A Project Report

On

ALERT AURA - WOMEN SAFETY APPLICATION

Submitted In Partial Fulfilment Of The
Requirement For The Award Of Degree Of

Bachelor of Engineering

In

Information Technology

Submitted by

N. Sushanth	1608-21-737-002
D. Shanmukhaditya	1608-21-737-036
R. Venkata Anirudh	1608-21-737-054

Under the Esteemed Guidance of

Dr. J. Srinivas

Associate Professor - I.T.

Department of Information Technology

Matrusri Engineering College

(An Autonomous Institution)

(Sponsored by Matrusri Education Society, Estd.1980)

(Affiliated to Osmania University, Approved by AICTE)

Saidabad, Hyderabad-500059

(2024-2025)

Department of Information Technology

Matrusri Engineering College

(An Autonomous Institution)
(Sponsored by Matrusri Education Society, Estd.1980)
(Affiliated to Osmania University, Approved by AICTE)
Saidabad, Hyderabad-500059
(2023-2024)

CERTIFICATE

This is to certify that the project report entitled " **Alert Aura** " submitted by **NERELLA SUSHANTH** bearing the H.T. No **: 1608 - 21 - 737 - 002 , D. SHANMUKHADITYA** bearing the H.T. No**: 1608 - 21 - 737 - 036** and **R. VENKATA ANIRUDH** bearing the H.T. No : **1608 - 21- 737 - 054**, in the partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering in Information Technology** is a Bonafide work carried by them. The results of the investigations enclosed in this report have been verified and found satisfactory.

DR. J. SRINIVAS
ASSOCIATE PROFESSOR
PROJECT GUIDE

DR. G. SHYAMA CHANDRA PRASAD
PROFESSOR & HEAD
DEPT. OF IT

EXTERNAL EXAMINER(S)

ACKNOWLEDGEMENT

We wish to express our gratitude to the project guide**,** for his valuable supervision and timely assistance in regard to our project work.

We wish to express our gratitude to project coordinator **Dr. J. Srinivas** , Associate Professor, Department of Information Technology, Matrusri Engineering College, for his indefatigable inspiration, guidance, cogent discussion, constructive criticisms and encouragement throughout this dissertation work.

We wish to express our gratitude to **Dr. G. Shyama Chandra Prasad,** Head of Department, Information Technology, Matrusri Engineering College for permitting us to do this project.

We wish to express our gratitude to **Dr. G. Amarendar Rao,** Principal of Matrusri Engineering College who permitted us to carry the project work as per the academics.

We would like to thank IT Department for providing us this opportunity to share and contribute ours part of work to accomplish the project in time, and all the teaching and supporting staff for their steadfast support and encouragement

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of my family members and friends, who made it possible and whose encouragement and guidance has been a source of inspiration.

DECLARATION

We hereby declare that the project entitled "**ALERT AURA - WOMEN SAFETY APPLICATION**" submitted to the Osmania University in partial fulfilment of the requirements for the award of the degree of B.E. in IT, is a bonafide and original work done by us under the guidance of Dr. J. Srinivas , Associate Professor, Information Technology and this project work have not been submitted to any other university for the award of any degree or diploma.

N. Sushanth 1608-21-737-002

D. Shanmukhaditya 1608-21-737-036

R. Venkata Anirudh 1608-21-737-054

ABSTRACT

This project proposes an AI-driven mobile application designed to enhance women's safety by autonomously detecting and responding to emergencies in real time. The application uses machine learning to analyze inputs from speech, video, motion sensors, and location-based crime data to identify potential threats. A key feature is gesture recognition through **video analysis**, which monitors for **distress signals** such as specific hand movements or body postures. When a gesture indicating distress is detected, the system triggers automated responses including sending SOS alerts, notifying authorities, and capturing evidence. Speech emotion analysis, achieved with an **LSTM model reaching an accuracy of 93.21%**, helps identify panic or distress in the user's voice. Additionally, **sensor-based monitoring employs Liquid Neural Networks** to detect abnormal movement patterns, **with a current accuracy of 78.05%**. Crime risk assessment adds contextual awareness based on the user's location, enhancing decision-making during high-risk situations. The application runs continuously in the background with minimal battery usage and prioritizes user privacy through secure data handling practices. By combining real-time data processing with intelligent automation, the application offers a proactive and personalized approach to safety, especially for women in vulnerable situations.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1. INTRODUCTION

Women's safety in emergency situations continues to be a major social and technological concern, often hindered by delayed response times and the dependence on manual activation of safety features. Traditional safety applications require users to press a button or send alerts actively, which may not always be possible during high-risk or unconscious situations. Recent advancements in artificial intelligence, especially in computer vision and sensor-based recognition, provide new opportunities to address these limitations. In this project, we propose an AI-powered mobile application capable of detecting threats autonomously using multimodal data inputs such as speech signals, motion sensor readings, geolocation crime data, and live video feeds. A key innovation of our system is the use of gesture-based distress recognition, where specific hand or body gestures—indicating panic or danger—are monitored through real-time video analysis. When a distress gesture is identified, the application triggers a series of automated safety actions, including SOS alerts, evidence collection, and communication with emergency contacts or local authorities. This combination of proactive monitoring and context-aware response aims to create a robust, intelligent safety solution for women, capable of operating independently without requiring user input in critical situations.

1.1. PROBLEM DEFINITION

A major technical challenge in modern safety systems is enabling autonomous detection and response to emergencies without relying on explicit user input. In critical situations—where an individual may be panicked, restrained, or unconscious—manual activation of alerts is often not possible, rendering traditional safety applications ineffective. Additionally, many existing solutions rely on isolated data streams, lacking the ability to holistically assess threats. This project seeks to address these gaps by developing a multimodal, AI-driven mobile application that intelligently interprets both user behavior and environmental cues. By integrating gesture-based distress detection through real-time video analysis, speech emotion recognition, motion sensor monitoring, and location-based crime risk assessment, the system can recognize potential danger autonomously. Upon identifying a threat—such as a distress gesture or panic in the user's voice—it initiates automatic safety protocols, including sending SOS alerts, recording video evidence, and notifying emergency contacts and authorities. The proposed solution aims to create a proactive,

context-aware safety framework that operates without requiring conscious user intervention, offering enhanced protection in vulnerable scenarios.

1.2. EXISTING SYSTEM

Current safety solutions aimed at emergency response primarily rely on manual user intervention and are limited in their ability to detect threats autonomously. Panic button applications, for instance, require users to actively trigger an alert, which may not be feasible in critical situations. Similarly, location-based alert systems use GPS tracking to notify contacts when a user leaves predefined safe zones, but they do not assess real-time danger or user behavior. Video surveillance tools are generally designed for fixed monitoring and lack integration with other sensory data, making them unsuitable for dynamic, real-time threat analysis. Safety wearables offer basic SOS features and location tracking but do not incorporate intelligent threat prediction or emotion detection. Additionally, standalone crime prediction models typically analyze historical data to identify risk-prone areas but fail to respond to immediate emergencies.

1.3. DISADVANTAGES:

- Heavy reliance on manual activation limits effectiveness in high-risk scenarios.
- Inability to detect emotional or physical distress through real-time behavioral cues.
- Lack of integration across multiple data sources such as video, audio, and sensors.
- Absence of proactive, AI-driven threat recognition and response.

1.4. PROPOSED SYSTEM

We focus on the development of a multimodal emergency response application, which differs significantly from existing manual safety tools by leveraging artificial intelligence for autonomous threat detection. Our work is the first to integrate gesture-based distress recognition using video analysis alongside speech emotion detection, sensor-based monitoring, and crime location awareness within a single mobile framework. The proposed system analyzes real-time data from the smartphone's front camera, microphone, accelerometer, gyroscope, and GPS to detect signs of distress without requiring user input. Specifically, the video module continuously monitors for

predefined distress gestures using computer vision models, while the audio module evaluates voice samples for emotional cues such as panic or fear. Motion sensors are used to detect sudden or erratic movements, which may indicate danger. Additionally, the system incorporates crime statistics from the user's current location to assess situational risk. Once a potential threat is identified, the application autonomously initiates an SOS protocol—alerting emergency contacts, notifying authorities, and recording video or audio evidence. Our proposed AI-driven solution addresses key limitations in current systems by offering a real-time, automated, and context-aware safety mechanism. By combining multiple data sources and machine learning models, this approach significantly improves emergency responsiveness, particularly in scenarios where the user is unable to manually seek help**.**

1.5. REQUIREMENT SPECIFICATION

1.5.1 HARDWARE REQUIREMENTS

Mobile Device	**Sensors:** Accelerometer, Gyroscope, Microphone, GPS, and Camera **Minimum RAM:** 3GB **Minimum Storage:** 32GB **Processor:** Qualcomm Snapdragon 845 or equivalent
Backend Server	**CPU:** Minimum 4 vCPUs **RAM:** Minimum 16GB **Storage:** Minimum 100GB SSD **Bandwidth:** 100 Mbps or higher (for real-time data processing)
Development and Testing Hardware	**Processor:** Intel Core i7 or AMD Ryzen 7 **RAM:** 16GB **Storage:** 512GB SSD **GPU:** NVIDIA GTX 1660 or equivalent (for ML model training)

1.5.2 SOFTWARE REQUIREMENTS

Operating System	Android 9.0 (Pie) or above for the mobile application
Programming Languages	Python, Java/Kotlin
IDEs/Tools	Android Studio, VS Code, Jupyter Notebook
Machine Learning Frameworks	TensorFlow, PyTorch
APIs	Google Maps API, CameraX API, Sensor API, Speech Recognition API
Database	Firebase Realtime Database or PostgreSQL
Cloud Services	AWS / Google Cloud / Azure (for hosting backend services)
Version Control	Git and GitHub

2. LITERATURE SURVEY

Various researchers have proposed AI-based systems to enhance women's safety through automation and real-time threat detection. In [1], the authors developed a mobile application that uses GPS-based location tracking and sends emergency alerts to pre-defined contacts during unsafe situations. The system, however, depends on manual SOS activation. In [2], an intelligent women safety device was proposed, which integrates sensors and GPS to send alerts to nearby police stations. It includes heart rate monitoring but lacks real-time decision-making based on user behavior.

Gupta et al. [3] presented a smart wearable device that detects abnormal conditions using body temperature and pulse sensors. The system activates an alert mechanism if parameters cross certain thresholds. In [4], a model was introduced that utilizes GSM and GPS modules integrated into a wearable band for real-time tracking and emergency messaging. While effective in sending alerts, the solution lacks the ability to analyze the user's emotional or behavioral state.

In [5], an AI model was developed for threat prediction using historical crime data. The system uses machine learning algorithms to identify crime-prone areas. Although useful for risk assessment, the model does not incorporate real-time user data such as gestures or speech input.

In [6], researchers explored real-time video analysis using computer vision to detect suspicious activities in public areas. However, their work was limited to surveillance infrastructure and not integrated into a mobile or wearable solution. In contrast, [7] proposed a deep learning-based model to detect distress emotions from audio samples using speech emotion recognition, demonstrating potential for use in mobile safety applications.

Collectively, these works lay the foundation for safety systems but often rely on single-modality inputs or manual activation, limiting their effectiveness. The proposed system in this project aims to overcome these limitations by integrating gesture recognition, voice emotion analysis, sensor data, and crime statistics into a unified, real-time mobile application for autonomous emergency response.

2.1 TABLE FOR LITERATURE SURVEY

S. No.	Name Of Author	Year	Technique Used	Advantages	Disadvantages
1.	Gupta et al.	2019	Body temperature and pulse sensors with GSM & GPS	Detects abnormal conditions and sends alerts via SMS	Lacks real-time behavioral analysis and gesture recognition
2.	Sharma et al.	2020	Panic button with GPS tracking	Simple UI, reliable location sharing	Depends entirely on manual activation
3.	Reddy et al.	2021	IoT-based wearable with sensor integration	Real-time location tracking and emergency alerts	Lacks emotion or gesture-based detection
4.	Mehta et al.	2022	Crime prediction using historical data with ML	Identifies high-risk zones for awareness	Cannot respond to real-time user emergencies
5.	Patel et al.	2022	Speech emotion recognition using SVM	Detects panic/stress in voice	Only supports audio input, lacks multi-modal capability
6.	Kopanati et al.	2024	TensorFlow Lite, audio sensing, cloud storage, geolocation (SDLC)	Automatically detects threats, records evidence, shares location	Limited to audio-based detection; no gesture or sensor fusion

3. DESIGN AND METHODOLOGY

3.1. SYSTEM ARCHITECTURE

We propose **Alert Aura**, an AI-driven, autonomous threat detection and response system. It collects data from multiple sources including a crime data API, microphone, camera, GPS, accelerometer, and gyroscope. These inputs are processed through modules for crime pattern analysis, speech and video processing, and sensor data analysis. Extracted features are fed into specialized machine learning models for crime risk assessment, emotion detection, video analysis, and anomaly detection. Model outputs are synthesized by a central Decision Engine using a decision algorithm. The system then triggers an Automated Response System for safety recommendations, evidence recording, and SOS alerting. By fusing multimodal data and machine intelligence, we provide comprehensive situational awareness.This architecture ensures intelligent, real-time threat response without human intervention. Its modular design offers scalability and reliability. Alert Aura is suited for both personal and public safety contexts. The system's architecture is shown in Figure 3.1.

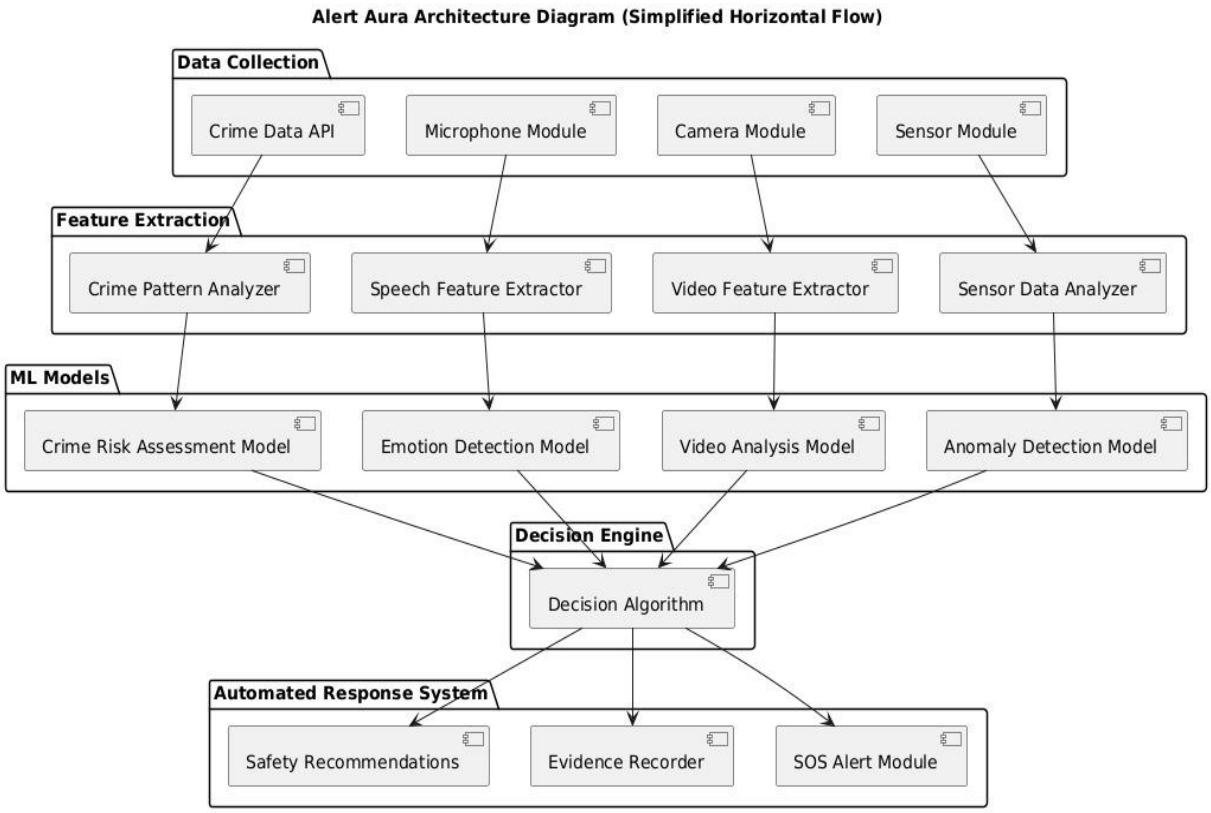


FIGURE 3.1.1 : SYSTEM ARCHITECTURE

3.2. PYTHON MODULES USED IN THE PROJECT

Python provides a powerful ecosystem of libraries and modules that support modular, scalable, and efficient development. In this project, which focuses on real-time women's safety through AI, several Python modules are employed for tasks such as machine learning, video processing, emotion recognition, geolocation, and user interaction.

TABLE 3.2.1 PYTHON MODULES USED

Module	Description
NumPy	Handles multi-dimensional arrays and performs numerical computations essential for video frame processing and sensor data transformation.
Pandas	Used for structured data handling, preprocessing crime datasets, sensor logs, and model outputs for analysis and training.
OpenCV	Core library for real-time image and video processing; captures frames, preprocesses images, and extracts visual features.
Keras	High-level API for building and training deep learning models for emotion recognition from speech and visual inputs.
PyTorch	Used for developing custom models for gesture and anomaly detection
Scikit-learn	Implements traditional ML algorithms (SVM, Random Forest, Decision Trees) for crime pattern analysis and sensor data classification.
Matplotlib & Seaborn	Used to create visualizations like risk heatmaps and performance metrics; supports analysis and interpretation.
FFmpeg	Handles audio/video stream operations such as converting, compressing,
MediaPipe	Enables real-time gesture and body pose detection; extracts hand, face, and body landmarks for distress gesture analysis.
Librosa	Extracts audio features (e.g., MFCCs) for speech analysis and emotion detection to

	identify panic or distress in voices.
BeautifulSoup	Performs web scraping to fetch/update crime data and emergency contacts; simplifies HTML parsing.
Geocoder	Converts GPS coordinates into human-readable addresses and vice versa; essential for geolocation and locating nearby safe places
Tkinter	Provides a GUI framework for desktop simulation and testing; enables user interaction and visualization of model outputs.

3.3. UML DIAGRAMS

UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. The goal of UML is to provide a standard notation that can be used by all object- oriented methods and to select and integrate the best elements of precursor notations. UML has been designed for a broad range of applications. Hence, it provides constructs for a broad range of systems and activities.

In addition to these core benefits, UML was standardized by the Object Management Group (OMG) in 1997, ensuring a globally recognized and maintained modeling language. One of the key strengths of UML is its flexibility; it supports various software development methodologies, including Agile, Waterfall, and Spiral models. UML includes 14 standard diagram types categorized into structure diagrams (such as class, object, component, and deployment diagrams) and behavior diagrams (such as use-case, sequence, activity, and state diagrams), enabling a comprehensive representation of both static and dynamic aspects of a system.

3.3.1 USE CASE DIAGRAM

A UML use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behavior (what), and not the exact method of making it happen (how). Use cases once specified can be denoted both textual and visual representation (i.e., use case diagram).

A key concept of use case modeling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behavior in the user's terms by specifying all externally visible system behavior.

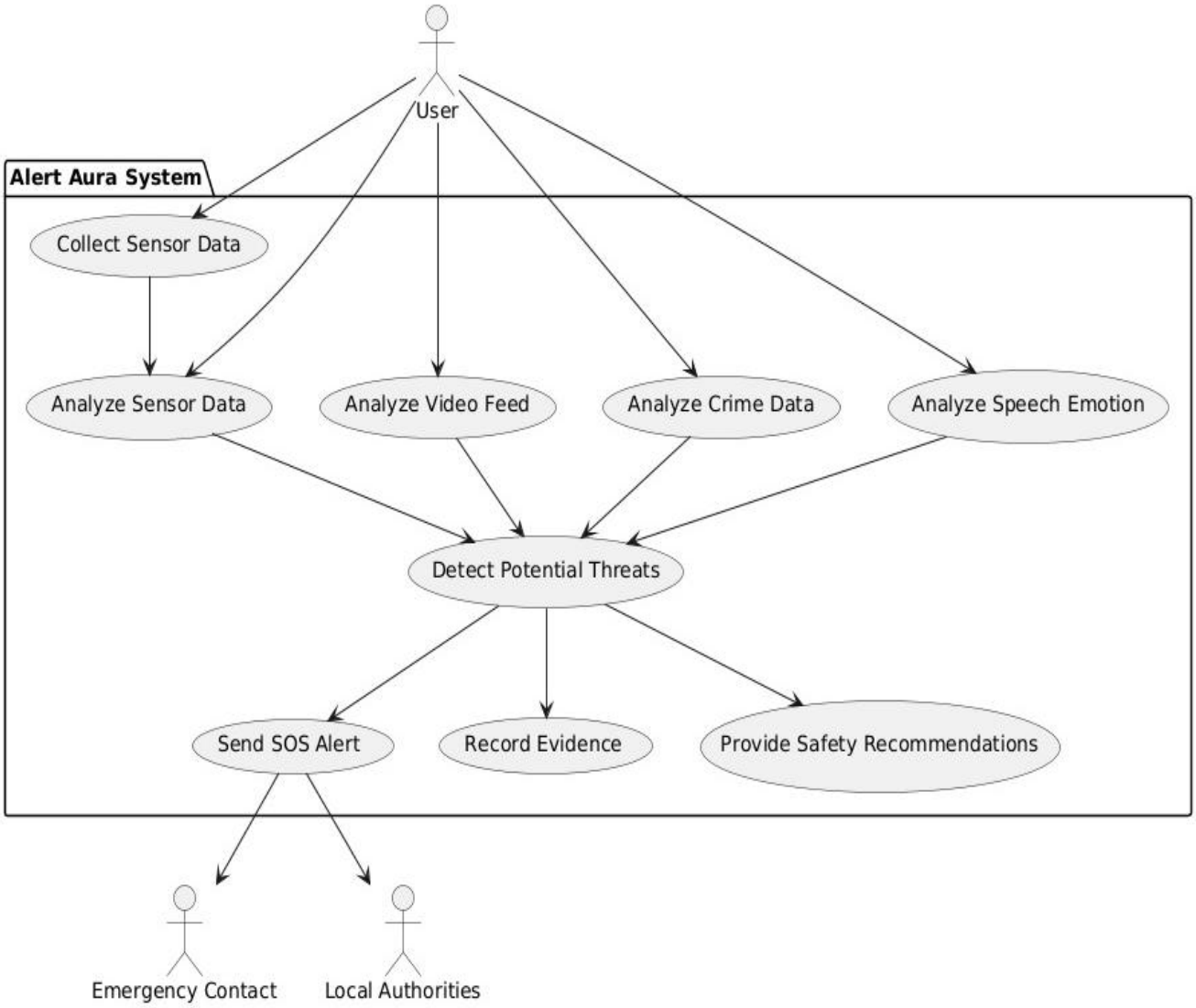


FIGURE 3.3.1 : USE CASE DIAGRAM

3.3.2 ACTIVITY DIAGRAM

Activity diagram is another important behavioral diagram in UML diagram to describe dynamic aspects of the system. Activity diagram is essentially an advanced version of flowchart that models the flow from one activity to another activity. Activity diagrams show the process from the start (the initial state) to the end (the final state). Each activity diagram includes an action, decision node, control flows, start node, and end node.

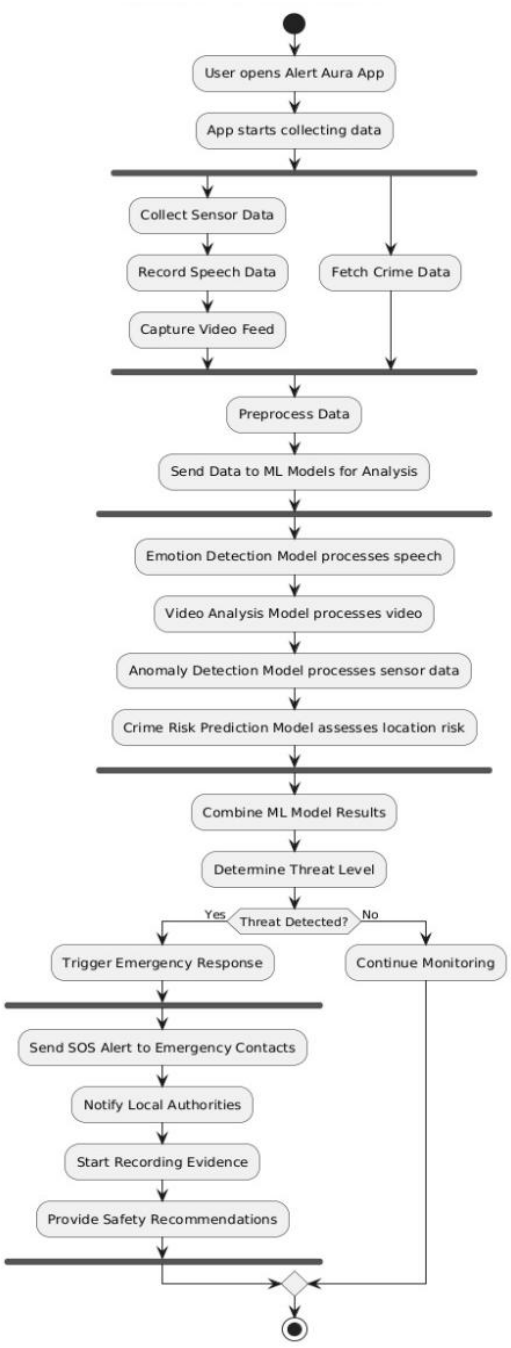


FIGURE 3.3.2 : ACTIVITY DIAGRAM

3.3.3 SEQUENCE DIAGRAM

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focused and they show the order of the interaction visually by using the vertical axis of the diagram to represent time, what messages are sent and when.

Sequence Diagrams captures:

- The interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams)
- High-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)

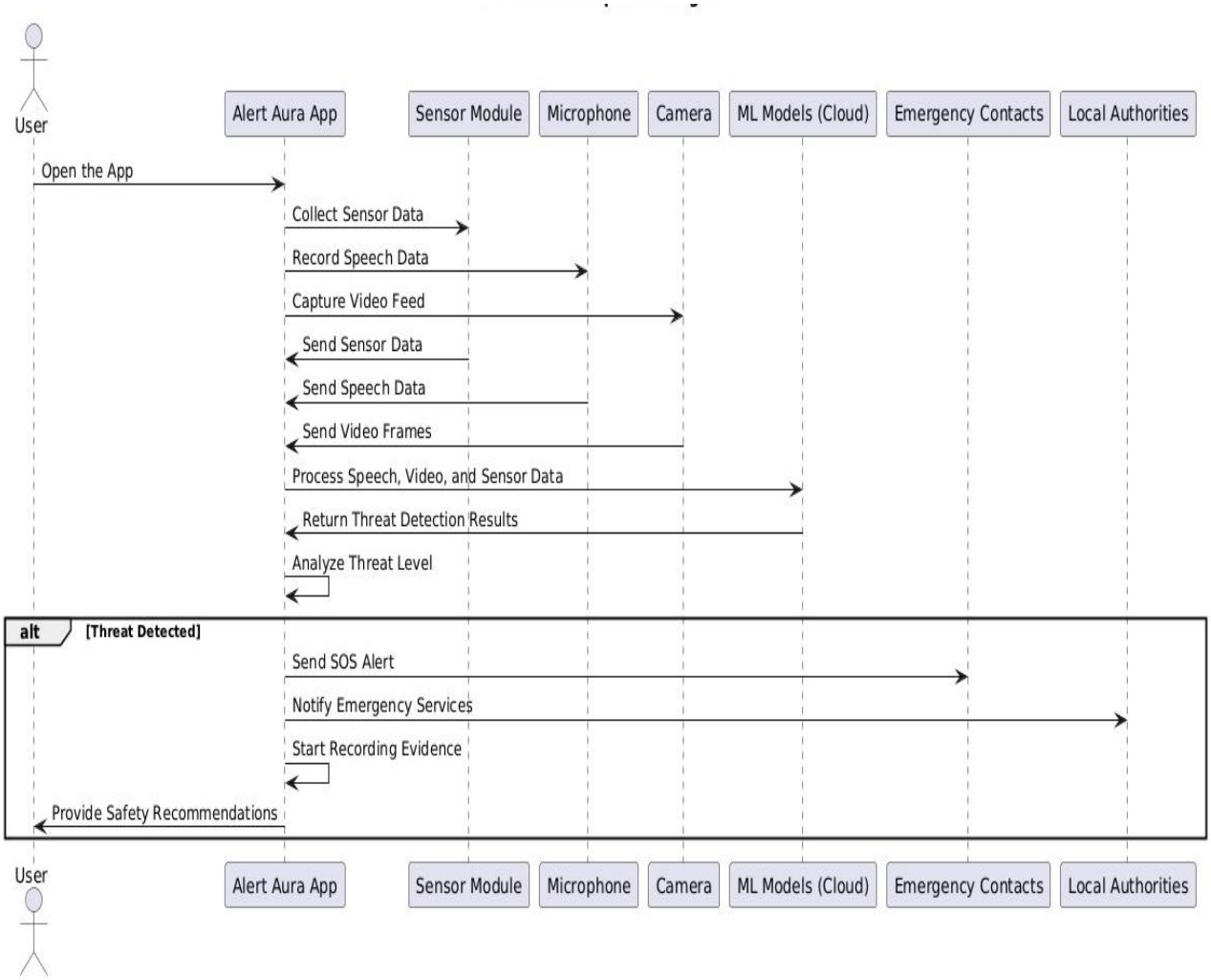


FIGURE 3.3.3 : SEQUENCE DIAGRAM

3.3.4 CLASS DIAGRAM

In software engineering, a class diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

A UML class diagram is made up of:

- A set of classes and
- A set of relationships between

The user performs an operation called giving input to the system. The system trains and tests the model and predicts the disease based on the inputs given to it and displays the result to the user.

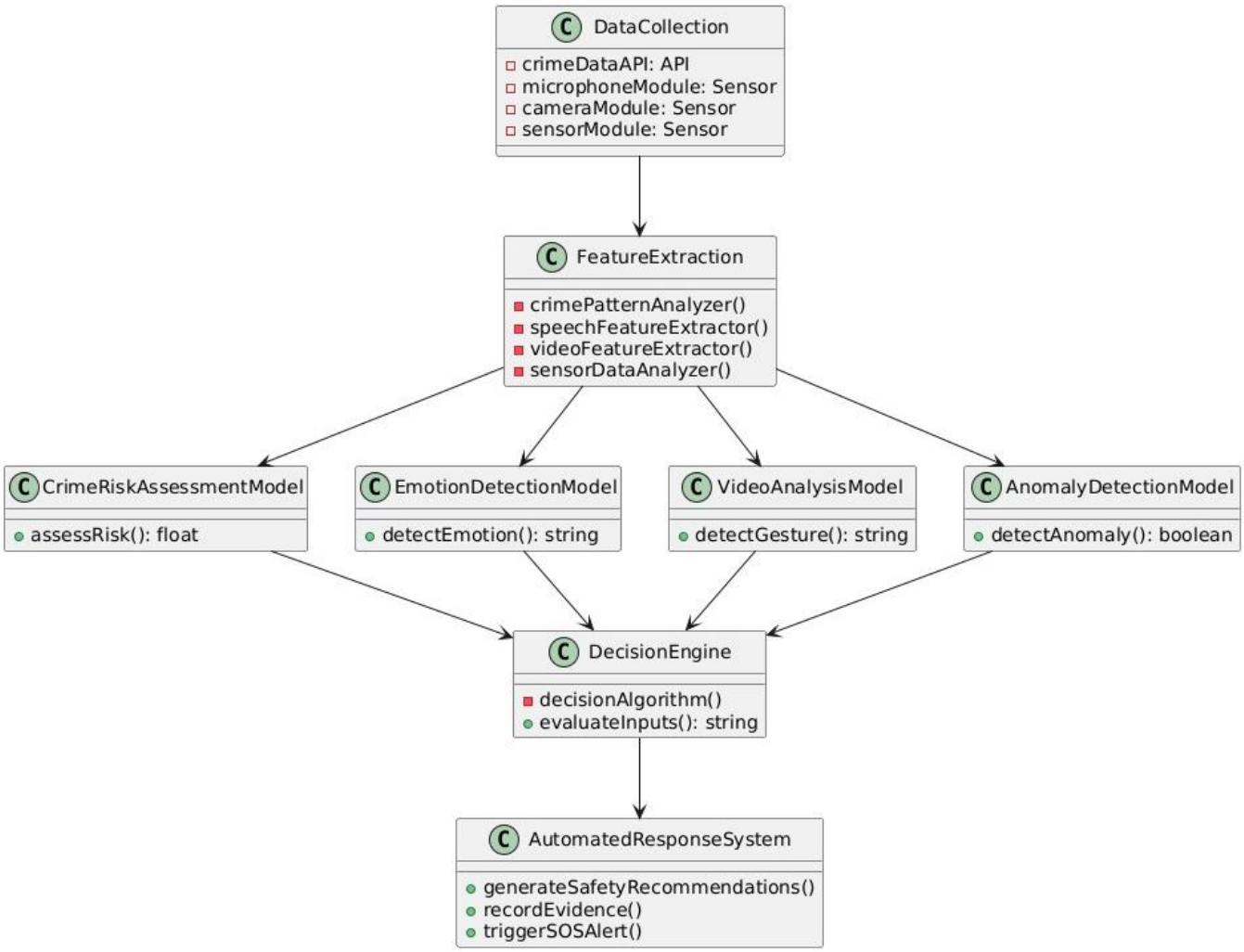


FIGURE 3.3.4 : CLASS DIAGRAM

3.4. SELECTED SOFTWARE - PYTHON

Python is an interpreted high-level programming language for general purpose programming. Created by Guido van Rossum, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

- It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management
- It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.
- Python interpreters are available for many operating systems.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect- oriented programming.

Python uses dynamic typing and a combination of reference counting and a cycle- detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

3.4.1 BENEFITS OF PYTHON:

- · Presence of Third-Party Modules
- · Extensive Support Libraries
- · Open Source and Community Development
- · Learning Ease and Support Available
- · User-friendly Data Structure
- · Productivity and Speed
- · Highly Extensible and Easily Readable Language.

3.4.2 FEATURES OF PYTHON:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English words frequently whereas other languages use punctuation, and it has fewer syntactic constructions than other languages.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

3.4.3 MACHINE LEARNING

Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field.

- Machine Learning is combination of correlations and relationships, most machine learning algorithms in existence are concerned with finding and/or exploiting relationship between datasets
- Once Machine Learning Algorithms can pinpoint on certain correlations, the model can either use these relationships to predict future observations or generalize the data to reveal interesting patterns.
- In Machine Learning there are various types of algorithms such as Regression, Linear Regression, Logistic Regression, Naive Bayes Classifier, Bayes theorem, KNN (K-Nearest Neighbor Classifier), Decision Trees, Entropy, ID3, SVM (Support Vector Machines), K-means Algorithm, Random Forest and etc.

3.4.4 FEATURES OF MACHINE LEARNING:

- Machine learning models involve machines learning from data without the help of humans or any kind of human intervention.
- Machine Learning is the science of making computers learn and act like humans by feeding data and information without being explicitly programmed.
- Machine Learning is totally different from traditionally programming, here data and output is given to the computer and in return it gives us the program which provides solution to the various problems
- It is nothing but automation.

- Writing software is a bottleneck.
- Getting computers to program themselves.
- Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover “hidden insights" through learning from historical relationships and trends in the data.

3.4.5. NEEDS FOR MACHINE LEARNING :

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other hand, AI is still in its initial stage and hasn't surpassed human intelligence in many aspects. Then the question is what is the need to make machine learning? The most suitable reason for doing this is, “to make decisions, based on data, with efficiency and scale”.

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programming logic, in the problems that cannot be programmed inherently. The fact is that we can’t do without human intelligence, but another aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

3.4.6. CATEGORIES OF MACHINE LEARNING

Machine learning tasks Machine learning tasks are typically classified into several broad categories:

- **Supervised learning**: The computer is presented with example inputs and their desired outputs, given by a "teacher”, and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback.
- **Semi-supervised learning:** The computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.

- **Active learning**: The computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labelling.
- **Unsupervised learning**: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).
- **Reinforcement learning**: Data (in the form of rewards and punishments) are given only as feedback to the program's actions in a dynamic environment,such as driving a vehicle or playing a game against an opponent. Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning.

4. TESTING AND IMPLEMENTATION RESULTS

4.1. TYPES OF TESTS

- **Unit Testing :** Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or systemconfiguration. Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.
- **Integration Testing :** Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.
- **Validation Testing :** An engineering validation test (EVT) is performed on first engineering prototypes, to ensure that the basic unit performs to design goals and specifications. It is important in identifying design problems, and solving them as early in the design Cycle, as possible, is the key to keeping projects on time and within budget. Too often, product design and performance problems are not detected until late in the product development cycle — when the product is ready to be shipped. The old adage holds true: It costs a penny to make a change in engineering, a dime in production and a dollar after a

product is in the field. Verification is a Quality control process that is used to evaluate whether or not a product, service, or system complies with regulations, specifications, or conditions imposed at the start of a development phase. Verification can be in development, scale- up, or production. Validation is a Quality assurance process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements.

- **System Testing :** System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system.System testing is a more limited type of testing; it seeks to detect defects both within the "inter- assemblages" and also within the system as a whole. System testing is performed on the entire system in the context of a Functional Requirement Specification (FRS) or System Requirement Specification (SRS).

The testing process overview is as follows:

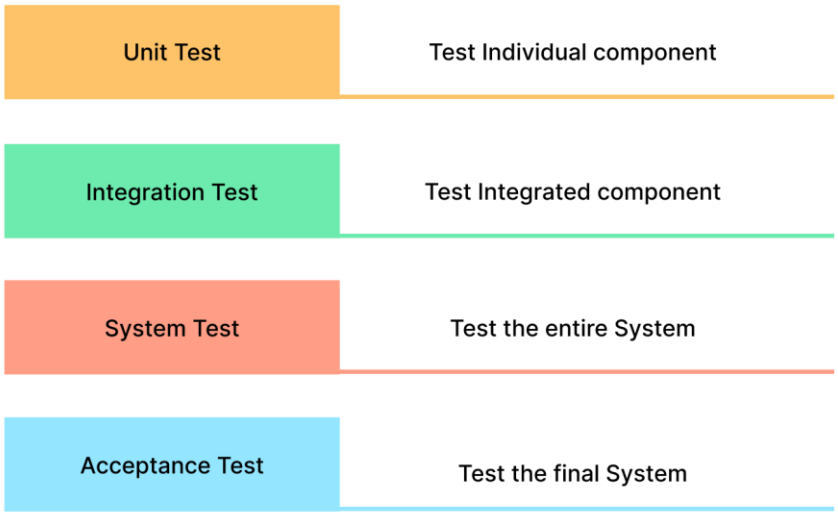


FIGURE 4.1. : TESTING PROCESS

4.2. TEST CASES MATRIX

Serial Number of Test Case	Module Under Test	Description	Input	Output	Remarks
TC 01	Speech Emotion Detection	System captures user's voice in real time and detects panic or distress emotion	Live microphone audio + trained speech model	Detected emotion label (e.g., "Panic") and confidence score displayed	Test Unsuccessful
TC 02	Gesture Detection (Video)	System monitors live video feed to detect predefined distress gestures	Live camera stream + trained gesture model	SOS trigger activated upon gesture detection	Test Unsuccessful
TC 03	Crime Risk Prediction	Predicts risk level by comparing user's location with static crime dataset from Indian government	User's GPS coordinates + preloaded crime dataset	Displays area-wise crime level (Low/Medium/ High) based on dataset mapping	Test Unsuccessful
TC 04	Automated SOS Alert	Sends alert to emergency contacts when danger is confirmed	Trigger from video/audio-based detection modules	Alert message with location sent to contacts and authorities	Test Unsuccessful
TC 05	Evidence Recorder	Starts audio/video recording upon detection of threat	Trigger from emotion or gesture model	Recorded video and audio files saved locally or to cloud	Test Unsuccessful

4.3. IMPLEMENTATION

1. **Speech Emotion Detection** : To recognize emotional states from voice input, we used the Toronto Emotional Speech Set (TESS) dataset, which includes over 2,800 audio samples categorized into seven emotions: angry, disgust, fear, happy, pleasant surprise, sad, and neutral. Audio preprocessing involved extracting MFCC features, which were then fed into a deep learning model trained to classify emotional tone. The model outputs the detected emotion along with a confidence score, which helps determine whether the user is in a potential state of distress.

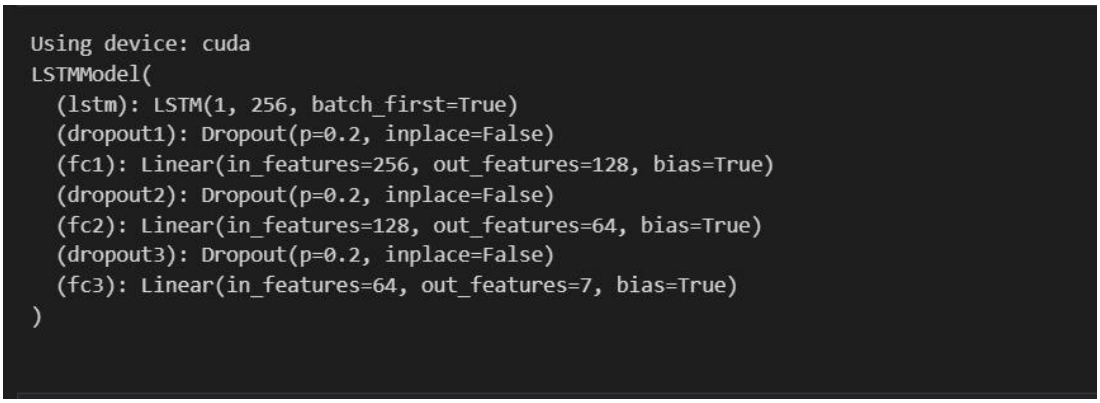


FIGURE 4.3.1. : L.S.T.M. (LONG SHORT-TERM MEMORY) ARCHITECTURE

Building the LSTM Model

- Sequential() – Just like in CNNs, we use Keras' Sequential API to stack our layers from input to output in a linear fashion.
- model.add(LSTM()) – LSTM layers are capable of capturing temporal dependencies. We use units like 64 or 128 and return_sequences=True for stacking multiple LSTM layers.
- model.add(Dropout()) – Dropout helps avoid overfitting by randomly disabling neurons during training.
- model.add(Dense()) – The Dense layer is added at the end to map the learned features to our emotion labels. Softmax is used for multi-class classification.
- model.add(BatchNormalization()) – This layer normalizes the outputs of the previous layer, which helps stabilize and accelerate training. It ensures that the inputs to the next layer are on a consistent scale.
- model.compile() – Before training, we compile the model using a loss function like categorical_crossentropy (for multi-class classification), an optimizer such as Adam.

```
Epoch 1/50, Train Loss: 41.6273, Train Acc: 0.9777, Val Loss: 10.7767, Val Acc: 0.9661
Epoch 2/50, Train Loss: 43.0124, Train Acc: 0.9362, Val Loss: 11.1931, Val Acc: 0.9232
Epoch 3/50, Train Loss: 42.1272, Train Acc: 0.9634, Val Loss: 10.8066, Val Acc: 0.9643
Epoch 4/50, Train Loss: 41.9050, Train Acc: 0.9679, Val Loss: 10.8951, Val Acc: 0.9536
Epoch 5/50, Train Loss: 42.4078, Train Acc: 0.9545, Val Loss: 10.9481, Val Acc: 0.9518
Epoch 6/50, Train Loss: 41.8319, Train Acc: 0.9705, Val Loss: 10.7115, Val Acc: 0.9732
Epoch 7/50, Train Loss: 41.9433, Train Acc: 0.9665, Val Loss: 10.9126, Val Acc: 0.9518
Epoch 8/50, Train Loss: 42.0938, Train Acc: 0.9629, Val Loss: 10.8647, Val Acc: 0.9554
Epoch 9/50, Train Loss: 42.8472, Train Acc: 0.9415, Val Loss: 10.9195, Val Acc: 0.9500
Epoch 10/50, Train Loss: 42.1782, Train Acc: 0.9612, Val Loss: 10.8983, Val Acc: 0.9554
Epoch 11/50, Train Loss: 41.8992, Train Acc: 0.9692, Val Loss: 11.0180, Val Acc: 0.9375
Epoch 12/50, Train Loss: 41.9915, Train Acc: 0.9656, Val Loss: 11.0222, Val Acc: 0.9411
Epoch 13/50, Train Loss: 41.7605, Train Acc: 0.9723, Val Loss: 10.8111, Val Acc: 0.9643
Epoch 14/50, Train Loss: 41.5244, Train Acc: 0.9790, Val Loss: 10.8067, Val Acc: 0.9661
Epoch 15/50, Train Loss: 41.5984, Train Acc: 0.9772, Val Loss: 10.7885, Val Acc: 0.9661
Epoch 16/50, Train Loss: 41.3310, Train Acc: 0.9848, Val Loss: 10.8234, Val Acc: 0.9625
Epoch 17/50, Train Loss: 42.2974, Train Acc: 0.9576, Val Loss: 10.7260, Val Acc: 0.9714
Epoch 18/50, Train Loss: 41.2379, Train Acc: 0.9875, Val Loss: 10.7164, Val Acc: 0.9768
Epoch 19/50, Train Loss: 41.3684, Train Acc: 0.9830, Val Loss: 10.7618, Val Acc: 0.9714
Epoch 20/50, Train Loss: 41.5174, Train Acc: 0.9795, Val Loss: 10.6845, Val Acc: 0.9786
Epoch 21/50, Train Loss: 41.5034, Train Acc: 0.9799, Val Loss: 10.7122, Val Acc: 0.9750
Epoch 22/50, Train Loss: 41.3487, Train Acc: 0.9839, Val Loss: 10.7044, Val Acc: 0.9768
Epoch 23/50, Train Loss: 41.3029, Train Acc: 0.9857, Val Loss: 10.6671, Val Acc: 0.9804
Epoch 24/50, Train Loss: 41.2492, Train Acc: 0.9871, Val Loss: 10.6190, Val Acc: 0.9839
Epoch 25/50, Train Loss: 41.1090, Train Acc: 0.9911, Val Loss: 10.5790, Val Acc: 0.9911
...
Epoch 47/50, Train Loss: 41.0872, Train Acc: 0.9915, Val Loss: 10.6211, Val Acc: 0.9857
Epoch 48/50, Train Loss: 41.5546, Train Acc: 0.9781, Val Loss: 12.9920, Val Acc: 0.7268
Epoch 49/50, Train Loss: 45.1822, Train Acc: 0.8723, Val Loss: 11.0930, Val Acc: 0.9339
Epoch 50/50, Train Loss: 43.0238, Train Acc: 0.9353, Val Loss: 11.1072, Val Acc: 0.9321
```

FIGURE 4.3.2 : L.S.T.M. TRAINING

2. **Speech-to-Text and Panic Word Detection :** For live voice transcription, the system integrates the Whisper model by OpenAI. This model converts real-time audio from the microphone into text. Once transcribed, a simple if-else keyword search is performed to detect panic-related words such as **“help”, “save me”, or “danger”.** If such words are found in the transcript, the system flags the situation as potentially dangerous and triggers the alert mechanism.
3. **Gesture Detection via MediaPipe :** The application uses MediaPipe Hands, a pre-trained model for extracting hand landmarks from video frames. Instead of training a new gesture recognition model, we implemented a rule-based if-else logic system that checks for specific gesture patterns (e.g., raised open palm or repeated waving motion). When such gestures are detected from the live camera feed, the system interprets them as distress signals.

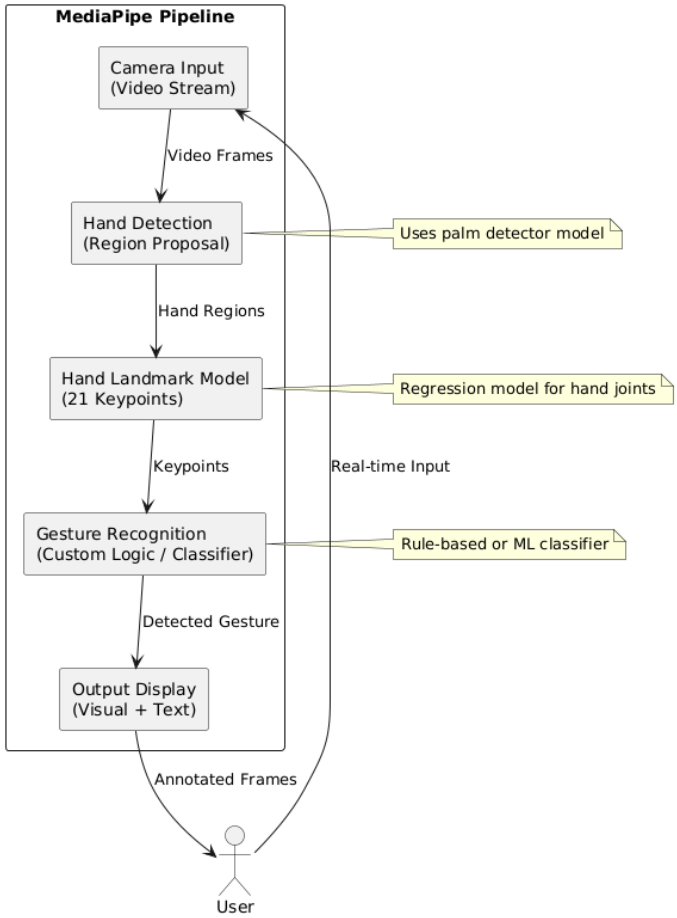


FIGURE 4.3.3 : MEDIAPIPE FOR HAND GESTURE DETECTION

4. **Location-Based Risk Evaluation :** To determine the risk level of the user's current location, we used a static crime dataset from the Indian government open data portal. The user's GPS coordinates are compared against entries in the dataset using basic coordinate matching logic. Depending on the matched area's crime rate, the system classifies it as a low, medium, or high-risk zone. This module is built using a straightforward if-else search algorithm, making it lightweight and suitable for mobile devices.

	Place	Latitude	Longitude	Total_Crime_2022	Zone
0	Cyberabad	17.4933	78.3986	2241	Red
1	Hyderabad	17.3850	78.4867	2165	Red
2	Karimnagar	18.4386	79.1288	283	Orange
3	Khammam	17.2473	80.1437	282	Orange
4	Nizamabad	18.6725	78.0941	576	Orange
5	Rachakonda	17.4239	78.6575	1803	Red
6	Ramagundam	18.7557	79.4710	251	Orange
7	Siddipet	18.0973	78.8487	185	Green
8	Warangal	17.9784	79.5910	385	Orange
9	Adilabad	19.6667	78.5333	128	Green
10	Bhadradri Kothagudem	17.4607	80.6480	146	Green
11	Jagtial	18.7904	78.9237	88	Green
12	Jayashankar Bhupalpalli	18.4385	79.1288	81	Green
13	Jogulamba Gadwal	16.2350	77.8000	271	Orange
14	Kamareddy	18.2643	78.3419	151	Green
15	Kumuram Bheem Asifabad	19.3762	79.3444	53	Green
16	Mahabubabad	17.5980	80.0022	102	Green
17	Mahabubnagar	16.7480	77.9975	220	Orange
18	Medak	18.0456	78.2607	98	Green
19	Nagarkurnool	16.4802	78.4258	252	Orange
20	Nalgonda	17.0541	79.2671	172	Green

FIGURE 4.3.4 : LOCATION-WISE CRIME INCIDENTS

5. **Sensor Data Classification :** For activity recognition, we used time-series data collected from the mobile device's accelerometer and gyroscope sensors. These include over 50 sensor parameters capturing movement, orientation, and position. Rather than training a complex model, we developed a threshold-based if-else classifier to detect physical states like walking, sitting, or sleeping. This allows the system to understand user context and verify whether abnormal motion patterns indicate a potential threat.

Building the Liquid Neural Network Model

For Liquid Neural Networks, we use a dynamic and adaptive recurrent layer inspired by spiking neural systems. These models are particularly good at handling irregular time series and small data.

- model.add(LiquidTimeConstant()) – This is a custom or experimental layer mimicking biological neuron behavior, ideal for real-time adaptive decision making.
- model.add(Dense()) – After temporal dynamics are processed, we use a Dense layer for classification or regression, depending on whether the task is emotion recognition, activity detection, or anomaly prediction.
- Liquid Neural Networks are ideal for adaptive learning from sensor data, especially when the data is non-linear and dynamic.

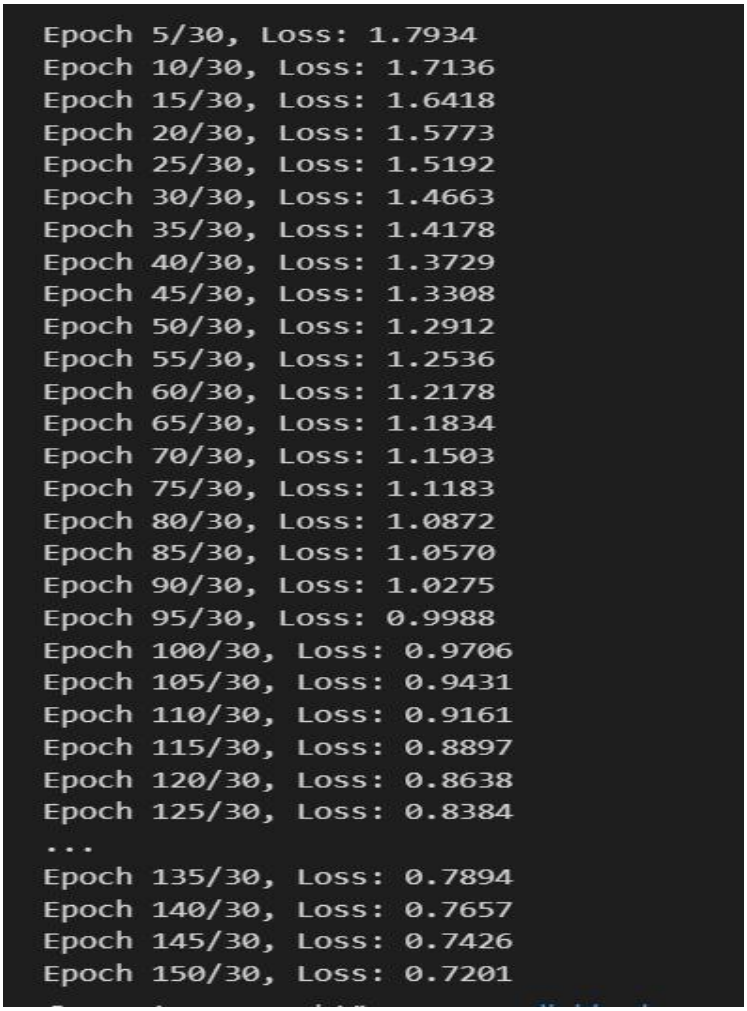


FIGURE 4.3.5 : TRAINING OF SENSOR MODEL

6. **System Integration and Response Mechanism :** Each of the above modules—emotion detection, speech transcription, gesture analysis, location checking, and sensor activity classification—runs independently but feeds into a common decision-making block built on if-else logic. This block evaluates:
    a. Emotional distress

b. Detected panic words

c. Suspicious gestures

d. High-risk location

e. Unusual movement

If one or more threat conditions are satisfied, the system automatically:

- Sends an SOS alert to emergency contacts with the user's location.
- Starts video and audio recording as evidence.
- Displays a warning or safe location suggestion on-screen.

This rule-based integration ensures low computational overhead, real-time performance, and interpretability, making it practical for deployment on smartphones and edge devices.

4.4 RESULTS

1. SENSOR MODEL

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyAccJerkMag-meanFreq()	fBodyBodyAccJerkMag-skewness()	fBodyBodyAccJerkMag-kurtosis()
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	0.346989	-0.516080	-0.802760
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.532061	-0.624871	-0.900160
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.660795	-0.724697	-0.928539
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.678921	-0.701131	-0.909639
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.559058	-0.528901	-0.858933
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
47	0.299665	-0.057193	-0.181233	-0.195387	0.039905	0.077078	-0.282301	0.043616	0.060410	0.210795	...	-0.066650	-0.076714	-0.426588
48	0.273853	-0.007749	-0.147468	-0.235309	0.004816	0.059280	-0.322552	-0.029456	0.080585	0.117440	...	-0.193946	-0.151879	-0.509448
49	0.273387	-0.017011	-0.045022	-0.218218	-0.103822	0.274533	-0.304515	-0.098913	0.332584	0.043999	...	0.007099	0.331457	0.083984
50	0.289654	-0.018843	-0.158281	-0.219139	-0.111412	0.268893	-0.310487	-0.068200	0.319473	0.101702	...	-0.109902	0.038175	-0.305554
51	0.351503	-0.012423	-0.203867	-0.269270	-0.087212	0.177404	-0.377404	-0.038678	0.229430	0.269013	...	-0.242448	-0.321950	-0.768542

2 rows × 350 columns

FIGURE 4.4.1.1. : TRAINING DATA FRAME

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle(tBodyAccMean,gravity)	angle(tBodyAccJer
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249	-0.674302	-0.894088	...	-0.705974	0.006462	
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401	-0.945823	-0.894088	...	-0.594944	-0.083495	
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735	-0.963483	-0.939260	...	-0.640736	-0.034956	
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471	-0.968897	-0.938610	...	-0.736124	-0.017067	
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953	-0.977346	-0.938610	...	-0.846595	-0.002223	

5 rows × 563 columns

FIGURE 4.4.1.2. : TEST DATA FRAME

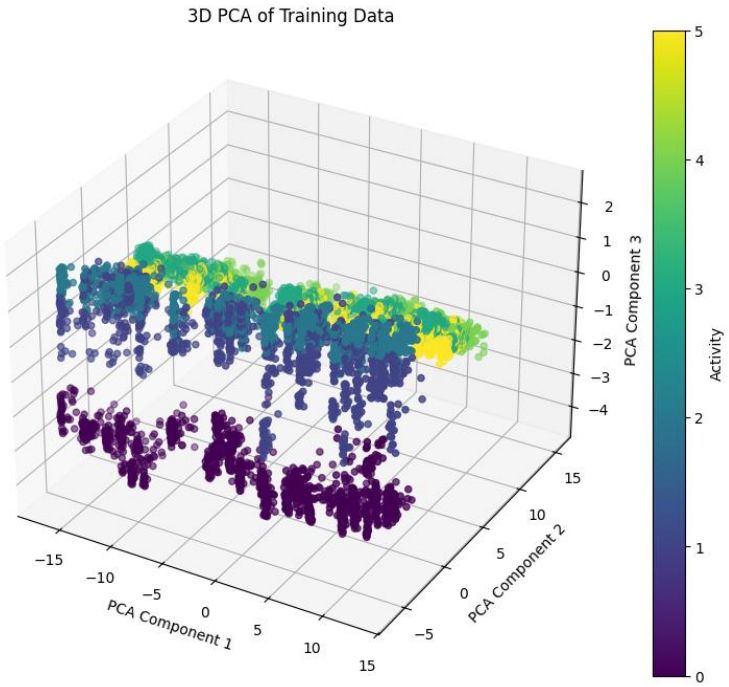


FIGURE 4.4.1. 3: 3D PCA OF TRAINING DATA

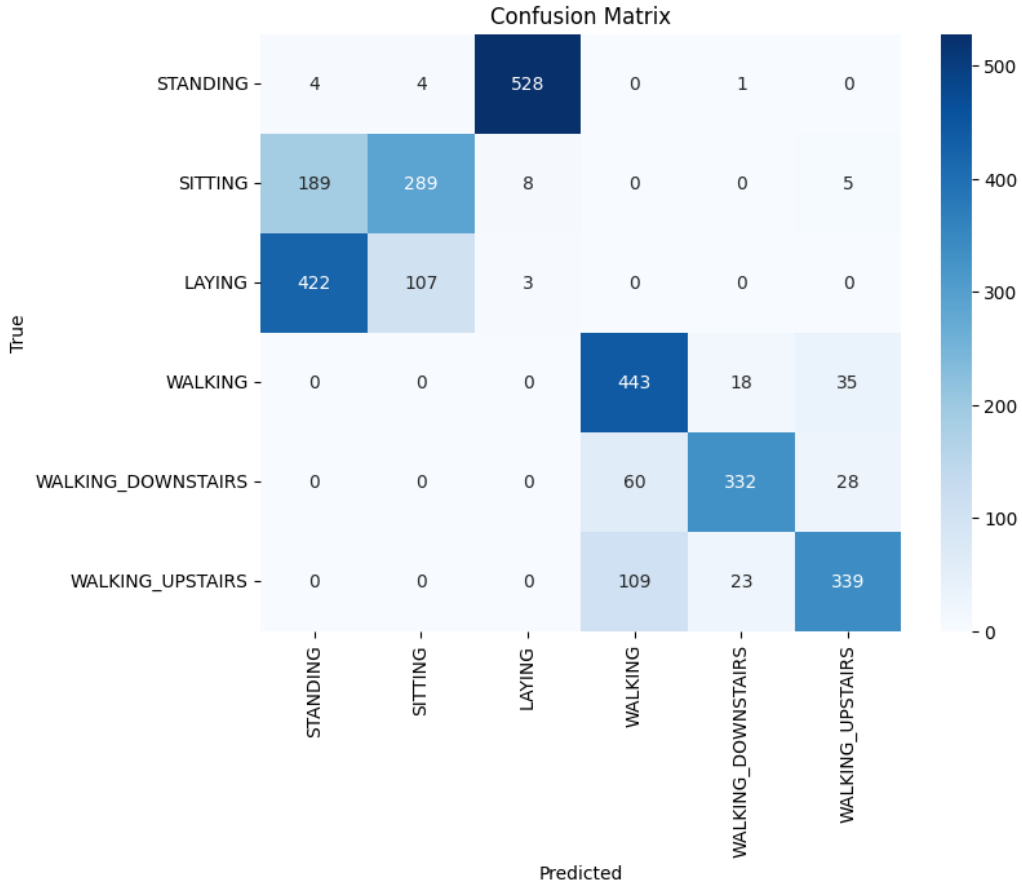


FIGURE 4.4.1.4. : CONFUSION MATRIX

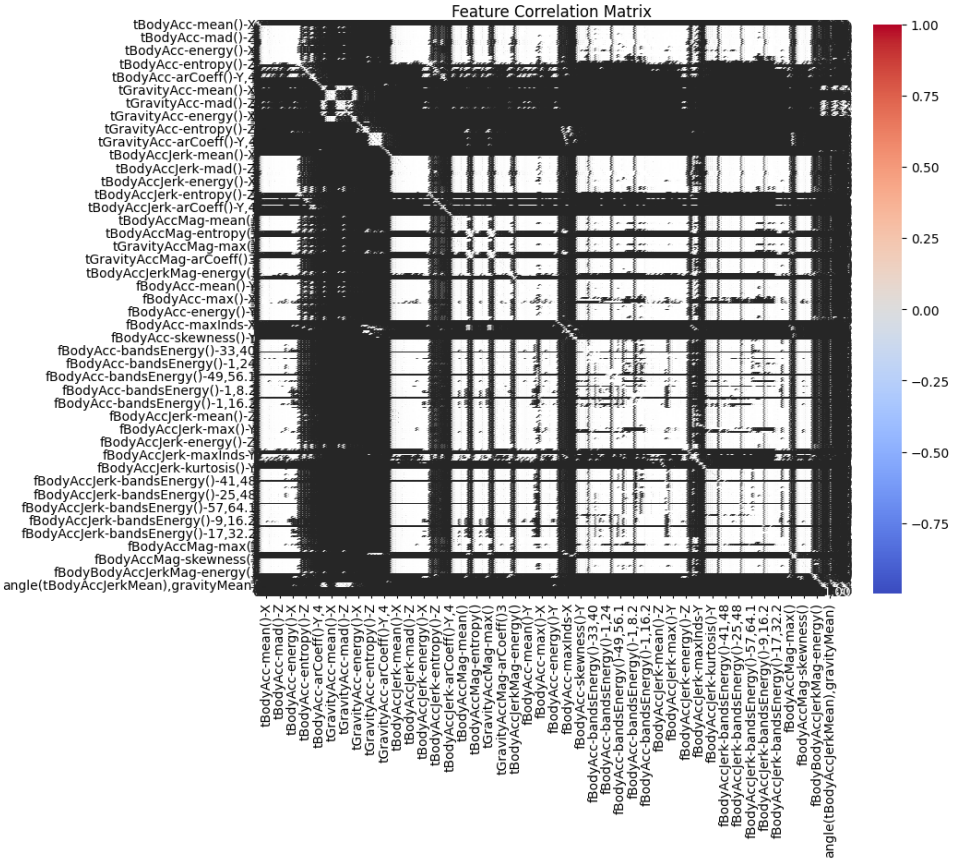


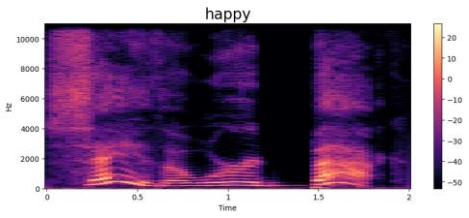
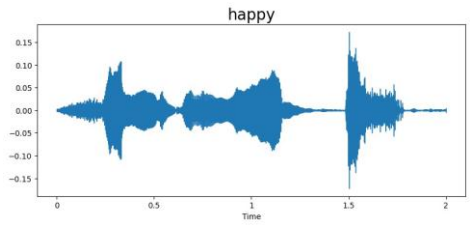
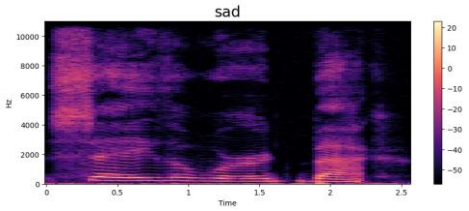
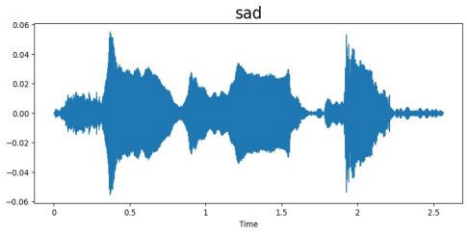
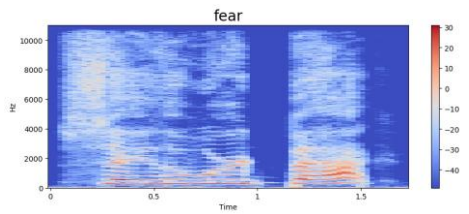
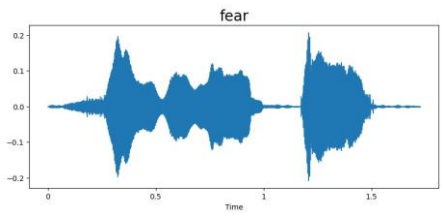
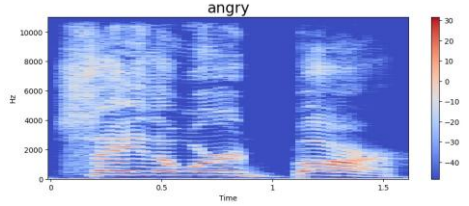
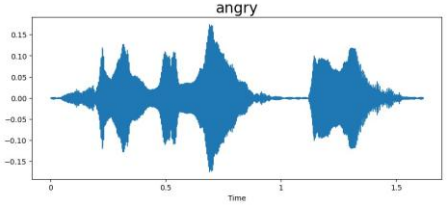
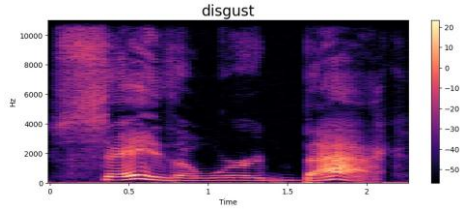
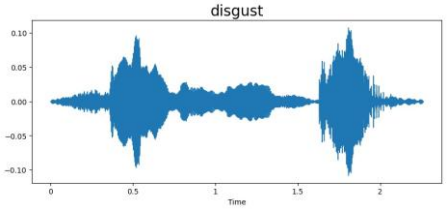
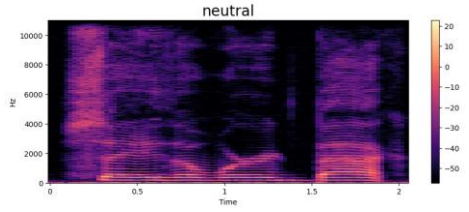
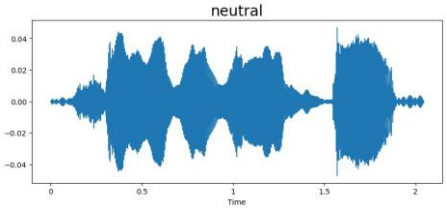
FIGURE 4.4.1.5. : CORRELATION MATRIX

```
Accuracy: 0.7815
F1 Score: 0.7781

Classification Report:
                    precision    recall  f1-score   support

          STANDING       0.74      0.75      0.74       532
           SITTING       0.71      0.69      0.70       491
            LAYING       0.96      0.96      0.96       537
           WALKING       0.72      0.89      0.79       496
WALKING_DOWNSTAIRS       0.82      0.55      0.66       420
  WALKING_UPSTAIRS       0.76      0.81      0.79       471

          accuracy                           0.78      2947
         macro avg       0.78      0.77      0.77      2947
      weighted avg       0.79      0.78      0.78      2947
```

FIGURE 4.4.1.6. : CLASSIFICATION REPORT

2. SPEECH EMOTION DETECTION

TABLE 4.4.2: SPEECH EMOTION TRAINING DATA

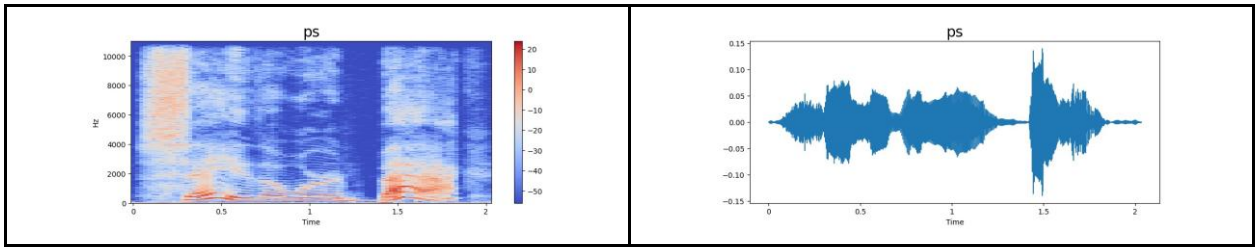


FIGURE 4.4.2.1 : TRAINING AND VALIDATION OF L.S.T.M.

3. DECIBEL METER

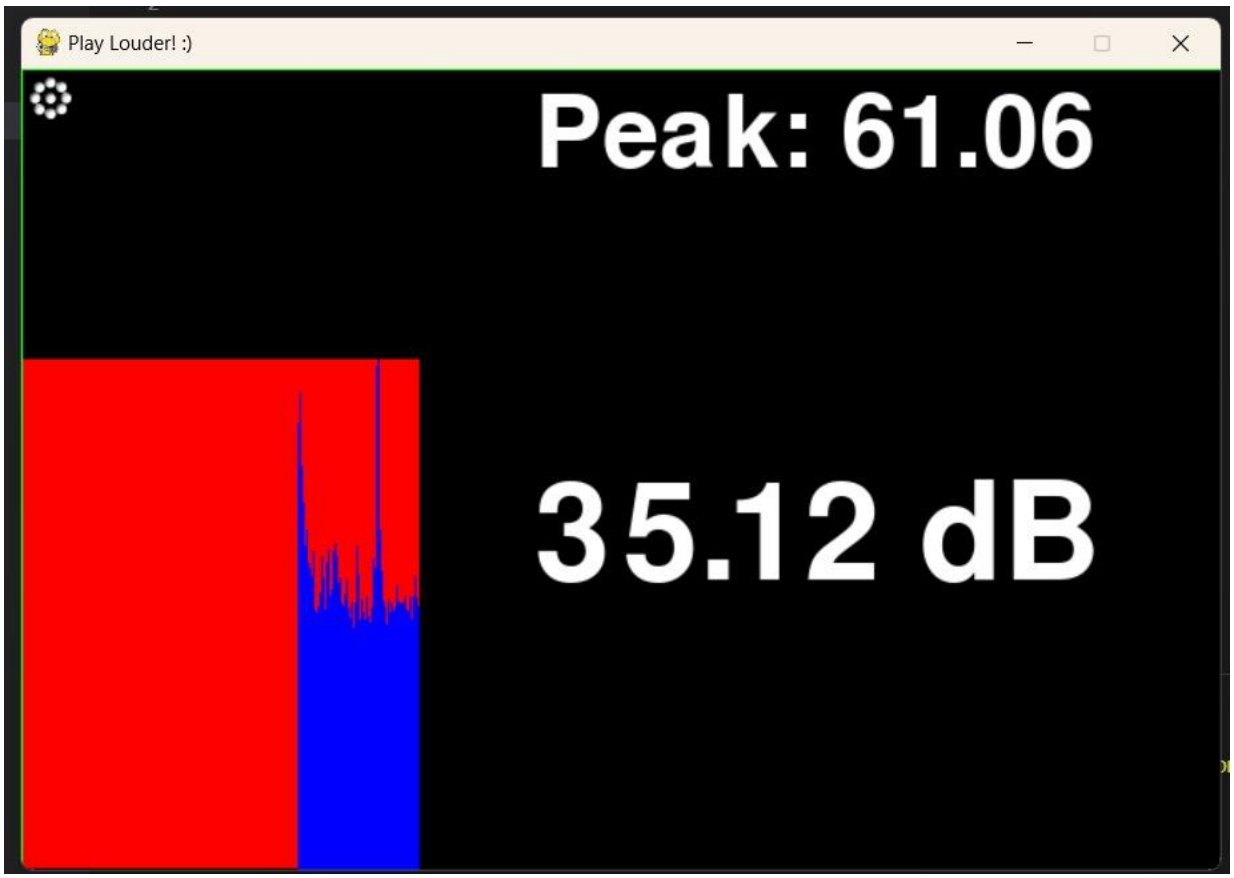


FIGURE 4.4.3.1. : DECIBEL METER

4.5. OUTPUT SCREENS

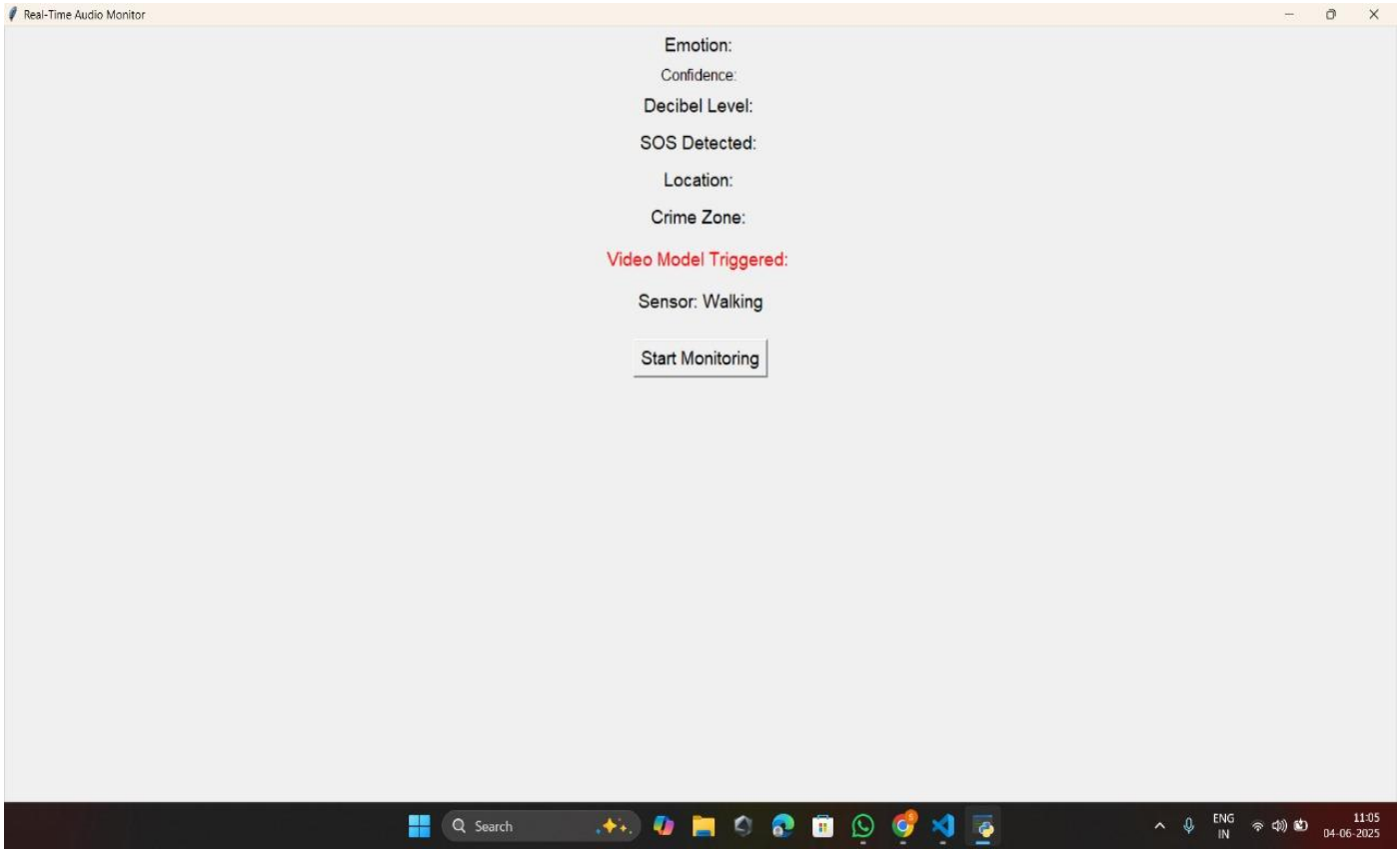


FIGURE 4.5.1 :OUTPUT SCREEN 1

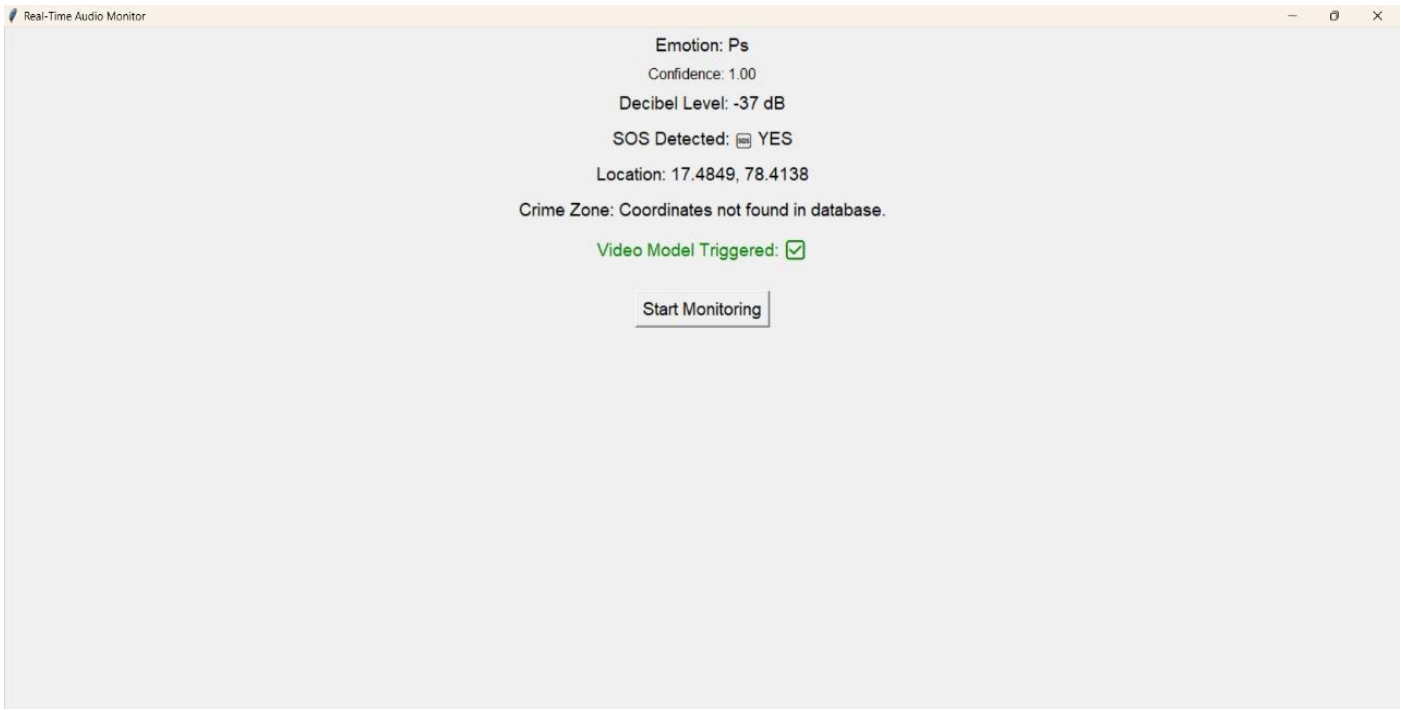


FIGURE 4.5.2 :OUTPUT SCREEN 2

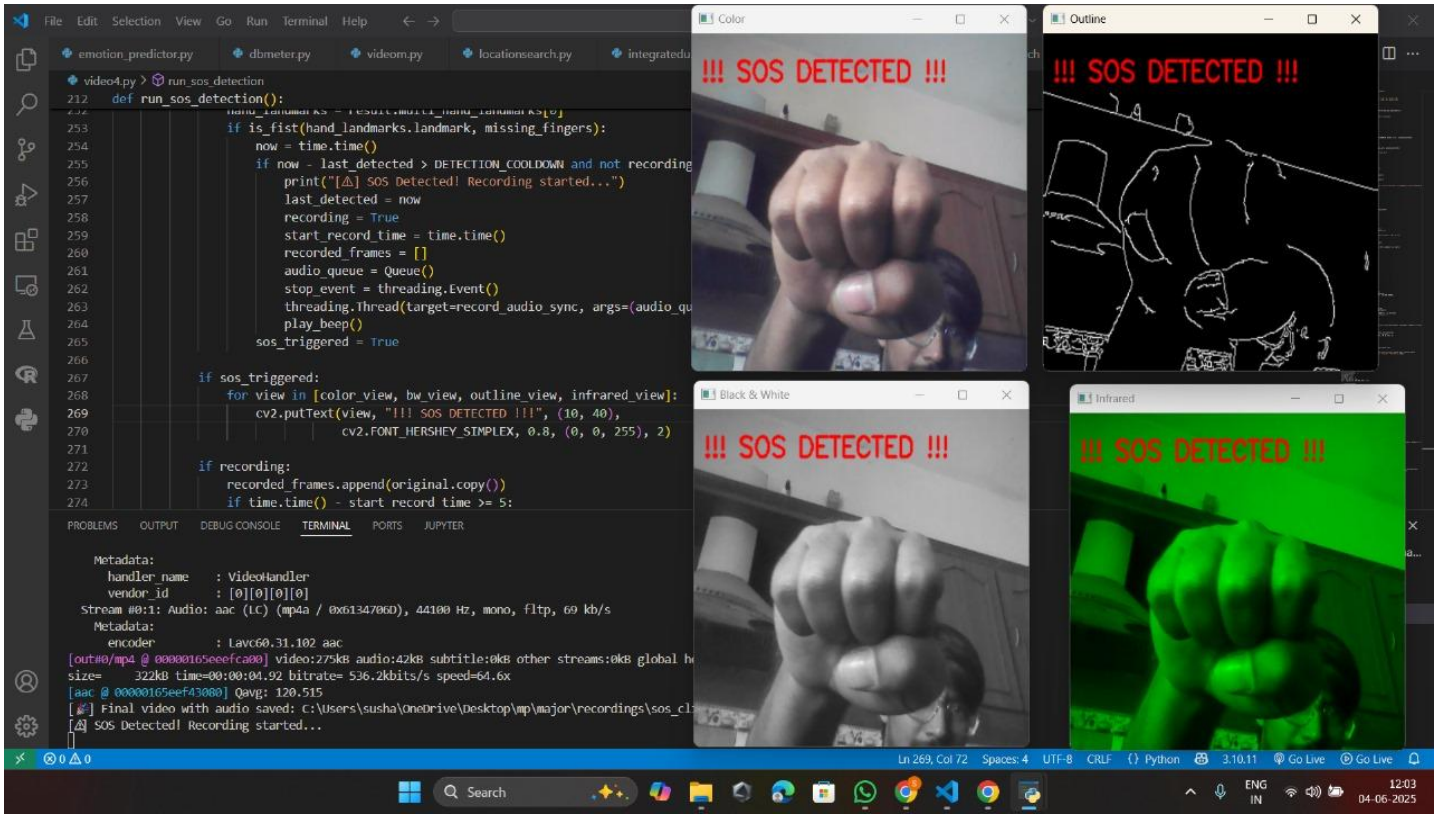


FIGURE 4.5.3. :OUTPUT SCREEN 3

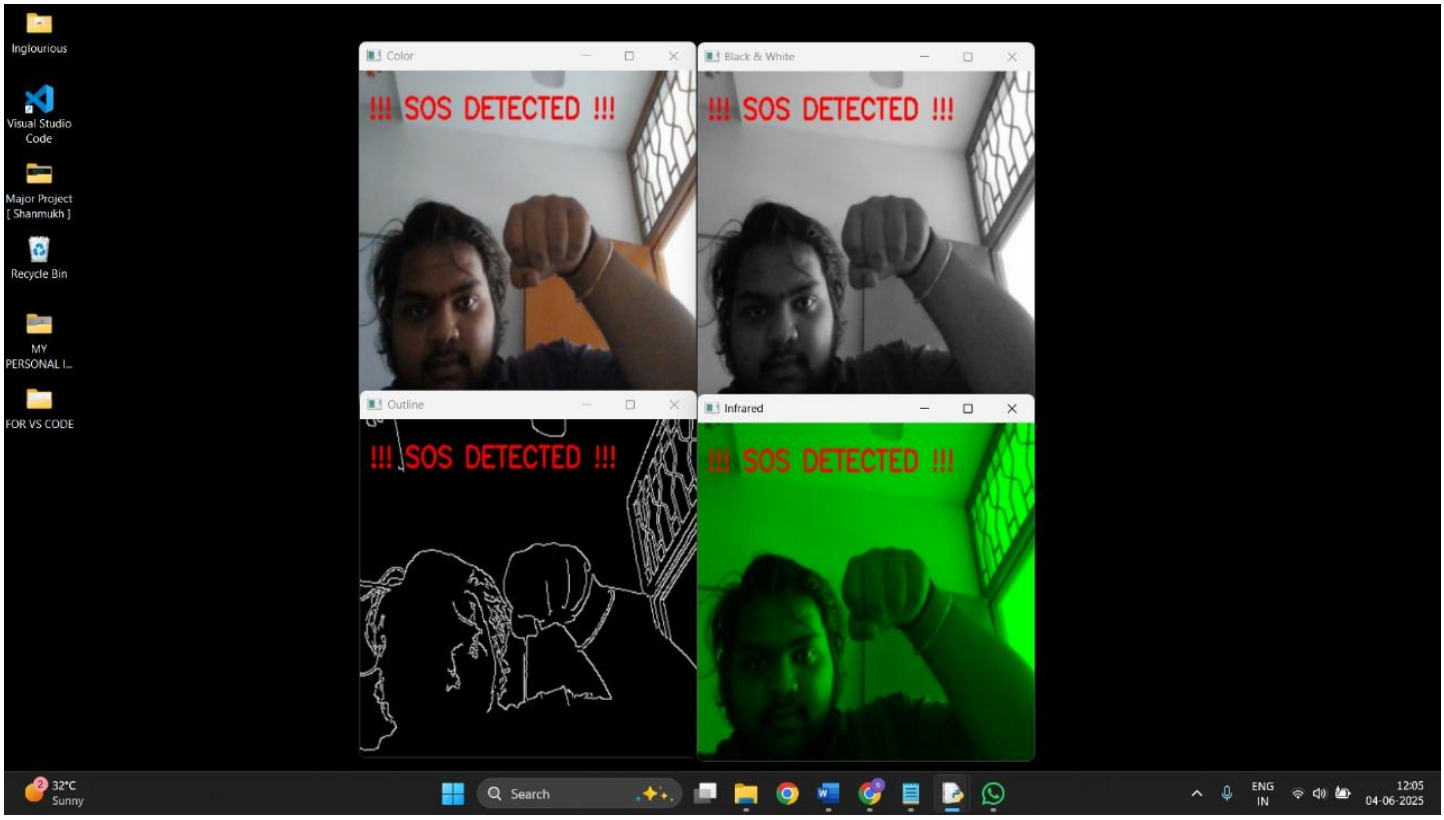


FIGURE 4.5.4. :OUTPUT SCREEN 4

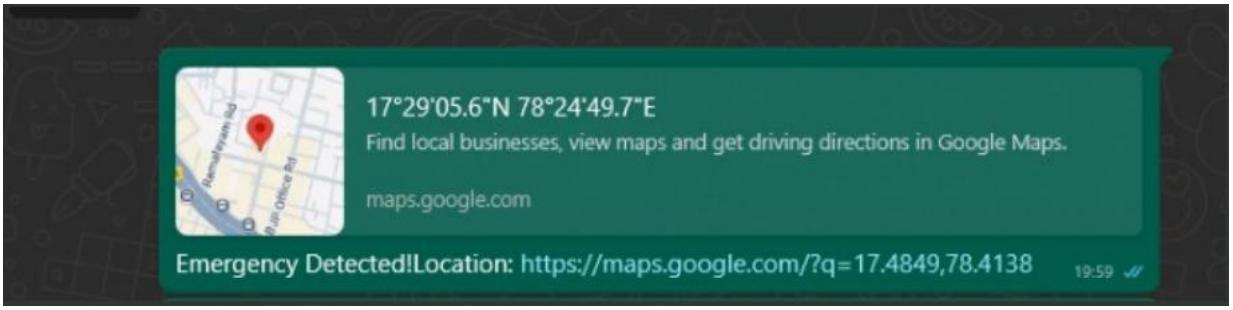


FIGURE 4.5.5. : OUTPUT SCREEN 5

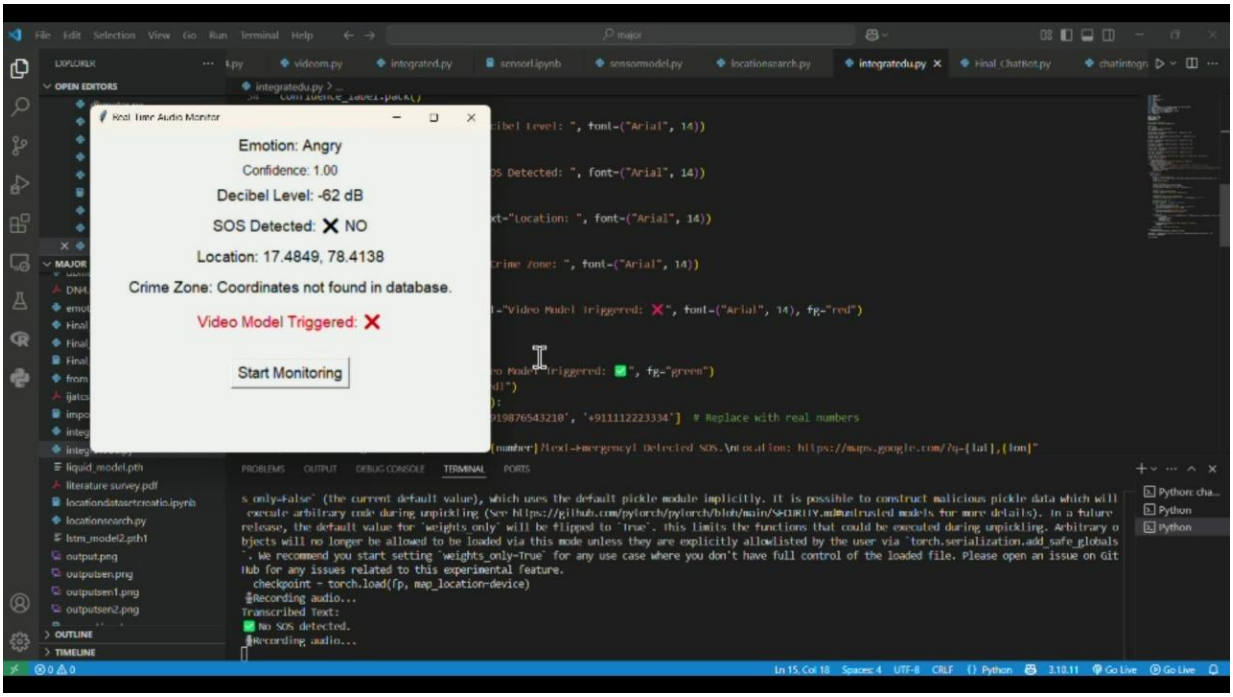


FIGURE 4.5.6. : OUTPUT SCREEN 6

5. CONCLUSION

We presented an AI-based mobile safety application designed to autonomously detect and respond to emergencies, with a special focus on enhancing women's safety. The system integrates multiple modules including speech emotion recognition, panic keyword detection using Whisper transcription, gesture recognition via MediaPipe, location-based crime risk assessment using government datasets, and sensor-based activity classification. Unlike traditional systems that rely on manual input, our approach combines real-time multimodal data and rule-based logic to trigger automated SOS alerts, capture evidence, and notify emergency contacts.

The application uses lightweight, threshold-driven if-else logic in place of complex algorithms, ensuring faster processing on mobile devices. Our modular and integrated design enables proactive response even when the user is unable to act. In future work, we aim to enhance gesture recognition using deep learning, improve context understanding through multimodal fusion, and optimize model deployment for edge devices to ensure wider accessibility and real-time protection in vulnerable scenarios.

5.1. FUTURE SCOPE

1. The future enhancements that can be projected for the project are:
2. Integration of advanced deep learning models for gesture recognition and emotion analysis.
3. Optimization for edge devices and deployment as a lightweight, fully functional mobile application.
4. Inclusion of voice-based assistant for hands-free user interaction in emergencies.
5. Real-time synchronization with local law enforcement databases for quicker threat validation.
6. Multi-language support for accessibility across diverse user groups.
7. Enhanced data visualization for better threat pattern analysis and user awareness.

6. REFERENCES

1. Latif, S., et al. (2020). Survey of deep learning-based techniques for speech emotion recognition. IEEE Access, 7, 49321–49335.
2. Zhang, Y., et al. (2018). Speech emotion recognition with transferred learning from speaker verification. Interspeech, 2456–2460.
3. Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.
4. Goodfellow, I., et al. (2016). Deep Learning. MIT Press.
5. Malekzadeh, M., et al. (2019). Protecting sensory data against sensitive inferences. Proceedings of the 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 193–204.
6. Shi, F., et al. (2020). A survey on deep learning for anomaly detection. ACM Computing Surveys (CSUR), 53(1), 1–36.
7. Gerber, M. S. (2014). Predicting crime using Twitter and kernel density estimation. Decision Support Systems, 61, 115–125.
8. Wang, Z., et al. (2019). DeepCrime: Attentive hierarchical recurrent networks for crime prediction. Pacific-Asia Conference on Knowledge Discovery and Data Mining, 400–412.
9. Atrey, P. K., et al. (2010). Multimodal fusion for multimedia analysis: A survey. Multimedia Systems, 16(6), 345–379.
10. Baltrusaitis, T., et al. (2018). Multimodal machine learning: A survey and taxonomy. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(2), 423–443.
11. Rahimi, A. M., et al. (2018). Mobile edge computing: A survey and taxonomy. IEEE Communications Surveys & Tutorials, 20(1), 359–380.
12. Shi, W., Dustdar, S. (2016). The promise of edge computing. Computer, 49(5), 78–81.
13. Farooq, U., et al. (2015). A review of emergency alert systems for developing regions. Journal of Network and Computer Applications, 47, 151–161.

7. APPENDIX

7.1. SPEECH EMOTION DETECTION CODE

```
import torch
import torch.nn as nn
import librosa
import numpy as np

SAMPLE_RATE = 16000
NUM_MFCC = 40

label_map = {
    0: "Disgust",
    1: "Happy",
    2: "Ps",
    3: "Fear",
    4: "Angry",
    5: "Sad",
    6: "Neutral"
}

class LSTMModel(nn.Module):
    def __init__(self):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size=1, hidden_size=256, batch_first=True)
        self.dropout1 = nn.Dropout(0.2)
        self.fc1 = nn.Linear(256, 128)
        self.dropout2 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, 64)
        self.dropout3 = nn.Dropout(0.2)
        self.fc3 = nn.Linear(64, 7)

    def forward(self, x):
        out, _ = self.lstm(x)
```

```
        out = self.dropout1(out[:, -1, :])
        out = torch.relu(self.fc1(out))
        out = self.dropout2(out)
        out = torch.relu(self.fc2(out))
        out = self.dropout3(out)
        out = self.fc3(out)
       return torch.softmax(out, dim=1)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = LSTMModel().to(device)
model.load_state_dict(torch.load('lstm_model2.pth1',
map_location=device,weights_only=True))
model.eval()

def predict_emotion(audio_array, sample_rate=SAMPLE_RATE):
    mfcc = librosa.feature.mfcc(y=audio_array, sr=sample_rate, n_mfcc=NUM_MFCC)
    mfcc = np.mean(mfcc.T, axis=0).reshape(NUM_MFCC, 1)
    input_tensor = torch.tensor(mfcc, dtype=torch.float32).unsqueeze(0).to(device)

    with torch.no_grad():
        output = model(input_tensor)
        pred_class = torch.argmax(output, dim=1).item()
        confidence = output[0][pred_class].item()

    return label_map[pred_class], confidence
```

7.2. CAMERA MODEL CODE

```
import cv2
import mediapipe as mp
import time
import os
import numpy as np
import threading
```

```
import winsound
from datetime import datetime
import warnings
import sounddevice as sd
import wave
import subprocess
from queue import Queue

warnings.filterwarnings("ignore", message=r"SymbolDatabase\.GetPrototype\(\) is
deprecated.*", category=UserWarning)

mp_hands = mp.solutions.hands

available_fingers = ["thumb", "index", "middle", "ring", "pinky"]
missing_fingers = []

print("Which fingers do you not have? (choose from: thumb, index, middle, ring, pinky)")
print("Enter up to 2, separated by commas (or press Enter if all are present): ")
response = input("Missing fingers: ").lower().strip()

if response:
    for finger in response.split(","):
        f = finger.strip()
        if f in available_fingers and f not in missing_fingers:
            missing_fingers.append(f)
        if len(missing_fingers) == 2:
            break

finger_tip_map = {"thumb": 4, "index": 8, "middle": 12, "ring": 16, "pinky": 20}
finger_pip_map = {"thumb": 3, "index": 6, "middle": 10, "ring": 14, "pinky": 18}

def is_fist(landmarks, missing_fingers):
    folded = []
    for finger in available_fingers:
```

```
        if finger in missing_fingers:
            continue
        if landmarks[finger_tip_map[finger]].y > landmarks[finger_pip_map[finger]].y:
            folded.append(finger)
    return len(folded) == (5 - len(missing_fingers))

SAVE_DIR = "C:\\Users\\Lenovo\\OneDrive\\Desktop\\Major Project [ Shanmukh ]\\Camera
Model\\vids"
os.makedirs(SAVE_DIR, exist_ok=True)

AUDIO_RATE = 44100
CHANNELS = 1

def record_audio_sync(queue: Queue, stop_event):
    audio_frames = []

    def callback(indata, frames, time_info, status):
        if stop_event.is_set():
            raise sd.CallbackStop()
        audio_frames.append(indata.copy())

    with sd.InputStream(samplerate=AUDIO_RATE, channels=CHANNELS, dtype='int16',
callback=callback):
        while not stop_event.is_set():
            time.sleep(0.1)

    audio = np.concatenate(audio_frames)
    queue.put(audio)

def merge_audio_video(temp_video_path, audio_data, output_path, sample_rate=44100):
    temp_audio_path = temp_video_path.replace(".mp4", "_temp.wav")
    with wave.open(temp_audio_path, 'wb') as wf:
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(2)
```

```
        wf.setframerate(sample_rate)
        wf.writeframes(audio_data.tobytes())

    try:
        subprocess.run([
            "ffmpeg",
            "-y",
            "-i", temp_video_path,
            "-i", temp_audio_path,
            "-c:v", "copy",
            "-c:a", "aac",
            "-strict", "experimental",
            output_path
        ], check=True)
        os.remove(temp_audio_path)
        os.remove(temp_video_path)
        print(f"[🎉] Final video with audio saved: {output_path}")
    except subprocess.CalledProcessError as e:
        print("[❌] FFmpeg merge failed:", e)

def save_clip_with_audio(frames, audio_data, fps=30):
    now = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    temp_video = os.path.join(SAVE_DIR, f"temp_{now}.mp4")
    final_output = os.path.join(SAVE_DIR, f"sos_clip_{now}.mp4")

    h, w = frames[0].shape[:2]
    out = cv2.VideoWriter(temp_video, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
    for f in frames:
        out.write(f)
    out.release()

    merge_audio_video(temp_video, audio_data, final_output)
```

```
def play_beep():
    threading.Thread(target=lambda: winsound.Beep(1000, 3000), daemon=True).start()

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 320)

DETECTION_COOLDOWN = 2
last_detected = 0
recording = False
recorded_frames = []
start_record_time = 0
fps = cap.get(cv2.CAP_PROP_FPS) or 30

with mp_hands.Hands(
    max_num_hands=1,
    min_detection_confidence=0.8,
    min_tracking_confidence=0.7,
    model_complexity=1
) as hands:
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            continue

        frame = cv2.flip(frame, 1)
        original = frame.copy()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 70, 200)

        color_view = original.copy()
        bw_view = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
        outline_view = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
        green_channel = cv2.normalize(gray, None, 0, 255, cv2.NORM_MINMAX)
```

```
        infrared_view = np.zeros_like(original)
        infrared_view[:, :, 1] = green_channel

        rgb = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)
        result = hands.process(rgb)
        sos_triggered = False

        if result.multi_hand_landmarks:
            hand_landmarks = result.multi_hand_landmarks[0]
            if is_fist(hand_landmarks.landmark, missing_fingers):
                now = time.time()
                if now - last_detected > DETECTION_COOLDOWN and not recording:
                    print("[⚠] SOS Detected! Recording started...")
                    last_detected = now
                    recording = True
                    start_record_time = time.time()
                    recorded_frames = []
                    audio_queue = Queue()
                    stop_event = threading.Event()
                    threading.Thread(target=record_audio_sync, args=(audio_queue, stop_event),
daemon=True).start()
                    play_beep()
                sos_triggered = True

        if sos_triggered:
            for view in [color_view, bw_view, outline_view, infrared_view]:
                cv2.putText(view, "!!! SOS DETECTED !!!", (10, 40),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)

        if recording:
            recorded_frames.append(original.copy())
            if time.time() - start_record_time >= 5:
                stop_event.set()
```

```
            time.sleep(0.2)
            audio_data = audio_queue.get()
            save_clip_with_audio(recorded_frames, audio_data, fps)
            recording = False

    views = [cv2.resize(v, (360, 360)) for v in [color_view, bw_view, outline_view,
infrared_view]]
    for title, view in zip(["Color", "Black & White", "Outline", "Infrared"], views):
        cv2.imshow(title, view)

    if cv2.waitKey(1) in [27, ord('q')]:
        break

cap.release()
cv2.destroyAllWindows()
```

7.3. LOCATION MODEL CODE

```
import pandas as pd

# Offline data for Telangana places
data = {
  "Place": [
    "Cyberabad", "Hyderabad", "Karimnagar", "Khammam", "Nizamabad", "Rachakonda",
"Ramagundam",
    "Siddipet", "Warangal", "Adilabad", "Bhadradri Kothagudem", "Jagtial", "Jayashankar
Bhupalpalli",
    "Jogulamba Gadwal", "Kamareddy", "Kumuram Bheem Asifabad", "Mahabubabad",
"Mahabubnagar",
    "Medak", "Nagarkurnool", "Nalgonda", "Nirmal", "Rajanna Sircilla", "Sanga Reddy",
    "Suryapet", "Vikarabad", "Wanaparthy", "RP Secunderabad"
  ],
  "Latitude": [
    17.4933, 17.3850, 18.4386, 17.2473, 18.6725, 17.4239, 18.7557,
```

```
        18.0973, 17.9784, 19.6667, 17.4607, 18.7904, 18.4385,
        16.2350, 18.2643, 19.3762, 17.5980, 16.7480,
        18.0456, 16.4802, 17.0541, 19.0968, 18.3852, 17.6280,
        17.1310, 17.3410, 16.3612, 17.4385
    ],
    "Longitude": [
        78.3986, 78.4867, 79.1288, 80.1437, 78.0941, 78.6575, 79.4710,
        78.8487, 79.5910, 78.5333, 80.6480, 78.9237, 79.1288,
        77.8000, 78.3419, 79.3444, 80.0022, 77.9975,
        78.2607, 78.4258, 79.2671, 78.3441, 78.8095, 78.0822,
        79.6237, 77.9040, 78.0667, 78.4983
    ],
    "Total_Crime_2022": [
        2241, 2165, 283, 282, 576, 1803, 251,
        185, 385, 128, 146, 88, 81,
        271, 151, 53, 102, 220,
        98, 252, 172, 63, 51, 210,
        155, 64, 192, 675
    ]
}


df = pd.DataFrame(data)

def assign_zone(crime):
    if crime >= 1000:
        return 'Red'
    elif crime >= 200:
        return 'Orange'
    else:
        return 'Green'

def get_crime_by_coords(lat, lon):
    match = df[(df['Latitude'] == lat) & (df['Longitude'] == lon)]
```

```
    if not match.empty:
        return int(match.iloc[0]["Total_Crime_2022"])
    else:
        return -1

def get_zone_by_coords(lat, lon):
    crime = get_crime_by_coords(lat, lon)
    if crime == -1:
        return "Coordinates not found in database."
    return assign_zone(crime)
```

7.4. SPEECH RECOGNITION MODEL CODE

```
'''import sounddevice as sd
import numpy as np
import time
import whisper
import scipy.io.wavfile
import io

SAMPLE_RATE = 16000
DURATION = 5  # seconds
NUM_MFCC = 40
model = whisper.load_model("base")
try:
    while True:
        print("\n Listening for 5 seconds...")
        audio = sd.rec(int(5 * 16000), samplerate=16000, channels=1, dtype='float32')
        sd.wait()
        audio = audio.flatten()
        result = model.transcribe(audio)
        text=result["text"].strip()

        if "help"in text.lower():
            print("sosdtected")
```

```
        else:
            print(text)
except KeyboardInterrupt:
    print("\n Stopped by user.")'"
import whisper
import numpy as np

# Load the Whisper model once
whisper_model = whisper.load_model("base")

def detect_sos_with_whisper(audio_array: np.ndarray, sample_rate: int = 16000) -> bool:
    """
    Detects if the word 'help' is present in the given audio using OpenAI's Whisper.

    Parameters:
        audio_array (np.ndarray): The audio waveform (mono) as a NumPy array.
        sample_rate (int): Sample rate of the audio.

    Returns:
        bool: True if 'help' is detected, False otherwise.
    """
    # Ensure float32 and shape [length]
    if audio_array.ndim > 1:
        audio_array = audio_array.flatten()
    if audio_array.dtype != np.float32:
        audio_array = audio_array.astype(np.float32)

    # Transcribe using Whisper
    result = whisper_model.transcribe(audio_array, fp16=False, language="en", task="transcribe")
    text = result["text"].strip()
    print(f"Transcribed Text: {text}")

    if "help" in text.lower():
```

```
        print("🆘 SOS detected!")
        return True
    else:
        print("✅ No SOS detected.")
        return False
```

7.5 SENSOR MODEL CODE

```
import torch
import torch.nn as nn
import numpy as np

class LSTMModel(nn.Module):
    def __init__(self, input_size=6, hidden_size=128, output_size=5, num_layers=2):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        out, _ = self.lstm(x, (h0, c0))
        return self.fc(out[:, -1, :])

# Load model once
model = LSTMModel()
model.load_state_dict(torch.load("liquid_model"))
model.eval()

def predict(input_list):

    input_array = np.array([input_list], dtype=np.float32)
    input_tensor = torch.from_numpy(input_array)
```

```
    with torch.no_grad():
        output = model(input_tensor)
        prediction = torch.argmax(output, dim=1).item()
        return prediction
```

7.6. DECIBEL METER CODE

```
import numpy as np

def calculate_decibel(audio_data: np.ndarray) -> int:

    if audio_data is None or len(audio_data) == 0:
        return 0

    # Root Mean Square (RMS) of the signal
    rms = np.sqrt(np.mean(audio_data**2))

    # Avoid log of zero
    if rms <= 1e-10:
        return 0

    # Convert to decibels
    db = 20 * np.log10(rms)

    # Normalize relative to full-scale (0 dBFS = 1.0 max in float32)
    dbfs = int(db)
    return dbfs
```

7.7. INTEGRATED CODE

```
import tkinter as tk
import threading
import time
import sounddevice as sd
import numpy as np
import torch
```

```
import librosa
import whisper
import geocoder
from locationsearch import get_zone_by_coords, get_crime_by_coords
from emotion_predictor import predict_emotion
from dbmeter import calculate_decibel
from speechreconw import detect_sos_with_whisper
from videom import run_sos_detection
import webbrowser

#  constants
SAMPLE_RATE = 16000
RECORD_SECONDS = 5
TRIGGERED = False

# Load whisper model once
whisper_model = whisper.load_model("base")

# GUI Setup
root = tk.Tk()
root.title("Real-Time Audio Monitor")
root.geometry("500x400")

emotion_label = tk.Label(root, text="Emotion: ", font=("Arial", 14))
emotion_label.pack(pady=5)

confidence_label = tk.Label(root, text="Confidence: ", font=("Arial", 12))
confidence_label.pack()

db_label = tk.Label(root, text="Decibel Level: ", font=("Arial", 14))
db_label.pack(pady=5)

sos_label = tk.Label(root, text="SOS Detected: ", font=("Arial", 14))
sos_label.pack(pady=5)
```

```
location_label = tk.Label(root, text="Location: ", font=("Arial", 14))
location_label.pack(pady=5)

zone_label = tk.Label(root, text="Crime Zone: ", font=("Arial", 14))
zone_label.pack(pady=5)

trigger_label = tk.Label(root, text="Video Model Triggered: ", font=("Arial", 14), fg="red")
trigger_label.pack(pady=10)


sensor_label = tk.Label(root, text="Sensor: Walking", font=("Arial", 14))
sensor_label.pack(pady=5)


def trigger_video_model():
    trigger_label.config(text="Video Model Triggered: ", fg="green")
    print(" Video Model Triggered!")


def monitor():
    global TRIGGERED
    while True:
        print(" Recording audio...")
        audio = sd.rec(int(RECORD_SECONDS * SAMPLE_RATE),
samplerate=SAMPLE_RATE, channels=1, dtype='float32')
        sd.wait()
        audio = audio.flatten()

        # Emotion Detection
        emotion, confidence = predict_emotion(audio)
        emotion_label.config(text=f"Emotion: {emotion}")
        confidence_label.config(text=f"Confidence: {confidence:.2f}")
```

```
        # Decibel Meter
        db_level = calculate_decibel(audio)
        db_label.config(text=f"Decibel Level: {db_level} dB")

        # SOS Detection
        sos_detected = detect_sos_with_whisper(audio)
        sos_label.config(text=f"SOS Detected: {' YES' if sos_detected else ' NO'}")

        # Location and Zone Detection
        g = geocoder.ip('me')
        lat, lon = g.latlng if g.latlng else (None, None)
        if lat and lon:
            zone = get_zone_by_coords(lat, lon)
            location_label.config(text=f"Location: {lat:.4f}, {lon:.4f}")
            zone_label.config(text=f"Crime Zone: {zone}")
        else:
            location_label.config(text="Location: Not Found")
            zone_label.config(text="Crime Zone: Unknown")
        if (emotion in ["Fear", "Sad"] and confidence > 0.6) or (db_level > 60) or sos_detected or
(zone == "Red"):
            if not TRIGGERED:
                trigger_video_model()
                TRIGGERED = True
                run_sos_detection()
        else:
            trigger_label.config(text="Video Model Triggered: ", fg="red")
            TRIGGERED = False
        time.sleep(5)
def start_monitoring():
    threading.Thread(target=monitor, daemon=True).start()
start_button = tk.Button(root, text="Start Monitoring", command=start_monitoring,
font=("Arial", 14))
start_button.pack(pady=20)
root.mainloop()
```