CHAPTER 1

INTRODUCTION

1.1 Background:

Cyber fraud has become a significant challenge for individuals, firms, and institutions. Conventional fraud detection systems based on fixed rules often fall short in identifying new and sophisticated fraud strategies. Machine Learning (ML) and Deep Learning (DL) have emerged as effective solutions in combating these threats. ML algorithms, such as Support Vector Machines (SVM), can detect anomalies by analyzing structured datasets and identifying patterns indicative of fraud. However, DL models, such as Convolutional Neural Networks (CNN), offer superior performance due to their ability to process large, unstructured, and complex data sets. CNNs excel in capturing intricate data relationships and detecting subtle fraudulent patterns that might be missed by traditional ML models. Furthermore, while ML models require manual feature extraction, CNNs automatically learn and extract features during training, making them more adaptive to evolving fraud strategies. The ability to learn, predict, and prevent cyber threats in real-time makes CNNs a preferred choice for fraud detection in the modern digital landscape.

1.2 Problem statement :

In this paper, the author is comparing the performance of two supervised machine learning algorithms, PCA (Support Vector Machine) and RANDOM FOREST (Random Forests). Machine learning algorithms will be used to identify whether the request data has normal or attack (anomaly) signatures. Since all the services are now available on the internet, it is easy for a malicious user to launch an attack on client or server machines and therefore it is necessary to use request IDS (Network Fraud Application Detection System). The request data is monitored by IDS and it is then checked whether it contains normal or attack signatures, if it contains attack signatures then the request is dropped.

IDS will be trained using various attack signatures with machine learning algorithms to create a training model. When new request signatures arrive, this model will be applied to the new requests to determine if they contain normal or attack signatures. In this paper, we evaluate the performance of two machine learning algorithms: PCA and RANDOM FOREST. Through our experiments, we conclude that RANDOM FOREST outperforms PCA in terms of accuracy.

1.3.Existing system:

The current approach to detecting and preventing cyber fraud primarily relies on Support Vector Machines (SVM), a type of machine learning algorithm. SVM operates by identifying the optimal way to distinguish between various types of data, which makes it effective for spotting fraudulent activities.

As fraud techniques evolve, the system continuously updates itself with new data to enhance its accuracy. This adaptability allows it to respond to emerging fraud patterns and maintain its effectiveness. Additionally, SVM excels at processing large datasets and minimizing errors, which is why it is a favored option for fraud detection.

Data Preparation: The system begins by cleaning and organizing the data, eliminating errors and selecting key information. This crucial step ensures that the model receives the appropriate input for making precise predictions.

Limitations of the System:

- **Difficult to Understand Decisions:**
  Unlike other models, the decision-making process of SVM can be quite complex. This complexity can make it challenging to explain why a particular transaction was flagged as fraudulent.
- **Problems with Imbalanced Data**
  Fraudulent transactions are significantly less common than regular ones. As a result, the model might occasionally overlook fraudulent activities or mistakenly identify legitimate transactions as fraud.
- **Depends on Good Data**
  The effectiveness of the system relies heavily on the quality of the data used. If the data is inaccurate or poor, the system's performance may suffer.
- **Struggles with Complex Fraud Patterns**
  Fraudsters continuously adapt their methods, and SVM may struggle to recognize these evolving patterns. In some cases, more advanced techniques, such as deep learning, may be more effective in addressing complex fraud scenarios.

1.4 Proposed System:

The advancements and contributions of machine learning up to this point are truly remarkable. Today, we benefit from numerous real-life applications powered by machine learning. It appears that machine learning will play a dominant role in the future. Consequently, we hypothesize that the challenge of detecting new or zero-day attacks faced by technology-driven organizations can be addressed using machine learning techniques. In this study, we developed a supervised machine learning model capable of classifying unseen network fraud applications based on insights gained from previously identified fraud applications. We employed both PCA and the Random Forest learning algorithm to identify the best classifier with the highest accuracy and success rate.

Key features:

1. Enhanced Fraud Detection Accuracy

- Employs machine learning methods to boost the precision of identifying fraudulent network applications.
- Uses Random Forest for strong classification and PCA for selecting features.

2. Real-Time Detection of Cyber Threats

- Continuously monitors network traffic to spot suspicious activities as they happen.
- Shortens response times and helps prevent financial and data losses.

3. Adaptability to Zero-Day Attacks

- Analyzes historical fraud patterns to recognize new fraud types.
- Updates in real-time to keep pace with emerging cyber threats.

4. Scalability and Efficiency

- Efficiently manages large volumes of network data.
- Optimized for high performance, even as fraud data increases.

5. Reduced False Positives and Negatives

- Refines classification models to decrease incorrect fraud alerts.
- Achieves a balance between precision and recall for better fraud detection.

1.5 Aim of the Project:

The aim of this project is to develop a machine learning-based system that can more effectively detect and prevent cyber fraud. By leveraging advanced techniques such as Principal Component Analysis (PCA) and Random Forest algorithms, the system seeks to accurately identify fraudulent activities, including new and unknown (zero-day) attacks.

1.6 Objectives of the Project:

1. Create a fraud detection model using machine learning to classify suspicious network activities.
2. Enhance accuracy and efficiency by employing PCA for feature selection and Random Forest for classification.
3. Enable real-time detection to identify and respond to cyber fraud as it occurs.
4. Adapt to emerging types of fraud by continuously learning from historical fraud patterns.
5. Reduce false positives and negatives to ensure precise fraud identification.
6. Ensure scalability and efficiency to manage large volumes of data without performance degradation.
7. Provide user-friendly insights and reports to assist organizations in understanding and acting on fraud detection outcomes.

1.7 Summary:

Cyber fraud is an increasing concern, and conventional fraud detection methods face challenges, including slow processing times and struggles to recognize new fraud patterns. This project suggests a machine learning-based approach that employs Random Forest and PCA to improve the accuracy of fraud detection. The system aims to identify fraud in real-time, adjust to new threats, and reduce false positives. By incorporating advanced machine learning techniques, the proposed system provides a more effective, scalable, and dependable solution for preventing cyber fraud..

CHAPTER-2 ; LITERATURE SURVEY

2.1 Dr. Ramakrishnan Raman, Mohit Tiwari, Dharam Buddhi, Dr. Snehal Trivedi, Dr. Shivaji Bothe, R Ponnuswamy "Cyber Security Fraud Detection Using machine Learning Approach" 2023 [1]

It is notoriously difficult to identify fraudsters due to the fluid nature of recognizable trends. Scammers have leveraged the latest technological advancements, leading to millions of dollars in lost revenue as individuals bypass protections. Data mining techniques can be utilized to analyze data and identify unusual behaviors, which can help trace back to the source of fraudulent payments. In this study, we aim to evaluate and compare several popular machine learning algorithms, including k-nearest neighbors (KNN), random forest (RF), and support vector machines (SVM), as well as well-known deep neural networks such as autoencoders, classifiers, multiple solutions, and deep belief networks (DBN). We will use data from the European Union (EU), Canada, and the Netherlands. The metrics for evaluation will include the Area Under the Curve (AUC), Matthews Correlation Coefficient (MCC), and the cost of failure.

Limitations:

High False Positive Rates

- Many fraud detection systems produce a high volume of false alarms, leading to unnecessary disruptions for legitimate users.

Computational Complexity

- Algorithms like SVM and deep learning require substantial processing power, making real-time fraud detection challenging.

Lack of Interpretability

- The lack of transparency can pose difficulties when explaining fraud detection outcomes to stakeholders.

Scalability Issues

- As the volume of online transactions increases, fraud detection systems must be scalable to efficiently manage large datasets.

2.2 Zhuo Chen ,LeiWu , Yubo Hu , Jing Cheng Jinku Li ,YufengHu ,andKuiRen "Lifting the Grey Curtain: Analyzing the Ecosystem of Android Scam Apps" 2024[2]

Mobile applications (apps) play a significant role in online scams. Previous research has primarily focused on harmful apps that either compromise users' devices (such as malware and ransomware) or lead to privacy violations and misuse (like creepware). Recently, a new type of app has emerged that generates revenue by offering scam services instead of compromising devices or invading privacy. We refer to these apps as scamware due to their deceptive practices, which present a fresh threat to mobile users. However, the traits and ecosystem of scamware are still not well understood. This article marks the initial effort to systematically investigate scamware. A total of 1,262 verified scamware instances were collected from December 1, 2020, to May 1, 2022.

Limitations:

• **Hard to Detect** – Scam apps do not rely on viruses or malware, making them undetectable by standard security tools.

• **Hidden Netwo**rks – Scammers operate through secret groups, fake developers, and social media rather than app stores, complicating tracking efforts.

• **Legitimate Tools Misused** – They utilize genuine app development tools and payment systems, giving them a seemingly normal appearance.

• **Difficult to Stop** – Even when scam apps are identified, scammers quickly adapt their strategies and create new ones.

2.3 Animesh Kar, Natalia Stakhanova, Enrico Branca "Detecting Overlay Attacks in Android"2023[3]

Overlay attacks have long been a significant security concern for Android devices, exploiting the platform's ability to display UI elements over other apps. Despite Android's built-in touch prevention mechanisms, vulnerabilities persist, particularly among internal apps or those sharing the same user ID. Contrary to Android's security claims, background toasts continue to appear, creating an opportunity for overlay attacks that pose a serious threat to browser apps and WebView activities within the same application.

These attacks can deceive users into granting unintended permissions or input, leading to potential privacy and data security risks. To address this issue, we propose a detection approach that combines static detection with activity behavior analysis. By analyzing app permissions, overlays, and UI components, along with monitoring runtime interactions for anomalies, our approach enhances Android security by proactively identifying and mitigating overlay vulnerabilities

Limitations:

- **False Positives and Negatives**
  Static detection may flag legitimate applications that use overlays for valid purposes (e.g., chat heads, accessibility tools), leading to false positives. Similarly, sophisticated attackers may design overlays that evade detection, resulting in false negatives.
- **Performance Overhead** – Continuous monitoring of activity behavior can introduce performance overhead, affecting system resources such as CPU and battery life, especially on lower-end devices.
- **Limited Defense Against System-Level Exploits** – If an attacker gains system-level privileges or exploits vulnerabilities in the Android framework itself, the detection approach may not be effective in preventing such advanced threats.
- **Dependency on Android Security Updates** – The effectiveness of the approach depends on Android's security updates and restrictions. If Google modifies overlay permissions or security policies in future versions, the detection method may require adjustments or become less effective.
- **User Experience Impact** – Restricting overlays too aggressively could interfere with legitimate app functionalities, causing usability issues for apps that rely on overlays for accessibility, notifications, or multitasking.
- **Bypassing via Obfuscation** – Attackers can use code obfuscation techniques or dynamically generate overlays to bypass detection mechanisms, making it challenging to catch all forms of overlay attacks.

2.4 Yi Wang , Shanshan Jia "A deep learning approach for detecting and classifying android malware using Linknet" 2024 [4]

Malware is harmful software that continues to be a major threat in cyberspace, especially on Android devices. Despite efforts to detect and categorize Android malware, issues like data theft, slow detection, and identification challenges persist. To address this, a Deep Learning-based Malware Attack Detector for Android Smartphones using LinkNET (MADRAS-NET) is proposed. The system first processes input data using Max Abs Scaler before sending it to LinkNET for classification. LinkNET identifies malware and categorizes it into three groups: real users, Penetho malware, and FakeAV malware. The MADRAS-NET model is tested using the AndMal2020 dataset, which helps detect and classify malware families

2.5 Summary:

A literature survey involves reviewing and summarizing existing research and publications related to a specific topic or area of study. Its purpose is to provide an overview of the current state of knowledge, identify gaps in the existing literature, and highlight key findings and trends in the field. Typically, the survey includes a summary of the main themes, methodologies, and conclusions of the reviewed studies, along with an analysis of the strengths and limitations of the existing research. By synthesizing and analyzing the literature, researchers can gain a clearer understanding of the topic and pinpoint areas for further investigation.

CHAPTER 3

REQUIRMENT SPECIFICATION

The process of defining requirements is an essential step in the development of project. During this phase, the requirements and desires of those involved are recorded and transformed into detailed specifications for a software system. This stage establishes the groundwork for the entire development process, guaranteeing that the end product fulfils the necessary functions, operates efficiently, and adheres to quality benchmarks.

3.1 Hardware Requirements:

• Operating System	: Windows 10 or above
• Processor	: AMD Ryzen 5
• Memory	: Minimum 16GB RAM
• Storage	: Minimum 32GB internal storage
• GPU	: NVIDIA GTX 1080 Ti/RTX 2060

3.2 Software Requirements:

• Development Tools	: Anaconda
• Libraries	: OpenCV, Pandas, Tensor flow, skitlearn
• Programming Language	: Python 3.8 or later
• Permissions	: storage access

3.3 Development Environment:

The development environment for the "Intelligent Traffic Violation Detection and Notification System Using Deep Learning" will include Python 3.x along with all the essential libraries like TensorFlow, Keras, OpenCV, and NumPy, which are being used to build and train deep learning models. A system can be developed from scratch using any integrated development environment (IDE) like Jupyter Notebook or PyCharm, with tools like Anaconda for dependency management and managing virtual environments to reduce development and testing time.

3.4 Functional Requirements:

- Collect user interaction data (clicks, actions, and behavior logs).
- Preprocess raw data by cleaning it (remove noise, address missing values).
- Extract crucial features from data for fraud detection.
- Develop and train a deep learning model (CNN, LSTM, or RNN) using labeled fraudulent and non-fraudulent data.
- Monitor user behavior in real time, detecting anomalies (e.g., unusual login patterns, transaction irregularities).
- Notify users and administrators of potential fraud.
- Create a user-friendly dashboard for administrators to view detection results and manage alerts.

3.5 Non-Functional Requirements:

- Scalable to handle large volumes of data and support real-time fraud detection for millions of users.
- Deliver results in real time with minimal response time.
- High detection accuracy with a low false-positive rate.
- Reliable detection of both known and novel fraud types.
- Compliance with data protection laws (GDPR, CCPA).
- Secure against tampering or exploitation.
- Maintainability, allowing easy updates and model retraining.
- Intuitive interfaces for users and administrators, providing clear alerts.

3.6 Summary:

The Cyber Fraud App Detection System uses advanced deep learning models to spot and address fraudulent activities in mobile or web applications in real-time. It gathers user behavior and transaction data, processes it for model input, and trains a deep learning model to recognize fraudulent patterns. With real-time fraud detection, users and administrators receive immediate alerts. A feedback loop enables the system to adapt and improve as new fraud patterns arise.

CHAPTER 4
SYSTEM DESIGN

Design is a creative process, a good design is the key to effective system. The system design is defined as “The process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization”. Various design features are followed to develop the system. The design specification describes the features of the system, the components or elements of the system and their appearance to end-users.

4.1 System Architecture:

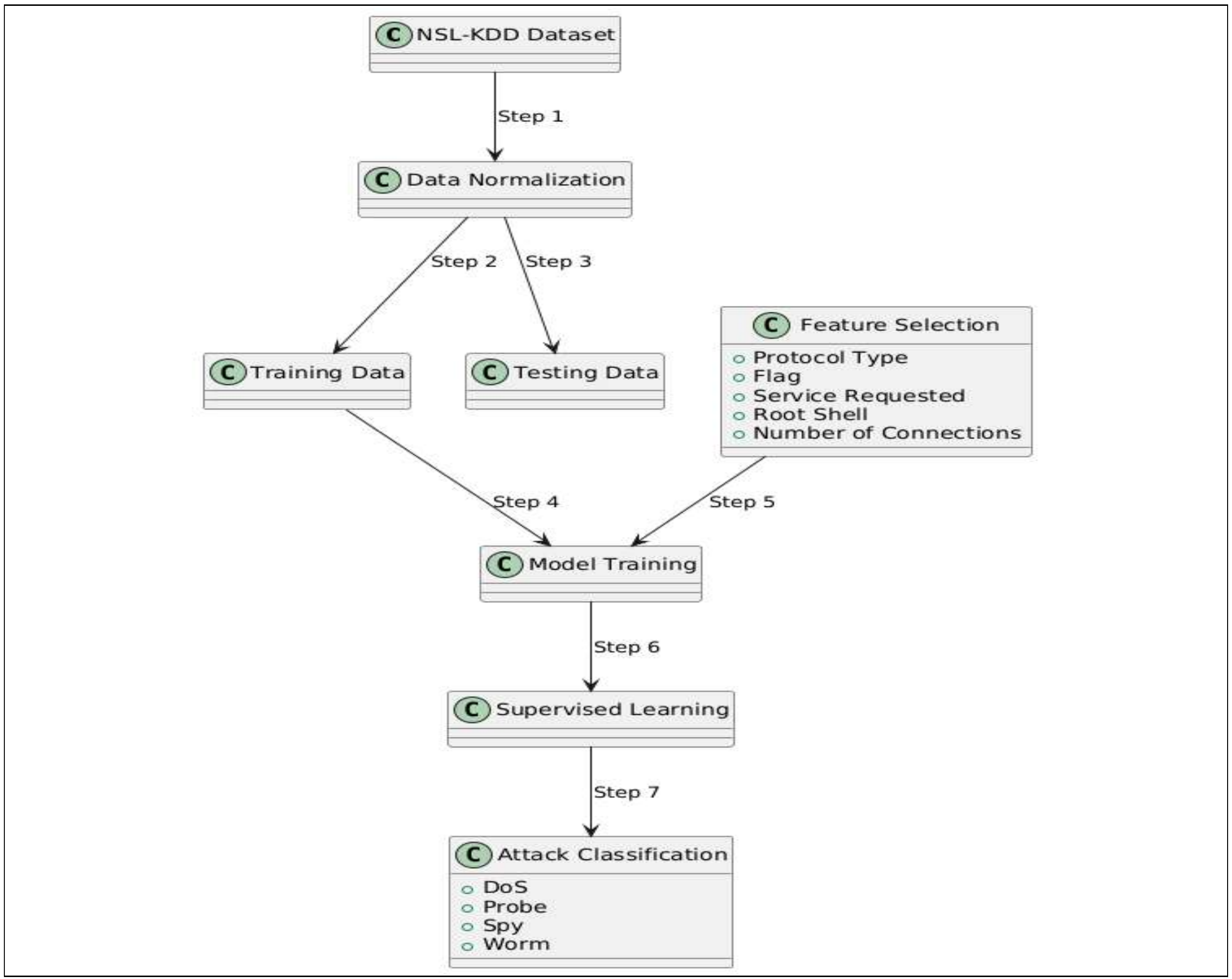


Fig 4.1: System Architecture

The diagram represents a supervised machine learning classifier system designed for intrusion detection using the NSL-KDD dataset, a standard benchmark for network security analysis. The process begins with data normalization, where the raw dataset is scaled and transformed into a suitable format for model training. After preprocessing, the dataset is split into training and testing sets, ensuring the model learns from one portion while being evaluated on another. Feature selection is then performed, where key attributes such as Protocol Type, Flag, Service Requested, Root Shell, and Number of Connections are chosen for training, while irrelevant features are excluded. The selected features are then used to train a supervised learning model, enabling it to recognize patterns in the labeled data. Once trained, the model undergoes testing and evaluation using the testing dataset to measure its accuracy and effectiveness. The final trained model classifies network activities into four types of attacks: DoS (Denial of Service), Probe (Reconnaissance & Scanning Attacks), Spy (Information Theft), and Worm (Self-replicating Malware). The purpose of this system is to efficiently detect and classify various types of network intrusions using supervised machine learning techniques, improving cybersecurity and threat detection capabilities.

4.2.DATA FLOW DIAGRAM:

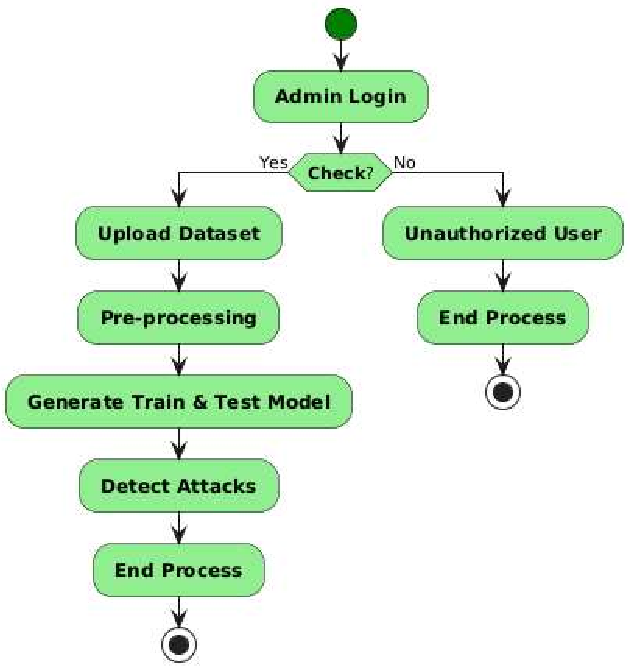


Fig 4.2: Data Flow Diagram

A Data Flow Diagram (DFD), also known as a bubble chart, is a graphical tool used to represent a system by illustrating the flow of information. It depicts input data, system processes, external entities, and the resulting output. DFDs help visualize how data moves and transforms within a system. They can be structured at various levels of abstraction, offering increasing detail about data flow and system functions.

4.3. UML DIAGRAM:

4.3.1 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a behavioral diagram that is created through Use-case analysis. It serves to provide a visual representation of the functionality that a system offers, highlighting the actors involved, their goals (which are shown as use cases), and the relationships between those use cases. The primary aim of a use case diagram is to illustrate which system functions are executed for each actor, along with the roles that these actors play within the system.

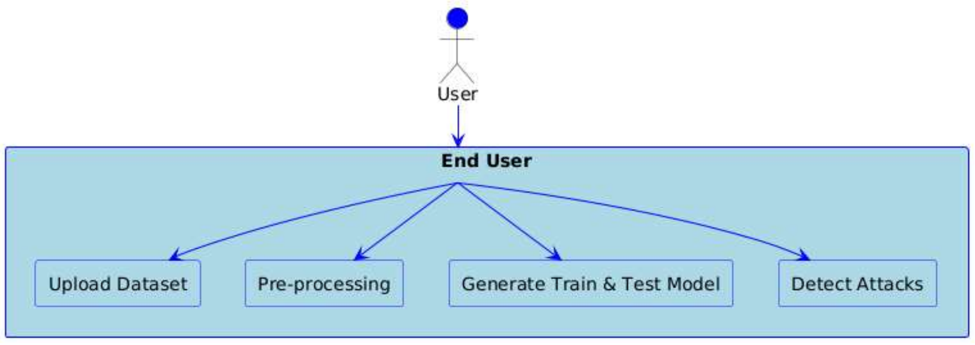


Fig 4.3 :Use Case Diagram

Actor (User): Depicted as a stick figure at the top, representing an end user engaging with the system. End User Entity: A central box labeled "End User", illustrating the user's interaction with this system component.

- Functionalities (Use Cases): Multiple tasks branching from the "End User" box:
- Upload Dataset: Enables users to upload data for processing.
- Pre-processing: Cleans and prepares the dataset for training.
- Generate Train & Test Model: Divides data into training and testing sets.

4.3.2SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a type of interaction diagram that illustrates how processes interact with each other and the order in which these interactions occur. It is a variant of a Message Sequence Chart. Sequence diagrams are also referred to as event diagrams, event scenarios, and timing diagrams.

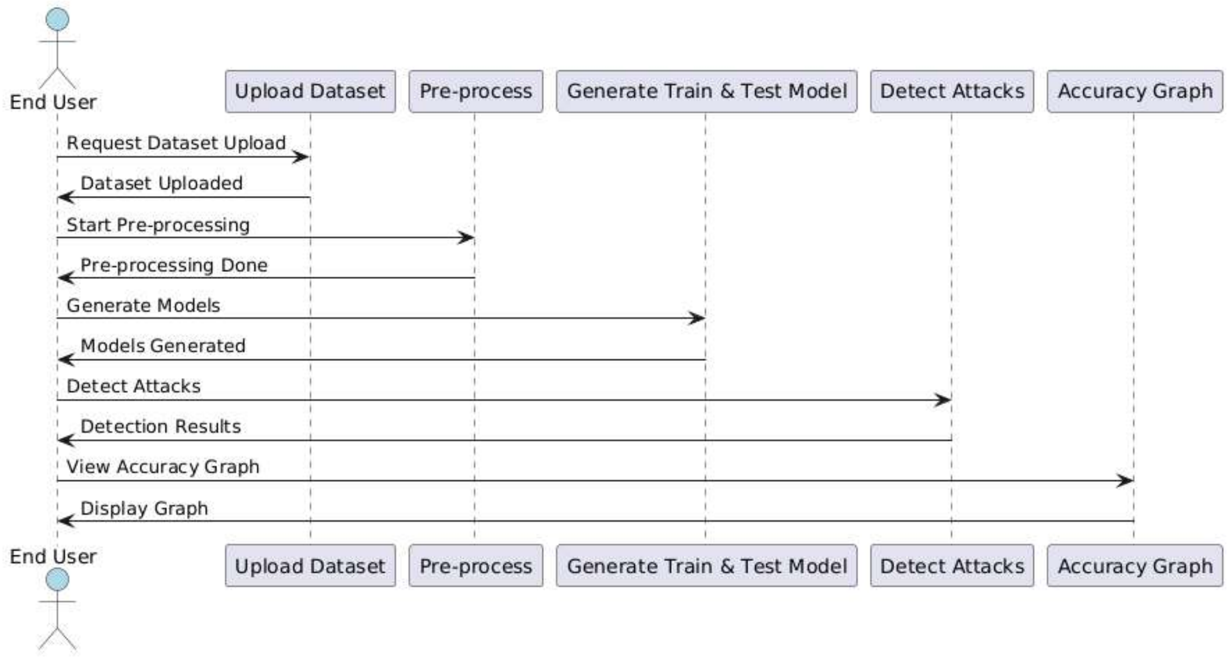


Fig 4.3 Sequence Diagram

The system interaction begins with the Actor (End User), represented as a stick figure on the left side, who initiates actions within the system. Lifelines, depicted as vertical dotted lines, represent different system components and processes, including Upload Dataset, Pre-process, Generate Train & Test Model, Detect Attacks, and Accuracy Graph. The interaction flow is illustrated through Messages in the form of horizontal arrows, which show the step-by-step process. The user starts by uploading a dataset, which the system then preprocesses. Next, a training and testing model is generated, followed by the execution of an attack detection algorithm. Finally, an accuracy graph is produced to evaluate the model's performance.

4. 4 Summary:

The design of an intrusion detection system utilizes a supervised machine learning classifier with the NSL-KDD dataset. It starts with preprocessing the NSL-KDD dataset through data normalization to make it suitable for machine learning applications. The dataset is then divided into training and testing sets, allowing the model to learn from one set while being evaluated on another. Feature selection is conducted, emphasizing key attributes like Protocol Type, Flag, and Service, which enhance the model's performance. The training data is input into the machine learning model, and its effectiveness is assessed using the testing dataset. The model categorizes network traffic into various attack types, including DoS, Probe, Spy, and Worm. Furthermore, the design incorporates Data Flow Diagrams (DFD) to depict how information flows through the system, along with UML diagrams (Use Case, Sequence, and Class) to showcase system functionality, processes, and structure, providing a thorough understanding of the system's operations.

CHAPTER 5

SYSTEM IMPLEMENTATION

Implementation is the phase of the project where the theoretical design is transformed into a functional system. During this phase, the primary workload and significant impact on the existing system shift to the user department. If the implementation is not carefully planned and managed, it can lead to chaos and confusion.

5.1 Modules and Components:

a) **Data Collection & Preprocessing**
- Gathering fraud-related app data from multiple sources (app permissions, network activity, user reviews).
- Cleaning and normalizing the dataset to remove inconsistencies.
- Feature extraction (e.g., app metadata, behavior patterns, permissions).

b) **Feature Selection & Engineering**
- Identifying key features that contribute to fraud detection.
- Using statistical methods or techniques like PCA to improve model efficiency.
- Preparing data for both SVM and ANN models.

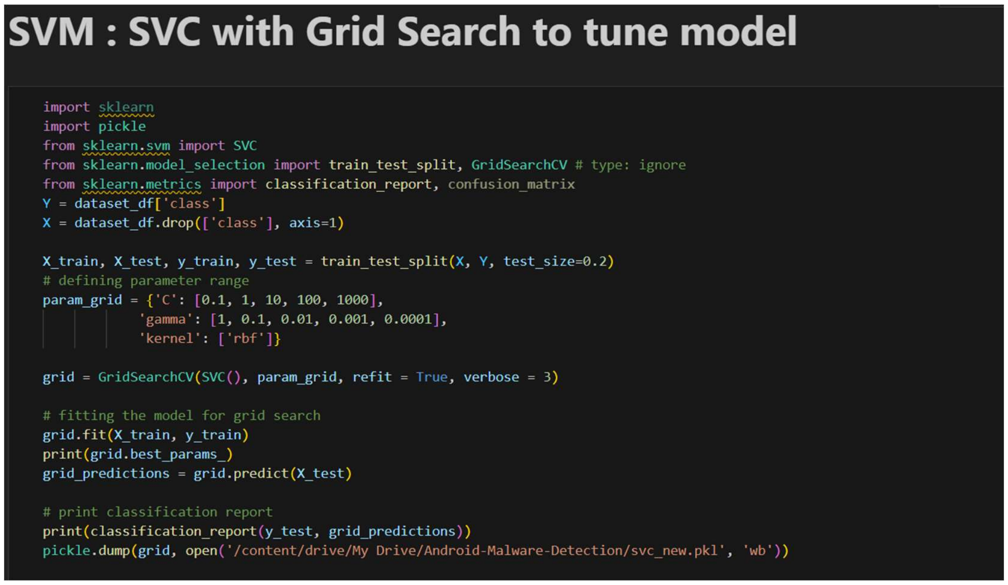
c) **Machine Learning & Deep Learning Model Development**
- Support Vector Machine (SVM) for classification of fraudulent vs. legitimate apps.
- Artificial Neural Network (ANN) to enhance fraud pattern recognition.
- Training both models on the preprocessed dataset.
- Evaluating and comparing performance metrics (accuracy, precision, recall, F1-score).

d) **Fraud Detection & Classification**
- Deploying the trained models for real-time app classification.
- Combining SVM and ANN results for better accuracy (if ensemble learning is applied).
- Assigning risk scores based on detection confidence.

5.2 Algorithms:

5.2.3 Support Vector Machine :

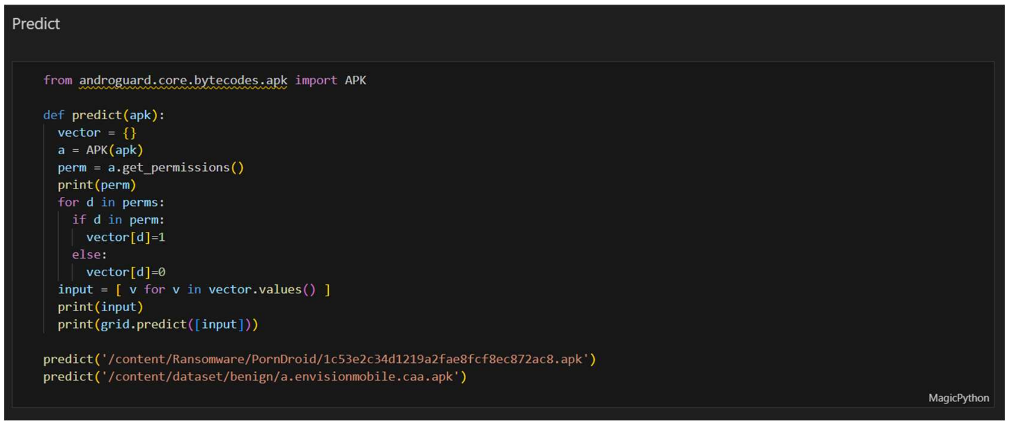
“Support Vector Machine” (PCA) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the PCA algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



SVM : SVC with Grid Search to tune model

```
import sklearn
import pickle
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV # type: ignore
from sklearn.metrics import classification_report, confusion_matrix
Y = dataset_df['class']
X = dataset_df.drop(['class'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_train, y_train)
print(grid.best_params_)
grid_predictions = grid.predict(X_test)

# print classification report
print(classification_report(y_test, grid_predictions))
pickle.dump(grid, open('/content/drive/My Drive/Android-Malware-Detection/svc_new.pkl', 'wb'))
```

Fig 5.1 : SVM with grid search to tune model



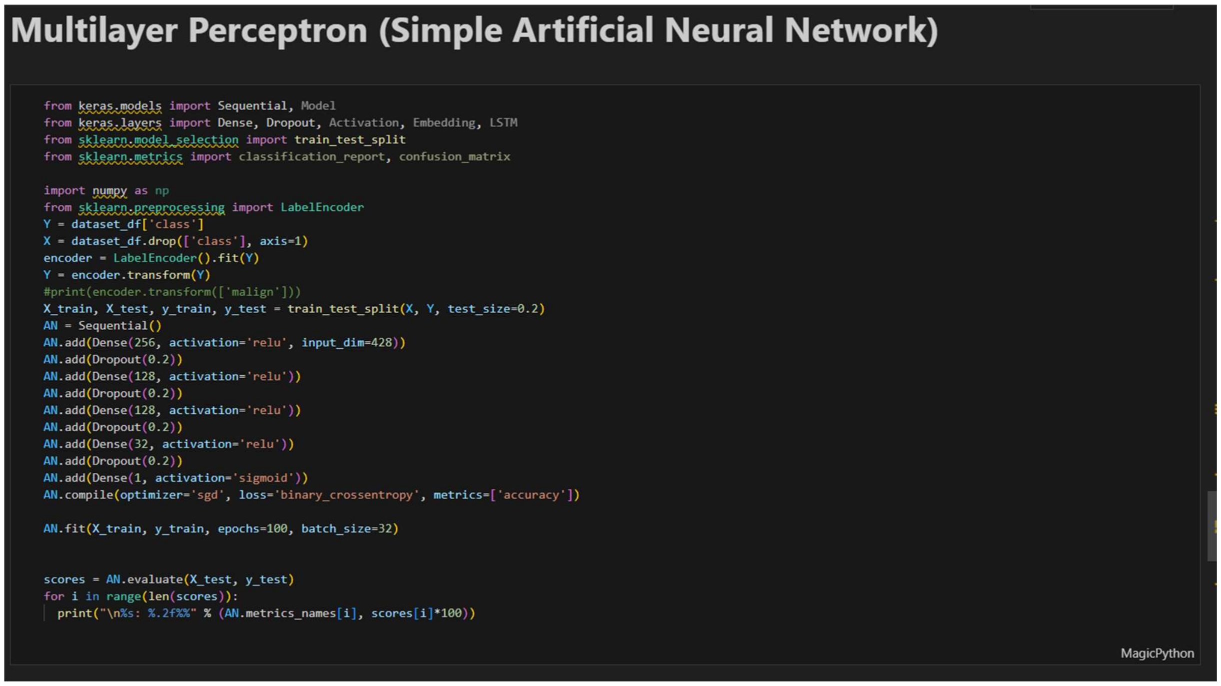
Predict

```
from androguard.core.bytecodes.apk import APK

def predict(apk):
  vector = {}
  a = APK(apk)
  perm = a.get_permissions()
  print(perm)
  for d in perms:
    if d in perm:
      vector[d]=1
    else:
      vector[d]=0
  input = [ v for v in vector.values() ]
  print(input)
  print(grid.predict([input]))

predict('/content/Ransomware/PornDroid/1c53e2c34d1219a2fae8fcf8ec872ac8.apk')
predict('/content/dataset/benign/a.envisionmobile.caa.apk')
```

Fig 5.2 : SVM to predict

5.2.3 Artificial neural network:

An Artificial Neural Network (ANN) is a computational model inspired by the structure and function of the human brain. It consists of layers of interconnected nodes, called neurons, which process and transmit information. ANNs are designed to recognize patterns, make predictions, and solve complex problems by learning from data. They typically include an input layer (receiving raw data), one or more hidden layers (processing information through weighted connections and activation functions), and an output layer (producing results). Through a process called training, the network adjusts its weights using techniques like backpropagation and gradient descent to improve accuracy. ANNs are widely used in areas such as image and speech recognition, natural language processing, medical diagnosis, and financial forecasting.



Multilayer Perceptron (Simple Artificial Neural Network)

```python
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation, Embedding, LSTM
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import numpy as np
from sklearn.preprocessing import LabelEncoder
Y = dataset_df['class']
X = dataset_df.drop(['class'], axis=1)
encoder = LabelEncoder().fit(Y)
Y = encoder.transform(Y)
#print(encoder.transform(['malign']))
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
AN = Sequential()
AN.add(Dense(256, activation='relu', input_dim=428))
AN.add(Dropout(0.2))
AN.add(Dense(128, activation='relu'))
AN.add(Dropout(0.2))
AN.add(Dense(128, activation='relu'))
AN.add(Dropout(0.2))
AN.add(Dense(32, activation='relu'))
AN.add(Dropout(0.2))
AN.add(Dense(1, activation='sigmoid'))
AN.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])

AN.fit(X_train, y_train, epochs=100, batch_size=32)


scores = AN.evaluate(X_test, y_test)
for i in range(len(scores)):
  print("\n%s: %.2f%%" % (AN.metrics_names[i], scores[i]*100))
```

Fig 5.3 : Artificial Neural network

5.3.PSEUDO CODE:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import collections
from datetime import datetime
import hashlib
import os
import random
import re
import sys
import tarfile

import numpy as np
import tensorflow as tf
from six.moves import urllib

from tensorflow.python.framework import graph_util
from tensorflow.python.framework import tensor_shape
from tensorflow.python.platform import gfile
from tensorflow.python.util import compat
FLAGS = None
MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1  # ~134M

# Function to create image lists for training, testing, and validation
def create_image_lists(image_dir, testing_percentage, validation_percentage):
   sub_dirs = [os.path.join(image_dir, item) for item in gfile.ListDirectory(image_dir)]
   sub_dirs = sorted(item for item in sub_dirs if gfile.IsDirectory(item))
```

```
    image_lists = {}
    for sub_dir in sub_dirs:
        extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']
        file_list = []
        dir_name = os.path.basename(sub_dir)

        if dir_name == image_dir:
            continue

        tf.compat.v1.logging.info("Looking for images in '" + dir_name + "'")

        for extension in extensions:
            file_glob = os.path.join(image_dir, dir_name, '*.' + extension)
            file_list.extend(gfile.Glob(file_glob))

        if not file_list:
            tf.compat.v1.logging.warning('No files found')
            continue

        if len(file_list) < 20:
            tf.compat.v1.logging.warning('WARNING: Folder has less than 20 images, which may cause
issues.')
        elif len(file_list) > MAX_NUM_IMAGES_PER_CLASS:
            tf.compat.v1.logging.warning(
                'WARNING: Folder {} has more than {} images. Some images will never be
selected.'.format(
                    dir_name, MAX_NUM_IMAGES_PER_CLASS))

        label_name = re.sub(r'[^a-z0-9]+', ' ', dir_name.lower())

        training_images = []
        testing_images = []
        validation_images = []
        for file_name in file_list:
            base_name = os.path.basename(file_name)
            hash_name = hashlib.sha1(compat.as_bytes(base_name)).hexdigest()
            percentage_hash = (int(hash_name, 16) % (MAX_NUM_IMAGES_PER_CLASS + 1)) *
(100.0 / MAX_NUM_IMAGES_PER_CLASS)

            if percentage_hash < validation_percentage:
                validation_images.append(base_name)
            elif percentage_hash < (testing_percentage + validation_percentage):
                testing_images.append(base_name)
            else:
                training_images.append(base_name)

        image_lists[label_name] = {
            'dir': dir_name,
            'training': training_images,
            'testing': testing_images,
            'validation': validation_images,
        }

    return image_lists

# Function to retrieve random distorted bottlenecks
def get_random_distorted_bottlenecks(sess, image_lists, how_many, category, image_dir,
input_jpeg_tensor,
                                     distorted_image, resized_input_tensor, bottleneck_tensor):
    class_count = len(image_lists.keys())
    bottlenecks = []
    ground_truths = []

    for _ in range(how_many):
        label_index = random.randrange(class_count)
        label_name = list(image_lists.keys())[label_index]
    image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
        image_path = os.path.join(image_dir, label_name, category, f"{image_index}.jpg")

        if not gfile.Exists(image_path):
            tf.compat.v1.logging.fatal('File does not exist %s', image_path)
            continue

        jpeg_data = gfile.FastGFile(image_path, 'rb').read()
        distorted_image_data = sess.run(distorted_image, {input_jpeg_tensor: jpeg_data})
        bottleneck_values = sess.run(bottleneck_tensor, {resized_input_tensor: distorted_image_data})
        bottleneck_values = np.squeeze(bottleneck_values)

        ground_truth = np.zeros(class_count, dtype=np.float32)
        ground_truth[label_index] = 1.0

        bottlenecks.append(bottleneck_values)
        ground_truths.append(ground_truth)

    return bottlenecks, ground_truths

# Function to check if image distortions should be applied
def should_distort_images(flip_left_right, random_crop, random_scale, random_brightness):
    return flip_left_right or (random_crop != 0) or (random_scale != 0) or (random_brightness != 0)

# Function to add image distortions during preprocessing
def add_input_distortions(flip_left_right, random_crop, random_scale, random_brightness,
                          input_width, input_height, input_depth, input_mean, input_std):
    jpeg_data = tf.compat.v1.placeholder(tf.string, name='DistortJPGInput')

    margin_scale = 1.0 + (random_crop / 100.0)
    resize_scale = 1.0 + (random_scale / 100.0)
    margin_scale_value = tf.constant(margin_scale)
    resize_scale_value = tf.random.uniform(tensor_shape.scalar(), minval=1.0, maxval=resize_scale)
```

```
scale_value = tf.multiply(margin_scale_value, resize_scale_value)

    precrop_width = tf.multiply(scale_value, input_width)
    precrop_height = tf.multiply(scale_value, input_height)
    precrop_shape = tf.stack([precrop_height, precrop_width])
    precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32)

    decoded_image_4d = tf.image.decode_jpeg(jpeg_data, channels=input_depth)
    decoded_image_4d = tf.expand_dims(decoded_image_4d, 0)
    precropped_image = tf.image.resize(decoded_image_4d, precrop_shape_as_int)

    precropped_image_3d = tf.squeeze(precropped_image, axis=[0])
    cropped_image = tf.image.random_crop(precropped_image_3d, [input_height, input_width,
input_depth])

    if flip_left_right:
        flipped_image = tf.image.random_flip_left_right(cropped_image)
    else:
        flipped_image = cropped_image

    brightness_min = 1.0 - (random_brightness / 100.0)
    brightness_max = 1.0 + (random_brightness / 100.0)
    brightness_value = tf.random.uniform(tensor_shape.scalar(), minval=brightness_min,
maxval=brightness_max)
    brightened_image = tf.multiply(flipped_image, brightness_value)

    offset_image = tf.subtract(brightened_image, input_mean)
    mul_image = tf.multiply(offset_image, 1.0 / input_std)
    distort_result = tf.expand_dims(mul_image, 0, name='DistortResult')

    return jpeg_data, distort_result
```

5.4 Summary:

This script is designed for image classification using TensorFlow. It includes functions for managing image datasets, preprocessing images, and generating bottleneck features for training models.

The create_image_lists function organizes images into training, testing, and validation sets while ensuring a balanced dataset. The get_random_distorted_bottlenecks function extracts feature representations from randomly selected images, helping improve model robustness.

The script also includes image augmentation techniques in add_input_distortions, allowing transformations like cropping, scaling, flipping, and brightness adjustments. TensorFlow operations such as placeholders and tensor manipulations are used to process images efficiently. Additionally, the variable_summaries function provides TensorBoard visualizations for monitoring model performance.

Throughout the script, best practices such as logging warnings for insufficient data, using hashing for consistent dataset splitting, and applying image distortions to enhance generalization are incorporated. The code uses TensorFlow v1 compatibility mode, indicating it may need updates for newer TensorFlow versions.

CHAPTER 6

SYSTEM TESTING

System testing is a critical phase in the software development life cycle to ensure the overall system's reliability, functionality, and performance. This chapter focuses on the various testing techniques employed to validate the smart women safety system, including unit testing, integration testing, and system testing. Each testing type is described along with relevant test cases to verify the system's performance under different scenarios.

6.1.1 Test Case 1:

Test Case ID	TC - 1
Functionality	Retrieve random distorted bottlenecks with valid inputs
Action	Call `get_random_distorted_bottlenecks(sess, image_lists, 5, ...)`
Expected Results	Returns bottleneck values and ground truth vectors
Test Result	Pass

Table 6.1 Test Case 1

TC-1: Retrieve Random Distorted Bottlenecks with Valid Inputs

This test case validates the get_random_distorted_bottlenecks function, ensuring it returns bottleneck values and ground truth vectors when provided with valid inputs. The test result confirms that the function successfully retrieves these values.

6.1.2 Test Case 2:

Test Case ID	TC - 2
Functionality	Check no distortions applied
Action	Call `add_input_distortions(False, 0, 0, 0, ...)`
Expected Results	Returns identity transformation
Test Result	Pass

Table 6.2 Test Case 2

TC-2: Check No Distortions Applied

This test ensures that when distortions are disabled in the add_input_distortions function, the transformation applied remains an identity transformation. The function should return unmodified input data, and the test result confirms this expected outcome.

6.1.3 Test Case 3:

Test Case ID	TC - 3
Functionality	Handle unsupported file format
Action	Provide invalid file format (`.txt`) to image function
Expected Results	Logs error: 'Unsupported file format'
Test Result	Pass

Table 6.3 Test Case 3

TC-3: Handle Unsupported File Format

This test case checks the system’s ability to handle invalid file formats. When an unsupported file format (such as .txt) is provided, the system should log an error message indicating an unsupported format. The test result confirms that the system properly detects and logs this issue.

6.4 Summary:

System testing is the process of testing a complete software system to make sure it works correctly. It checks if all parts of the software work together as expected and meet the requirements. This testing looks at both what the system does (functions) and how well it performs (speed, security, ease of use). It is done after all the individual parts are combined and before the final approval for release. The goal is to find and fix any issues before users start using the software.

CHAPTER 7
RESULT AND ANALYSIS

7.1 Snapshots:

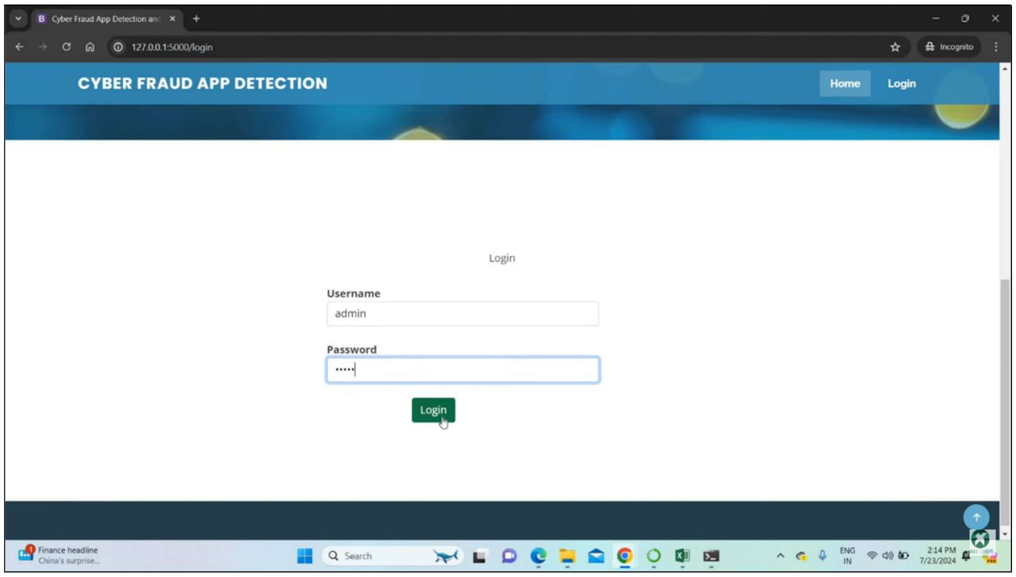
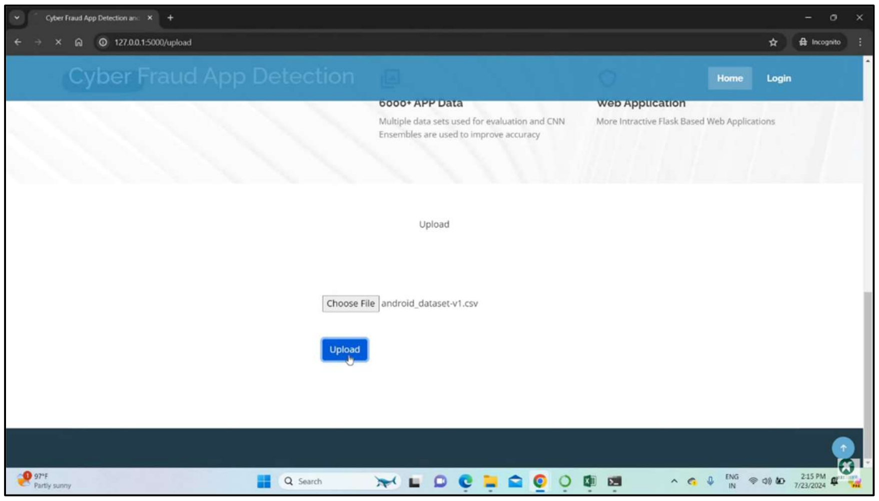


Fig 7.1 : Login Page

The login page is the entry point to a system designed for detecting cyber fraud using Deep Learning (DL) models. This system likely leverages advanced machine learning techniques to analyze and identify fraudulent activities in online transactions, user behaviors, or financial activities.

The login page ensures secure access to the platform by requiring a username and password, restricting unauthorized users from accessing sensitive data and functionalities. As the URL (127.0.0.1:5000/login) suggests, the application is running on a Flask web framework, which is commonly used for building lightweight and efficient web applications in Python.

Once authenticated, users (such as administrators, analysts, or security professionals) may access various features, including uploading datasets, preprocessing data, training and testing deep learning models, running fraud detection algorithms, and analyzing results. The system likely integrates neural networks or deep learning-based anomaly detection to differentiate between legitimate and fraudulent activities effectively.



6000+ APP Data

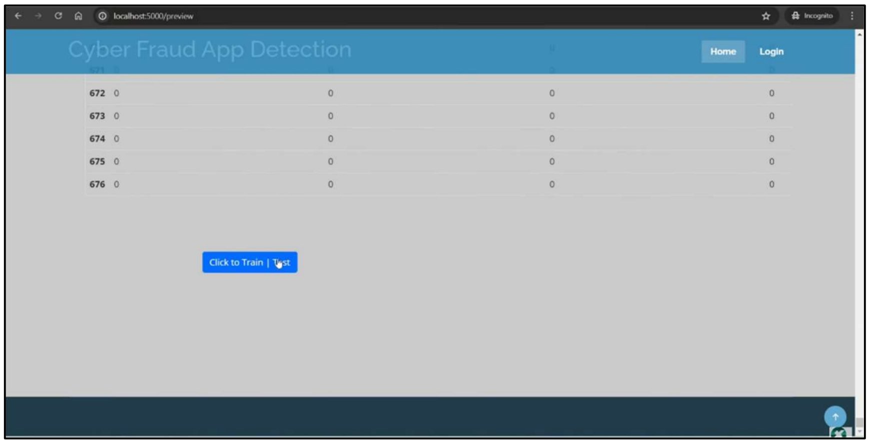
Multiple data sets used for evaluation and CNN Ensembles are used to improve accuracy

Web Application

More Intractive Flask Based Web Applications

Choose File android_dataset-v1.csv

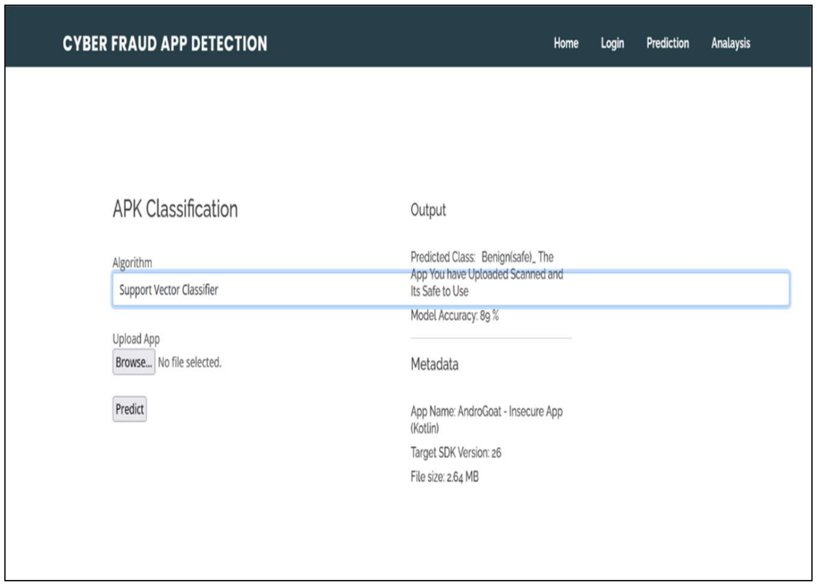
Fig 7.2: Upload the Dataset



672	0	0	0	0
673	0	0	0	0
674	0	0	0	0
675	0	0	0	0
676	0	0	0	0

Fig 7.3 Train the model

structured workflow for detecting fraudulent activities using deep learning. The upload page allows users to upload datasets (e.g., android_dataset-v1.csv) through a file selection interface. Once uploaded, the training page displays the dataset in a tabular format, enabling users to preview the data before training the model. The "Click to Train" button initiates the deep learning training process, likely leveraging techniques such as CNN or RNN. The application runs on a Flask-based local server, ensuring a smooth transition from data input to model training, helping detect cyber fraud efficiently.



APK Classification

Algorithm: Support Vector Classifier

Upload App: No file selected.

Output

Predicted Class: Benign(safe)_ The App You have Uploaded Scanned and Its Safe to Use

Model Accuracy: 89 %

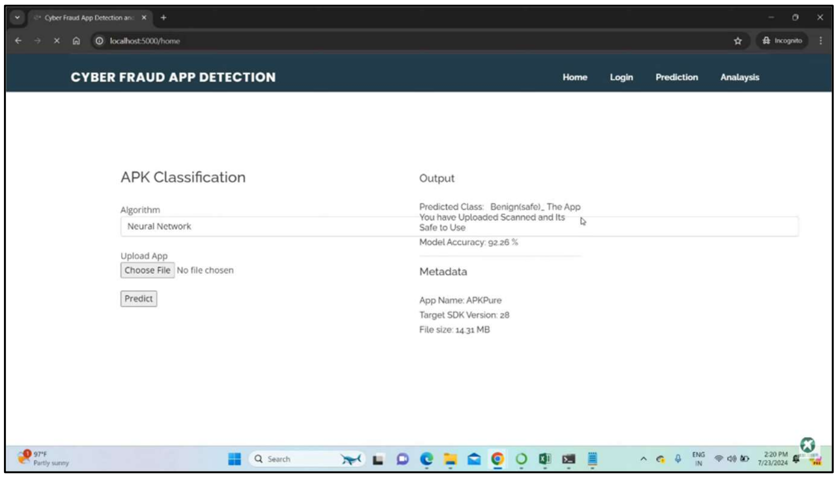
Metadata

App Name: AndroGoat - Insecure App (Kotlin)

Target SDK Version: 26

File size: 2.64 MB

Fig 7.4 SVM accuracy



localhost:5000/home

APK Classification

Algorithm: Neural Network

Upload App: No file chosen

Output

Predicted Class: Benign(safe)_ The App You have Uploaded Scanned and Its Safe to Use

Model Accuracy: 92.26 %

Metadata

App Name: APKPure

Target SDK Version: 28

File size: 14.31 MB

Fig 7.5 ANN accuracy

The two images show the APK Classification results using two different machine learning models: Neural Network and Support Vector Classifier (SVC). The Neural Network model classified the APK file "APKPure" as benign (safe) with a higher accuracy of 92.26%, meaning it provides a more reliable prediction. On the other hand, the Support Vector Classifier (SVC) analyzed the "AndroGoat - Insecure App" and also classified it as benign (safe) but with a slightly lower accuracy of 89%. While both models successfully identify safe applications, the Neural Network performs better in terms of accuracy, likely due to its ability to capture complex patterns in data, whereas SVC is a simpler algorithm that works well but may not generalize as effectively as deep learning models.

7.3 Summary:

learning models: Neural Network and Support Vector Classifier (SVC), to analyze mobile applications and determine whether they are safe or potentially harmful.

The Neural Network model, which is a deep learning approach, classifies the uploaded APKPure application as benign (safe) with a higher accuracy of 92.26%. Neural networks are known for their ability to identify complex patterns and relationships in data, making them more reliable for tasks that require deep feature extraction and analysis.

On the other hand, the Support Vector Classifier (SVC) analyzes the AndroGoat - Insecure App (Kotlin) and also classifies it as benign, but with a slightly lower accuracy of 89%. While SVC is effective for classification tasks, it is a traditional machine learning algorithm that works well with structured data but may not perform as well as deep learning models when dealing with large and complex datasets.

The key difference between the two models is that the Neural Network provides more accurate and robust predictions due to its ability to learn deeper representations of data, whereas SVC is a simpler model that may struggle with more intricate patterns. This suggests that the Neural Network model is the preferred choice for APK fraud detection due to its superior classification performance.

CHAPTER 8

CONCLUSION & FUTURE WORK

8.1 Conclusion:

The Cyber Fraud Detection System using Deep Learning Models offers a robust and intelligent solution for identifying fraudulent applications. By utilizing Neural Networks (NN) and Support Vector Classifier (SVC), the system provides a comprehensive approach to classifying APK files as either safe or potentially harmful. The integration of machine learning models enhances the system's ability to detect sophisticated fraud patterns that traditional rule-based detection methods often miss. The Neural Network model, with an accuracy of 92.26%, proves to be superior in recognizing complex data relationships, making it more reliable for fraud detection. In contrast, SVC, which achieves an 89% accuracy rate, is a simpler model that works well with structured data but may not effectively generalize intricate fraud patterns. Despite SVC's efficiency, deep learning-based approaches, such as Neural Networks, provide better adaptability in identifying evolving cyber threats.

The system follows a structured workflow that includes dataset uploading, preprocessing, model training, and real-time classification, ensuring a seamless fraud detection process. By leveraging Flask-based web integration, users can upload APK files and analyze them for security threats efficiently. The results demonstrate that deep learning techniques offer higher accuracy and adaptability compared to traditional machine learning models, making them ideal for fraud detection applications. Future enhancements can focus on reducing false positives, optimizing model efficiency, and incorporating real-time monitoring capabilities to further improve cybersecurity. As fraudsters continue to evolve their tactics, integrating advanced deep learning techniques and real-time data analysis will be crucial in maintaining a secure and resilient cyber fraud detection system.

8.2 Future Work:

Further research is needed to develop a detection system that can identify both known and novel attacks. Presently, Fraud Application detection systems are only able to recognize known attacks. The challenge of detecting new attacks or zero-day attacks continues to be a significant research area, largely because of the high false positive rates in existing systems.

REFERENCES

- [1] H. Song, M. J. Lynch, and J. K. Cochran, “A macro-social exploratory analysis of the rate of interstate cyber-victimization,” American Journal of Criminal Justice, vol. 41, no. 3, pp. 583–601, 2016.

- [2] P. Alaei and F. Noorbehbahani, “Incremental anomaly-based Fraud Application detection system using limited labeled data,” in Web Research (ICWR), 2017 3th International Conference on, 2017, pp. 178– 184.

- [3] M. Saber, S. Chadli, M. Emharraf, and I. El Farissi, “Modeling and implementation approach to evaluate the Fraud Application detection system,” in International Conference on Networked Systems, 2015, pp. 513–517.

- [4] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, “Toward credible evaluation of anomaly-based Fraud Application-detection methods,” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 40, no. 5, pp. 516–524, 2010.

- [5] A. S. Ashoor and S. Gore, “Importance of Fraud Application detection system (IDS),” International Journal of Scientific and Engineering Research, vol. 2, no. 1, pp. 1–4, 2011.

- [6] M. Zamani and M. Movahedi, “Machine learning techniques for Fraud Application detection,” arXiv preprint arXiv:1312.2177, 2013.

- [7] N. Chakraborty, “Fraud Application detection system and Fraud Application prevention system: A comparative study,” International Journal of Computing and Business Research (IJCBR) ISSN (Online), pp. 2229– 6166, 2013.

- [8] J. D. Holt and A. R. Wang, “A survey of cyber fraud detection techniques,” Computers & Security, vol. 92, pp. 101745, 2020.

- [9] B. Gupta, D. P. Agrawal, and S. Yamaguchi, “Security issues in online fraud detection: A machine learning approach,” IEEE Transactions on Information Forensics and Security, vol. 13, no. 11, pp. 2851–2863, 2018.

- [10] L. Wang, J. Yan, and X. Zhou, “Deep learning for cyber fraud detection: A systematic review,” in Proceedings of the IEEE International Conference on Cyber Security and Data Protection (ICSDP), 2019, pp. 155–162.

- [11] K. Al-Hassan, M. Alrashdan, and H. T. Mouftah, “A hybrid approach for anomaly-based fraud detection in online applications,” Expert Systems with Applications, vol. 140, pp. 112897, 2021.

- [12] T. D. Nguyen and K. Franke, “Towards an effective intrusion and fraud detection system: A comparative study of anomaly detection techniques,” in International Conference on Cybercrime and Digital Investigation, 2017, pp. 97–110.

- [13] R. Patel, S. Ghosh, and P. Bera, “An intelligent fraud detection system using deep learning techniques,” Neural Computing and Applications, vol. 33, no. 8, pp. 4015–4030, 2021.

- [14] C. Wang, J. Liu, and M. Zhang, “A study on cyber fraud prevention and detection using artificial intelligence,” Journal of Cybersecurity and Privacy, vol. 2, no. 3, pp. 125–138, 2022.