Sign Language Detection

Pranav Irlapale
Department Of Computer
Science and Engineering
DYPCET Kolhapur, India
pranavirlapale@gmail.com

Rushikesh Mantri
Department of Computer
Engineering,
VIIT Pune, India
rushimantri1@gmail.com

Abstract: **"Sign Language Detection"** is a real-time system developed to bridge the communication gap between the deaf-mute community and the wider population by converting sign language gestures into text. The project leverages **MediaPipe** for accurate and efficient hand gesture tracking and integrates it with a lightweight **TensorFlow** model trained on datasets of **Indian Sign Language (ISL)** and **International Sign Language (ASL)** sourced from Kaggle.

The system takes video input, detects and interprets hand gestures frame by frame, and translates them into meaningful text in real time. The frontend is built using **HTML and CSS**, with a backend powered by **Flask** for API integration and **MongoDB** for managing gesture data and user records. Designed to run efficiently even on low-resource systems, this project provides an accessible and scalable solution for enhancing communication for individuals with hearing and speech impairments.

Chapter 1-Introduction

1.1 Introduction

Communication is an essential aspect of human interaction. For individuals with hearing or speech impairments, sign language serves as a primary means of communication. However, sign language is not universally understood by the general public, creating a significant communication barrier. This gap often leads to difficulties in education, employment, healthcare, and social settings for people who use sign language. The project titled "**Conversion of Sign Language to Text**" aims to address this challenge by developing a system that can detect and recognize hand gestures in real-time using computer vision and machine learning techniques, and convert them into human-readable text. By leveraging technologies like MediaPipe, OpenCV, and a CNN-based classifier, we aim to provide an efficient and affordable solution to facilitate communication between the deaf-mute community and society.

1.2 Motivation

India has millions of individuals who are either deaf or have speech impairments. While sign language provides them a way to communicate, most people in society are not trained to understand it. This often leads to the marginalization of these individuals and restricts their ability to participate fully in social and economic activities. The motivation behind this project stems from the desire to use technology as a bridge between these communities and the rest of society.
Furthermore, current solutions for sign language recognition are either limited in capability or too expensive for widespread

use. This project uses a cost-effective, open-source, and easily deployable framework that can run on personal computers without the need for specialized hardware. The integration of computer vision and machine learning provides a scalable way to improve accuracy over time. Our motivation also includes contributing to inclusive digital transformation and creating opportunities for disabled communities.

1.3 Purpose

The purpose of this project is to develop a real-time, webcam-based gesture recognition system that converts hand signs into text. It aims to:

- Provide a communication aid for people with hearing or speech impairments.
- Facilitate understanding of sign language for non-signers.
- Demonstrate the application of machine learning and deep learning in human- computer interaction.
- Create a base system that can be enhanced in the future to support real-time audio or text-to-sign animation.

By achieving this purpose, the project supports inclusivity, accessibility, and the practical application of AI technologies for social good.

1.4 Problem Statement

Deaf and mute individuals struggle with communication due to limited understanding of sign language and inaccurate translation tools. We need to develop a system that quickly and accurately converts sign language from video into text and translates text into animated sign language, improving real-time communication for these individuals.

1.5 Objectives

- To develop a real-time gesture recognition system using webcam input.
- To implement hand landmark detection using MediaPipe for accurate feature extraction.
- To build and train a CNN model using Keras for static gesture classification.
- To integrate the backend model and logic into a user-friendly web application using Streamlit.
- To overlay the detected text on live video using OpenCV, enhancing usability.
- To ensure high accuracy and real-time performance without requiring expensive hardware.
- To create a scalable framework that can be expanded to support more gestures and languages in the future.

Chapter 2- Literature Survey

2.1 Existing System

2.1.1 Referred Journal/Conference Papers

1. **"Real-Time American Sign Language Recognition Using Deep Learning"** – IEEE Conference (2020)

- Utilized CNN and RNN networks to identify ASL letters and words.
- Demonstrated accuracy of \~95% but required GPU-based processing.
- Focused only on ASL and static signs.

2. **"Hand Gesture Recognition Using MediaPipe and Deep Learning"** – Springer (2021)

- Combined landmark-based detection with a neural network classifier.
- Emphasized low-cost implementation using a webcam and MediaPipe.
- Highlighted challenges with real-time dynamic gesture classification.

3. **"Sign Language to Text Translation Using CNN and TensorFlow"** – IJCA (2020)

- Used OpenCV for frame capturing and CNNs for gesture classification.
- Demonstrated practical implementation but limited to digits and a few letters.
- Required user to hold pose for several seconds for accurate recognition.

2.1.2 Elaborate on Existing System Applications / Examples

- **MotionSavvy Uni :-**

MotionSavvy Uni worked by using a special device called Leap Motion to track the movements of your hands. This information was then sent to a special app on a tablet. The app used artificial intelligence to understand the signs you were making and translated them into text or speech. In other words, user would make a sign, and the device would try to figure out what it meant. It would then show user the words or say them out loud.

- **SignAll :-**

SignAll is a technology that can translate sign language into text or speech. It uses cameras to capture sign language and then uses artificial intelligence to understand and translate it. This allows for real-time communication between deaf and hearing people.

- **Hand Talk :-**

It is a mobile app that helps people who use Brazilian Sign Language (Libras) communicate with people who speak Portuguese. It can change spoken Portuguese into Libras signs. It uses a 3D character named Hugo to show the signs.

- **Google's Teachable Machine:**

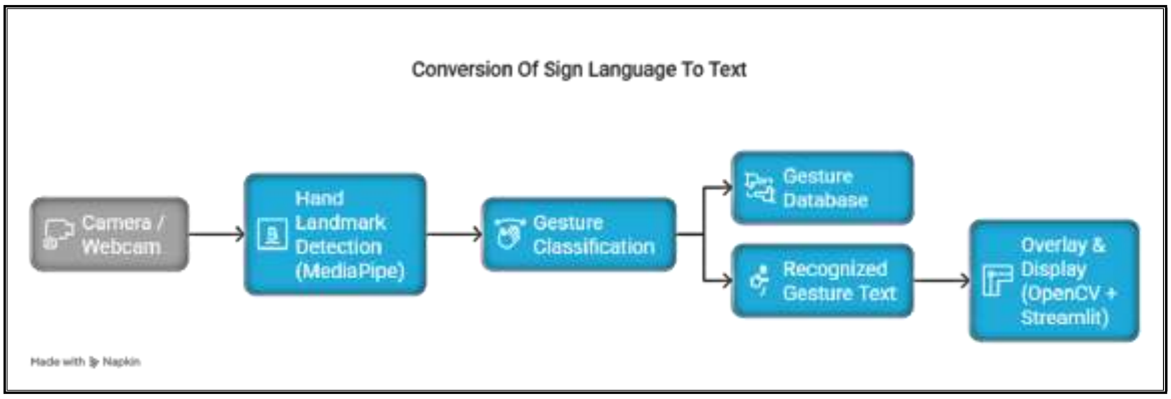
An online tool for training basic image or gesture recognition models using a webcam. It's simple and accessible, but not optimized for detailed hand gesture classification or real- time use in production environments.

2.1.3 Limitations or Challenges in Existing System

- Hardware Limitations: Most existing systems rely on additional hardware like gloves, Kinect sensors, or depth cameras.
- Static Gesture Only: Many models only recognize static signs, which limits vocabulary and expressiveness.
- Limited Languages: Focus is generally on American Sign Language (ASL), with little to no support for other regional variants like Indian Sign Language (ISL).
- Complex Setup: Some applications are difficult to install, configure, or require technical expertise.
- Latency Issues: Real-time systems may experience lag due to high computational demands

2.2 Proposed System with block diagram

2.2.1 Block Diagram


Conversion Of Sign Language To Text
Camera / Webcam
Hand Landmark Detection (MediaPipe)
Gesture Classification
Gesture Database
Recognized Gesture Text
Overlay & Display (OpenCV + Streamlit)
Made with Napkin

Block Diagram

2.2.2 Modules: -

1. Camera/Webcam Input:

- **Function**: Captures live video stream from the user's webcam.
- **Details**:
  - Initiates real-time video feed using OpenCV.
  - Acts as the primary input source for gesture recognition.
  - Continuously sends frames to the hand detection module.

2. Hand Landmark Detection (MediaPipe):

- **Function**: Detects key hand landmarks (joints and finger positions).
- **Details**:
  - Utilizes Google's MediaPipe library for real-time hand tracking.
  - Extracts 21 key landmark points from each detected hand.
  - Converts hand shapes into numerical data for gesture interpretation.
  - Supports both single and multi-hand detection.

3. Gesture Classification:

- **Function**: Identifies the gesture from hand landmark data.
- **Details**:
  - Takes landmark points as input and feeds them to a **classification model** (e.g., CNN or rule-based logic).
  - Differentiates between **static gestures** (like letters) and **dynamic gestures** (like waving).
  - May use thresholding and smoothing techniques to improve accuracy.

- ○ Output is a gesture label (e.g., “A”, “B”, “Hello”).

4. Gesture Database (Optional – Used for Lookup/Mapping):

- **Function**: Stores the mapping between gestures and corresponding text.
- **Details**:
  - ○ Contains predefined mappings between gesture classes and text outputs.
  - ○ Acts as a lookup table to convert recognized gestures to meaningful text or phrases.
  - ○ Easily extendable to include new signs.

5. Recognized Gesture Text:

- **Function**: Converts classified gesture into readable text.
- **Details**:
  - ○ Receives gesture ID/class from the classifier.
  - ○ Uses gesture database to map it to the corresponding text (e.g., “Thank you”).
  - ○ Supports buffering and string formation for continuous text construction.

6. Overlay & Display (OpenCV + Streamlit):

- **Function**: Displays the recognized text and live video.
- **Details**:
  - ○ Uses OpenCV to overlay recognized text on the video frame.
  - ○ Streamlit web interface is used to display live video and results.
  - ○ Provides an interactive and user-friendly UI.
  - ○ May include additional UI components like start/stop buttons or gesture logs.

2.3 Feasibility Study

Objectives:-

- **Evaluate:** Technical and operational feasibility.
- **Check:** Compliance with legal requirements. Feasibility Study include:
- **Technical Feasibility:** Uses open-source Python libraries. MediaPipe and OpenCV handle video and landmark detection efficiently. No need for GPU or advanced hardware.
- **Economic Feasibility:** Completely free to use. All tools and libraries are open- source. Requires only a basic webcam and a personal computer.
- **Operational Feasibility:** Easy-to-use interface. No prior technical knowledge required to use the system. Streamlit offers a smooth UI/UX experience.

Operational feasibility may include:

➢ **User Needs**

- ○ **For:** Deaf/mute individuals, educators, and healthcare professionals.
- ○ **Features:** Real-time sign-to-text and text-to-sign conversion.

➢ **Stakeholders**

○ **Primary Users:** Deaf/mute individuals.

○ **Secondary Users:** Educators, interpreters, and healthcare professionals.

➢ **Maintenance**

○ Regular updates for improving accuracy, fixing bugs, and cloud services for backups and scaling.

• **Legal Feasibility:** MediaPipe, OpenCV, Keras, and Streamlit are open-source under permissive licenses (Apache, MIT). No proprietary dependency issues.

Chapter 3-Project Scope and Requirement Analaysis

3.1 Project Scope

• **Scope Description:**

The system is designed to recognize static hand gestures and translate them into readable text in real time using a webcam. It is meant to work on local systems and can be expanded to include more gestures, multiple sign languages, or dynamic gesture recognition in the future.

• **Objective**: Develop a system that translates sign language gestures into written text and text into animated sign language.

• **Core Functionality**:

➢ Convert video input of sign language to text.

➢ Provide text-to-sign language animation for improved communication.

• **User Focus**: Enable better interaction for deaf and mute individuals with non-sign language users.

• **Technology**:

➢ Machine learning models for gesture recognition.

➢ Web application with cloud-based deployment.

• **Inclusions**: Support for multiple sign languages (e.g., International, Indian)

• **In-Scope:**

➢ Real-time static gesture recognition.

➢ Pre-trained CNN model for classification.

➢ MediaPipe-based hand landmark detection.

- Streamlit web interface with webcam and file upload support.
- Text overlay and display of recognized gesture.

- **Out-of-Scope:**
  - Audio-to-sign or text-to-sign (reverse translation).
  - Dynamic sequence recognition (e.g., full sign sentences).
  - Multi-user gesture recognition.
  - Gesture recognition in low-light or occluded conditions.

- **Deliverables:**
  - Fully working web application.
  - Trained CNN model file.
  - Codebase and documentation.
  - Project report and user guide.

3.2 Requirement Gathering & Analysis

Objective:

To develop a system that converts real-time sign language gestures captured via webcam into readable text output.

Core Functionality:

- Detect and classify hand gestures from a live webcam feed.
- Translate gestures into corresponding text using deep learning.

User Focus:

- Enable smooth communication for deaf and mute individuals by providing a real-time gesture-to-text translation tool.

Technology:

- MediaPipe for hand landmark detection
- TensorFlow for gesture classification
- Flask for web interface
- OpenCV for video and image processing

Inclusions:

- Support for static gesture recognition
- Pre-trained models using standard sign language datasets (e.g., WLASL)

- Text overlay and live display in the UI

3.2.1 Requirement Gathering

1. Functional Requirements

i) Data Ingestion:

- Input only through live webcam feed via the Flask interface
- No file upload or batch processing

ii) Data Preprocessing:

- Use OpenCV to capture and preprocess frames from the webcam.
- Apply MediaPipe to detect hand landmarks.
- Normalize and prepare the frame for classification.

iii) Gesture Recognition:

- Use CNN model trained with WLASL data.
- Apply custom rules on landmark positions when needed.
- Output gesture labels with high accuracy.

iv) Text Conversion:

- Map classified gesture to corresponding text using JSON-based lookup.
- Display recognized words on screen.

v) User Interaction:

- Flask-based interface with a simple webcam activation and result display.
- Real-time feedback loop for users to see recognized text instantly.

vi) Data Augmentation (Training Phase):

- Geometric transformations (rotation, flip, scale) using OpenCV.
- Add frame-level variety to improve model generalization.

vii) Data Quality Assurance:

- Evaluate the model using metrics like accuracy, precision, recall, and F1-score.

2. Non-Functional Requirements

i) Performance:

- Real-time processing with latency <1 second per gesture.

ii) Scalability:

- Designed to scale to support more gestures or new sign languages.

iii) Usability:

- Simple, responsive UI with large, readable text.

iv) Reliability:

- Should work across different backgrounds and lighting conditions.

3.2.2. Requirement Analysis

1. Data Source:

- Use the created dataset for training.
- Use associated JSON files for labels and annotations.

2. Gesture Recognition Techniques:

- Use CNN for static image-based gesture classification.
- Hand landmark-based rules for position-based logic.

3. Data Augmentation Techniques:

- Apply OpenCV-based flipping, rotation, and scaling during training.
- Frame variation to simulate different speeds.

4. Text Mapping and Generation:

- Map classified gestures to words using JSON dictionaries.

5. Dataset Format:

- JSON format for annotations, labels, and mappings.

6. Evaluation Metrics:

- Accuracy
- Precision
- Recall
- F1-Score

7. Additional Considerations:

- Design a clean UI using Flask.
- Use standard open-source libraries only.

Chapter 4-Project Design and Modelling Details

4.1 Software Requirement Specification (SRS)

4.1.1 Introduction

- **Purpose**

The purpose of this document is to define the functional and non-functional requirements of the “Conversion of Sign Language to Text” application. This system aims to bridge the communication gap between hearing-impaired individuals and others by converting sign language gestures into readable text using a webcam and real-time machine learning-based detection.

- **Scope**

This project provides a real-time web-based solution to interpret hand gestures and convert them into meaningful text using a CNN model trained on hand gesture images. The platform supports both static image recognition and dynamic real-time detection via webcam. It is intended for educational, assistive, and accessibility-based applications.

- **Definitions, Acronyms, and Abbreviations**

- CNN: Convolutional Neural Network
- UI: User Interface
- OpenCV: Open Source Computer Vision Library
- Streamlit: A Python-based web framework for building data-driven web applications
- MediaPipe: A Google framework for real-time hand/face/body landmark detection

- **References**

- "Gesture Recognition using CNN and MediaPipe", IEEE Conference Proceedings, 2023
- "Real-Time Sign Language Recognition with Deep Learning", ArXiv, 2022
- Official documentation: MediaPipe, Keras, Streamlit

- **Overview**

This document outlines the system requirements for the "Sign Language to Text" converter, covering hardware/software components, functional and non-functional specifications, system models, and module descriptions.

Overall Description

Product Perspective

The application is a standalone web-based tool that processes video input through the browser and displays real-time textual output. It combines computer vision, deep learning, and frontend development tools to ensure fast and accurate gesture recognition.

- **Product Features**

➢ Real-Time Detection: Convert hand gestures into text live via webcam.

➢ Static Gesture Recognition: Identify and classify static gestures using a CNN model.

➢ Custom UI: Web-based interface with logo branding and accessible design.

➢ Visual Output: Detected text overlaid on video stream.

- **User Classes and Characteristics**

➢ Hearing-impaired Users: Use the tool to communicate via gesture recognition.

➢ Educators and Trainers: Demonstrate sign language basics in classrooms.

➢ Researchers/Developers: Analyze or extend the detection model.

➢ General Public: Use for learning or interacting with sign language users.

- **Operating Environment**

➢ System OS: Windows 11 / Linux

➢ Frontend: Streamlit (Web-based UI)

➢ Backend: Python 3.x, OpenCV, MediaPipe, NumPy, Keras

➢ IDE: Visual Studio Code / Jupyter Notebook

➢ Model File: CNN_KERAS_MODEL.h5

- **Design and Implementation Constraints**

➢ Requires a stable internet connection and a working webcam.

➢ Must run on a machine capable of handling live video processing.

➢ The model and personal data should be used in compliance with ethical AI practices.

- **Specific Requirements**

➢ Functional Requirements

➢ User Interface Requirements

➢ Streamlit-based UI with sections for webcam input, file upload, and text display.

➢ CSS customization for titles and logo placement.

- **Gesture Detection**

➢ Real-time webcam-based hand tracking using MediaPipe.

➢ Display live feedback and recognized gestures in text form.

- **CNN Integration**

➢ Process uploaded static images for classification using CNN_KERAS_MODEL.h5
➢ Output the recognized gesture label based on trained data.

- **Text Display and Overlay**

➢ Overlay recognized gesture text on the video feed using OpenCV.

Non-functional Requirements

➢ **Performance:**

The system should detect and classify gestures with reasonable accuracy, even if recognition takes slightly more than 1 second depending on hardware conditions.
The gestures detected must be correct and match the intended meaning with minimal false positives.

➢ **Usability:**

The interface should be intuitive and easy to navigate.

It must include large buttons, accessible layout, and supportive visuals for non-technical users.

➢ **Reliability:**

The trained CNN model should consistently perform across different sessions.

The system should function smoothly during extended use with minimal crashes or freezes.

➢ **Security:**

Uploaded data should not be stored permanently.

The application should securely handle webcam access and prevent unauthorized use.

System Models

- Data Flow Diagrams
- Activity Diagram
- Sequence Diagram
- Block Digram
- Class Diagram

4.2 System Modules

1. Camera/Webcam Input:

- **Function**: Captures live video stream from the user’s webcam.

- **Details**:
  - Initiates real-time video feed using OpenCV.
  - Acts as the primary input source for gesture recognition.
  - Continuously sends frames to the hand detection module.

2. **Hand Landmark Detection (MediaPipe):**

- **Function**: Detects key hand landmarks (joints and finger positions).
- **Details**:
  - Utilizes Google's MediaPipe library for real-time hand tracking.
  - Extracts 21 key landmark points from each detected hand.
  - Converts hand shapes into numerical data for gesture interpretation.
  - Supports both single and multi-hand detection.

3. **Gesture Classification:**

- **Function**: Identifies the gesture from hand landmark data.
- **Details**:
  - Takes landmark points as input and feeds them to a **classification model** (e.g., CNN or rule-based logic).
  - Differentiates between **static gestures** (like letters) and **dynamic gestures** (like waving).
  - May use thresholding and smoothing techniques to improve accuracy.
  - Output is a gesture label (e.g., "A", "B", "Hello").

4. **Gesture Database (Optional – Used for Lookup/Mapping):**

- **Function**: Stores the mapping between gestures and corresponding text.
- **Details**:
  - Contains predefined mappings between gesture classes and text outputs.
  - Acts as a lookup table to convert recognized gestures to meaningful text or phrases.
  - Easily extendable to include new signs.

5. **Recognized Gesture Text:**

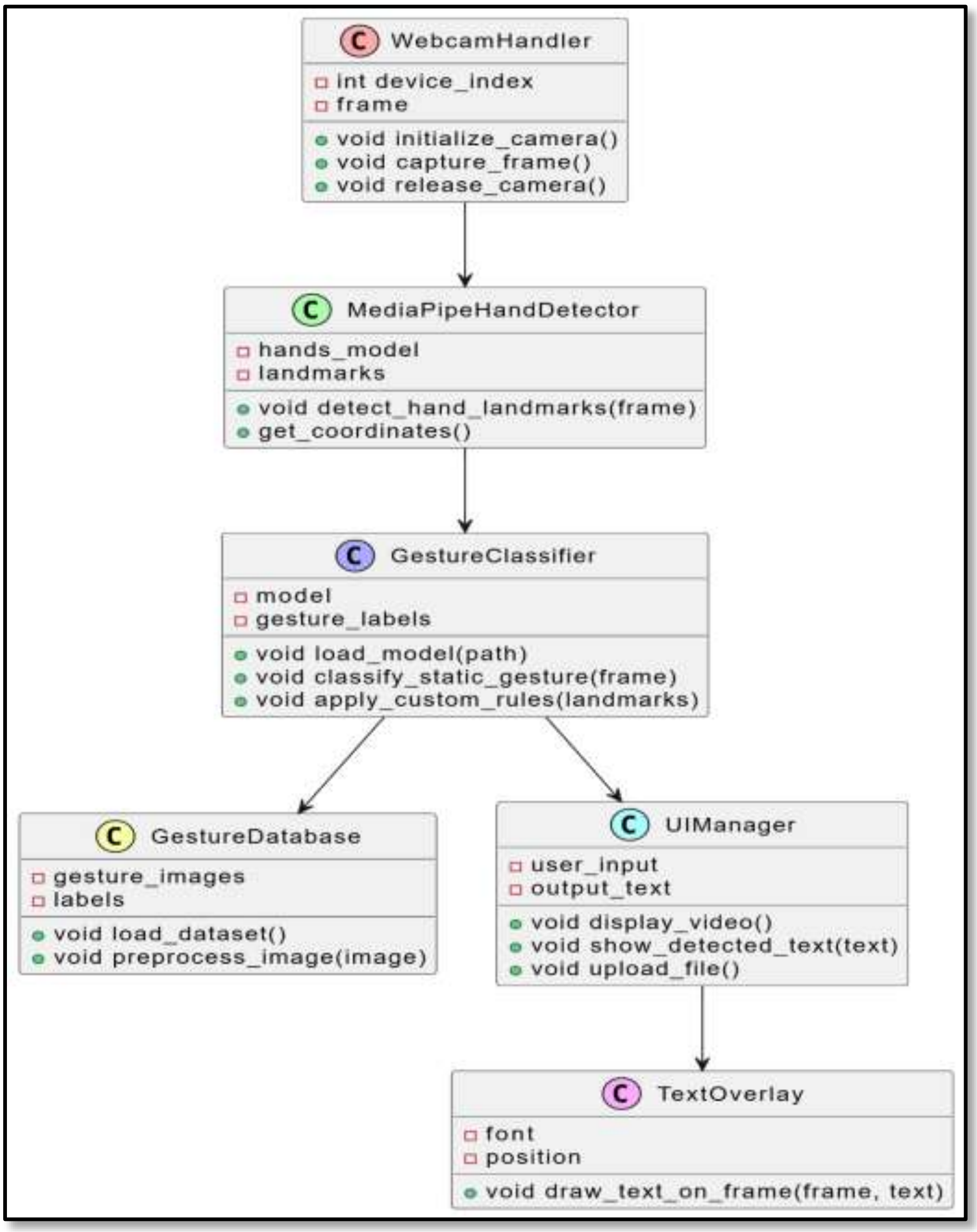
- **Function**: Converts classified gesture into readable text.
- **Details**:
  - Receives gesture ID/class from the classifier.
  - Uses gesture database to map it to the corresponding text (e.g., "Thank you").
  - Supports buffering and string formation for continuous text construction.

6. Overlay & Display (OpenCV + Streamlit):

- **Function**: Displays the recognized text and live video.
- **Details**:
  - Uses OpenCV to overlay recognized text on the video frame.
  - Streamlit web interface is used to display live video and results.
  - Provides an interactive and user-friendly UI.
  - May include additional UI components like start/stop buttons or gesture logs

4.3 System Modeling & Design

4.3.1 Class Diagram



The Class diagram has following contents:

- **WebcamHandler**

➢ **Purpose:** Manages webcam initialization and frame capture.

➢ **Attributes:**

○ **device_index:** ID of the webcam device.

○ **frame:** Stores the current video frame.

➢ **Methods:**

○ initialize_camera(): Starts the webcam.

○ capture_frame(): Captures one frame from the webcam.

○ release_camera(): Releases the webcam resource.

- **MediaPipeHandDetector**

➢ **Purpose:** Detects hand landmarks using MediaPipe.

➢ **Attributes:**

○ hands_model: The MediaPipe model instance.

○ landmarks: List of detected hand landmark coordinates.

➢ **Methods:**

○ detect_hand_landmarks(frame): Processes a frame to find hands.

○ get_coordinates(): Returns x, y coordinates of detected hand landmarks.

- **GestureClassifier**

➢ **Purpose**: Recognizes hand gestures using either CNN or custom rules.

➢ **Attributes:**

○ model: CNN model for static gesture classification.

○ gesture_labels: Mapping of classes to gesture names.

➢ **Methods**:

○ load_model(path): Loads a saved CNN model.

○ classify_static_gesture(frame): Classifies static hand gestures.

○ apply_custom_rules(landmarks): Uses coordinate logic for dynamic gestures.

- **GestureDatabase**

➢ **Purpose**: Stores and processes labeled gesture data.

➢ **Attributes:**

○ gesture_images: Images of hand gestures.

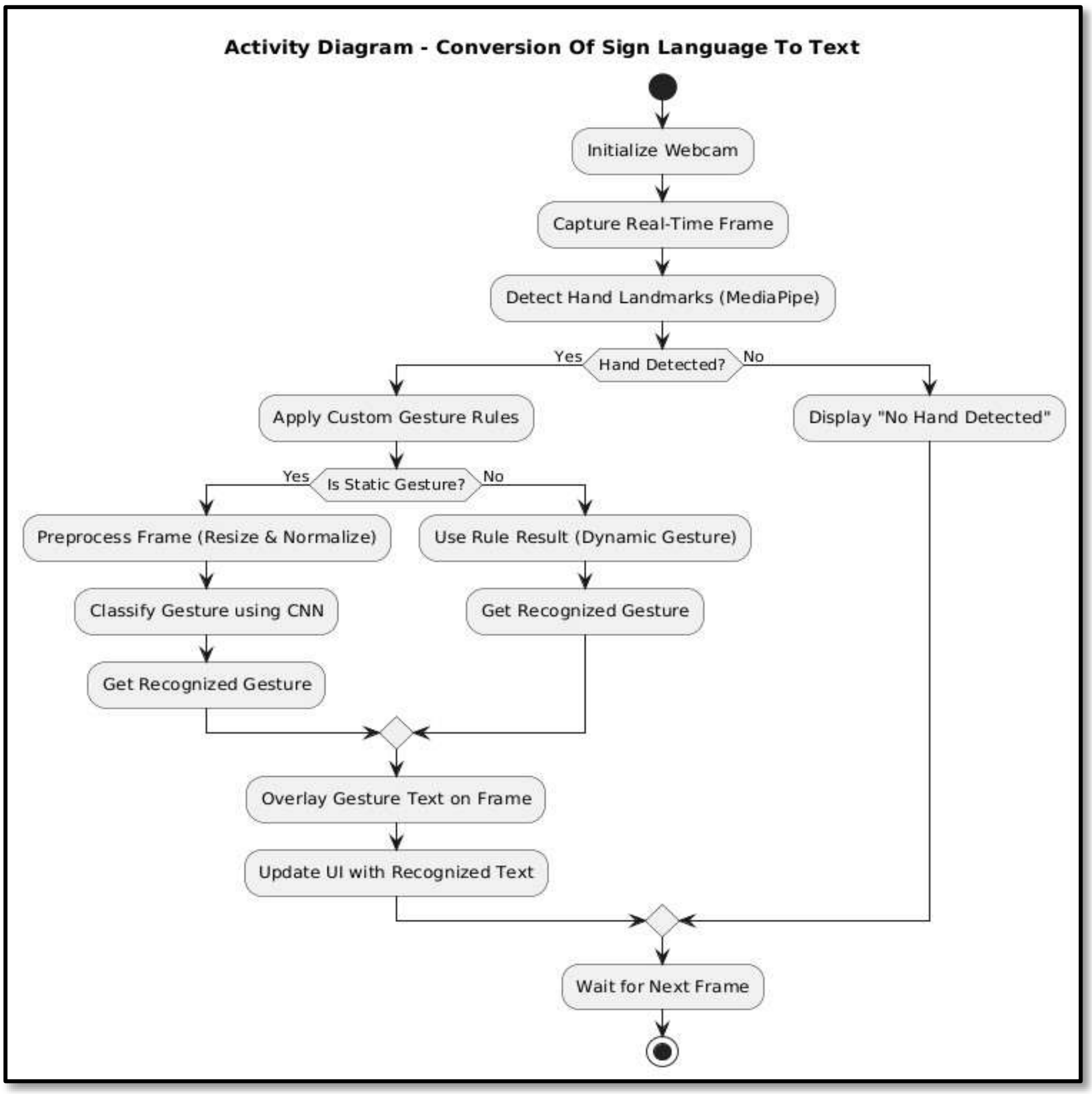
○ labels: Class labels for each gesture.

- **Methods**:
  - load_dataset(): Loads gesture image dataset.
  - preprocess_image(image): Normalizes and resizes input images.

- **UIManager**
  - **Purpose:** Handles the web interface and user interaction.
  - **Attributes:**
    - user_input: Uploaded images or video.
    - output_text: Text corresponding to recognized gesture.
  - **Methods:**
    - display_video(): Shows webcam stream in UI.
    - show_detected_text(text): Displays recognized text
    - upload_file(): Allows user to upload gesture images.

- **TextOverlay**
  - **Purpose**: Renders recognized text on top of video.
  - **Attributes**:
    - font: Font style for overlay.
    - position: Coordinates where text appears.
  - **Methods:**
    - draw_text_on_frame(frame, text): Adds gesture text to video frame

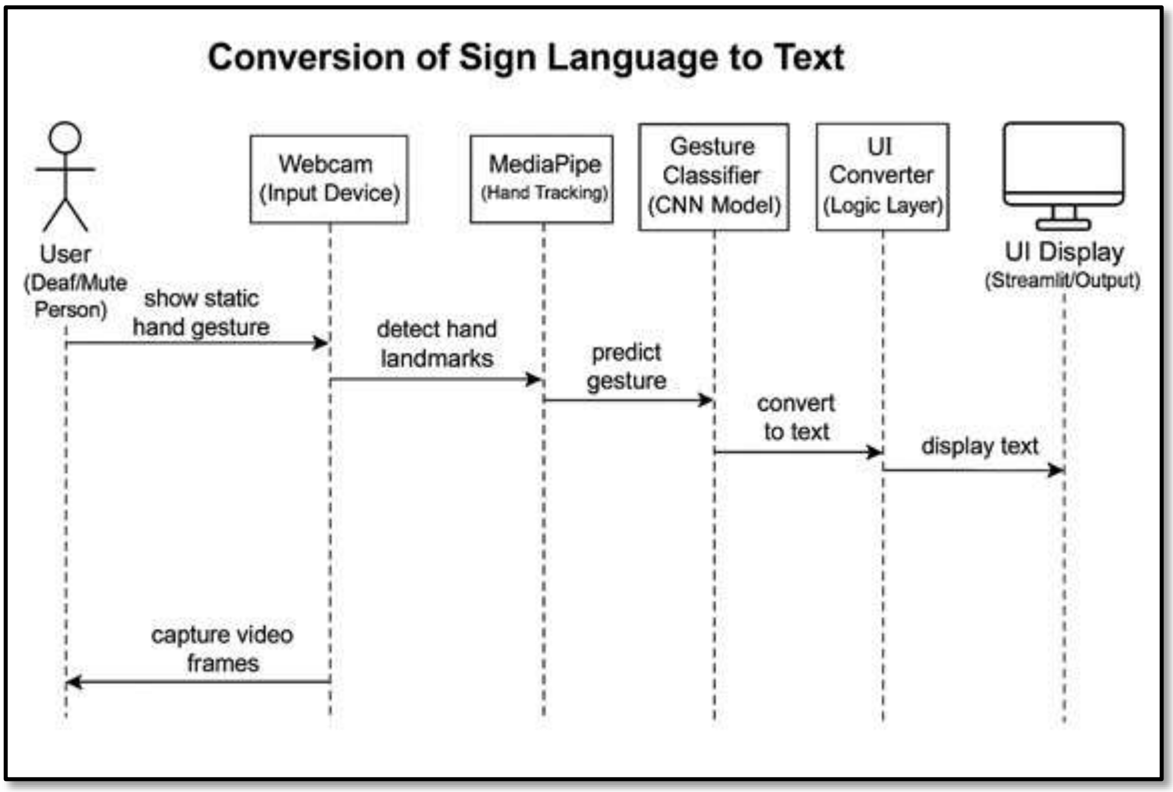
4.3.2 Activity Diagram


Activity Diagram - Conversion Of Sign Language To Text
Initialize Webcam
Capture Real-Time Frame
Detect Hand Landmarks (MediaPipe)
Hand Detected?
Yes
No
Apply Custom Gesture Rules
Display "No Hand Detected"
Is Static Gesture?
Yes
No
Preprocess Frame (Resize & Normalize)
Use Rule Result (Dynamic Gesture)
Classify Gesture using CNN
Get Recognized Gesture
Get Recognized Gesture
Overlay Gesture Text on Frame
Update UI with Recognized Text
Wait for Next Frame

This diagram shows the step-by-step **workflow of the system**:

- The webcam is initialized and starts capturing real-time frames.
- MediaPipe detects hand landmarks in the frame.
- If no hand is detected, a message is displayed. If a hand is detected:
  - Custom rules are applied to determine gesture type.
  - For **static gestures**, the frame is preprocessed and passed to a CNN for classification.
  - For **dynamic gestures**, predefined rules are used to recognize them.
- The recognized gesture is then overlaid as text on the frame, and the UI is updated.
- The system loops to process the next frame.

4.3.3 Sequence Diagram



This sequence diagram represents the flow of data in the sign language to text conversion system. It includes user input via webcam, hand landmark detection using MediaPipe, gesture recognition through a CNN model, conversion to readable text, and display on a user interface.

Purpose: To convert sign language gestures into text format for better accessibility.

- **Components:**

➢ User: A deaf/mute person showing static hand gestures.

➢ Webcam: Captures video frames of the user's gestures.

➢ MediaPipe: Utilizes hand tracking to detect landmarks of the user's hands.

➢ Gesture Classifier (CNN Model): Predicts the gesture based on the detected landmarks.

➢ UI Converter: Converts the predicted gesture into text.

➢ UI Display: Outputs the converted text for display on screen.

- **Flow of Actions:**

➢ User shows a static hand gesture.

➢ Webcam captures video frames of the gesture.

➢ MediaPipe detects hand landmarks from the video.

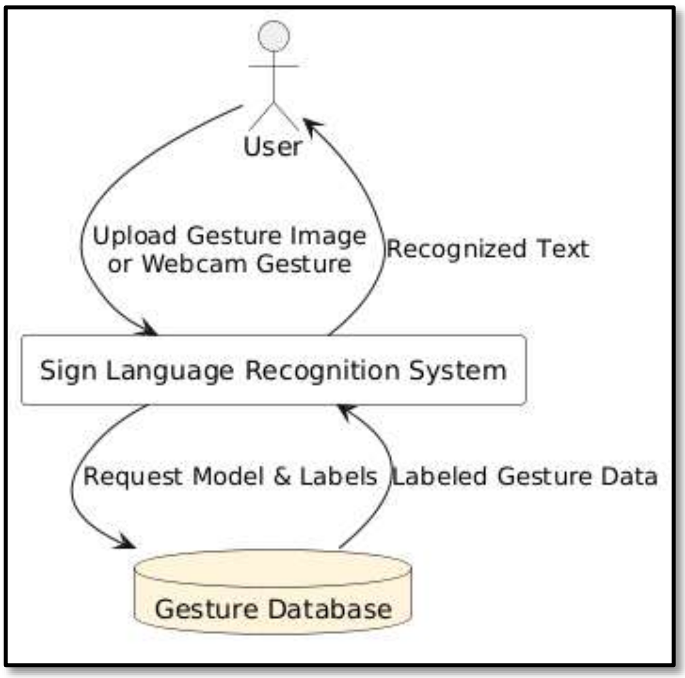
➢ Gesture Classifier predicts the gesture identified.

➢ The UI Converter converts the gesture prediction into text.

➢ The text is displayed on the UI Display.

4.3.4 Data Flow Diagram

Level 0



- **User (External Entity)**

Interacts with the system by either uploading a static gesture image or presenting live gestures via webcam. Receives the final output ("Recognized Text") back from the system.

- **Sign Language Recognition System (Process)**

➢ Core black-box process that:

○ Accepts user input (video stream or image).

○ Detects and classifies hand gestures.

○ Generates the corresponding textual output.

➢ Gesture Database (Data Store / External Entity)

○ Stores all trained models and labeled gesture data required for classification.

○ Supplies the system with the CNN weights, labels, and any rule-based mappings when requested.

➢ **Data Flow: "Upload Gesture Image or Webcam Gesture"**

Arrow from User → System: Carries raw input (live video or uploaded image) into the recognition process.

➢ **Data Flow: "Request Model & Labels"**

Arrow from System → Gesture Database: System asks for the trained data needed to interpret gestures.

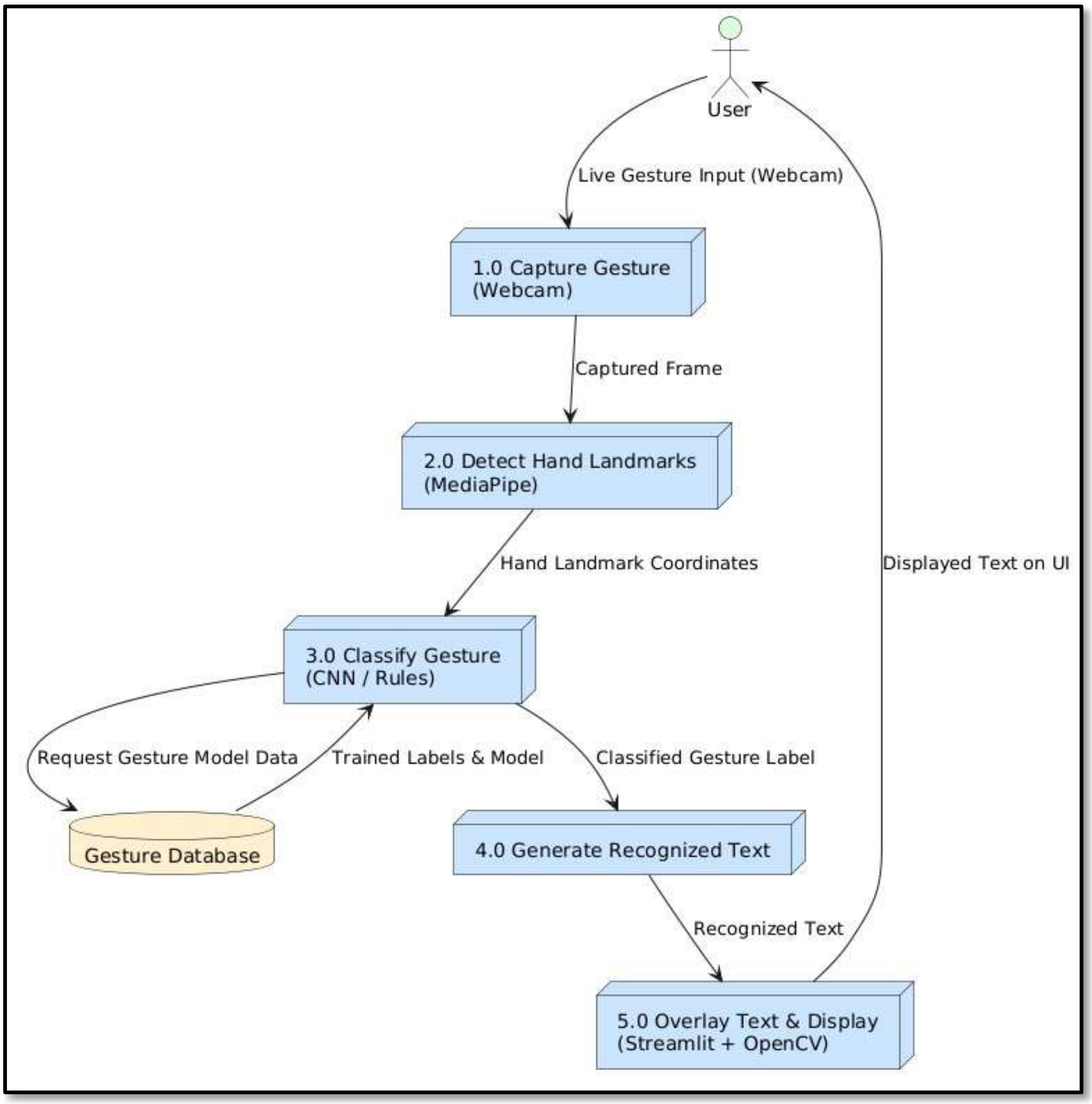
➢ **Data Flow: "Labeled Gesture Data"**

Arrow from Gesture Database → System: Returns model parameters and label mappings for classification.

➢ **Data Flow: "Recognized Text"**

Arrow from System → User: Conveys the final interpreted text back to the user interface

DFD Level 1


User
Live Gesture Input (Webcam)
1.0 Capture Gesture (Webcam)
Captured Frame
2.0 Detect Hand Landmarks (MediaPipe)
Hand Landmark Coordinates
Displayed Text on UI
3.0 Classify Gesture (CNN / Rules)
Request Gesture Model Data
Trained Labels & Model
Classified Gesture Label
Gesture Database
4.0 Generate Recognized Text
Recognized Text
5.0 Overlay Text & Display (Streamlit + OpenCV)

The above diagram has following contents:

- **User (External Entity)**

Provides live gesture input via webcam or uploads a static gesture file. Receives the final displayed text on the UI.

- **Capture Gesture (Webcam)**

Grabs a video frame from the webcam or reads the uploaded image. Outputs a raw Frame for further processing.

- **Detect Hand Landmarks (MediaPipe)**

Takes the incoming frame and runs MediaPipe to locate hand keypoints. Produces Hand Landmarks (sets of x,y coordinates).

- **Classify Gesture (CNN / Rules)**

Receives landmarks and/or preprocessed frame.

Requests trained labels and model data from the Gesture Database.

Uses a CNN model for static gestures or rule-based logic for dynamic ones. Outputs a Gesture Label (e.g., “A”, “Hello”, “5”).

- **Gesture Database (Data Store)**

Stores trained model weights, label mappings, and gesture metadata.

On request, returns Trained Labels & Model to the classification process.

- **Generate Recognized Text**

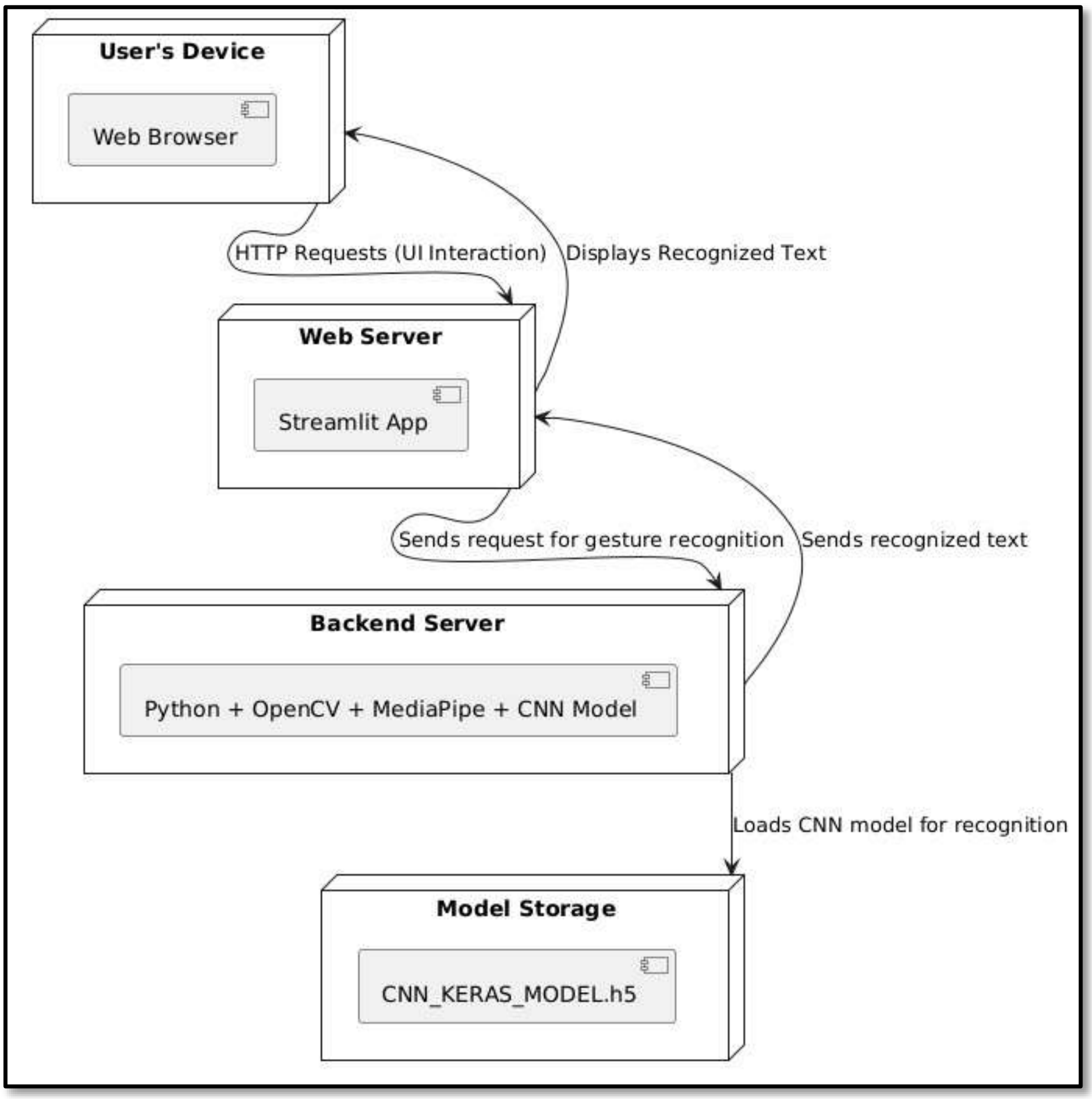
Converts the numeric or categorical gesture label into actual text. Outputs Recognized Text string.

- **Overlay Text & Display (Streamlit + OpenCV)**

Takes the recognized text and overlays it onto the live video frame.

Presents the combined video + text stream back to the User as Displayed Text.

4.3.4 Deployment Diagram


User's Device
Web Browser
HTTP Requests (UI Interaction)
Displays Recognized Text
Web Server
Streamlit App
Sends request for gesture recognition
Sends recognized text
Backend Server
Python + OpenCV + MediaPipe + CNN Model
Loads CNN model for recognition
Model Storage
CNN_KERAS_MODEL.h5

The deployment diagram represents the architecture of the **"Conversion of Sign Language to Text"** system. It showcases the physical and logical components involved in the deployment process.

Key Elements:

➢ **User's Device (Web Browser)**: The user's device where the frontend (Streamlit app) is accessed via a web browser. It allows the user to interact with the system.

➢ **Web Server** (Streamlit App): This server handles the user interface (UI) logic and communicates with the backend to send gesture recognition requests.

➢ **Backend Server** (Python + OpenCV + MediaPipe + CNN Model): The server responsible for processing the user's input, recognizing hand gestures using OpenCV and MediaPipe, and running the gesture recognition model (CNN) to convert sign language into text.

➢ **Model Storage (CNN_KERAS_MODEL.h5):** A storage node where the pre- trained CNN model is stored, which is loaded by the backend server for gesture recognition.

Flow:

➢ The user interacts with the UI via the web browser.

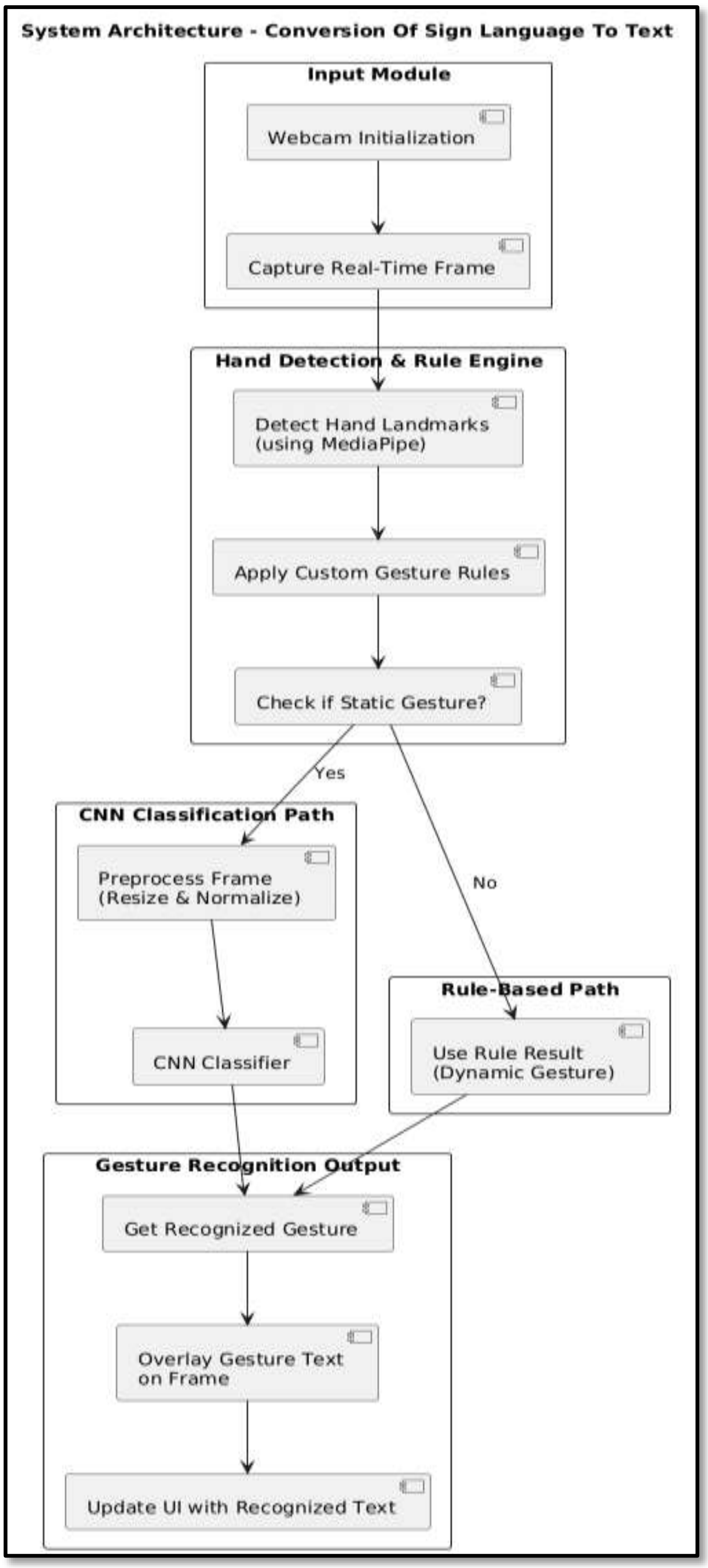
➢ The web server sends requests to the backend for gesture recognition.

➢ The backend loads the necessary model from the model storage.

➢ The recognized gesture is then converted into text and sent back to the web server, which displays it on the user's device.

➢ This deployment diagram illustrates the physical distribution and interaction between different system components to achieve the sign language-to-text conversion functionality.

4.5 System Architecture


System Architecture - Conversion Of Sign Language To Text
Input Module
Webcam Initialization
Capture Real-Time Frame
Hand Detection & Rule Engine
Detect Hand Landmarks (using MediaPipe)
Apply Custom Gesture Rules
Check if Static Gesture?
Yes
No
CNN Classification Path
Preprocess Frame (Resize & Normalize)
CNN Classifier
Rule-Based Path
Use Rule Result (Dynamic Gesture)
Gesture Recognition Output
Get Recognized Gesture
Overlay Gesture Text on Frame
Update UI with Recognized Text

Chapter 5-Implementation and Coding

5.1 Algorithms

5.1.1 Gesture Recognition Algorithm

Function: recognizeGesture(frame)

- Get the current image from the webcam.
- Use MediaPipe to find hand landmarks in the image.
- If hand landmarks are found:

➢ Get the x and y positions of the 21 landmarks.

➢ Use specific rules based on these positions to identify the gesture.

➢ If the gesture is a still image:

i. Prepare the image (resize and normalize).

ii. Use the CNN model to recognize the gesture.

iii. Return the recognized gesture as text.

- If no hand is detected:

➢ Return "No gesture detected."

5.1.2 Real-time Video Processing Algorithm

Function: processVideo()

- Start the webcam using OpenCV.
- Continuously capture images from the webcam:

➢ Call recognizeGesture(frame) for each image.

➢ Show the recognized gesture text on the image using OpenCV.

➢ Display the processed image in the Streamlit app.

- If the user stops the webcam:

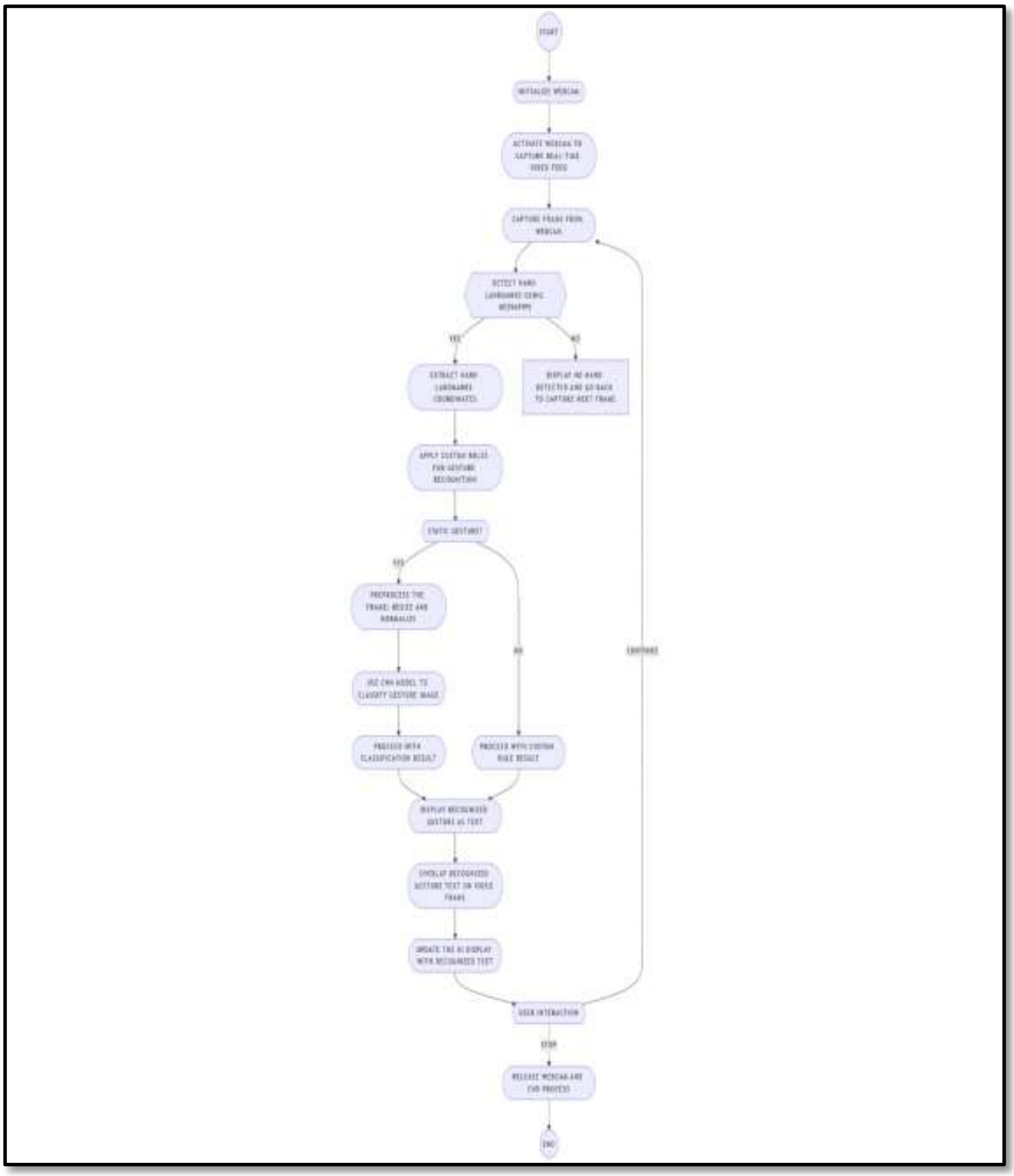
➢ Release the webcam and close the display.

5.1.3 Text Display Algorithm

Function: displayDetectedText(gestureText)

- Update the Streamlit app with the recognized gesture text.
- Make sure the text is easy to see.
- Optionally, change the text color based on the recognized gesture.
- Allow users to clear the displayed text with a button.

5.2 Flowchart


START
INITIALIZE WEBCAM
ACTIVATE WEBCAM TO CAPTURE REAL-TIME VIDEO FEED
CAPTURE FRAME FROM WEBCAM
DETECT HAND LANDMARKS USING MEDIAPIPE
YES
NO
EXTRACT HAND LANDMARK COORDINATES
DISPLAY NO HAND DETECTED AND GO BACK TO CAPTURE NEXT FRAME
APPLY CUSTOM RULES FOR GESTURE RECOGNITION
STATIC GESTURE?
YES
NO
PREPROCESS THE FRAME: RESIZE AND NORMALIZE
USE CNN MODEL TO CLASSIFY GESTURE IMAGE
PROCEED WITH CLASSIFICATION RESULT
PROCEED WITH CUSTOM RULE RESULT
DISPLAY RECOGNIZED GESTURE AS TEXT
OVERLAY RECOGNIZED GESTURE TEXT ON VIDEO FRAME
UPDATE THE UI DISPLAY WITH RECOGNIZED TEXT
CONTINUE
USER INTERACTION
STOP
RELEASE WEBCAM AND END PROCESS
END

This flowchart describes the step-by-step process of how the gesture recognition system operates. It starts with activating the webcam and capturing video frames, then moves on to detecting hand gestures, processing those gestures, and finally allowing for user interaction. The system is designed to provide real-time feedback and ensure a smooth user experience while effectively recognizing hand gestures.

- **Flowchart Description**: Overall Process of Gesture Recognition System

1. **Start**- The process begins when the system is initiated. This is the starting point of the gesture recognition application.

2. **Webcam Setup-**

➢ Activate Webcam: The system turns on the webcam to start capturing live video. This allows the application to see what is happening in front of the camera.
➢ Capture Frame: The system takes a single frame (or picture) from the continuous video feed provided by the webcam. This frame is what the system will analyze to detect hand gestures.

3. **Hand Detection-**

➢ Detect Hand Landmarks: The system uses specialized software, such as MediaPipe, to check if there are any hands visible in the captured frame. This software is designed to identify key points on the hands, which are essential for recognizing gestures.

4. **Decision on Hand Detection-**

➢ Hand Detected? : The system checks if a hand is detected in the frame.

➢ If a hand is detected (Yes):

o Extract Landmarks: The system identifies and extracts key points (landmarks) on the hand. These landmarks are specific locations on the hand that help in analysing the gesture being made.
o Apply Gesture Recognition Rules: The system then applies predefined rules to recognize specific hand gestures based on the extracted landmarks. These rules help the system understand what gesture the user is making.
➢ If no hand is detected (No):

o Display Message: The system shows a message on the screen saying "No hand detected." This informs the user that the system did not see any hands in the current frame. After displaying this message, the system goes back to capturing the next frame from the webcam to check again.

5. **Gesture Processing-**

➢ Is it a Static Gesture? : The system determines whether the detected gesture is static (a still hand position) or dynamic (a moving gesture).

➢ If it's a static gesture (Yes):

o Preprocess Frame: The captured image of the hand is resized and normalized. This means adjusting the image to a standard size and format, which helps improve the accuracy of the analysis.
o Classify Gesture: A Convolutional Neural Network (CNN) model is used to analyze the preprocessed image and classify the gesture. The CNN has been trained to recognize different hand gestures based on the features it learns from the images.
o Proceed with Classification Result: Based on the result from the classification, the system will take the next steps.

○ Display Recognized Gesture: The recognized gesture is shown as text on the screen, allowing the user to see what gesture was identified.

○ Overlay Gesture on Video: The recognized gesture text is displayed over the live video feed, providing real-time feedback to the user.

○ Update UI: The user interface is updated to reflect the recognized gesture, ensuring that the user has the latest information about what the system has detected.

➢ If not a static gesture (No):

○ Proceed with Custom Results: The system follows custom rules for further processing of dynamic gestures. This may involve different methods of analysis or interaction based on the type of gesture detected.

6. User Interaction-

➢ User Interaction: The system waits for the user to interact with it. This could involve the user making a gesture, clicking a button, or providing some other input.Depending on the user's input, the user can choose to stop the process at any time. This allows for flexibility and control over the application.

7. End Process-

➢ Release Resources: Once the user decides to stop the process, the system releases the resources it was using, such as turning off the webcam. This ensures that the application concludes properly and frees up any resources that were in use.

5.3 Software Requirements with Relevant Justification

- **Python (Latest Version)**

Justification: Python is the primary programming language used for developing the application. It is favored for its simplicity and versatility, making it an ideal choice for rapid development. Python supports a wide range of libraries that are crucial for our project, particularly OpenCV and MediaPipe. These libraries are essential for implementing computer vision techniques and real-time processing, which are fundamental to the functionality of our gesture recognition system.

- **Streamlit (Latest Version)**

Justification: Streamlit is a modern framework that simplifies the creation of interactive web applications. It is particularly well-suited for our project as it allows developers to build user interfaces quickly and efficiently. Streamlit enables the display of real-time gesture recognition outputs and provides user controls in an intuitive manner, enhancing the overall user experience and making the application more accessible to users.

- **OpenCV (Version 4.5 or Higher)**

Justification: OpenCV (Open Source Computer Vision Library) is a comprehensive library that provides a wide array of tools for video capture and image processing. It is crucial for our application as it handles webcam input and performs the necessary image processing to detect hand gestures accurately. The specified version (4.5 or higher) ensures that we have access to the latest features, optimizations, and bug fixes, which are vital for maintaining the performance and reliability of the application

- **MediaPipe (Latest Version)**

Justification: MediaPipe is a framework developed by Google that specializes in efficient hand landmark detection. It is instrumental in our project as it allows for the recognition of hand gestures with high accuracy in real-time scenarios. By utilizing the latest version of MediaPipe, we can leverage the most recent advancements in gesture recognition technology, ensuring that our application performs optimally and meets user expectations.

- **Keras (Latest Version)**

Justification: Keras is a high-level neural networks API that simplifies the process of building and training convolutional neural networks (CNNs). This is essential for our application, as we need to classify hand gestures effectively. Keras provides a user-friendly interface and integrates seamlessly with TensorFlow, making it easier to experiment with different neural network architectures. The latest version ensures that we benefit from improved functionalities and support for various model types.

- **NumPy (Latest Version)**

Justification: NumPy is a fundamental package for numerical computations in Python. It is particularly useful for handling and processing data arrays, which are integral to the CNN model used for gesture classification. NumPy's efficient array operations and mathematical functions enable us to perform complex calculations quickly, which is essential for the performance of our application. Using the latest version ensures compatibility with other libraries and optimizations for performance.

5.4 Hardware Requirements with Relevant Justifications

- **Development Machines**

➢ **Specifications:** Intel Core i5 (or equivalent), 8GB RAM, 256GB SSD

➢ **Justification:** The specified hardware configuration is designed to ensure the smooth operation of development tools, including Integrated Development Environments (IDEs) and local server instances for testing the application. An Intel Core i5 processor or its equivalent provides sufficient processing power for running development tools and executing code efficiently. The 8GB of RAM allows for multitasking and running multiple applications simultaneously, while the 256GB SSD ensures fast data access and storage, which is crucial for development efficiency.

- **Server**

➢ **Specifications:** 4GB RAM, 500GB SSD, Dual-core processor

➢ **Justification:** The server must be capable of handling multiple concurrent users while efficiently processing video input. The specified 4GB of RAM is necessary to support the demands of real-time video processing and to ensure a responsive user experience. The 500GB SSD provides ample storage for application data, user progress, and logs,

while the dual-core processor ensures that the server can manage multiple tasks simultaneously without significant performance degradation.

- **Webcam**

➢ **Specifications:** 720p HD or better

➢ **Justification:** A high-quality webcam is crucial for capturing clear video input, which is necessary for accurately detecting hand gestures in real-time. The specified resolution of 720p HD or better ensures that the video feed is of sufficient quality to facilitate effective gesture recognition. A clear and stable video input is essential for the application to function correctly and provide accurate results.

- **Network**

➢ **Specifications:** High-speed internet (10+ Mbps)

➢ **Justification:** A reliable and fast internet connection is required for the seamless delivery of video and real-time interactions within the web application. The specified bandwidth of 10 Mbps or higher ensures that users experience minimal lag or buffering, which is vital for maintaining the application's responsiveness and overall user satisfaction. A stable connection is essential for real-time communication and interaction, particularly when processing video streams.

- **Backup Solution**

➢ **Specifications:** External hard drives or cloud backup

➢ **Justification:** Implementing a robust backup solution is essential for regular backups to prevent data loss. This is particularly important for preserving user progress and educational content, ensuring reliability in gesture data logs and overall application performance. Regular backups safeguard against potential data corruption or loss, providing peace of mind and ensuring that critical information is always retrievable. Utilizing external hard drives or cloud storage solutions offers flexibility and security for data management.

5.5 Implementation Details

5.5.1 CNN Model Training Process

Function: trainCNNModel(datasetPath)

- Load the dataset from the specified location, ensuring images are sorted by gesture.
- Prepare the images:

➢ Resize them to a standard size.

➢ Normalize the pixel values to be between 0 and 1.

➢ Split the dataset into training, validation, and test sets.

- Define the CNN structure:

➢ Add layers to extract features from the images.

➢ Include layers to reduce the size of the data.

➢ Add dropout layers to avoid overfitting.

➢ Add dense layers to classify the gestures.

- Compile the model using a loss function and optimizer.
- Train the model using the training set and validate it with the validation set.
- Save the trained model as CNN_KERAS_MODEL.h5 for later use.

5.5.2 User Interface Implementation

- **Streamlit UI Components**

Function: createStreamlitUI()

➢ Set up the layout of the Streamlit app:

○ Add a title and description of the app.

○ Include an option for users to upload gesture images (if needed).

○ Create a button to start/stop the webcam.

➢ Show the recognized gesture text in real-time.

➢ Use CSS to customize the app's look, including logos and colors.

➢ Make sure the app is easy to use and responsive.

- **Interaction Handling**

Function: handleUser Input()

➢ Monitor user actions in the app.

➢ If the user clicks the start button:

○ Start the webcam and begin processing video.

➢ If the user clicks the stop button:

○ Stop the webcam and show a message that it has stopped.

➢ Allow users to reset the displayed text.

Chapter 6-Testing

6.1 Fundamentals of Testing

Testing is a critical step in developing the sign language to text conversion application. It ensures that the application functions correctly, is secure, and meets the needs of its users, especially those who rely on sign language for communication. The testing process involves several methods tailored to the specific requirements of real-time gesture recognition software.

Here's a closer look at the key testing methodologies we will use:

- **Usability Testing**

Usability testing is all about how easy and pleasant the application is to use. It is essential to ensure that the interface is user-friendly, especially for individuals who may not be very familiar with technology.

➢ Interface Design: The application should have a clean and straightforward layout. This means that buttons, menus, and instructions should be clearly visible and easy to understand. Users should be able to quickly learn how to use the webcam and perform gestures without confusion.

➢ Accessibility: The design must consider users with different abilities. This includes making sure that people with disabilities can also use the application effectively. For example, using larger buttons or providing audio instructions can help users who may have visual impairments.

➢ Feedback Mechanisms: It's important for users to receive immediate feedback when they perform gestures. For instance, if a user makes a gesture, the application should quickly display the corresponding text. This instant response helps users feel engaged and confident that the application is working correctly.

- **Functional Testing**

Functional testing checks that all features of the application work as intended. This involves testing various aspects of the application to ensure everything functions properly.

➢ Webcam Input: We need to verify that the webcam activates correctly and captures video input without any delays. A smooth video feed is crucial for accurate gesture recognition, as any lag can lead to misinterpretation of gestures.

➢ Gesture Recognition: This part of testing focuses on how accurately the application recognizes and translates gestures into text. We will test various gestures to ensure that the application can identify them correctly and consistently.

➢ Text Display: It is essential to check that the detected text appears on the screen promptly and accurately. Users should see the text representation of their gestures without any delays, as this is key to effective communication.

- **Performance Testing**

Performance testing evaluates how well the application performs under different conditions. This includes checking its responsiveness and stability.

➢ Real-time Processing: The application should process gestures in real-time without noticeable delays, even when lighting conditions change. This ensures that users can communicate smoothly without interruptions, which is especially important for sign language users.

➢ System Load: We will test how the application performs when multiple users are using it at the same time or during extended use. The goal is to ensure that the application remains responsive and does not slow down, even when under heavy usage.

6.2 Test Plan of the Project

Test Phase	**Objectives**	**Timeline**
Unit Testing	Verify individual components function correctly	Weeks 6-7
Integration Testing	Ensure components work together seamlessly	Weeks 8-9
System Testing	Validate the entire system against requirements	Week 10
User Acceptance Testing	Gather feedback from target users (deaf community, educators)	Week 11

Table No: 6.2.1 Test Plan of the Project

6.3 Test Cases and Test Results

In this section, we will describe specific tests related to the user module of the application. Each test is designed to evaluate a particular aspect of the application's functionality.

Webcam Activation: This test checks that when the user activates the webcam, the application successfully displays the live video feed. This is crucial because the webcam feed is necessary for the gesture recognition process. The expected outcome is that the webcam feed is displayed correctly, and this test passed successfully.

Gesture Recognition: In this test, users perform specific gestures that the application is programmed to recognize. The expected output is that the application accurately translates these gestures into text, which is then displayed on the screen. This test is essential to ensure that the core functionality of the application works as intended, and it also passed successfully.

Real-time Detection: This test evaluates the application's ability to continuously recognize gestures and update the displayed text in real-time. It is important for providing a smooth user experience, as users expect immediate feedback when they perform gestures. The expected outcome is that the text updates in real-time, and this test also passed successfully.

Display Detected Text: In this test, users perform a series of gestures, and the application should display the corresponding text for each gesture accurately and promptly. This ensures that users can see the text representation of their gestures without any delays. The expected outcome is that all corresponding texts are displayed correctly, and this test passed successfully.

This chapter outlines a comprehensive testing strategy for the sign language to text conversion application. It emphasizes the importance of usability, functionality, and performance testing to ensure that the application meets the needs of its users. By conducting thorough testing across various phases and focusing on specific test cases, we aim to deliver a reliable and user-friendly application that effectively supports communication for those who use sign language.

Chapter 7-Project Plan & Schedule

7.1 Project Planning and Project Resources

The sign language to text conversion project will be carried out in distinct phases. This structured approach ensures that the application is developed systematically and meets the needs of its users. Each phase includes specific tasks, deadlines, and milestones to track progress effectively. Here's a detailed breakdown of each phase:

- **Project Planning-**

1. **Phase 1: Requirement Analysis**

- Interactive Sessions: We will hold discussions with various stakeholders, including members of the deaf community, educators, and technical experts. These sessions will help us gather insights into their needs and expectations for the application.

- Functional Requirements: During this phase, we will identify and document the essential features the application must have. This includes:

➢ Webcam Integration: Ensuring the application can access and use the webcam for capturing video.

- Gesture Recognition Capabilities: Defining how the application will recognize and interpret different hand gestures.
- Text Display Features: Determining how the recognized gestures will be converted into text and displayed to the user.
- Non-Functional Requirements: We will also define important non-functional aspects, such as:
- Application Responsiveness: The application should respond quickly to user inputs.
- Processing Speed: The system should process gestures and display text without noticeable delays.
- User Safety: Ensuring that the application is safe to use and does not expose users to any risks.
- Data Security: Protecting user data and ensuring compliance with relevant data protection laws.

2. Phase 2: Design

- User Interface Design: We will create high-fidelity wireframes and clickable prototypes for the user interface. The focus will be on making the application easy to use for individuals who communicate using sign language.
- Consistent Design Theme: The design will be visually appealing and accessible to all users. This includes using colors, fonts, and layouts that are easy to read and navigate.
- System Architecture Design: We will design the overall structure of the application using:
- Class Diagram: This will illustrate the relationships between different objects in the application.
- Use Case Diagram: This will map out how users interact with the application, highlighting different user scenarios.

3. Phase 3: Development

- Frontend Development: We will use Streamlit to build an interactive web-based user interface. This will allow users to easily interact with the application.
- Backend Development: The backend will be developed using Python. We will integrate libraries like OpenCV and MediaPipe to enable gesture recognition.
- Feature Implementation: Key features to be implemented include:
- Real-time Webcam Input: Capturing live video from the webcam.
- Gesture Detection: Recognizing hand gestures in real-time.
- Text Display: Converting recognized gestures into text and displaying them on the screen.

4. Phase 4: Testing

- Testing Types: We will conduct various types of testing to ensure that all functionalities work as intended. This includes:
- Unit Testing: Testing individual components of the application to ensure they function correctly.
- Integration Testing: Checking that different components work together seamlessly.
- User Acceptance Testing: Gathering feedback from actual users to ensure the application meets their needs.
- Usability and Performance: We will ensure that the application is usable for all users and performs well under

different conditions. Additionally, we will verify that it complies with data protection laws.

- **Project Resources-**

1. Human Resources

To successfully complete the project, we will need a skilled team with various roles:

- Frontend Developers: Responsible for building the user interface and ensuring it is user- friendly.
- Backend Developers: Focused on developing the server-side logic and integrating gesture recognition features.
- AI/ML Specialist: An expert in artificial intelligence and machine learning who will work on gesture recognition algorithms.
- UI/UX Designer: Responsible for creating an appealing and accessible design for the application.
- Quality Assurance Tester: Ensures that the application is tested thoroughly and meets quality standards.
- Project Manager: Oversees the project, ensuring that it stays on schedule and within budget.

2. Software Resources-

We will utilize various software tools and libraries to develop the application:

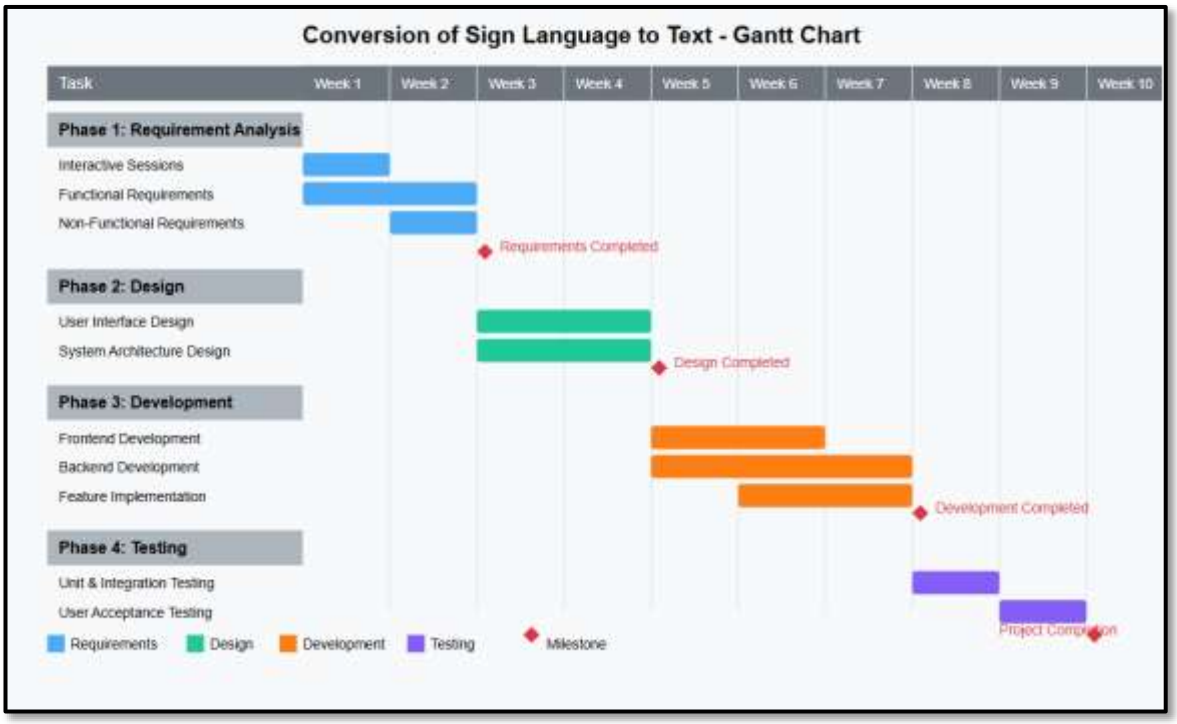
- Streamlit: This framework will be used for frontend development, allowing us to create an interactive web application quickly.
- OpenCV and MediaPipe: These libraries will be used for gesture recognition, enabling the application to interpret hand movements accurately.
- Python: The primary programming language for backend logic and functionality.
- GitHub: A platform for version control and collaboration among team members, allowing us to manage code changes and work together efficiently.

3. Hardware Resources

To support the development and deployment of the application, we will require specific hardware resources:

- Developer Machines: Each developer will need a machine with at least an Intel Core i5 processor, 8GB of RAM, and a 256GB SSD to ensure smooth development and testing processes.
- Server Specifications: For hosting the application and storing data, we will need a server with a minimum of 4GB RAM and a 500GB SSD. This will ensure that the application runs efficiently and can handle user requests effectively.

7.2 Project Scheduling



7.3 Effort Estimation

Phase	Estimated Effort (Person-Hours)	Percentage of Total Effort
Requirement Analysis	80	10%
Design	120	15%
Frontend Development	200	25%
Backend Development	200	25%
Integration	80	10%
Testing	120	15%
Total	800	100%

Diagram title: Effort Estimation

This structured project plan and schedule ensure that the sign language to text conversion application is developed efficiently, meeting the needs of its users while adhering to timelines and resource constraints.

Chapter 8-Risk Management and Analysis

8.1 Project Risk Identification

Finding possible problems early is an important part of managing the **"Conversion of Sign Language to Text "** project. These problems could affect how well the system works, how fast it runs, how long it takes to finish, or how easy it is to use. Below are some of the main risks that could happen during this project.

- **Technical Risks:**

➢ Gesture not recognized correctly: The system might not detect hand signs properly if the lighting is poor, the hand is tilted, or part of the hand is hidden.
➢ Slow performance: The app might run slowly if OpenCV, MediaPipe, and the CNN model don't work well together during live video.
➢ Difficult to connect parts: It may be hard to make the backend (logic) and the front end (Streamlit UI) work smoothly, especially during real-time video detection.

- **Resource Risks:**

➢ Not enough powerful computers: We may not have access to strong computers (like ones with GPUs) that are needed to train and test the CNN model properly.
➢ Lack of skilled team members: There might not be enough team members who know deep learning or computer vision well, especially when they're most needed.
➢ Problems with tools we depend on: Since we are using free tools like MediaPipe, Streamlit, and OpenCV, there could be issues if the versions don't work well together or if they are updated unexpectedly.

- **Schedule Risks:**

➢ Delays in dataset collection and preprocessing, which may postpone model training.

➢ Extended debugging time required to fine-tune gesture classification accuracy and UI responsiveness.

- **Security and Compliance Risks:**

➢ Unauthorized access to webcam or image data if the Streamlit interface is deployed online without proper security controls.
➢ Data privacy concerns if user-uploaded videos or webcam inputs are stored or misused without consent.

- **Financial Risks:**

➢ Spending more money than planned: We might need to buy tools, datasets, or use paid cloud services for testing, which can increase the total cost.
➢ Future costs for updates: Later, we may need to spend extra money to fix, update, or retrain the model as new data becomes available

8.2 Project Risk Analysis-

Risk analysis means carefully looking at the problems (risks) that might affect our project. We check how likely each problem is to happen and how big its effect could be if it does happen. Based on that, we decide which risks are the most serious and need to be handled first. This helps us plan how to reduce or avoid these problems before they cause any damage.Risk analysis involves evaluating the likelihood and impact of identified risks, prioritizing them for mitigation planning. Below is an analysis of the risks categorized by severity:

Risk Category	Risk Description	Likelihood	Impact	Mitigation Strategy
Technical	Inaccurate gesture recognition due to lighting or occlusions	High	High	Use data augmentation, improve model generalization, optimize lighting
Technical	Real-time latency in detection and text conversion	Medium	High	Optimize code, reduce frame processing load, test on various systems
Technical	Integration issues between Streamlit UI and backend	Medium	Mediu m	Modular development, unit testing of each component
Resource	Limited access to GPU or hardware for model training/testing	Medium	Mediu m	Use cloud services, schedule shared hardware usage
Resource	Shortage of experienced team members for deep learning	Medium	High	Team training, assign mentors, refer to documentation/tutorials
Schedule	Delay in dataset collection and annotation	Medium	High	Begin early, set weekly goals, automate preprocessing where possible
Schedule	Debugging and performance tuning delays	Medium	Mediu m	Follow Agile development, conduct weekly reviews
Security	Unauthorized access to webcam/video input	Low	High	Use secure permissions, restrict access, implement warnings in UI
Security	Privacy concerns over user-uploaded videos	Low	Mediu m	Avoid data storage, use session-based inputs only
User Acceptance	Unfriendly or non-intuitive UI design	Medium	Mediu m	Conduct user testing, iterate UI based on feedback

User Acceptance	Low model accuracy leading to user dissatisfaction	High	High	Continuous model training, gather diverse data

Chapter 9-Configuration Management

9.1 Installation/Uninstallation

- **Prerequisites:**
  - ➢ Python
  - ➢ Streamlit
  - ➢ OpenCV
  - ➢ MediaPipe
  - ➢ NumPy
  - ➢ Keras
  - ➢ TensorFlow

- **Installation**
  - ➢ Set Up Python Environment:
    - ○ Install Python 3.11+ from Python's official site.
    - ○ Create a virtual environment: python -m venv sign_language_env
    - ○ Activate the virtual environment:
      - ▪ On Windows: .\sign_language_env\Scripts\activate
      - ▪ On macOS/Linux:source sign_language_env/bin/activate
  - ➢ Install Dependencies:
    - ○ Install the required libraries:pip install streamlit opencv-python mediapipe numpy keras tensorflow
  - ➢ Download or Clone the Repository:
    - ○ Clone the project repository or download the source code to your local machine.
  - ➢ Set Up the Model:
    - ○ Download the pre-trained model CNN_KERAS_MODEL.h5 or train your model using theprovided script.
    - ○ Ensure the model is placed in the correct directory for the app to access it.
  - ➢ Run the Application:
    - ○ Navigate to the project directory and run the Streamlit app: streamlit run app.py
  - ➢ Verify Installation:

○ Open your browser and go to http://localhost:8501/ to verify the app is working.

- **Uninstallation Steps:**

➢ Deactivate Virtual Environment:

○ Deactivate the environment by running: deactivate

➢ Uninstall Dependencies:

○ If you want to remove all installed dependencies, you can delete the virtual environment folder or manually uninstall the packages: pip uninstall streamlit opencv-python mediapipe numpy keras tensorflow

➢ Remove the App Files:

○ Delete the project files or remove the cloned repository.

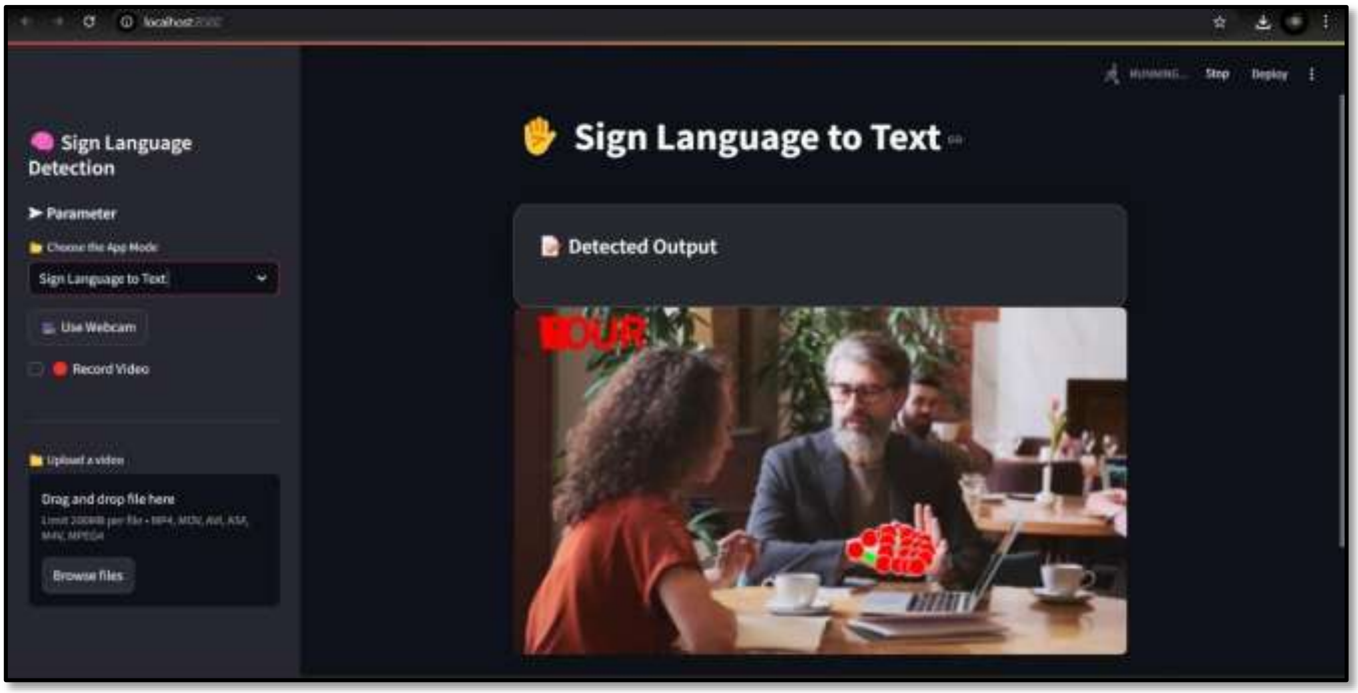
➢ Remove Python Environment:

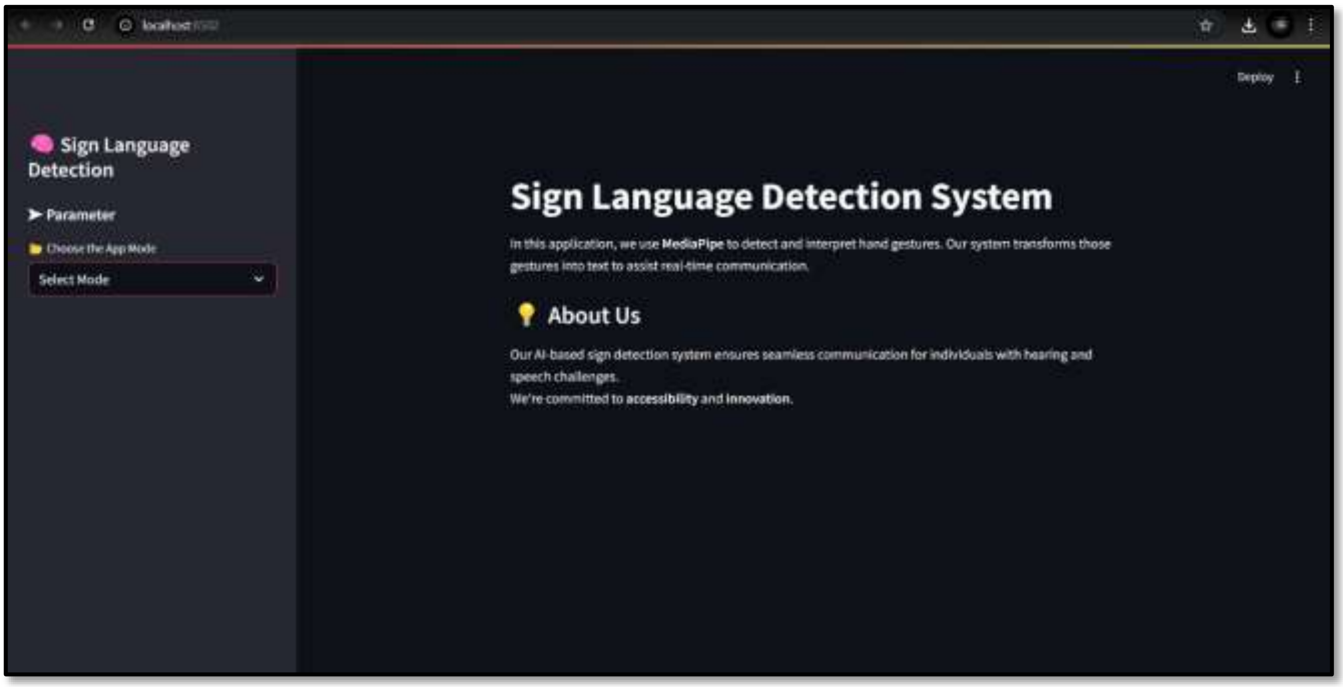
○ Delete the sign_language_env folder if no longer needed.

➢ Backup Data (if required):

○ Ensure any user data generated by the app is backed up before removing any files.

9.2 User Manual



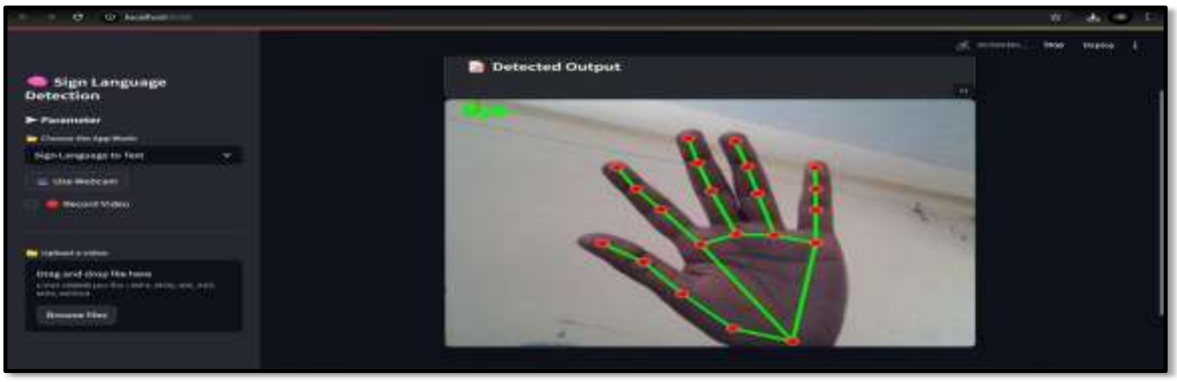


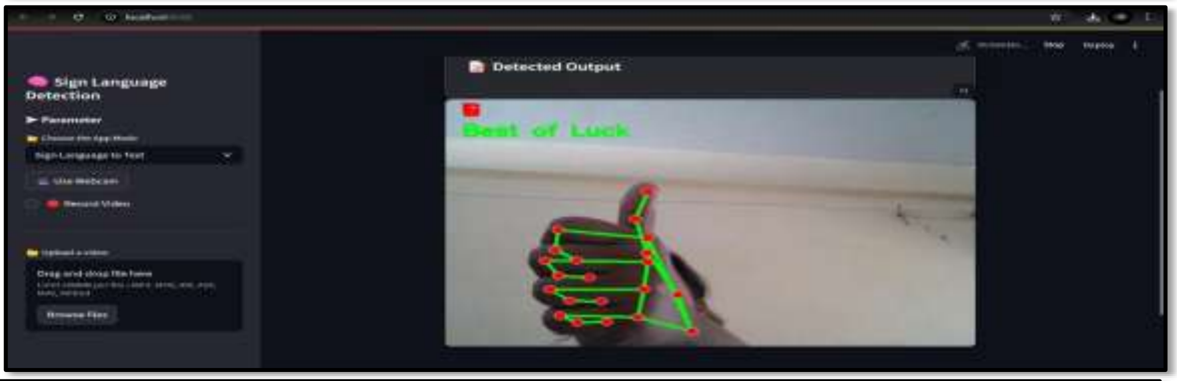
Sign Language Detection System

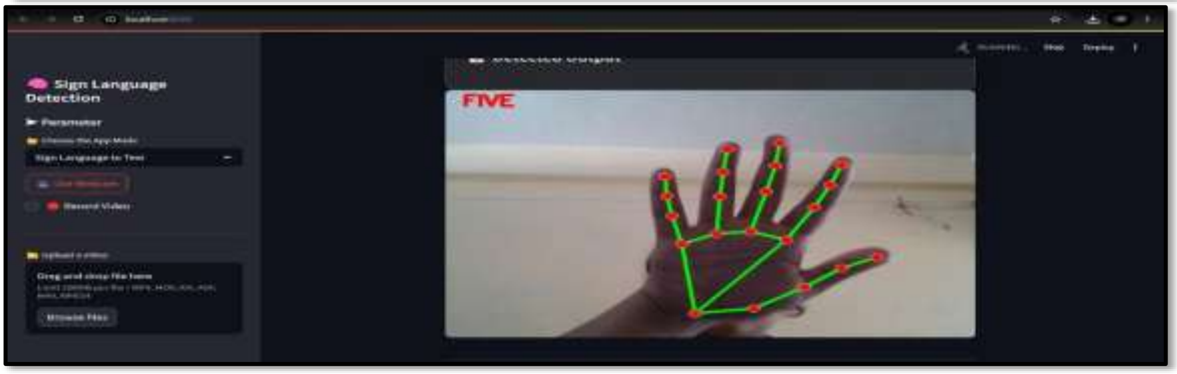
In this application, we use **MediaPipe** to detect and interpret hand gestures. Our system transforms those gestures into text to assist real-time communication.

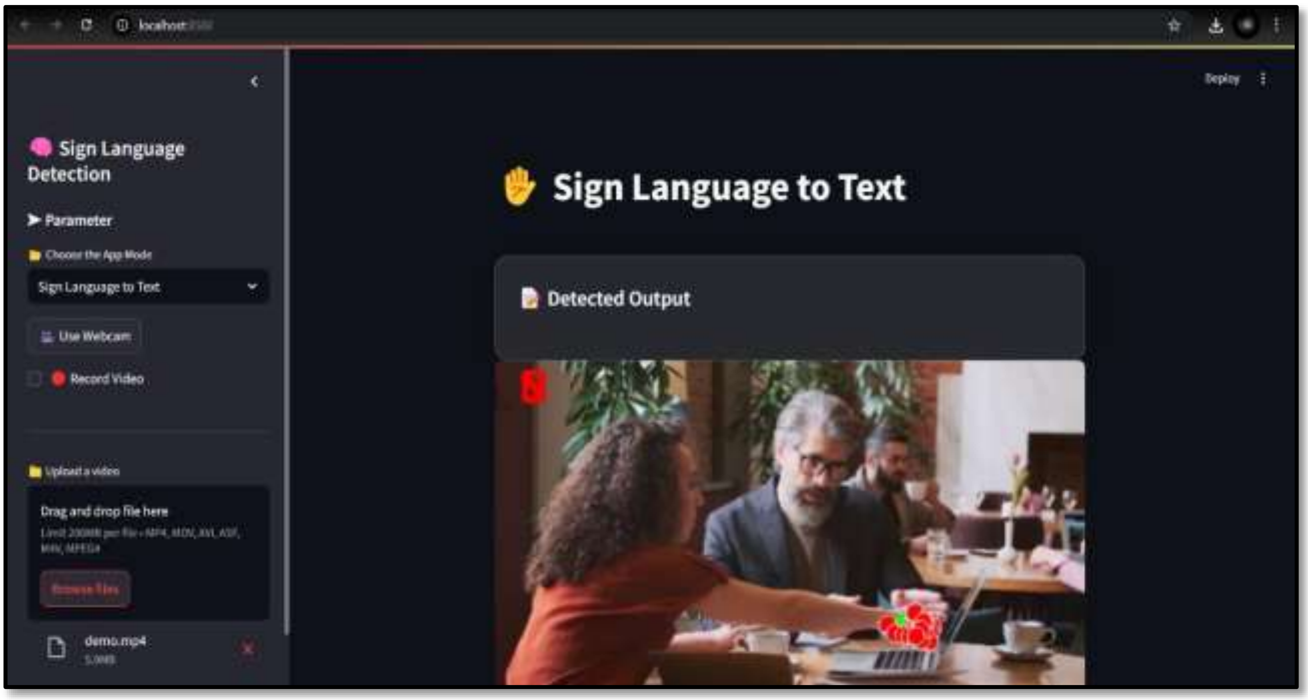
About Us

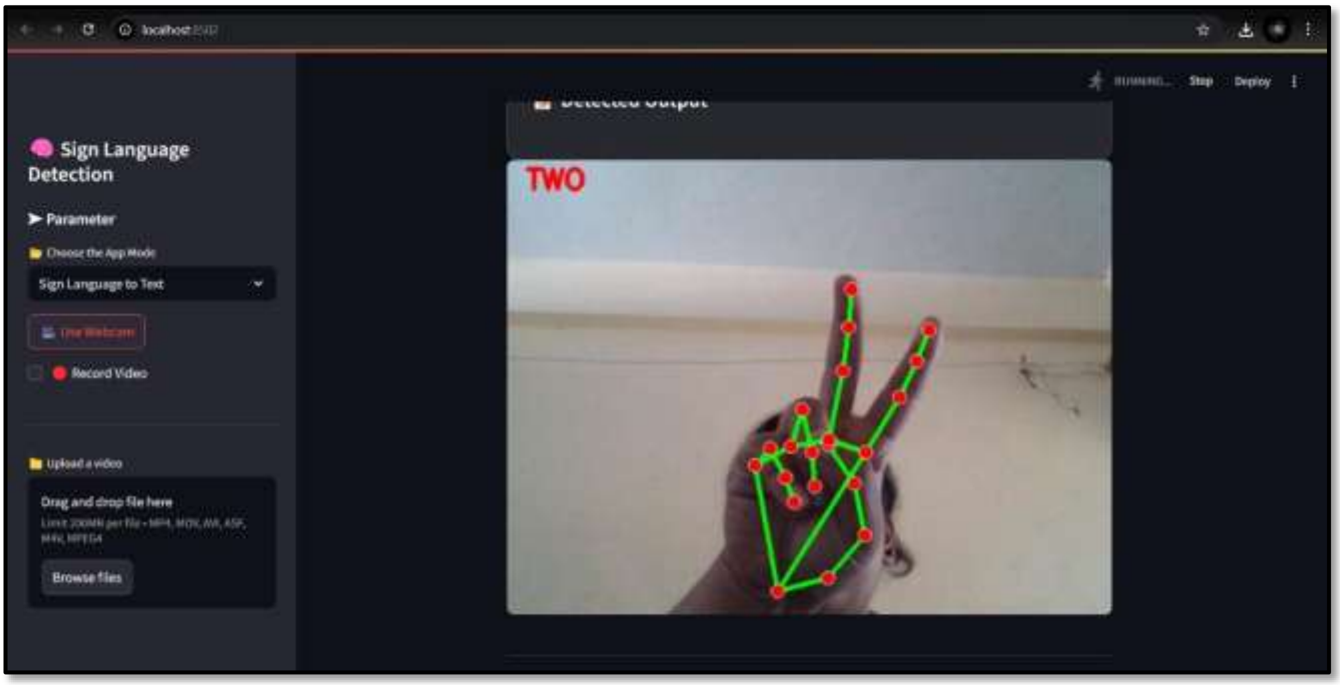
Our AI-based sign detection system ensures seamless communication for individuals with hearing and speech challenges.
We're committed to **accessibility** and **innovation**.

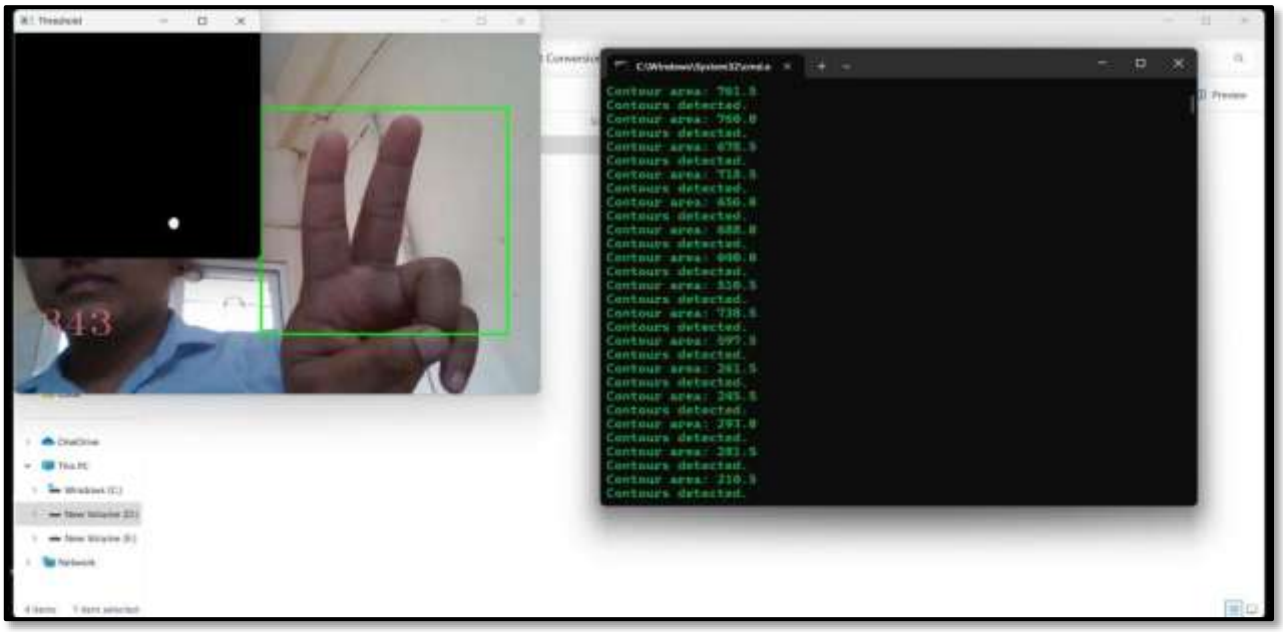




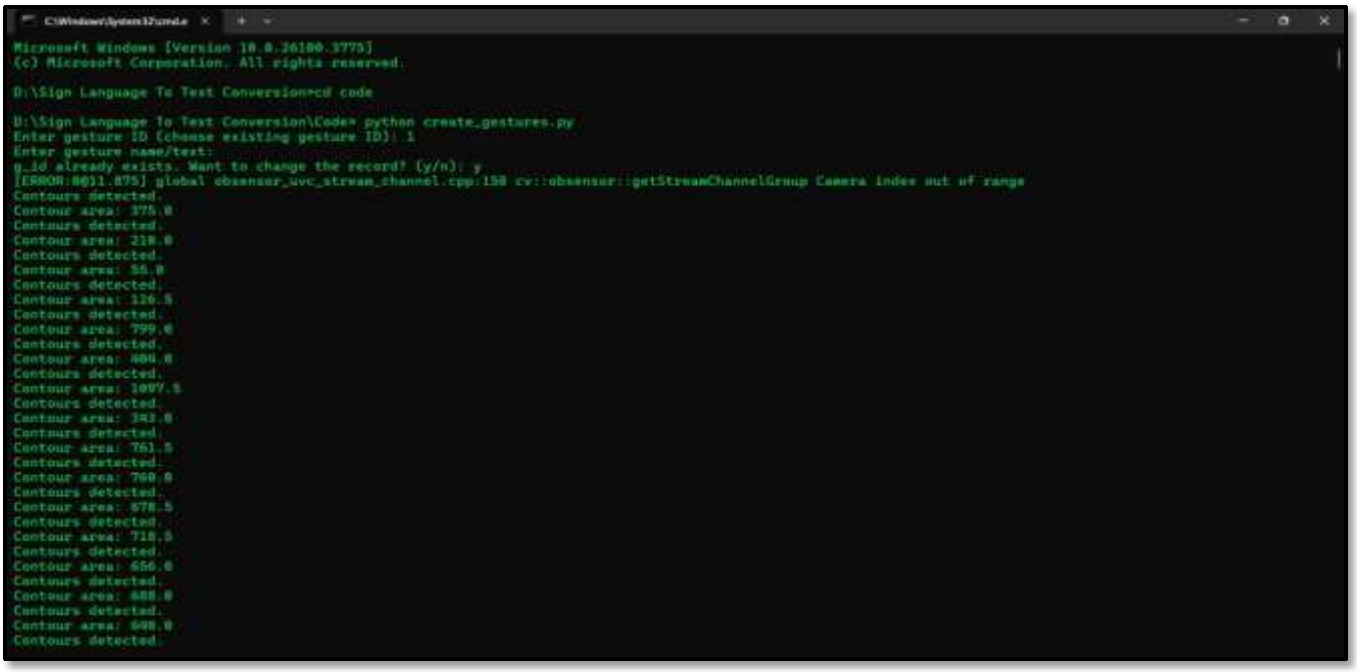



localhost
Deploy
Sign Language Detection
Parameter
Choose the App Mode
Sign Language to Text
Use Webcam
Record Video
Upload a video
Drag and drop file here
Limit 200MB per file • MP4, MOV, AVI, ASF, M4V, MPEG4
Browse files
demo.mp4
Sign Language to Text
Detected Output


localhost
RUNNING...
Stop
Deploy
Sign Language Detection
Parameter
Choose the App Mode
Sign Language to Text
Use Webcam
Record Video
Upload a video
Drag and drop file here
Limit 200MB per file • MP4, MOV, AVI, ASF, M4V, MPEG4
Browse files
Detected Output
TWO



```
Contour area: 761.5
Contours detected.
Contour area: 760.0
Contours detected.
Contour area: 678.5
Contours detected.
Contour area: 718.5
Contours detected.
Contour area: 656.0
Contours detected.
Contour area: 688.0
Contours detected.
Contour area: 698.0
Contours detected.
Contour area: 510.5
Contours detected.
Contour area: 738.5
Contours detected.
Contour area: 597.5
Contours detected.
Contour area: 261.5
Contours detected.
Contour area: 245.5
Contours detected.
Contour area: 293.0
Contours detected.
Contour area: 283.5
Contours detected.
Contour area: 210.5
Contours detected.
```



```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

D:\Sign Language To Text Conversion>cd code

D:\Sign Language To Text Conversion\Code> python create_gestures.py
Enter gesture ID (choose existing gesture ID): 1
Enter gesture name/text:
g_id already exists. Want to change the record? (y/n): y
[ERROR:0@11.875] global obsensor_uvc_stream_channel.cpp:158 cv::obsensor::getStreamChannelGroup Camera index out of range
Contours detected.
Contour area: 375.0
Contours detected.
Contour area: 218.0
Contours detected.
Contour area: 55.0
Contours detected.
Contour area: 126.5
Contours detected.
Contour area: 799.0
Contours detected.
Contour area: 404.0
Contours detected.
Contour area: 1097.5
Contours detected.
Contour area: 343.0
Contours detected.
Contour area: 761.5
Contours detected.
Contour area: 760.0
Contours detected.
Contour area: 678.5
Contours detected.
Contour area: 718.5
Contours detected.
Contour area: 656.0
Contours detected.
Contour area: 688.0
Contours detected.
Contour area: 698.0
Contours detected.
```

Chapter 10-Conclusion and Future Scope

10.1 Conclusion

Communication is a basic human need, and for people who are deaf or mute, sign language plays a vital role. However, due to the lack of awareness and understanding of sign language among the general public, many people with hearing and speech impairments face difficulties in daily life. They often struggle to express themselves clearly to others who do not

know sign language, leading to feelings of isolation and frustration.
This project, “Conversion of Sign Language to Text,” was developed to help reduce that communication gap. The main goal is to support the deaf and mute community by providing a tool that helps them communicate more easily with people who do not understand sign language.
We created a real-time system that uses a regular webcam to detect and recognize static hand gestures. The system uses technologies such as MediaPipe for hand landmark detection, OpenCV for image processing, and a Convolutional Neural Network (CNN) to classify the gestures. The recognized gesture is then converted into human-readable text, which can be displayed on the screen for others to read.
Our system is designed to be:

Affordable – No need for expensive tools like sensor gloves or 3D cameras.

Accessible – Can be used on a regular computer with a webcam.

User-friendly – Easy interface suitable for both technical and non-technical users. During the development process, we focused on building a system that is accurate, fast, and works efficiently in real-time. We trained our model on a custom dataset of hand gestures, which includes various signs used for basic communication. The system performs well in identifying these signs, even under different lighting conditions and backgrounds.
In addition to the technical goals, this project also helps create social awareness about the importance of inclusive communication. It shows how technology can be used to support people with disabilities and make society more understanding and connected.
We also tested the system for usability and performance, and based on feedback, we found that it could be very helpful in real-world situations like schools, hospitals, public places, or even homes where hearing-impaired individuals need to express themselves quickly and clearly.
This project not only fulfills an academic purpose but also carries social value. It demonstrates how computer science and machine learning can contribute to real-life problem-solving and improve the quality of life for many individuals.

10.2 Future Scope

This project has great potential for further development to make it more advanced, useful, and accessible. Below are some areas where the system can be improved in the future:

1. Dynamic Gesture Recognition

Currently, the system works only with static hand signs (like showing one letter at a time). In the future, it can be upgraded to recognize moving gestures or sequences of signs. This would allow it to understand full sentences in sign language, making communication more natural.

2. Support for More Signs and Languages

Right now, the system supports only a few gestures. In the future:

It can include all alphabets (A–Z), numbers, and commonly used words.

It can also be trained for different sign languages, such as ASL (American Sign Language), ISL (Indian Sign Language), or BSL (British Sign Language).
It could also handle regional gestures used in different parts of the world.

3. Voice Output Integration

After detecting a hand gesture and converting it into text, the system can be improved to speak the text using text-to-speech (TTS) technology. This would help the deaf-mute user communicate with people who prefer to hear responses rather than read them.

4. Mobile App Development

A mobile version of the system can be developed, allowing users to: Use their smartphone cameras instead of a webcam. Access the system anytime, anywhere, making it more flexible and convenient.

5. Two-way Communication

In the future, the system can support both directions: Sign to text/speech (as it currently does).
Speech or text to animated sign language, helping hearing people also communicate back to the deaf-mute user.

6. Improved Accuracy with Bigger Datasets

To make the system more accurate and reliable.

Larger and more diverse datasets can be used for training.

Data with different hand shapes, skin tones, lighting, and backgrounds can help the model perform better in real-life situations.

7. Multi-hand Detection

Future versions can detect both hands at once, allowing more complex gestures that require two-hand signs (common in full sign language communication).

8. Context Understanding

The system can be trained to understand the context of gestures. For example, the same sign might mean different things in different situations, and context-aware processing can improve communication quality.

9. Web Integration

The system can also be turned into a web application, making it accessible through browsers without downloading any software, which is useful for educational institutions or public service centers.

10. User Personalization

Future updates could allow users to customize gestures or add their own signs, which is helpful for people with unique needs or communication styles.

References

1) Journal/Conference Papers:

- American Sign Language Recognition System Shivashankara S and Srinath S 08 August 2018.
- Video-Based Sign Language Recognition without Temporal Segmentation Jie Huang, Wengang

Zhou, Qilin Zhang, Houqiang Li, Weiping Li

- ➢ A Review on Indian Sign Language Recognition Anuja V. Nair Bindu V.

2) Web References:

- https://www.indiegogo.com/projects/motionsavvy-uni-1st-sign-language-to- voicesystem#/
- https://www.accessibility.com/org-spotlight/business/signall- technologies 3. https://www.handtalk.me/en/plugin/

3) Book References:

- ➢ International Perspectives on Sign Language Interpreter Education 1st Edition Edited by Jemina Napier.
- ➢ International Sign Linguistic, Usage, and Status Issues Edited by Rachel Rosenstock & Jemina Napier.