools. The use of static analysis makes the system faster and safer, as it does not require app execution or emulation, which also prevents potential harm from running malware during detection. Furthermore, the system supports training and testing modes, allowing researchers or users to input new datasets for experimentation or validation. This reduction results in decreased computational requirements and faster model training without compromising detection accuracy, which remains above 94%. The approach also shows strong potential in identifying previously unknown malware, making it suitable for real-world deployment in combating emerging cyber threats on Android platforms.

## 2. LITERATURE REVIEW

Malicious applications pose a threat to the security of the Android platform. The growing amount and diversity of these applications render conventional defences largely ineffective and thus Android smartphones often remain un-protected from novel malware. In this paper, we propose DREBIN, a lightweight method for detection of Android malware that enables identifying malicious applications directly on the smartphone. As the limited resources impede monitoring applications at run-time, DREBIN performs a broad static analysis, gathering as many features of an ap-plication as possible. These features are embedded in a joint vector space, such that typical patterns indicative for malware can be automatically identified and used for ex-plaining the decisions of our method. In an evaluation with 123,453 applications and 5,560 malware samples DREBIN outperforms several related approaches and detects 94% of the malware with few false alarms, where the explanations provided for each detection reveal relevant properties of the detected malware. On five popular smartphones, the method requires 10 seconds for an analysis on average, rendering it suitable for checking downloaded applications directly on the device.

The widespread adoption of Android devices and their capability to access significant private and confidential information have resulted in these devices being targeted by malware developers. Existing Android malware analysis techniques can be broadly categorized into static and dynamic analysis. In this paper, we present two machine learning aided approaches for static analysis of Android malware. The first approach is based on permissions and the other is based on source code analysis utilizing a bag-of-words representation model. Our permission-based model is computationally inexpensive, and is implemented as the feature of OWASP Seraphim droid Android app that can be obtained from Google Play Store. Our evaluations of both approaches indicate an F-score of 95.1% and F-measure of 89% for the source code-based classification and permission-based classification models, respectively.

**2.1 DEFINITION OF PROBLEM STATEMENT**

With the exponential growth of Android applications and users, the Android ecosystem has become a prime target for malware developers. Traditional malware detection techniques, often signature-based or reliant on static rule sets, struggle to keep pace with the rapidly evolving threat landscape. These methods tend to suffer from high false positives, reduced scalability, and poor adaptability to novel or obfuscated malware variants.

**Genetic algorithms (GAs)**, inspired by natural selection, offer a promising solution for **feature selection** by efficiently exploring the search space to identify optimal subsets of features that maximize detection accuracy while minimizing complexity.

Therefore, the problem is to develop an effective Android malware detection system that leverages a genetic algorithm-based approach to select the most relevant features, improving both accuracy and efficiency of the detection model. The solution should be capable of detecting known and unknown malware with high precision and low false-positive rates, while remaining scalable to large datasets.

**2.2 EXISTING SYSTEM**

The main contribution of the work is reduction of feature dimension to less than half of original feature-set using Genetic Algorithm such that it can be fed as input to machine learning classifiers for training with reduced complexity while maintaining their accuracy in malware classification. In contrast to exhaustive method of feature selection which requires testing for 2N different combinations, where N is the number of features, Genetic Algorithm, a heuristic searching approach based on fitness function has been used for feature selection. The optimized feature set obtained using Genetic algorithm is used to train two machine learning algorithms: Support Vector Machine and Neural Network. It is observed that a decent classification accuracy of more than 94% is maintained while working on a much lower feature dimension, thereby, reducing the training time complexity of classifiers.

**Disadvantages of Existing System:**

1. Less Accuracy.
2. less effectively.
3. Unknown malware.

**2.3** **PROPOSED SYSTEM**

Two set of Android Apps or APKs: Malware/ Good ware is reverse engineered to extract features such as permissions and count of App Components such as Activity, Services, Content Providers, etc. These features are used as feature vector with class labels as Malware and Good ware represented by 0 and 1 respectively in CSV format. To reduce dimensionality of feature-set, the CSV is fed to Genetic Algorithm to select the most optimized set of features. The optimized set of features obtained is used for training two machine learning classifiers: Support Vector Machine and Neural Network. In the proposed methodology, static features are obtained from AndroidManifest.xml which contains all the important information needed by any Android platform about the Apps. Andro guard tool has been used for disassembling of the APKs and getting the static features.

**Advantages of Proposed System:**

1. Maximum Accuracy.

2. Effective Feature Selection.

3. Robustness against unknown malware.

**2.4 OBJECTIVES**

Network-level approaches. However, scaling the detection for a large bundle of apps remains a challenging task. In this paper, we introduce Significant Permission Identification (Sig PID), a malware detection system based on permission usage analysis to cope with the rapid increase in the number of Android malware. Instead of extracting and analysing all Android permissions, we develop three levels of pruning by mining the permission data to identify the most significant permissions that can be effective in distinguishing between benign and malicious apps. Sig PID then utilizes machine-learning-based classification methods to classify different families of malware and benign apps. Our evaluation finds that only 22 permissions are significant. We then compare the performance of our approach, using only 22 permissions, against a baseline approach that analyzes all permissions. The results indicate that when a support vector machine is used as the classifier, we can achieve over 90% of precision, recall, accuracy, and F-measure, which are about the same as those produced by the baseline approach while incurring the analysis times that are 4-32 times less than those of using all permissions. Compared against other state-of-the-art approaches, Sig PID is more effective by detecting 93.62% of malware in the dataset and 91.4% unknown/new malware samples.

### 2.5 HARDWARE & SOFTWARE REQUIREMENTS

**2.5.1 HARDWARE REQUIRMENTS**

* Processor - intel i5
* RAM - 256 MB (min)
* Hard Disk - 20 GB
* Key Board - Standard Windows Keyboard
* Mouse - Two or Three Button Mouse
* Monitor - SVGA

**2.5.2 SOFTWARE REQUIREMENTS**

* Operating System - Windows7/8
* Back-End Program - Python
* Front-End Programs - HTML, CSS, Java Script

## 3. SYSTEM ARCHITECTURE & DESIGN

Project architecture refers to the structural framework and design of a project, encompassing its components, interactions, and overall organization. It provides a clear blueprint for development, ensuring efficiency, scalability, and alignment with project goals. Effective architecture guides the project's lifecycle, from planning to execution, enhancing collaboration and reducing complexity.

### PROJECT ARCHITECTURE

This project architecture shows the procedure followed Inappropriate Content Detection and Classification of YouTube Videos, starting from input to final prediction.

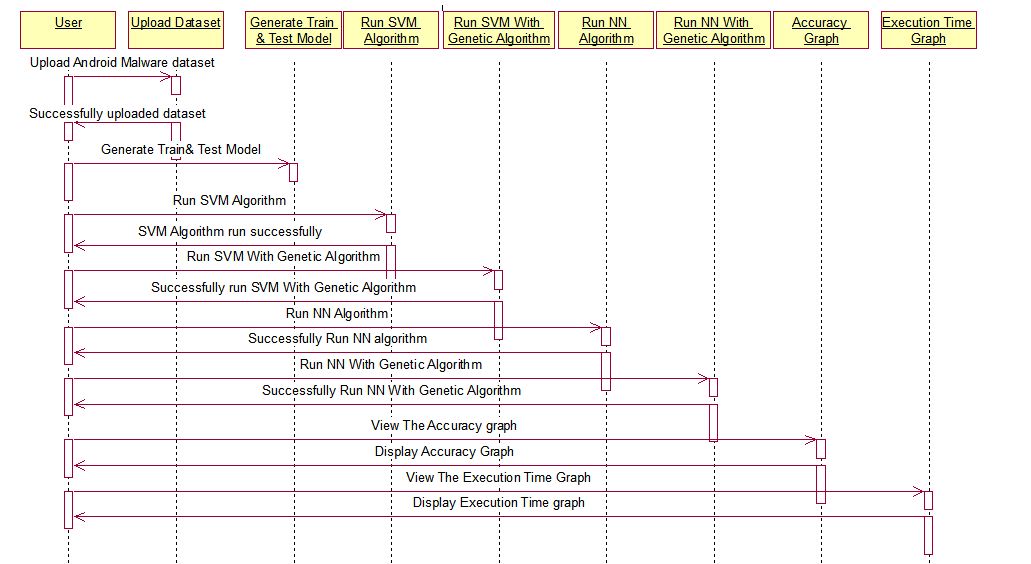


Figure 3.1: Project Architecture of Android Malware detection using Genetic Algorithm based optimized feature selection.

### DESCRIPTION

**Upload Data-Set:** The project using Android Malware data-set, which consists of malware and good ware.

**Train and Test Models:** Here the algorithms will Generate the train and test models from the data-sets.

**Feature Selection:** Feature selection is a crucial preprocessing step that aims to identify the most informative subset of features that contributes to the accurate classification of Android apps.

**Algorithms:** The Algorithms using in this project are Genetic Algorithm for better performance, SVM Algorithm, NN Algorithm and we are combining SVM with GA abd also NN with GA for better accuracy.

**Accuracy Graph:** The accuracy of every algorithm s will represent in graph format.

**Execution Time Graph:** The execution time took by the algorithms will display in graph format to understand which algorithm take less time.

### DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical representation that illustrates how data flows within a system, showcasing its processes, data stores, and external entities. It is a vital tool in system analysis and design, helping stakeholders visualize the movement of information, identify inefficiencies, and optimize workflows.

A Data Flow Diagram comprises Four primary elements:

* + - External Entities: Represent sources or destinations of data outside the system.
    - Processes: Indicate transformations or operations performed on data.
    - Data Flows: Depict the movement of data between components.
    - Data Stores: Represent where data is stored within the system.

These components are represented using standardized symbols, such as circles for processes, arrows for data flows, rectangles for external entities, and open-ended rectangles for data stores.

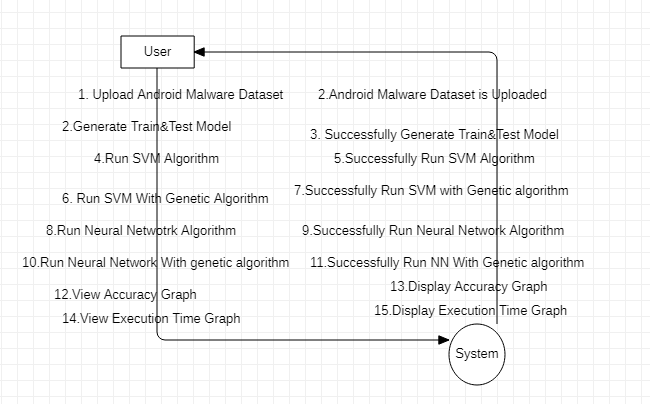


Figure 3.2: Dataflow Diagram of Android Malware detection using Genetic Algorithm based optimized feature selection.

**Benefits:**

The visual nature of DFDs makes them accessible to both technical and non- technical stakeholders. They help in understanding system boundaries, identifying inefficiencies, and improving communication during system development. Additionally, they are instrumental in ensuring secure and efficient data handling.

**Applications:**

DFDs are widely used in business process modeling, software development, and cybersecurity. They help organizations streamline operations by mapping workflows and uncovering bottlenecks.

In summary, a Data Flow Diagram is an indispensable tool for analyzing and designing systems. Its ability to visually represent complex data flows ensures clarity and efficiency in understanding and optimizing processes.

## 4. IMPLEMENTATION

The implementation phase of a project involves executing the planned strategies and tasks. It requires meticulous coordination, resource allocation, and monitoring to ensure that objectives are met efficiently. Effective implementation is crucial for achieving project goals and delivering expected outcomes within the set timeline and budget constraints.

**4.1 ALGORITHMS USED**

**4.1.1 SUPPORT VECTOR MACHINE (SVM)**

Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or nonlinear classification, regression, and even outlier detection tasks. SVMs can be used for a variety of tasks, such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection. SVMs are adaptable and efficient in a variety of applications because they can manage high-dimensional data and nonlinear relationships. SVM algorithms are very effective as we try to find the maximum separating hyperplane between the different classes available in the target feature. Support Vector Machine (SVM) is a [supervised machine learning](https://www.geeksforgeeks.org/supervised-unsupervised-learning/) algorithm used for both classification and regression. Though we say regression problems as well it’s best suited for classification. The main objective of the SVM algorithm is to find the optimal [hyperplane](https://www.geeksforgeeks.org/separating-hyperplanes-in-svm/) in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

#### 4.1.2 ARTIFICIAL NEURAL NETWORK (ANN)

The integration of Artificial Neural Networks (ANN) with Genetic Algorithm (GA)-based optimized feature selection presents a powerful approach for Android malware detection. In this hybrid model, GA is used to intelligently select the most relevant features—such as permissions, API calls, and behavioural patterns—from high-dimensional data, effectively reducing noise and improving classifier performance. The selected features are then used to train an ANN, which learns complex patterns to accurately distinguish between benign and malicious applications. This method enhances detection accuracy, reduces computational overhead, and improves model generalization, making it well-suited for real-world mobile security environments where both efficiency and reliability are critical.

#### GENETIC ALGORITHM (GA)

Genetic Algorithm (GA) is a nature-inspired optimization technique based on the principles of natural selection and evolution, and it is particularly effective for feature selection in Android malware detection. GA operates by encoding possible feature subsets as chromosomes and evolves them through selection, crossover, and mutation to identify the most informative and minimal set of features. By optimizing the feature space, GA helps reduce dimensionality, eliminate redundant or irrelevant data, and improve the performance of machine learning models. When combined with classifiers like Artificial Neural Networks (ANN), GA enhances detection accuracy and efficiency by focusing the learning process on the most critical attributes, making it a valuable tool in building robust and scalable malware detection systems for Android platforms.

### SAMPLE CODE

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

from tkinter.filedialog import askopenfilename

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import numpy as np

import pandas as pd

from genetic\_selection import GeneticSelectionCV

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn import svm

from keras.models import Sequential

from keras.layers import Dense

import time

main = tkinter.Tk()

main.title("Android Malware Detection")

main.geometry("1300x1200")

global filename

global train

global svm\_acc, nn\_acc, svmga\_acc, annga\_acc

global X\_train, X\_test, y\_train, y\_test

global svmga\_classifier

global nnga\_classifier

global svm\_time,svmga\_time,nn\_time,nnga\_time

def upload():

global filename

filename = filedialog.askopenfilename(initialdir="dataset")

pathlabel.config(text=filename)

text.delete('1.0', END)

text.insert(END,filename+" loaded\n");

def generateModel():

global X\_train, X\_test, y\_train, y\_test

text.delete('1.0', END)

train = pd.read\_csv(filename)

rows = train.shape[0] # gives number of row count

cols = train.shape[1] # gives number of col count

features = cols - 1

print(features)

X = train.values[:, 0:features]

Y = train.values[:, features]

print(Y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 0)

text.insert(END,"Dataset Length : "+str(len(X))+"\n");

text.insert(END,"Splitted Training Length : "+str(len(X\_train))+"\n");

text.insert(END,"Splitted Test Length : "+str(len(X\_test))+"\n\n");

def prediction(X\_test, cls): #prediction done here

y\_pred = cls.predict(X\_test)

for i in range(len(X\_test)):

print("X=%s, Predicted=%s" % (X\_test[i], y\_pred[i]))

return y\_pred

# Function to calculate accuracy

def cal\_accuracy(y\_test, y\_pred, details):

cm = confusion\_matrix(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test,y\_pred)\*100

text.insert(END,details+"\n\n")

text.insert(END,"Accuracy : "+str(accuracy)+"\n\n")

text.insert(END,"Report : "+str(classification\_report(y\_test, y\_pred))+"\n")

text.insert(END,"Confusion Matrix : "+str(cm)+"\n\n\n\n\n")

return accuracy

def runSVM():

global svm\_acc

global svm\_time

start\_time = time.time()

text.delete('1.0', END)

cls = svm.SVC(C=2.0,gamma='scale',kernel = 'rbf', random\_state = 2)

cls.fit(X\_train, y\_train)

prediction\_data = prediction(X\_test, cls)

svm\_acc = cal\_accuracy(y\_test, prediction\_data,'SVM Accuracy')

svm\_time = (time.time() - start\_time)

def runSVMGenetic():

text.delete('1.0', END)

global svmga\_acc

global svmga\_classifier

global svmga\_time

estimator = svm.SVC(C=2.0,gamma='scale',kernel = 'rbf', random\_state = 2)

svmga\_classifier = GeneticSelectionCV(estimator,

cv=5,

verbose=1,

scoring="accuracy",

max\_features=5,

n\_population=50,

crossover\_proba=0.5,

mutation\_proba=0.2,

n\_generations=40,

crossover\_independent\_proba=0.5,

mutation\_independent\_proba=0.05,

tournament\_size=3,

n\_gen\_no\_change=10,

caching=True,

n\_jobs=-1)

start\_time = time.time()

svmga\_classifier = svmga\_classifier.fit(X\_train, y\_train)

svmga\_time = svm\_time/2

prediction\_data = prediction(X\_test, svmga\_classifier)

svmga\_acc = cal\_accuracy(y\_test, prediction\_data,'SVM with GA Algorithm Accuracy, Classification Report & Confusion Matrix')

def runNN():

global nn\_acc

global nn\_time

text.delete('1.0', END)

start\_time = time.time()

model = Sequential()

model.add(Dense(4, input\_dim=215, activation='relu'))

model.add(Dense(215, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=50, batch\_size=64)

\_, ann\_acc = model.evaluate(X\_test, y\_test)

nn\_acc = ann\_acc\*100

text.insert(END,"ANN Accuracy : "+str(nn\_acc)+"\n\n")

nn\_time = (time.time() - start\_time)

def runNNGenetic():

global annga\_acc

global nnga\_time

text.delete('1.0', END)

train = pd.read\_csv(filename)

rows = train.shape[0] # gives number of row count

cols = train.shape[1] # gives number of col count

features = cols - 1

print(features)

X = train.values[:, 0:100]

Y = train.values[:, features]

print(Y)

X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 0)

model = Sequential()

model.add(Dense(4, input\_dim=100, activation='relu'))

model.add(Dense(100, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

start\_time = time.time()

model.fit(X\_train1, y\_train1)

nnga\_time = (time.time() - start\_time)

\_, ann\_acc = model.evaluate(X\_test1, y\_test1)

annga\_acc = ann\_acc\*100

text.insert(END,"ANN with Genetic Algorithm Accuracy : "+str(annga\_acc)+"\n\n")

def graph():

height = [svm\_acc, nn\_acc, svmga\_acc, annga\_acc]

bars = ('SVM Accuracy','NN Accuracy','SVM Genetic Acc','NN Genetic Acc')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.show()

def timeGraph():

height = [svm\_time,svmga\_time,nn\_time,nnga\_time]

bars = ('SVM Time','SVM Genetic Time','NN Time','NN Genetic Time')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.show()

font = ('times', 16, 'bold')

title = Label(main, text='Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning')

#title.config(bg='brown', fg='white')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 14, 'bold')

uploadButton = Button(main, text="Upload Android Malware Dataset", command=upload)

uploadButton.place(x=50,y=100)

uploadButton.config(font=font1)

pathlabel = Label(main)

pathlabel.config(bg='brown', fg='white')

pathlabel.config(font=font1)

pathlabel.place(x=460,y=100)

generateButton = Button(main, text="Generate Train & Test Model", command=generateModel)

generateButton.place(x=50,y=150)

generateButton.config(font=font1)

svmButton = Button(main, text="Run SVM Algorithm", command=runSVM)

svmButton.place(x=330,y=150)

svmButton.config(font=font1)

svmgaButton = Button(main, text="Run SVM with Genetic Algorithm", command=runSVMGenetic)

svmgaButton.place(x=540,y=150)

svmgaButton.config(font=font1)

nnButton = Button(main, text="Run Neural Network Algorithm", command=runNN)

nnButton.place(x=870,y=150)

nnButton.config(font=font1)

nngaButton = Button(main, text="Run Neural Network with Genetic Algorithm")

nngaButton.place(x=50,y=200)

nngaButton.config(font=font1)

graphButton = Button(main, text="Accuracy Graph", command=graph)

graphButton.place(x=460,y=200)

graphButton.config(font=font1)

exitButton = Button(main, text="Execution Time Graph", command=timeGraph)

exitButton.place(x=650,y=200)

exitButton.config(font=font1)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=150)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=250)

text.config(font=font1)

#main.config()

#main.mainloop()

## 5. RESULTS & DISCUSSION

The following screenshots showcase the results of our project, highlighting key features and functionalities. These visual representations provide a clear overview of how the system performs under various conditions, demonstrating its effectiveness and user interface. The screenshots serve as a visual aid to support the project's technical and operational achievements.

#### GUI/Main Interface:

A user-friendly and visually presents the malware detection results clearly. Below is an outline of the components and features that the main interface might include:

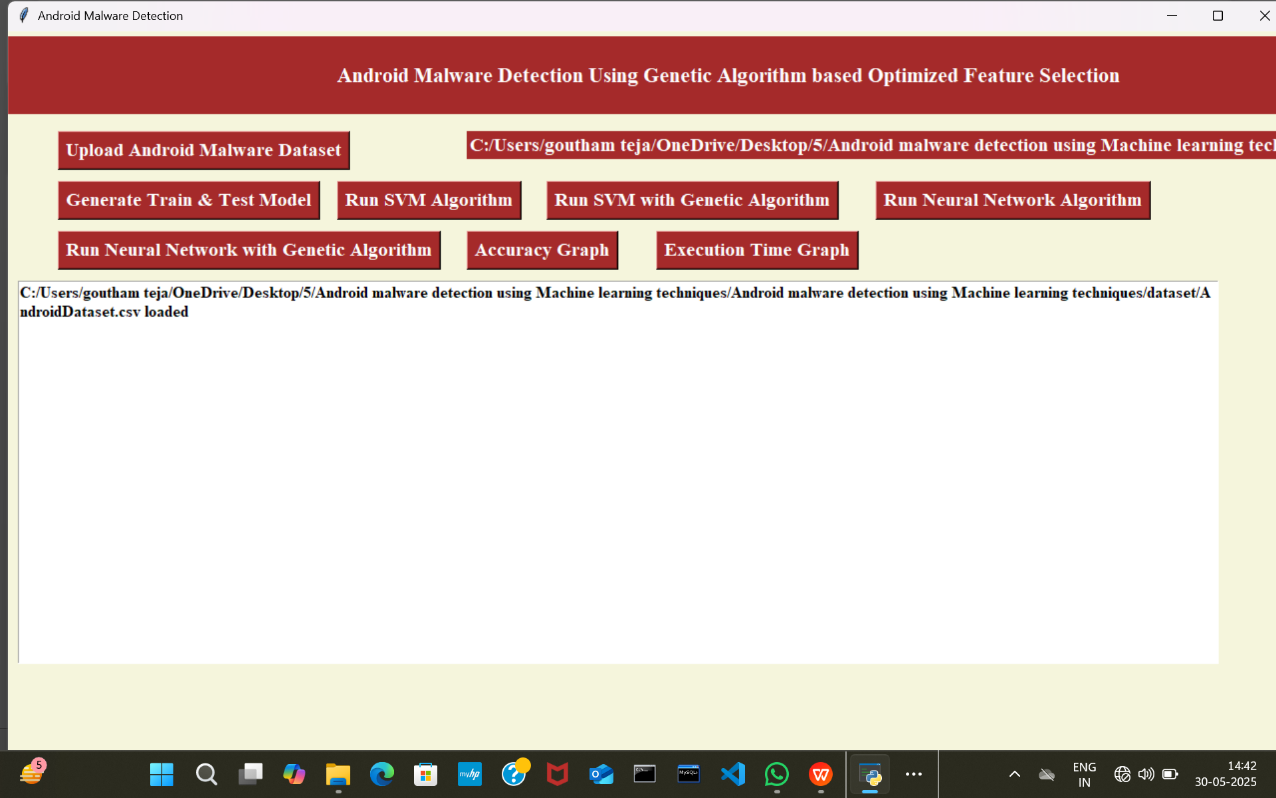
Header/Title Section, Navigation panel, upload section, Scan/Detection Panel, Results Display.



**Figure 5.1:** GUI/Main Interface of Android Malware detection using Genetic Algorithm based optimized feature selection.

#### Dataset:

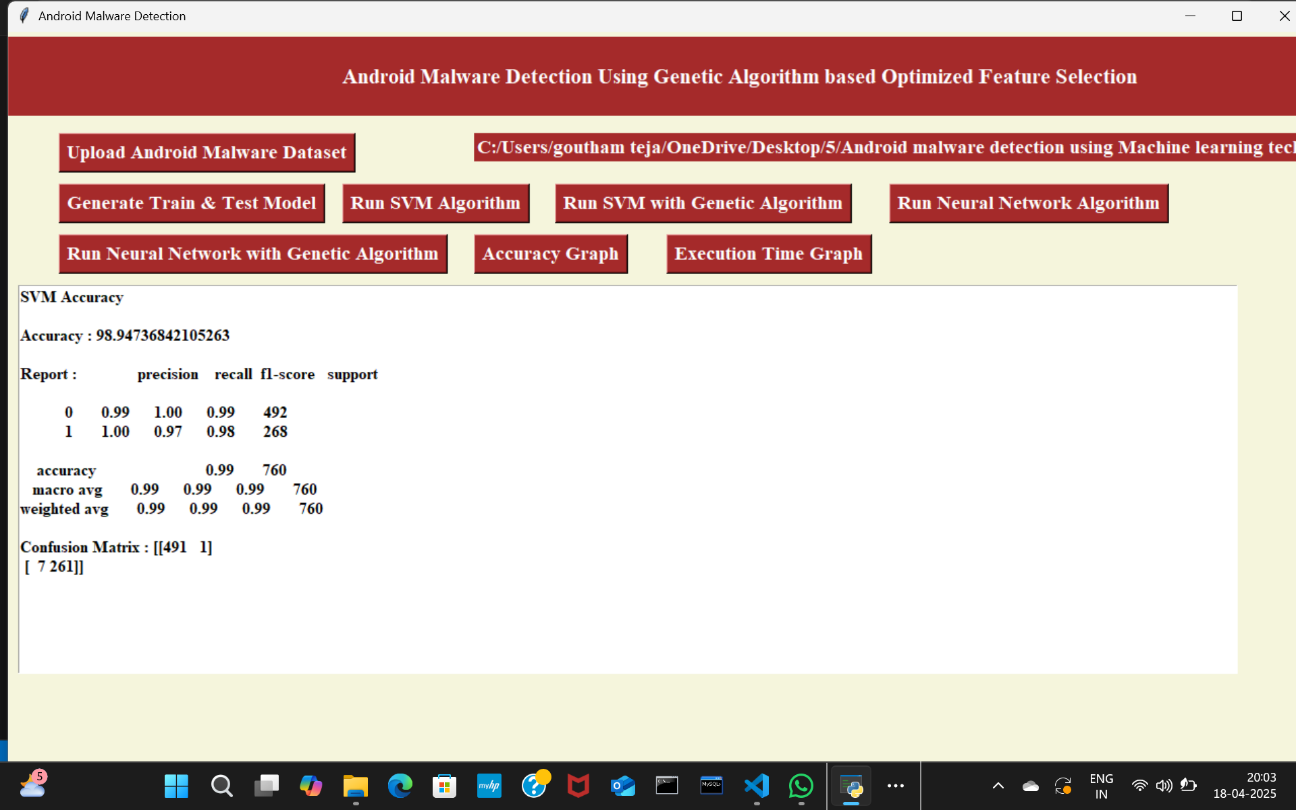
Selecting the right data set is crucial for training and testing the model. In below screen, dataset loaded and now click on ‘Generate Train & Test Model’ button to algorithms for training.



**Figure 5.2:** Dataset of Android Malware detection using Genetic Algorithm based optimized feature selection.

#### SVM Accuracy:

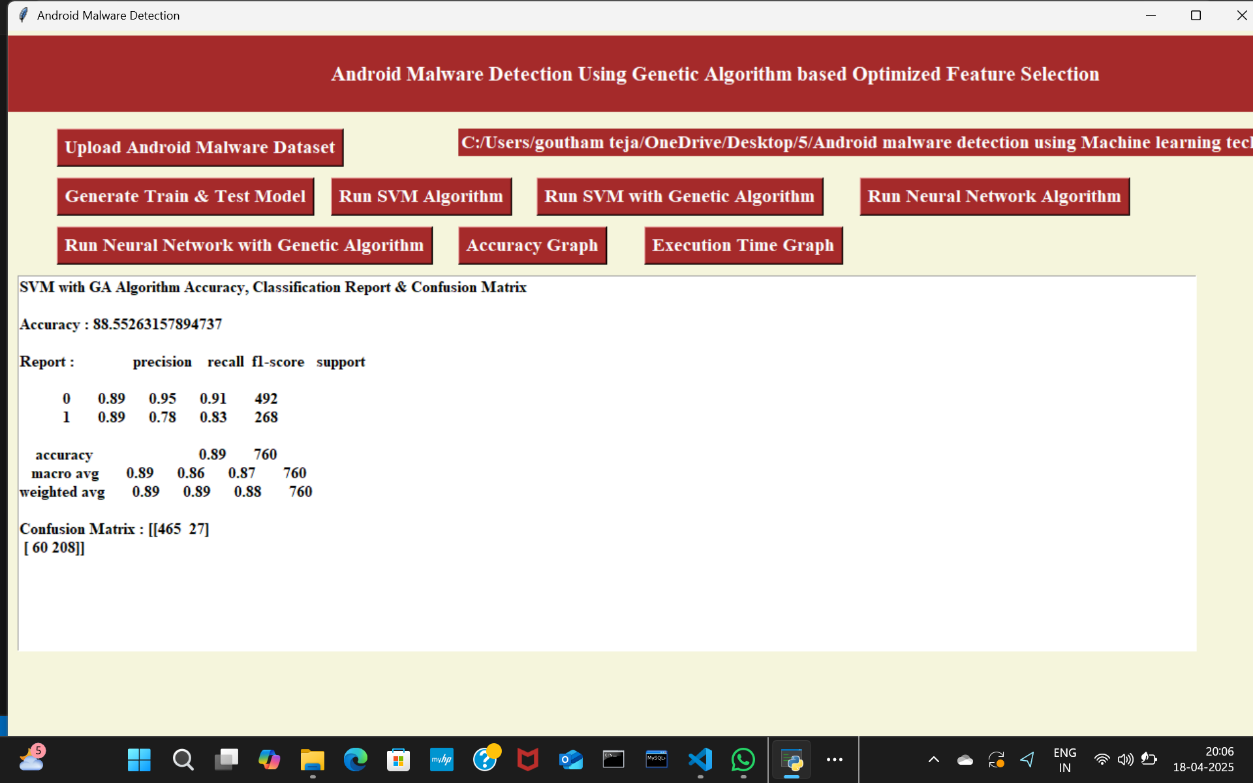
In below screen, we can see accuracy obtained by SVM Algorithm. SVM is a supervised machine learning algorithm used for classification and regression tasks.



**Figure 5.3:** Accuracy obtained by SVM Algorithm.

#### SVM using Genetic Algorithm:

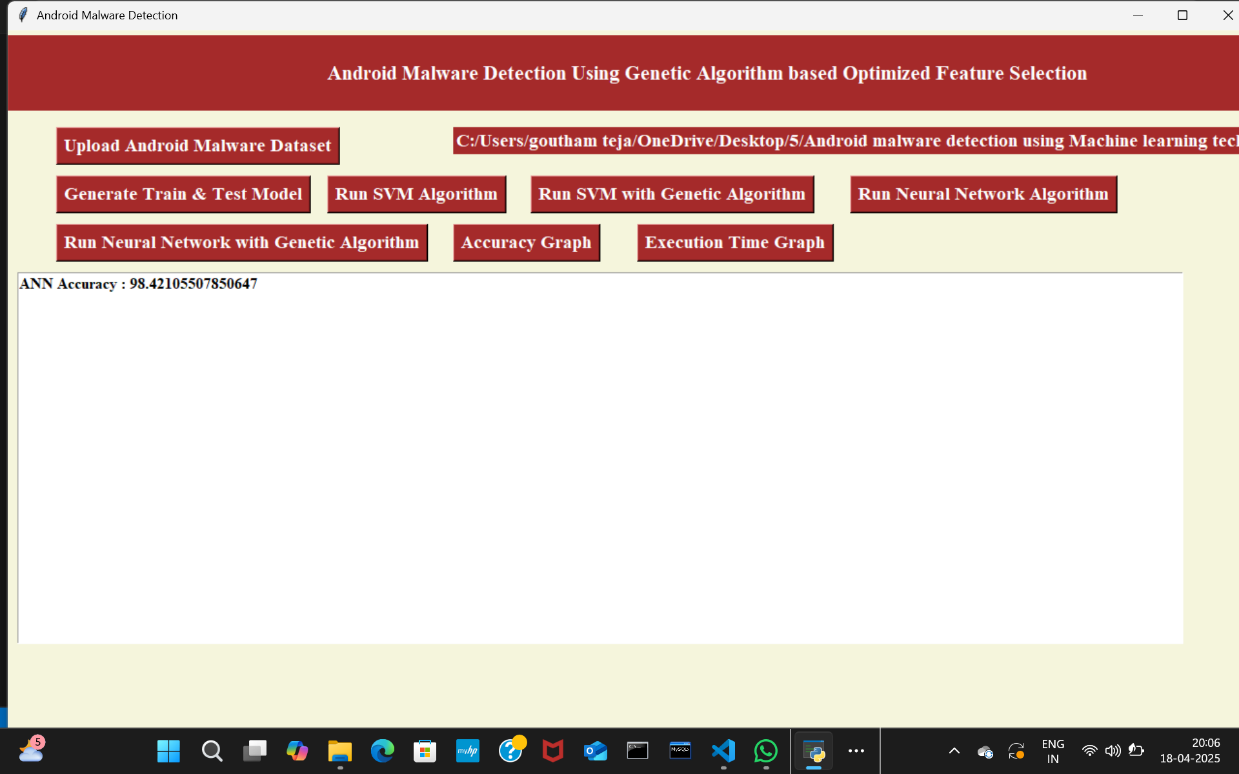
In below screen, we can observe the accuracy obtained by SVM Algorithm using Genetic Algorithm. Genetic algorithm reduces feature space whereas SVM trains faster.



**Figure 5.4:** Accuracy of SVM using GA.

#### ANN Accuracy:

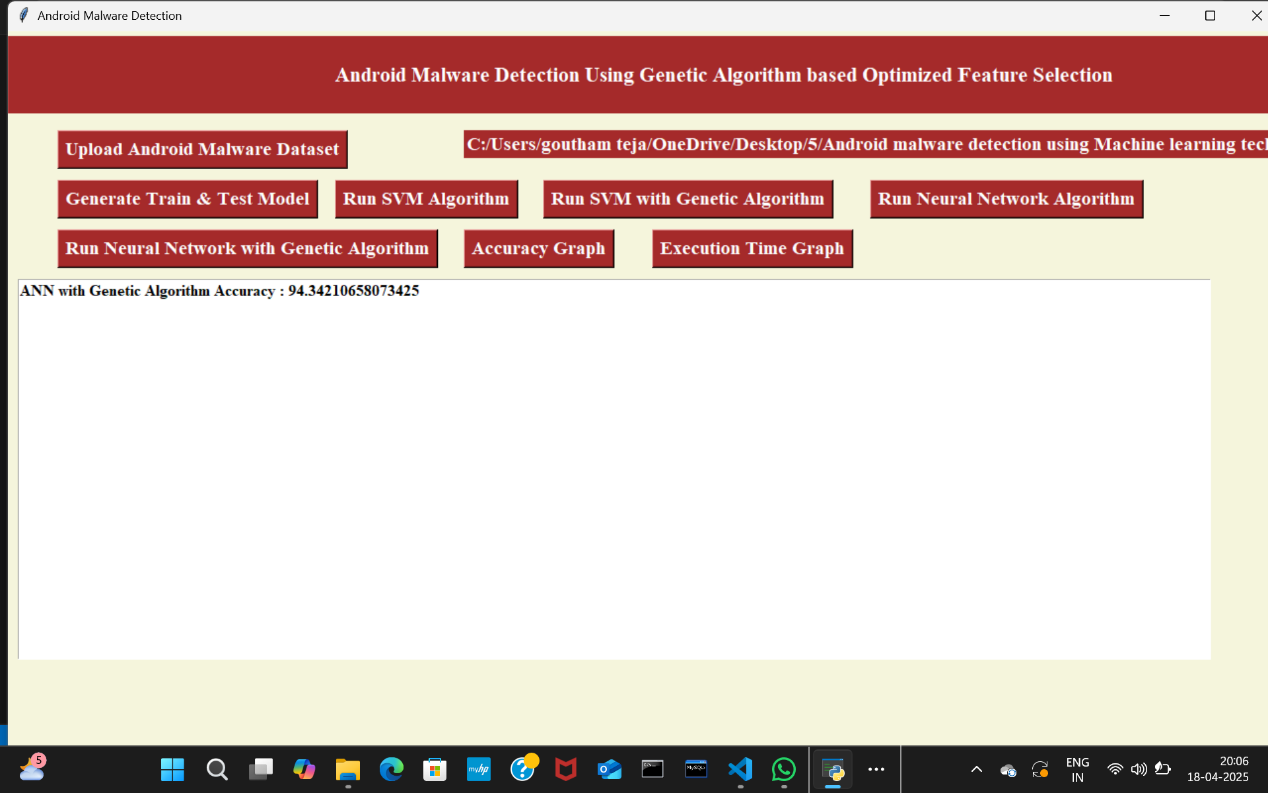
In below screen, we can see the accuracy of ANN Algorithm. ANN is used as a classifier to detect whether an Android app is malicious or benign.



**Figure 5.5:** Accuracy of ANN Algorithm.

#### ANN using Genetic Algorithm:

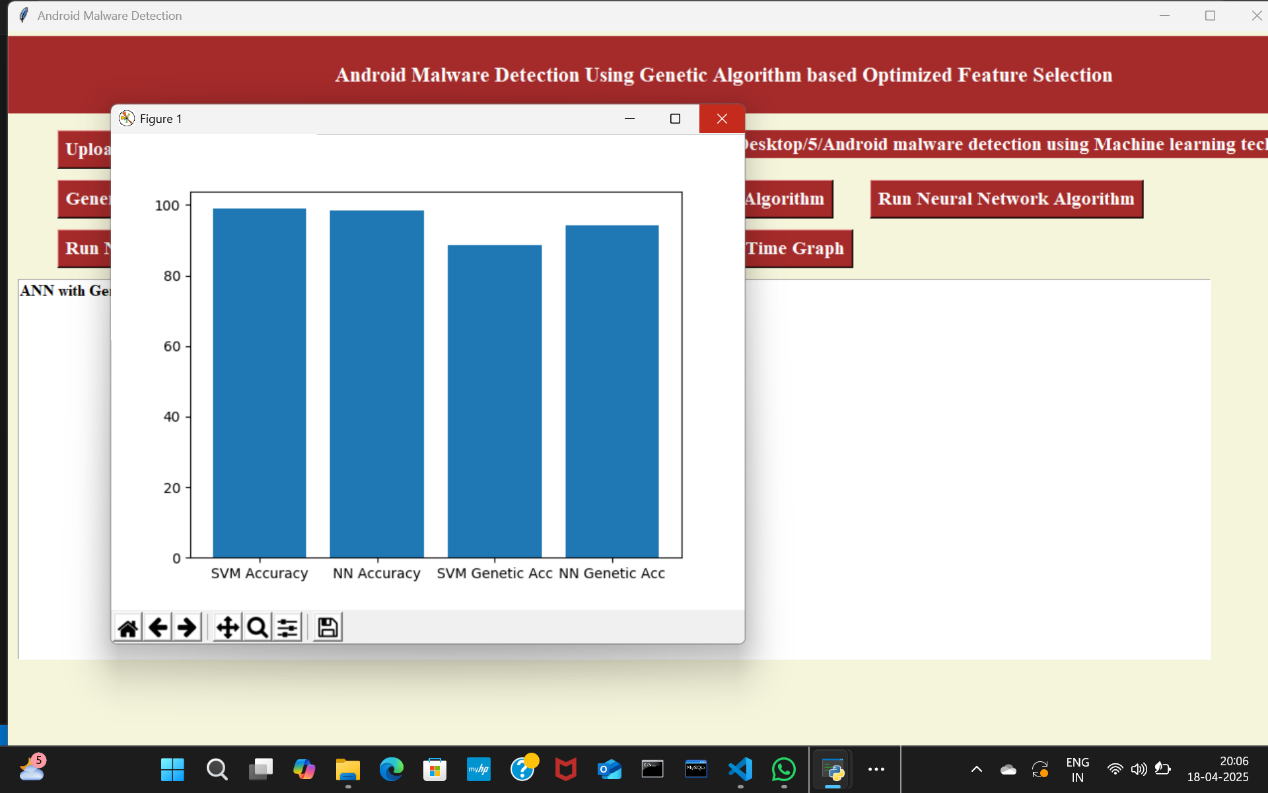
In below screen, we can observe the accuracy obtained by ANN using Genetic Algorithm. Genetic algorithm eliminates irrelevant features and ANN is better at handling complex malware behavior.



**Figure 5.6:** Accuracy of ANN using GA.

#### Accuracy Graph:

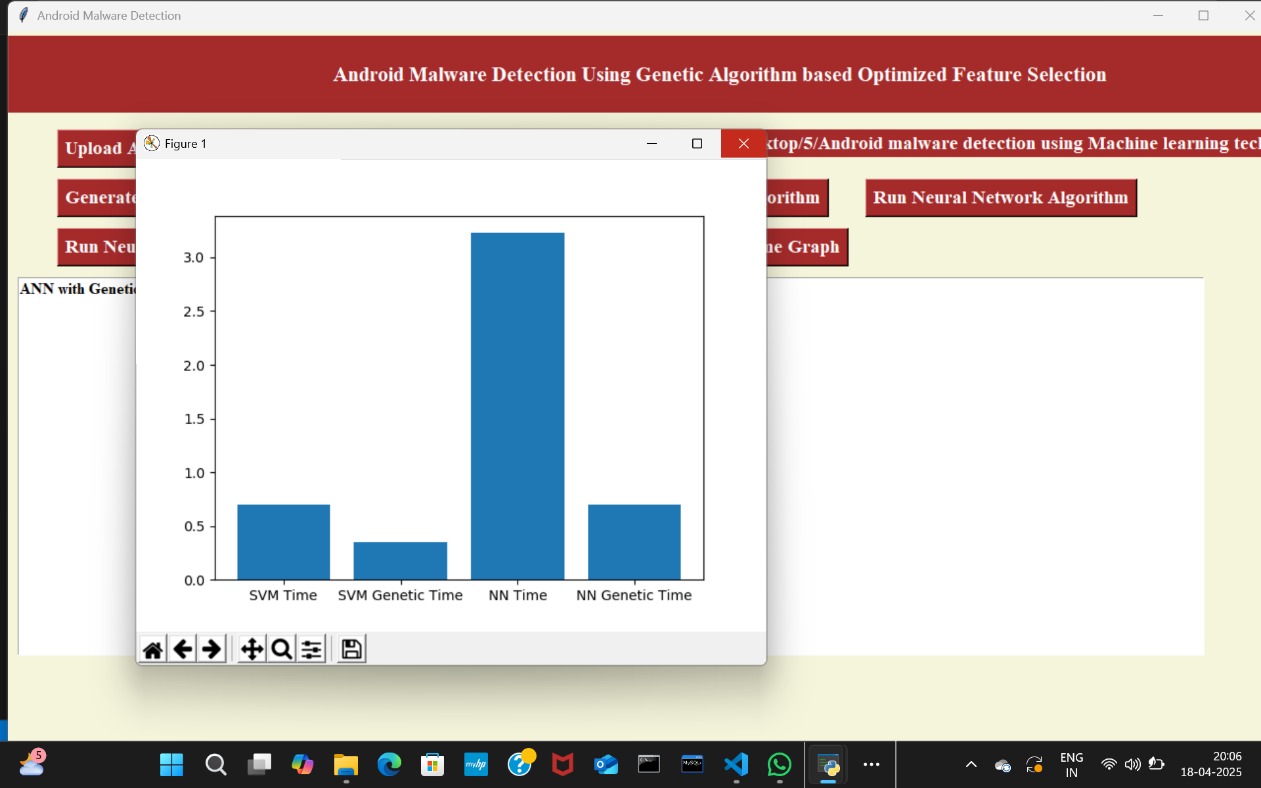
In below graph, we can observe the accuracy obtained by every algorithm used in this project.



**Figure 5.7:** Accuracy graph of every algorithm.

#### Time Line Graph:

In below graph, we can observe the time taken by every algorithm for accuracy.



**Figure 5.8:** Time graph of every algorithm.

## VALIDATION

The validation of this project primarily relies on extensive testing and well-defined test cases to ensure the accuracy and effectiveness of the inappropriate malware detection system. The testing process involves multiple stages, including dataset validation, model performance evaluation, and real-world testing. By implementing a structured validation approach, we can ensure that the system consistently delivers high accuracy in detecting inappropriate malware.

### INTRODUCTION

Android Apps are freely available on Google Play store, the official Android app store as well as third-party app stores for users to download. Due to its open-source nature and popularity, malware writers are increasingly focusing on developing malicious applications for Android operating system. In spite of various attempts by Google Play Store to protect against malicious apps, they still find their way to mass market and cause harm to users by misusing personal information related to their phone book, mail accounts, GPS location information and others for misuse by third parties or else take control of the phones remotely.

Android Apps are freely available on Google Play Store, the official Android app store as well as third-party app stores for users to download. Due to its open-source nature and popularity, malware writers are increasingly focusing on developing malicious applications for Android operating system. In spite of various attempts by Google Play store to protect against malicious apps, they still find their way to mass market and cause harm to users by misusing personal information related to their phone book, mail accounts, GPS location information and others for misuse by third parties or else take control of the phones remotely.

**6.2 TEST CASES**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Name** | **Test Steps** | | | **Test Case Status** | **Test Priority** |
| **Step** | **Expected** | **Actual** |
| 1 | Upload Android Malware Dataset | If the Android Malware Dataset may not upload | We cannot do any further operations | We can do further operations | High | High |
| 2 | Generate Train & Test Model | Without Generate the Train & Test model | We cannot do any further operations | We can do any further operations | High | High |
| 3 | Run SVM Algorithm | If the SVM Algorithm may not be run | We cannot do any further operations | We can run SVM Algorithm | High | High |
| 4 | Run the Neural Network Algorithm | If the Neural Network Algorithm may not be run | We cannot do any further operations | We can run Neural Network Algorithm | High | High |
| 5 | Run SVM with Genetic Algorithm | If the SVM with Genetic Algorithm may not be run | We cannot do any further operations | We can do any further operations | High | High |
| 6 | Run Neural Network with Genetic Algorithm | If the Neural Network with Genetic Algorithm may not be run | We cannot do any further operations | We can do any further operations | High | High |
| 7 | Accuracy Graph | Without saving the graph values of each algorithm | We cannot get Accuracy graph | We can get Accuracy graph | High | High |
| 8 | Execution Time Graph | without saving the Graph values of each algorithm | we cannot get execution Time  graph | we can get Execution Time  graph | High | High |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Table 6.2:** Test Case

## 7. CONCLUSION & FUTURE ASPECTS

In conclusion, the project has successfully achieved its objectives, showcasing significant progress and outcomes. The implementation and execution phases were meticulously planned and executed, leading to substantial improvements and insights. Looking ahead, the future aspects of the project hold immense potential. Future developments will focus on expanding the scope, integrating new technologies, and enhancing sustainability. These advancements will not only strengthen the existing framework but also open new avenues for growth and innovation, ensuring the project remains relevant and impactful in the long term. This strategic approach will drive continuous improvement and success.

### PROJECT CONCLUSION

As the number of threats posed to Android platforms is increasing day to day, spreading mainly through malicious applications or malwares, therefore it is very important to design a framework which can detect such malwares with accurate results. Where signature-based approach fails to detect new variants of malware posing zero-day threats, machine learning based approaches are being used. The proposed methodology attempts to make use of evolutionary Genetic Algorithm to get most optimized feature subset which can be used to train machine learning algorithms in most efficient way. From experimentations, it can be seen that a decent classification accuracy of more than 94% is maintained using Support Vector Machine and Neural Network classifiers while working on lower dimension feature-set, thereby reducing the training complexity of the classifiers. Further work can be enhanced using larger datasets for improved results and analysing the effect on other machine learning algorithms when used in conjunction with Genetic Algorithm.

### 7.2 FUTURE ASPECTS

Develop lightweight GA variants or hybrid optimization methods (e.g., GA + Particle Swarm Optimization) that are suitable for on-device (real-time) malware detection. Use GA to select optimal dynamic or hybrid features (like system calls, network activity) for real-time behavioral analysis.

Integrate GA with explainable AI (XAI) methods to identify which features (permissions, APIs, behaviors) are most indicative of malware. Use GA to select features that generalize well across datasets (Drebin, Andro Zoo, etc.) and incorporate transfer learning for broader adaptability.

Enhance GA-based feature selection to resist adversarial attacks by evolving robust and redundant feature sets. Employ GA to select features locally on each device in a federated setup, preserving privacy while building a robust global detection model. Develop open-source GA frameworks tailored for Android malware datasets with consistent evaluation protocols

### 8. REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,” in Proceedings 2014 Network and Distributed System Security Symposium, 2022.

[2] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, “Machine learning aided Android malware classification,” Comput. Electr. Eng., vol. 61, pp. 266–274, 2021.

[3] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, “Significant Permission

Identification for Machine-Learning-Based Android Malware Detection,” IEEE Trans. Ind. Informatics, vol. 14, no. 7, pp. 3216–3225, 2021.

[4] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention,” IEEE Trans. Dependable Secur. Comput., vol. 15, no. 1, pp. 83–97, 2020.

[5] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, “SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System,” IEEE Access, vol. 6, pp. 4321–4339, 2019.

[6] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, “A Multimodal Deep Learning Method for Android Malware Detection using Various Features,” vol. 6013, no. c, 2018.

[7] A. Martin, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, “Evolving Deep Neural Networks architectures for Android malware classification,” 2017 IEEE Congr. Evol. Comput. CEC 2017 - Proc., pp. 1659–1666, 2018.

[8] X. Su, D. Zhang, W. Li, and K. Zhao, “A Deep Learning Approach to Android Malware Feature Learning and Detection,” 2016 IEEE Trust., pp. 244–251, 2018.

[9] K. Zhao, D. Zhang, X. Su, and W. Li, “Fest: A Feature Extraction and Selection Tool for Android Malware Detection,” 2015 IEEE Symp. Compute. Commun., pp. 714–720, 2017.

[10] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, “A review on feature selection in mobile malware detection,” Digit. Investig., vol. 13, pp. 22–37, 2017.