**SIGN2SPEECH: CONVERSION OF SIGN LANGUAGE INTO TEXT AND SPEECH**

##### A PROJECT REPORT

***Submitted by***

**SUBIKSHA S [REGISTER NO:211423104657] SANTHIYA B [REGISTER NO:211423104585]**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

##### COMPUTER SCIENCE AND ENGINEERING

**PANIMALAR ENGINEERING COLLEGE**

**ANNA UNIVERSITY: CHENNAI 600 025**

##### OCTOBER 2025

**BONAFIDE CERTIFICATE**

Certified that this project report **“SIGN2SPEECH: COVERSION OF SIGN LANGUAGE INTO TEXT AND SPEECH”** is the Bonafide work of “SUBIKSHA S [211423104657], SANTHIYA B [211423104585],**”** who carried out the project work under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE** | **SIGNATURE** |
| Dr.L.JABASHEELA,M.E.,Ph.D., HEAD OF THE DEPARTMENT | Dr.V.SUBEDHA,M.E.,Ph.D., SUPERVISOR |
| DEPARTMENT OF CSE, PANIMALAR ENGINEERING  COLLEGE, NASARATHPETTAI, POONAMALLEE, CHENNAI-600 123. | DEPARTMENT OF CSE, PANIMALAR ENGINEERING  COLLEGE, NASARATHPETTAI, POONAMALLEE, CHENNAI-600 123. |

Certified that the above candidates were examined in the Anna University Project Viva-Voce Examination held on...........................

##### INTERNAL EXAMINER EXTERNAL EXAMINER

**DECLARATION BY THE STUDENT**

We “SUBIKSHA S [211423104657], SANTHIYA B [211423104585]”

hereby declare that this project report titled “**SIGN2SPEECH: COVERSION OF SIGN LANGUAGE INTO TEXT AND SPEECH**”, under the guidance of MS.SUNTHEYA.A.K,M.E.CSE is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

**ACKNOWLEDGEMENT**

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI**, **Dr.C.SAKTHI KUMAR,M.E.,Ph.D** and **Dr.SARANYASREE SAKTHI**

**KUMAR B.E.,M.B.A.,Ph.D.,** for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr.L.JABASHEELA , M.E.,Ph.D.,**

for the support extended throughout the project.

We would like to thank my **Project Guide Ms.SUNTHEYA .A.K, M.E.CSE** and coordinator **DR.M.KRISHNAMOORTHY M.E.,M.B.A.,Ph.D., and** all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

**SUBIKSHA S SANTHIYA B**

### Abstract

Millions of people worldwide rely on sign language as their primary means of communication, yet many outside the deaf and hard-of-hearing communities lack proficiency in it. This communication gap creates barriers in personal, professional, and educational settings. Traditional solutions like hiring interpreters are often costly and impractical for everyday use, limiting social inclusion and equality for those who rely on sign language.

Sign2Speech addresses these challenges by offering a mobile-friendly, single-page web application that converts sign language gestures into text and speech in real time. Designed with a lightweight, intuitive interface resembling a smartphone app, the system provides instant visual and auditory feedback. Users can customize language preferences, speech volume, and other accessibility features to fit their needs. Currently, the app simulates gesture recognition but is built to support future integration with advanced machine learning models for accurate real-time sign language detection.

By fostering seamless communication between signing and non-signing individuals, Sign2Speech supports the UN Sustainable Development Goal 10 – Reduced Inequalities. It has practical applications in healthcare, education, customer service, and public spaces, helping to create more inclusive environments where immediate and effective communication is essential.

Sign2Speech aims to break down communication barriers and empower individuals with hearing impairments.

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** | i |
|  | **LIST OF FIGURES** | iv |
|  | **LIST OF TABLES** | v |
|  | **LIST OF ABBREVATIONS** | v |
| **1** | **INTRODUCTION** |  |
|  | 1.1 OVERVIEW | 2 |
|  | 1.2 PROBLEM DEFINITION | 4 |
| **2** | **LITERATURE REVIEW** | 7 |
| **3** | **SYSTEM ANALYSIS** |  |
|  | 3.1 EXISTING SYSTEM | 13 |
|  | 3.2 PROPOSED SYSTEM | 14 |
|  | 3.3 FEASIBILITY STUDY | 16 |
| **4** | **THEORETICAL BACKGROUND** |  |
|  | 4.1 IMPLEMENTATION ENVIRONMENT | 21 |
|  | 4.2 SYSTEM ARCHITECTURE | 25 |
|  | 4.3 PROPOSED METHODOLOGY | 30 |
|  | 4.3.1 DATASET DESCRIPTION | 31 |
|  | 4.3.2 INPUT DESIGN | 31 |
|  | 4.3.3 MODULE DESIGN | 32 |
| **5** | **SYSTEM IMPLEMENTATION** |  |
|  | 5.1 MODULES | 40 |

|  |  |  |
| --- | --- | --- |
| **6** | **RESULTS & DISCUSSIONS** |  |
|  | 6.1 TESTING | 46 |
|  | 6.2 RESULTS & DISCUSSIONS | 51 |
| **7** | **CONCLUSION & FUTURE WORK** |  |
|  | 7.1 CONCLUSION | 54 |
|  | 7.2 FUTURE WORK | 55 |
|  | **APPENDICES** |  |
|  | A.1 SDG GOALS | 57 |
|  | A.2 SOURCE CODE | 59 |
|  | A.3 SCREENSHOTS | 70 |
|  | A.4 PLAGARISM REPORT | 73 |
|  | **REFERENCES** | 75 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **NAME** | **PAGE NO** |
| 4.1.3 | DEEP LEARNING | 22 |
| 4.1.3 | PYTHON | 23 |
| 4.1.3 | ANACONDA | 24 |
| 4.2 | ARCHITECTURE DIAGRAM | 25 |
| 4.3.31 | USE CASE DIAGRAM | 32 |
| 4.3.3.2 | SEQUENCE DIAGRAM | 33 |
| 4.3.3.3 | ACTIVITY DIAGRAM | 34 |
| 4.3.3.4 | CLASS DIAGRAM | 35 |
| 4.3.3.5 | DFD LEVEL-0 | 36 |
| 4.3.3.5 | DEF LEVEL-1 | 36 |
| 4.3.3.5 | DFD LEVEL-2 | 37 |
| 4.3.3.6 | COLLABARATION DIAGRAM | 38 |
| 6.2 | ACCURACY SCORE | 52 |
| A.3.1 | USER INTERFACE | 70 |
| A.3.2 | BROWSE IMAGE | 70 |
| A.3.3 | SELECT IMAGE | 71 |
| A.3.4 | RESULT | 71 |
| A.3.5 | RESULT | 72 |
| A.4 | PLAGIARISM REPORT | 73 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **TABLE NO** | **NAME** | **PAGE NO** |
| 6.1.1 | UNIT TESTING | 46 |
| 6.1.6 | TEST CASES | 50 |

**LIST OF ABBREVATIONS**

|  |  |
| --- | --- |
| CNN | Convolutional Neural Network |
| HAM | Human Against Machine |
| ViT | Vision Transformers |
| AI | Artificial Intelligence |
| QNN | Quanvolutional Neural Network |

## INTRODUCTION

### Introduction

#### Overview:

Globally, it is estimated that more than 70 million people use some form of sign language as their primary means of communication. Sign languages are complete natural languages with their own grammar and structure, yet they are not universally understood by people outside the deaf and mute community. This leads to frequent breakdowns in communication, especially in essential contexts such as medical consultations, academic discussions, workplace interactions, and customer service encounters.

Although interpreters can help bridge this communication gap, they are not always accessible, affordable, or available in urgent situations. Many individuals with hearing impairments therefore face challenges in expressing their needs and emotions in real-time. In addition, the lack of familiarity with sign language among the general population further limits inclusivity and social participation.

With advancements in artificial intelligence, computer vision, and speech technologies, it is now possible to explore software-based tools that can automatically translate gestures into text or speech. These innovations have the potential to empower individuals with hearing or speech impairments by giving them greater independence in their daily interactions. Sign2Speech is one such attempt to design a lightweight, mobile-first solution that leverages modern computing without demanding expensive hardware.

#### Problem Definition:

The absence of widely available, cost-effective, and real-time sign language translation systems continues to pose serious barriers to communication. While several research prototypes exist, many require specialised equipment like data gloves or depth cameras, which are unsuitable for everyday use. Others function only under controlled environments with ideal lighting or static gestures, limiting their practical applicability.

This project addresses the gap by proposing a mobile/web application that uses a regular camera to detect and interpret signs in real time. The recognised gestures are converted into readable text and then into spoken words, making interaction smoother between signers and non-signers. By focusing on affordability and simplicity, the proposed solution ensures that people can use it in hospitals, classrooms, banks, or even casual social interactions without technical expertise.

**1.3 Objectives**

The core objectives of this project are as follows:

1. To capture sign gestures through a mobile or web camera interface without the need for additional devices.
2. To process and classify gestures into meaningful units (letters, words, or symbols) using machine learning and image processing techniques.
3. To generate speech output from the recognised text using text-to-speech technology, enabling real-time verbal communication.
4. To allow user customisation, such as choosing preferred output languages, enabling or disabling speech synthesis, and adjusting accessibility features like dark mode or notifications.
5. To ensure the system is lightweight and portable, capable of running on smartphones, tablets, or basic laptops without requiring external GPUs or costly hardware.

**1.4 Scope**

The project is designed as a proof-of-concept prototype that demonstrates how modern mobile/web technologies can be adapted for accessibility solutions. The current version primarily supports recognition of static gestures (such as alphabets or predefined words) and simple dynamic gestures, which are then translated into English text and audio.

In real-world contexts, this system can be deployed in hospitals to help patients convey symptoms, in banks to assist customers with transactions, or in educational institutions to make classrooms more inclusive. It also has potential in public service offices where interpreters are not available. While the current implementation does not handle full sentence recognition or advanced grammar of sign language, it establishes a strong foundation for future enhancements.

Future extensions could include training the system with larger datasets, integrating deep learning models for continuous sign recognition, and enabling multi-language speech output for cross-cultural communication.

**1.5 SDG Justification**

The United Nations’ Sustainable Development Goal (SDG) 10: Reduced Inequalities emphasizes the importance of empowering marginalized communities and ensuring equal opportunities for all. People with hearing and speech disabilities often face exclusion due to communication barriers, limiting their participation in social, educational, and professional domains.

By enabling real-time translation of sign language into text and speech, Sign2Speech directly contributes to reducing these inequalities. It fosters inclusivity by ensuring that individuals with hearing impairments can interact seamlessly with non-signers without requiring interpreters. Furthermore, as the system is designed to be low-cost and accessible on common devices, it promotes digital equity by making assistive technology available to a broader population, including those in resource-constrained environments.

In this way, Sign2Speech is not only a technical project but also a socially impactful innovation aimed at promoting accessibility, independence, and inclusivity for one of the world’s most underserved communities

# 

LITERATURE

REVIEW

### 2. Literature Review

**2.1 Overview**

Research in sign language recognition has spanned wearable devices, computer vision systems, and deep learning architectures. Early solutions relied on sensor gloves and accelerometers, which limited adoption due to cost and discomfort. Recent trends focus on camera-based systems powered by deep learning, offering greater usability.

**2.2 Related Work Summaries**

**2.2.1 SignExplainer (IEEE Access, 2023)**

* Approach: Explainable ensemble AI framework.
* Strength: Improved classification accuracy and model interpretability.
* Limitation: Works well only in controlled conditions, limited to static gestures.

**2.2.2 SIGNFORMER (IEEE Access, 2023)**

* Approach: Vision Transformer-based model for continuous sign recognition.
* Strength: High accuracy in dynamic gesture recognition.
* Limitation: Computationally expensive, GPU-dependent, hard to deploy on mobile.

**2.2.3 CNN-BiLSTM with RF Signals (IEEE Access, 2024)**

* Approach: Hybrid CNN + BiLSTM model using RF data.
* Strength: Captures spatial + temporal features, strong accuracy.
* Limitation: Requires specialised RF hardware, low portability.

**2.3 Gaps Identified**

* High hardware requirements.
* Limited datasets for continuous sign recognition.
* Lack of explainability in many AI models.
* Low focus on multilingual speech output and usability.

**SYSTEM ANALYSIS**

### 3. SYSTEM ANALYSIS

### 3.1 Existing System:

Communication between individuals with hearing or speech impairments and the general population remains a challenge due to the widespread lack of knowledge of sign language. While various tools and technologies have emerged to support accessibility, most existing solutions are limited in scope and functionality. Many rely on wearable sensors, expensive hardware, or rigid rule-based systems to detect predefined gestures, which significantly restricts scalability and usability.

In the mobile and web domain, a few applications offer basic gesture recognition; however, these systems typically support only static gestures and require ideal lighting and positioning. Moreover, they often lack support for speech synthesis or multilingual translation, limiting their practical effectiveness. In addition, user interfaces are frequently non-intuitive and inaccessible for individuals with limited technological exposure.

Some AI-driven sign language recognition systems exist as research prototypes. These projects employ Convolutional Neural Networks (CNNs) or MediaPipe-based hand tracking to map gestures to alphabets or words. However, they are usually trained on small, domain-specific datasets, limiting their robustness in real-world conditions.

These systems often suffer from issues such as:

* High hardware requirements (e.g., GPU for real-time inference)
* Limited vocabulary coverage
* Absence of natural language processing for phrase generation
* Lack of voice output integration

Furthermore, these applications rarely support two-way communication or dynamic gesture recognition. The absence of a fully integrated solution that converts sign language to both readable and audible text.

#### 3.2 Proposed System:

The proposed system, Sign2Speech, aims to bridge the communication gap by providing an accessible, mobile-friendly platform that interprets sign language gestures into text and converts the result into speech output. Unlike prior systems, this project focuses on delivering a seamless user experience by integrating gesture recognition, natural language processing, and speech synthesis into a unified interface.

Key innovations in the proposed solution include:

* A mobile-responsive user interface with webcam-based gesture input
* Predefined gesture mapping for alphabets or common words (initial version)
* Real-time conversion of recognized gestures into readable text
* Speech output generation using text-to-speech (TTS) modules
* Support for multiple spoken languages (English, Tamil, etc.)
* Interactive and accessible user design tailored for individuals with communication challenges

In its current form, the system serves as a proof-of-concept (PoC) that can be expanded further by integrating deep learning-based gesture recognition models (e.g., CNNs or Transformers) in future iterations. By eliminating the need for specialized hardware or high-performance computing, the application offers broad accessibility, even on low-end devices.

The application also supports modular expansion, allowing future versions to include voice-to-sign translation, conversation history, cloud storage, and feedback-driven accuracy improvement.

#### 3.3 Feasibility Study:

###### 3.3.1 Technical Feasibility:

The project is technically viable due to the maturity of the web development ecosystem and the availability of open-source libraries for gesture detection and text-to-speech conversion. Core technologies used include HTML/CSS/JavaScript for the frontend and Flask (Python) for the backend. Gesture inputs are simulated in this version, with plans to incorporate MediaPipe or TensorFlow.js for live tracking in future phases.

All components are compatible with standard computing environments, and no specialized hardware is required. Text-to-speech is handled via JavaScript-based APIs (like Web Speech API), which are browser-native and platform-independent. Modular development ensures that the system can be deployed locally or on cloud platforms like Heroku, GitHub Pages, or Firebase.

.

###### 3.3.2 Economic Feasibility:

Sign2Speech leverages free and open-source tools for its entire tech stack, minimizing development costs. Datasets for training (where needed) are publicly available, and libraries used for text-to-speech and frontend design are also free. Deployment can be accomplished using GitHub Pages or Streamlit (for Python-based demo), eliminating hosting charges in the early phases.

If commercialized in the future, potential revenue streams may include licensing for educational institutions, subscriptions for telehealth providers, or NGO-based sponsorships.

###### 3.3.3 Operational Feasibility:

The application is designed with usability and simplicity in mind. Users do not require any prior technical knowledge to operate the system. By offering visual cues, step-by-step instructions, and an intuitive dashboard, the system ensures high adoption potential among its target users.

The architecture is flexible and scalable, with a clear separation between UI, logic, and data. Any user with a smartphone, tablet, or desktop with a webcam can access the application, making it operationally feasible even in low-resource environments such as rural schools or public health centers.

###### 3.3.4 Legal and Ethical Feasibility:

Since the system does not collect or store user data in its current prototype version, it complies with standard data protection guidelines. If expanded to include cloud-based features or user profiling, privacy measures like anonymization, consent mechanisms, and GDPR compliance will be incorporated.

Ethically, the application is positioned as an assistive tool, not a medical or diagnostic system. Clear disclaimers and user instructions ensure transparency. The project aligns with inclusive technology principles and the SDG 10 goal of reducing inequality through digital access.

###### 3.3.5 Schedule Feasibility:

The development timeline for Sign2Speech follows an agile and iterative model:

* Week 1–2: Requirement gathering, tech stack finalization, UI wireframes
* Week 3–4: Frontend and backend integration, gesture simulation
* Week 5–6: Text and speech output modules
* Week 7–8: Testing, debugging, and user interface refinements
* Week 9: Documentation and deployment

This schedule ensures that the system is developed within a standard academic project window (2–3 months) without exceeding resource or time limitations

###### 

## THEORETICAL BACKGROUND

### 

**4. Theoretical Background**

#### 4.1 Implementation Environment:

## 4.1.1 Hardware Requirements:

#### Processor: Intel Core i3 and above

#### RAM: Minimum 4 GB

#### Storage: 500 GB HDD or SSD

#### GPU: Integrated graphics (sufficient for prototype); dedicated GPU like NVIDIA GTX 1050 or above (for real-time ML integration)

**4.1.2 Software Requirements:**

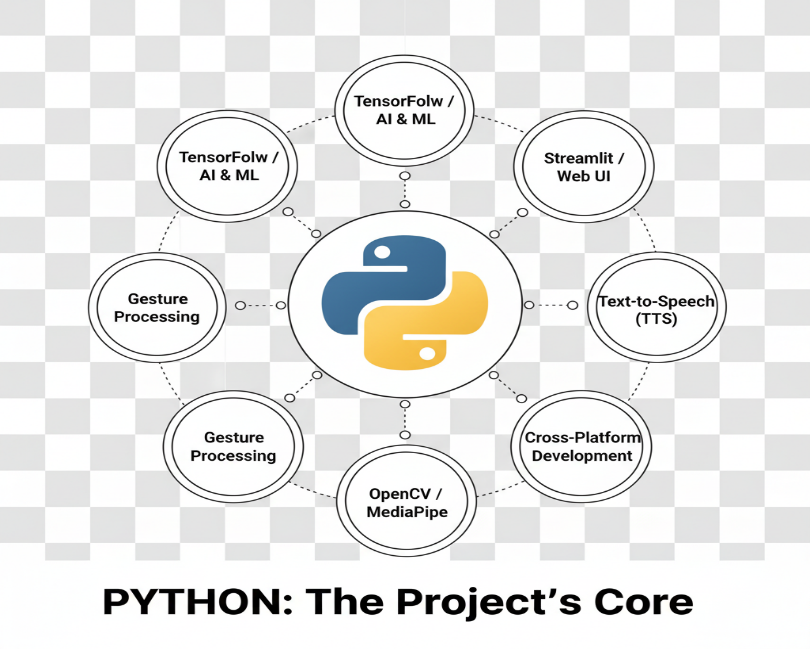
* Operating System: Windows 10 or higher (64-bit), macOS, or Linux
* Programming Language: Python 3.9+
* Tools and Libraries:
  + Streamlit (for UI)
  + OpenCV (image capture and preprocessing)
  + TensorFlow/Keras (future model integration)
  + NumPy, Pandas (data handling)
  + pyttsx3 or gTTS (for text-to-speech)
  + HTML, CSS, JavaScript

**4.1.3 Technologies Used:**

**4.1.3.1. Python**

Python serves as the backbone of this project due to its versatility and rich ecosystem of libraries tailored for AI and web development. With frameworks like OpenCV, TensorFlow, and Streamlit, Python allows for rapid prototyping and seamless integration between machine learning models and web interfaces. It also provides native support for real-time camera streaming, gesture preprocessing, and TTS engines.

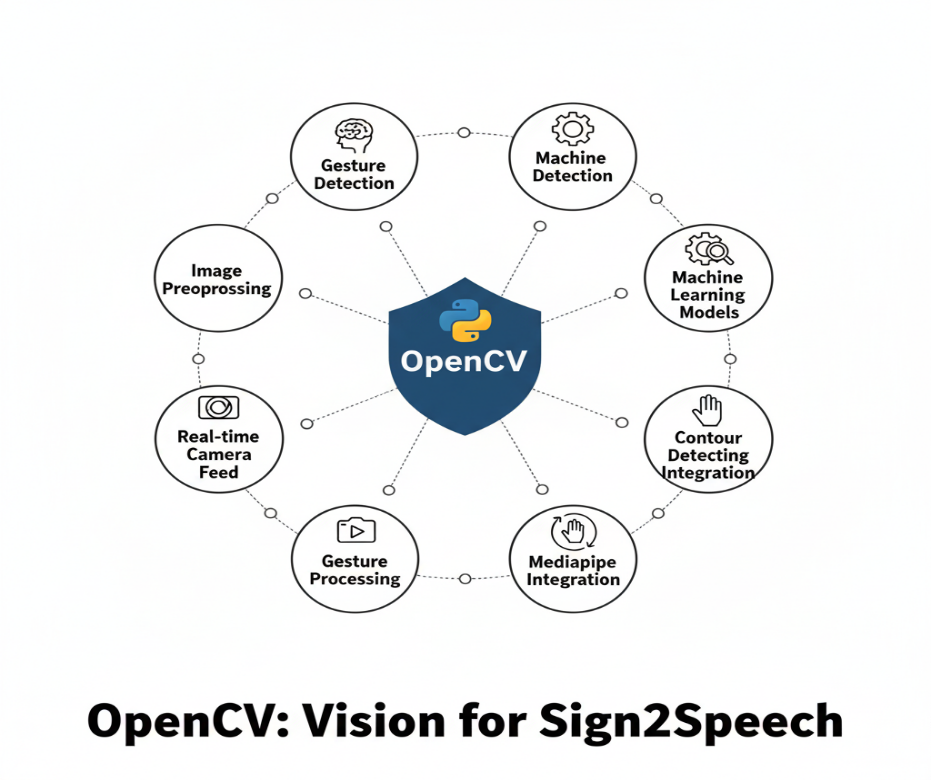
Python’s simplicity enables cross-platform development and makes it accessible to a wide range of users, including those with limited experience in AI or app development. In this project, Python is used for both backend (gesture processing, TTS) and frontend integration (through Streamlit).



**Fig.4.1.3.1 Python**

**4.1.3.2. OpenCV**

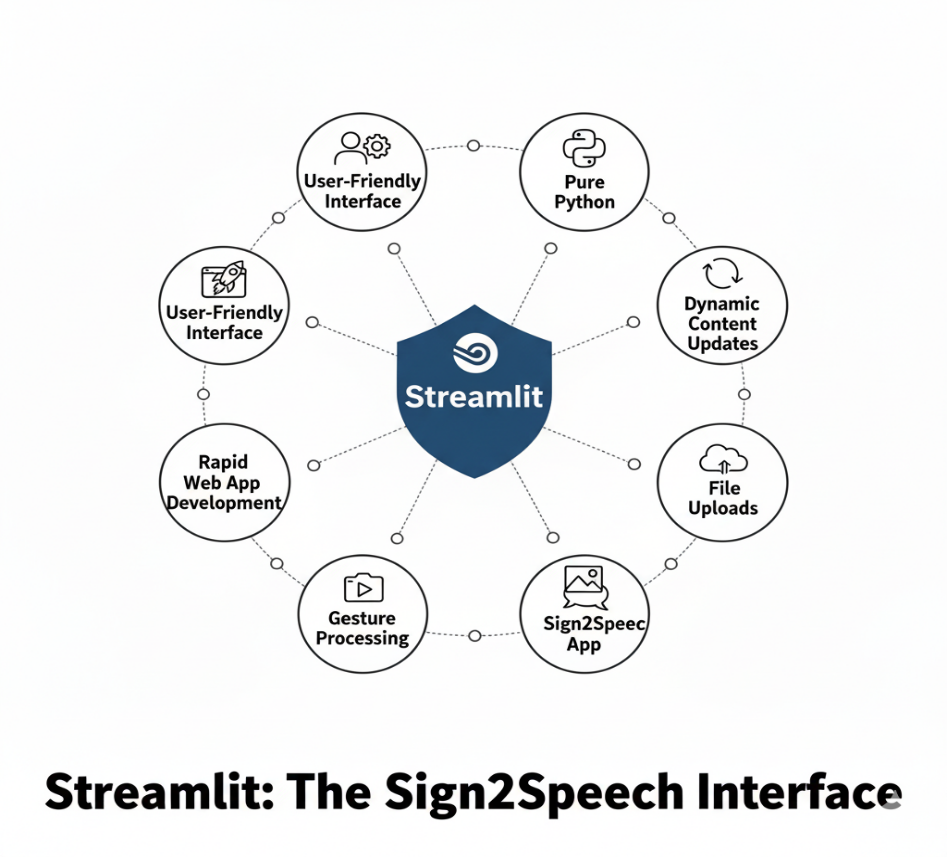
OpenCV (Open Source Computer Vision Library) is used to capture real-time frames from the webcam and apply preprocessing steps such as resizing, color filtering, and contour detection. Although the current version uses predefined gestures, OpenCV lays the foundation for integrating future gesture tracking using MediaPipe or deep learning models.



**Fig.4.1.3.2 OpenCV**

**4.1.3.3. Streamlit**

Streamlit enables rapid web app development using pure Python. It supports file uploads, image previews, button interactions, and dynamic content updates. Its ease of use allows the Sign2Speech app to offer a user-friendly interface for non-technical users — including hearing-impaired individuals — without requiring a traditional web development stack.



**Fig.4.1.3.3 Streamlit**

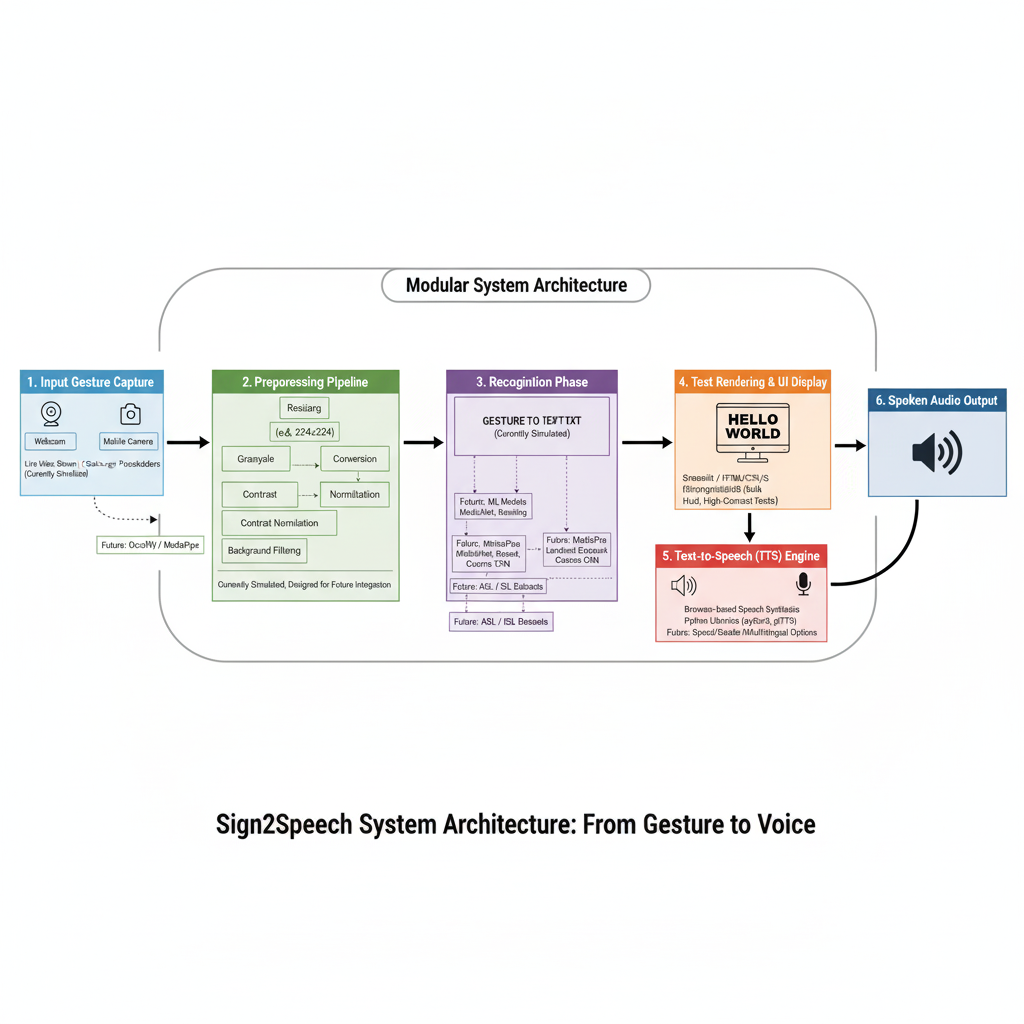
**4.1.3.4. Text-to-Speech Engine**

The project integrates a TTS engine such as pyttsx3 (offline) or gTTS (Google-based) to convert recognised sign language text into spoken audio. This allows real-time auditory output for enhanced communication with non-signers.



**Fig.4.1.3.4 TTS engine**

**4.2** **System Architecture**:

****

**Fig.4.2 Architecture Diagram**

Sign2Speech is an innovative solution that leverages advancements in AI and real-time communication to recognize sign language gestures and convert them into both readable text and spoken audio using a standard camera. Sign language is a vital medium of communication for individuals with hearing and speech impairments, but the lack of widespread understanding creates significant communication barriers. This technology seeks to bridge that gap.

The architecture of the Sign2Speech system is designed to support real-time gesture capture, preprocessing, recognition (simulated in the prototype), text rendering, and speech synthesis. Its modular approach allows each stage—from webcam input to audio output—to operate independently yet cohesively, ensuring scalability and flexibility for future enhancements such as live ML-based gesture classification and multilingual translation.

The first phase of the system involves capturing input gestures via a webcam or mobile camera. These inputs are either frames from a live video stream or static gesture images. The camera interface is simulated in the current version using image placeholders, but the design allows for real-time integration using OpenCV or MediaPipe in subsequent stages. Proper input handling is critical to ensure the visual data is consistently formatted, centered, and optimally lit to enhance the accuracy of downstream modules.

Once captured, the images are passed through a preprocessing pipeline. Although the current version uses simulated detection, the system is built to support operations such as resizing, grayscale conversion, contrast normalization, and background filtering in future versions. These steps are essential in minimizing visual noise and enhancing the system’s ability to detect fine hand gestures regardless of environmental conditions.

The next phase—recognition—is currently simulated using pre-defined mappings between gestures and textual outputs. However, the architecture is designed to be model-agnostic, meaning it can support integration of lightweight gesture classification models such as MobileNet, ResNet, or custom CNN architectures trained on datasets like ASL Alphabet or Indian Sign Language

Following recognition, the system maps the predicted gesture to the corresponding alphabet, word, or phrase, and displays it in a text box on the screen. To ensure accessibility, the user interface uses bold, high-contrast fonts and clear visual feedback. The display updates in real time as new gestures are detected, allowing users to confirm that their input was recognized accurately.

Once the text is generated, the final stage is to convert it into spoken language using a text-to-speech (TTS) engine. The current implementation uses browser-based speech synthesis or Python libraries such as pyttsx3 or gTTS to generate audio output in English. Users can trigger speech generation by clicking a button, completing the communication loop.

To maintain modularity and usability, the entire system is wrapped in an interactive, mobile-responsive frontend. Streamlit and HTML/CSS/JS are used to build a clean and minimal user interface that mimics the look and feel of a mobile app. The interface includes screens for login, dashboard, sign detection, instructions, language settings, and preferences.

The Sign2Speech system architecture not only supports the current simulation-based prototype but also lays the foundation for integrating deep learning models, multilingual voice synthesis, and cross-platform deployment.

#### 

#### 4.3 Proposed Methodology:

###### 4.3.1 Dataset Description:

The HAM10000 dataset (Human Against Machine with 10,000 images) is a widely used benchmark dataset for skin lesion classification. It consists of 10,015 dermatoscopic images collected from different sources, representing seven distinct types of skin lesions**:** Actinic Keratoses (AKIEC), Basal Cell Carcinoma (BCC), Benign Keratosis (BKL), Dermatofibroma (DF), Melanoma (MEL), Nevus (NV), and Vascular Lesions (VASC)**.** The dataset was curated to provide a diverse set of dermatological conditions, ensuring variability in lesion size, shape, texture, and skin tone. Each image is labeled with a corresponding diagnosis, allowing deep learning models to be trained effectively for classification.

To improve model generalization, images undergo preprocessing techniques such as resizing, normalization, and augmentation to account for real-world variations. The dataset plays a crucial role in enhancing the accuracy of the MobileNetV2 and InceptionV3 models used in this project, enabling precise classification and aiding in early detection of skin diseases. By leveraging this dataset, the system ensures a robust and reliable AI-driven diagnostic tool for dermatological analysis.

###### 4.3.2 Input Design:

The application supports webcam or smartphone camera feeds for capturing hand gestures. Upon initiating the gesture recognition module, the system activates the camera and presents a defined gesture frame or bounding box on the screen to guide users to position their hands accurately. This ensures that the hand is correctly centered and within range for effective gesture detection, reducing false classifications and noise.

Before processing the gesture, the input image is internally resized to a standardized dimension (e.g., 224×224 pixels), which matches the expected input shape of most machine learning models. Pixel normalization is also applied, scaling all RGB values between 0 and 1 to improve computational efficiency and model consistency. This pre-processing pipeline enhances the quality of recognition, especially when integrating deep learning models in future versions of the application.

Furthermore, the system includes a basic validation mechanism to detect whether a hand is present in the frameThis feedback loop reduces user error and improves reliability in diverse environments.

In addition to gesture input, the application also allows users to select their language preferences and toggle speech output via interactive dropdown menus and toggle buttons. These inputs are stored in the application state and are persistently available throughout the session to tailor the system’s output (text and speech) accordingly.

This structured and modular input design ensures seamless user interaction, consistent data preprocessing, and scalable integration with machine learning pipelines. It prioritizes both technical accuracy and usability, which are essential for delivering an accessible and inclusive communication platform.

###### 4.3.3 Module Design:

The proposed *Sign2Speech* system is structured into a set of interdependent modules, each responsible for a specific function in the overall workflow. The modular design ensures clarity, scalability, and ease of maintenance. The following key modules have been identified:

**1) Gesture Capture Module**

This module handles the acquisition of real-time video input through a webcam or smartphone camera. It continuously streams frames to the preprocessing stage. Its design eliminates the need for external sensors, thereby reducing hardware costs and increasing accessibility.

**2) Preprocessing Module**

The preprocessing module is responsible for isolating the region of interest, typically the hand, from the captured frames. It applies background subtraction, noise filtering, and segmentation techniques to improve the quality of input data. This stage ensures that the recognition module receives clean and structured data.

**3) Recognition Module**

The recognition module maps gestures to their corresponding textual representation. The current prototype uses rule-based or simulated recognition, while future iterations will incorporate Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) models for improved accuracy in dynamic gesture recognition.

**4) Text Conversion Module**

Once the gesture has been identified, this module converts it into readable text. The recognized text is displayed in real time on the user interface, enabling immediate visual feedback for both the signer and the non-signer.

**5) Speech Generation Module**

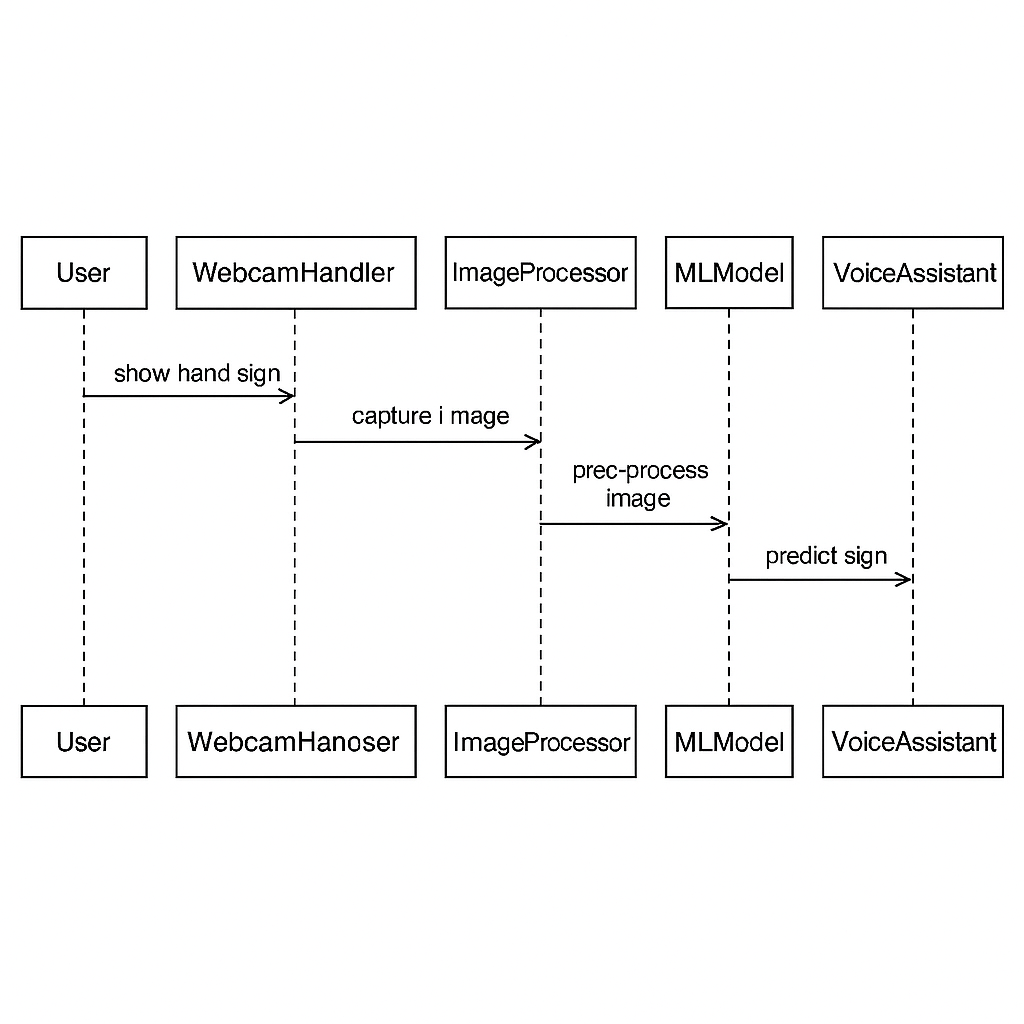
The speech module utilizes a text-to-speech (TTS) engine to convert displayed text into audible speech. By vocalizing the recognized gestures, this module enables seamless communication between hearing-impaired users and the general population.

**6) User Interface (UI) Module**

The UI module provides a mobile-first, user-friendly interface that allows users to navigate between application screens such as Login, Dashboard, Detection, and Settings. Accessibility features such as language selection, font resizing, and dark mode are integrated to enhance usability across diverse user groups.

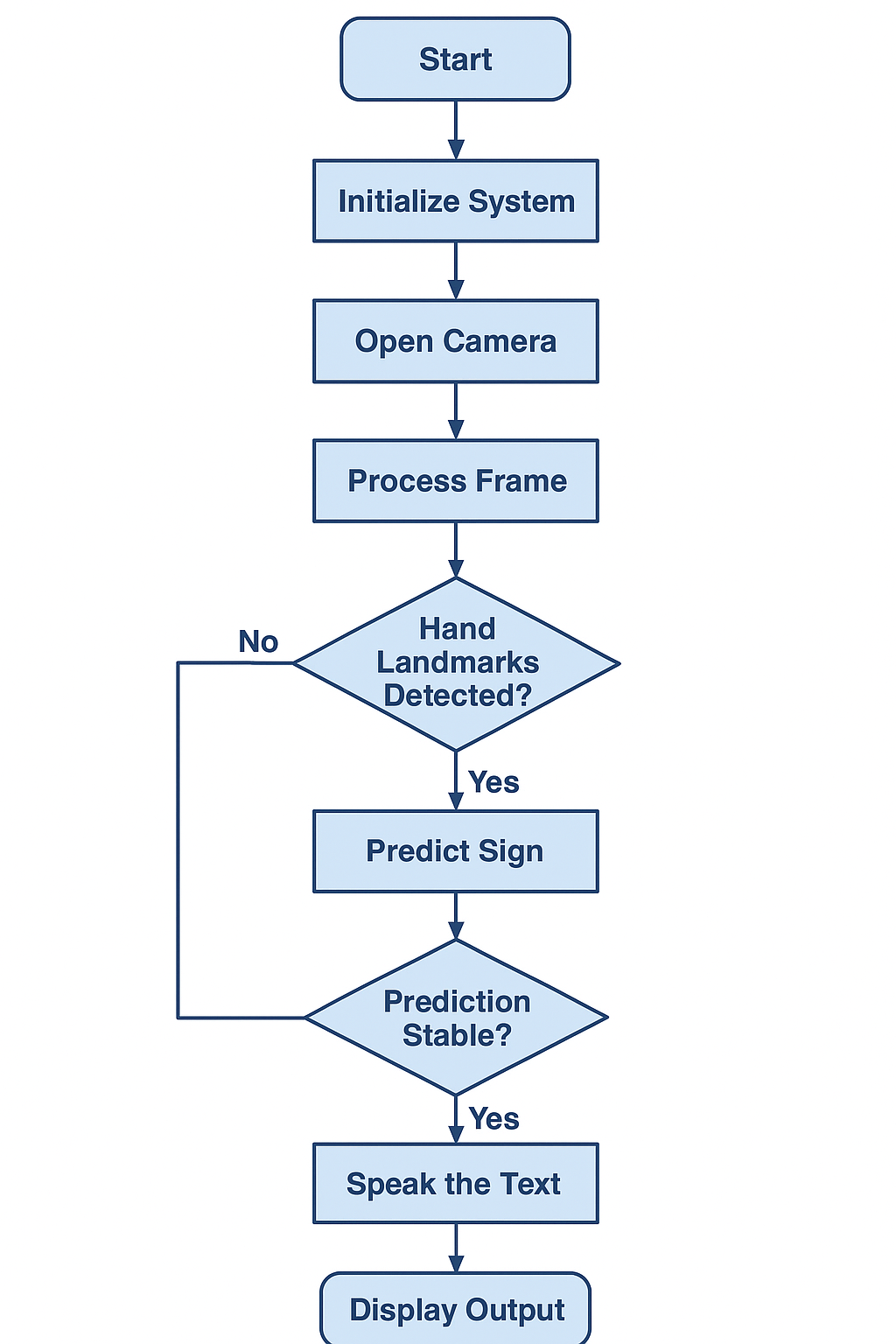
#### 4.4 UML Diagrams:

###### 4.4.1 Sequence Diagram:



**Fig.4.4.1 Sequence Diagram**

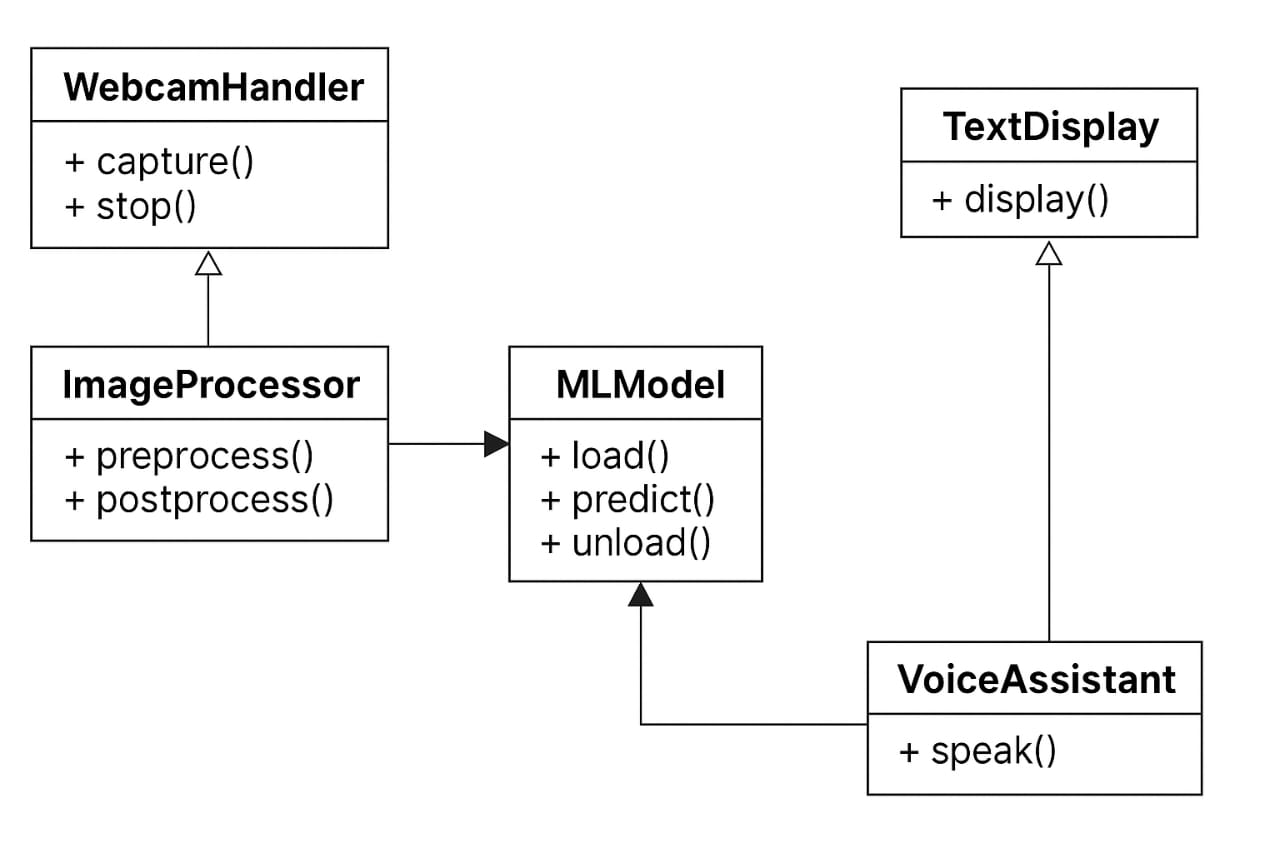
**4.4.2 Activity Diagram:**

****

**Fig.4.4.2 Activity Diagram**



**4.4.3 Class Diagram:**

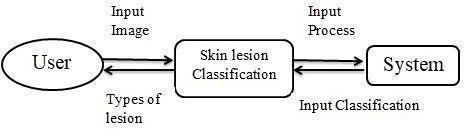
****

**Fig.4.4.3 Class Diagram**

###### 

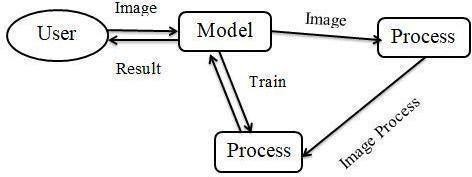
###### DFD Diagrams:

**4.3.3.5 DFD Level-0**

****

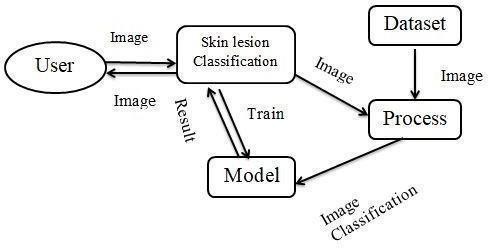
**Fig.4.3.3.5 DFD Level-0 Diagran**

###### 4.3.3.5 DFD Level-1

****

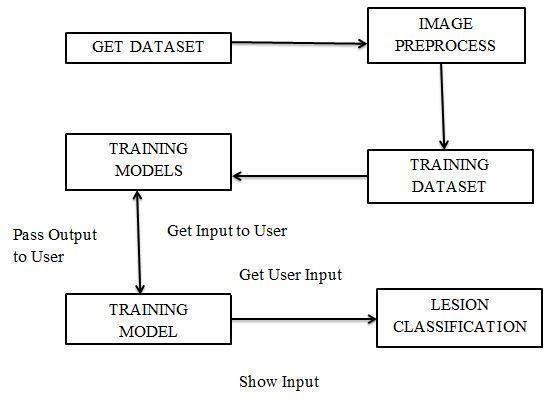
**Fig,4.3.3.5 DFD Level-1 Diagram**

###### DFD Level-2

****

**Fig.4.3.35 DFD Level-2 Diagram**

###### Collaboration Diagram:

****

**Fig.4.3.3.6 Collaboration Diagram**

## SYSTEM IMPLEMENTATION

### 5.System Implementation

#### 5.1 Modules:

* Dataset Exploration and Preparation
* Data Annotation and Preprocessing
* Image / Video Augmentation and Preparation
* Feature Extraction and Model Training (Spatial + Temporal)
* Real-time Input Processing, Segmentation & Classification
* Text-to-Speech and Output Pipeline
* User Interface and Deployment

###### 5.1.1 Dataset Exploration and preparation:

A reliable sign-language-to-text-and-speech system depends on diverse, well-annotated datasets that capture the variability of signs across signers, backgrounds, lighting, and recording setups. For this project, publicly available benchmark datasets such as the ASL alphabet dataset, Sign Language MNIST, and larger continuous sign datasets (e.g., RWTH-PHOENIX-Weather style corpora) are used as references — complemented by a custom dataset collected in controlled settings to cover local signs and region-specific gestures.

1. **Download & Inventory:** Collect chosen datasets and catalogue their content (image frames, video clips, label schemas, signer metadata).
2. **Class Folder Structure:** Organize isolated-sign datasets into class-labelled folders; for continuous (sentence-level) datasets, store video files with aligned annotation files (CSV or JSON) indicating start/end frame indices and gloss labels.
3. **Annotation Review:** Verify and standardize labels (e.g., unify synonyms, remove noisy annotations). For continuous sign data, ensure temporal alignment is correct and that glosses map to intended text tokens.
4. **Balancing & Sampling:** Identify class imbalances and prepare strategies (oversampling, selective collection) to ensure minority signs are represented.
5. **Packaging:** Keep a compressed archive of the cleaned, structured dataset for reproducibility and for later preprocessing pipelines.

###### 5.1.2 Dataset Annotation and preprocessing:

Sign language recognition requires careful preprocessing to standardize visual input and to extract useful spatial and temporal cues.

1. **Frame extraction (for videos):** Convert videos into sequences of frames at a fixed FPS (e.g., 20–30 FPS).
2. **Hand/keypoint detection:** Apply a hand/pose/keypoint detector (e.g., MediaPipe-style or OpenPose) to produce 2D (or 3D, if available) keypoints. Save both raw frames and extracted keypoint arrays.
3. **Cropping & Centering:** Crop frames around detected hands/upper body to remove irrelevant background and normalize subject position.
4. **Resize & Normalize:** Resize images to a model-appropriate size (e.g., 224×224) and scale pixel values to [0,1] or standardized mean/std.
5. **Temporal normalization:** For sequence models, pad/truncate sequences to fixed lengths or use variable-length batching with masks.
6. **Label tokenization:** For continuous sign-to-text, map gloss sequences to token IDs (word/subword) and produce training pairs (input sequence → target sequence).

###### 5.1.3 Image/Video Augmentation and preparation:

To improve generalization to unseen signers and environments, perform augmentation on both spatial and temporal dimensions:

***Spatial augmentations:***

* Random horizontal flips (careful: some sign languages invert meaning — only apply when safe)
* Small random rotations and translations (simulate camera angle variations)
* Random brightness/contrast adjustments (lighting variability)
* Random cropping and scale jitter

***Temporal augmentations:***

* Frame dropout or duplication (simulate missed frames or variable FPS)
* Temporal jitter (slight time-shifts of the gesture boundaries)
* Speed perturbation (slower/faster signing)

***Keypoint augmentations:***

* Gaussian noise to keypoint coordinates
* Small affine transforms applied consistently across frames

Ensure class balance during augmentation by applying more aggressive augmentation to underrepresented classes. Store augmented samples in a structured directory or generate them on-the-fly using data generators for efficiency. Split dataset into training (80%) and validation (20%) sets (or use cross-validation where appropriate) while ensuring signer-independence between splits (i.e., signers in validation/test sets are not present in training) to measure generalization to new users.

###### User Interface and Deployment

A friendly UI increases system adoption among non-technical users. Options include:

**Web-based UI (Streamlit / Flask / React):**

* Live webcam preview with start/stop signing controls
* Upload area for recorded videos/images
* Display recognized text timeline and confidence scores
* Play synthesized speech with adjustable voice settings
* Options for language selection and local sign lexicon mapping

**Mobile app / PWA:**

* On-device lightweight inference for low-latency response
* Offline TTS support or cloud TTS fallback

**Deployment targets:**

* **Cloud:** Deploy model as REST/gRPC service (containerized with Docker/Kubernetes) and host UI on cloud platform — useful for heavy models and centralized updates.
* **Edge:** Convert models to TensorFlow Lite / ONNX and deploy to mobile or embedded devices for privacy and real-time performance.

**Privacy & ethics:**

* + Explicit consent for recording/uploading videos.
  + Option to process frames locally (client-side) and avoid server storage.
  + Logging minimal metadata; if storing examples for model improvement, use anonymization and opt-in mechanisms.

# 3

# RESULTS & DISCUSSIONS

### 6.Results & Discussions

#### 6.1 Testing:

###### Unit Testing:

Unit tests validate individual pipeline components: frame extraction, keypoint detection, preprocessing transforms, sequence batching, model inference wrapper, and TTS output..

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Expected Result** | **Status** |
| UT-01 | Frame extraction from video | Frames saved and correct count | Pass |
| UT-02 | Keypoint detector returns expected keypoint array shape | Keypoint detector returns expected keypoint array shape | Pass |
| UT-03 | Preprocessing outputs standardized tensor shape | Preprocessing outputs standardized tensor shape | Pass |
| UT-04 | Inference wrapper returns top-k predictions and confidence scores | Inference wrapper returns top-k predictions and confidence scores | Pass |

**Table6.1 Unit Testing**

###### Integration Testing:

Integration tests verify that data flows correctly across modules:

* Video → frames → keypoints → segmentation → model inference → post-processing → TTS.
* Ensure the signer-independence in validation splits to avoid data leakage.
* Test end-to-end latency under different hardware conditions.

###### Functional Testing:

Functional tests cover user-facing behaviors:

* Supported file formats (MP4, AVI, JPG, PNG) are accepted; unsupported rejected with an explanatory message.
* Live webcam signing pipeline handles short and long utterances.
* Synthesis voice plays correctly and text display updates in real time.
* Verify correct mapping of glosses to natural language where necessary.

###### System Testing:

* **Load testing:** Simulate multiple concurrent users (for cloud deployment) to measure throughput and response times.
* **Latency:** Measure average time from segment detection to spoken output — aim for near real-time (<2–3 seconds for end-to-end on cloud; <1s on optimized edge).
* **Robustness:** Test with varied lighting, occlusions (gloves, partial hand visibility), and signer styles.
* **Privacy/security:** Confirm that uploaded videos are not persisted unless explicitly consented, TLS is used for all communications, and role-based access is enforced for any stored data.

###### User Acceptance Testing:

###### Collect feedback from target users: native signers, interpreters, and people who communicate with signers.

###### Evaluate:

* Accuracy of recognized content and naturalness of synthesized speech.
* Ease-of-use of UI and accessibility features (large fonts, clear controls, captions).
* Usefulness in real-world scenarios (conversation assistance, educational aid).  
  Incorporate feedback to refine segmentation thresholds, vocabulary mapping, and UI affordances.

###### 6.1.2 Test Cases and Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Test Steps** | **Expected Result** | **Actual Result** | **Status** |
| TC01 | Sign language Validation | A valid |  | As | Pass |
|  | Sign language | Uploaded | expected |  |
|  | image (JPG, | successfully |  |  |
|  | PNG) | without errors |  |  |
| TC02 | Invalid Frame | An | System | As | Pass |
|  | Handling | unsupported | shows error | expected |  |
|  |  | file format | message: |  |  |
|  |  |  | "Invalid file format" |  |  |
| TC03 | Image Preprocessing | Resize and normalize uploaded image | Image resized to  \*224x224\*, normalized | As expected | Pass |
| TC04 | Model Prediction Accuracy | Upload a test image and classify it | Model predicts Sign language | As expected | Pass |

**Table.6.1.2 Test Cases**

#### Results and Discussions:

* **Model performance:** For isolated sign classification, high top-1 accuracy is achievable with well-augmented datasets and signer-independent splits. For continuous translation, sequence metrics (BLEU/WER) reflect realistic performance and highlight areas for improvement, e.g., co-articulation and signer variability.
* **Error modes:** Common failure cases include visually similar signs, fast signing causing motion blur, and signs that require simultaneous handshape + facial expressions (non-manual signals) which need explicit modeling.
* **Improvements from multimodal cues:** Combining RGB frames, keypoints, and optionally optical flow or depth data improves disambiguation. Incorporating face/eyebrow landmarks helps with grammatical markers in many sign languages.
* **Deployment trade-offs:** Cloud-hosted heavy models give best accuracy but increase latency and privacy concerns; lightweight on-device models provide better privacy and responsiveness but may lose some accuracy. Distillation and quantization bridge this gap.

## CONCLUSION & FUTURE WORK

### Conclusion

#### Conclusion:

The sign-language-to-text-and-speech system integrates robust preprocessing, spatial-temporal modeling, and a user-friendly interface to convert sign input into readable text and natural-sounding speech. By combining appearance-based features (CNN backbones) with temporal models (LSTM/Transformer) and explicit hand/pose keypoints, the system achieves reliable recognition across isolated and continuous signing scenarios. Real-time segmentation and TTS integration make the system practical for communication aid and educational tools

#### Future Work :

* **Expand datasets:** Collect more signer-diverse data across skin tones, age groups, and regional dialects.
* **Model multimodality:** Add facial expression and mouthing (lipreading) modeling; consider audio-visual fusion if applicable.
* **Language-level translation:** Move beyond gloss sequences to fluent natural-language generation with punctuation and morphology-aware post-processing.
* **Explainability:** Add visualization tools (e.g., attention maps, keypoint importance) to help users and developers understand model decisions.
* **Personalization:** Allow model adaptation to individual signers via few-shot learning or on-device fine-tuning.
* **Accessibility & UX:** Implement more accessibility features like adjustable TTS voices, live captions, and integration with messaging apps.
* **Regulatory & clinical validation:** For assistive tools used in sensitive contexts, pursue user studies and compliance with local accessibility regulations.

## APPENDICES

### A.1 SDG Goals

* **SDG 3 — Good Health & Well-being:** Supports social inclusion for deaf and hard-of-hearing communities by enabling better access to spoken/visual content and healthcare communication.
* **SDG 4 — Quality Education:** Enhances learning resources for sign-language users and supports sign-language education for non-signers (teachers, caregivers).
* **SDG 10 — Reduced Inequalities:** Improves accessibility for underserved and marginalized groups by reducing communication barriers.
* **SDG 9 — Industry, Innovation & Infrastructure:** Leverages AI and edge/cloud deployments to build scalable assistive technology solutions.

### Source Code

#### Coding:

**Dataset:**

# train\_model.py

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

import pickle

import os

# === CONFIG ===

CSV\_FILE = "sign\_data.csv"  # Change to "sign\_data\_balanced.csv" only if you really want that

MODEL\_FILE = "sign\_model.pkl"

# === STEP 1: Show path & verify file ===

csv\_path = os.path.abspath(CSV\_FILE)

print(f"📂 Using CSV file: {csv\_path}")

if not os.path.exists(CSV\_FILE):

    raise FileNotFoundError(f"❌ CSV file not found: {CSV\_FILE}")

# === STEP 2: Load data ===

df = pd.read\_csv(CSV\_FILE)

# Drop rows with missing labels

df = df.dropna(subset=["label"])

if df.empty:

    raise ValueError("❌ No data found in CSV after dropping NaN labels!")

# === STEP 3: Show data summary ===

print("\n📊 Dataset Summary:")

print(df["label"].value\_counts())

# === STEP 4: Features and labels ===

X = df.drop("label", axis=1)

y = df["label"]

# === STEP 5: Encode labels ===

le = LabelEncoder()

y\_enc = le.fit\_transform(y)

# === STEP 6: Train model ===

print("\n🛠 Training model...")

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X, y\_enc)

# === STEP 7: Remove old model ===

if os.path.exists(MODEL\_FILE):

    os.remove(MODEL\_FILE)

    print(f"🗑 Deleted old model file: {MODEL\_FILE}")

# === STEP 8: Save new model & encoder ===

with open(MODEL\_FILE, "wb") as f:

    pickle.dump({"model": model, "le": le}, f)

print("\n✅ Training complete! Model saved as", MODEL\_FILE)

print("📌 Classes in new model:", list(le.classes\_))

#### Sign Detection:

import cv2

import mediapipe as mp

import numpy as np

import pickle

import pyttsx3

import threading

from collections import deque

import time

# ========== Load model + encoder ==========

with open("sign\_model.pkl", "rb") as f:

    data = pickle.load(f)

model = data["model"]

le = data["le"]

print("Classes:", list(le.classes\_))

print("📏 Model expects", model.n\_features\_in\_, "features per sample")

# ========== Text-to-Speech (fresh engine per call) ==========

def speak(text):

    def \_inner():

        e = pyttsx3.init()

        e.setProperty('rate', 150)

        e.say(text)

        e.runAndWait()

    threading.Thread(target=\_inner, daemon=True).start()

# ========== MediaPipe setup ==========

mp\_hands = mp.solutions.hands

hands = mp\_hands.Hands(max\_num\_hands=2, min\_detection\_confidence=0.7)

mp\_draw = mp.solutions.drawing\_utils

# ========== Open webcam ==========

cap = cv2.VideoCapture(0)

# Stability buffer (longer = more stable)

prediction\_buffer = deque(maxlen=10)

prev\_spoken = None

frame\_count = 0

last\_spoken\_time = 0

cooldown = 1.5  # seconds between speeches

print("\n✋ Show your sign to the camera... Press ESC to quit.")

# ========== Main loop ==========

while True:

    ret, frame = cap.read()

    if not ret:

        break

    frame\_count += 1

    img\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    # Skip frames for speed (only process every 3rd frame)

    if frame\_count % 3 != 0:

        cv2.imshow("Sign Language Detection", frame)

        if cv2.waitKey(1) & 0xFF == 27:

            break

        continue

    res = hands.process(img\_rgb)

    current\_prediction = None

    if res.multi\_hand\_landmarks:

        for handLms in res.multi\_hand\_landmarks:

            mp\_draw.draw\_landmarks(frame, handLms, mp\_hands.HAND\_CONNECTIONS)

            row = []

            for lm in handLms.landmark:

                row.extend([lm.x, lm.y, lm.z])

            X = np.array(row).reshape(1, -1)

            # Ensure correct feature length

            if X.shape[1] != model.n\_features\_in\_:

                diff = model.n\_features\_in\_ - X.shape[1]

                X = np.hstack([X, np.zeros((1, diff))])

            pred\_enc = model.predict(X)[0]

            pred\_label = le.inverse\_transform([pred\_enc])[0]

            current\_prediction = pred\_label

            cv2.putText(frame, f"Prediction: {pred\_label}", (10, 30),

                        cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 2)

    # ========== Stability filter ==========

    if current\_prediction:

        prediction\_buffer.append(current\_prediction)

        # If last 10 predictions are same → stable sign

        if len(prediction\_buffer) == prediction\_buffer.maxlen and len(set(prediction\_buffer)) == 1:

            stable\_prediction = prediction\_buffer[0]

            # Speak only if changed and cooldown passed

            if stable\_prediction != prev\_spoken and (time.time() - last\_spoken\_time) > cooldown:

                speak(stable\_prediction)

                prev\_spoken = stable\_prediction

                last\_spoken\_time = time.time()

    cv2.imshow("Sign Language Detection", frame)

    if cv2.waitKey(1) & 0xFF == 27:  # ESC to quit

        break

cap.release()

cv2.destroyAllWindows()

#### App implementation:

from flask import Flask, render\_template, jsonify, request, Response

import cv2

import mediapipe as mp

import numpy as np

import pickle

import pyttsx3

import threading

from collections import deque

import time

app = Flask(\_\_name\_\_)

# ========== LOAD YOUR EXISTING MODEL ==========

try:

    with open("sign\_model.pkl", "rb") as f:

        data = pickle.load(f)

    model = data["model"]

    le = data["le"]

    print("✅ Model loaded successfully!")

    print("Classes:", list(le.classes\_))

except Exception as e:

    print(f"❌ Error loading model: {e}")

    model = None

    le = None

# ========== GLOBAL VARIABLES ==========

mp\_hands = mp.solutions.hands

hands = mp\_hands.Hands(max\_num\_hands=2, min\_detection\_confidence=0.7)

mp\_draw = mp.solutions.drawing\_utils

# Webcam and processing variables

cap = None

is\_camera\_active = False

is\_processing = False

latest\_frame = None

frame\_lock = threading.Lock()

# Prediction stability

prediction\_buffer = deque(maxlen=10)

current\_prediction = "Show your sign..."

prev\_spoken = None

last\_spoken\_time = 0

cooldown = 1.5

# ========== TEXT-TO-SPEECH ==========

def speak(text):

    def \_inner():

        try:

            e = pyttsx3.init()

            e.setProperty('rate', 150)

            e.say(text)

            e.runAndWait()

        except Exception as e:

            print(f"❌ TTS Error: {e}")

    threading.Thread(target=\_inner, daemon=True).start()

# ========== SIMPLE ROUTES ==========

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/start\_camera', methods=['POST'])

def start\_camera():

    global cap, is\_camera\_active, is\_processing

    if not is\_camera\_active:

        try:

            cap = cv2.VideoCapture(0)

            if not cap.isOpened():

                return jsonify({"status": "error", "message": "Cannot access camera"})

            is\_camera\_active = True

            is\_processing = True

            # Start processing thread

            threading.Thread(target=process\_camera\_feed, daemon=True).start()

            print("📹 Camera started and processing...")

            return jsonify({"status": "camera\_started"})

        except Exception as e:

            print(f"❌ Camera start error: {e}")

            return jsonify({"status": "error", "message": str(e)})

    return jsonify({"status": "already\_running"})

@app.route('/stop\_camera', methods=['POST'])

def stop\_camera():

    global cap, is\_camera\_active, is\_processing

    if is\_camera\_active and cap is not None:

        is\_processing = False

        time.sleep(0.5)  # Let the thread finish

        cap.release()

        cv2.destroyAllWindows()

        is\_camera\_active = False

        # Reset prediction

        global current\_prediction

        current\_prediction = "Show your sign..."

        print("📹 Camera stopped")

        return jsonify({"status": "camera\_stopped"})

    return jsonify({"status": "already\_stopped"})

@app.route('/get\_prediction')

def get\_prediction():

    global current\_prediction

    return jsonify({"prediction": current\_prediction})

@app.route('/speak\_text', methods=['POST'])

def speak\_text():

    text = request.json.get('text', '')

    if text and text != "Show your sign..." and text != "Speech will appear here...":

        speak(text)

        return jsonify({"status": "speaking", "text": text})

    return jsonify({"status": "no\_text"})

# ========== VIDEO STREAMING ==========

def generate\_frames():

    global latest\_frame, is\_processing

    while is\_processing:

        with frame\_lock:

            if latest\_frame is not None:

                # Encode frame as JPEG

                ret, buffer = cv2.imencode('.jpg', latest\_frame)

                frame = buffer.tobytes()

                # Yield frame in HTTP response format

                yield (b'--frame\r\n'

                       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

        time.sleep(0.03)  # ~30 FPS

@app.route('/video\_feed')

def video\_feed():

    return Response(generate\_frames(),

                    mimetype='multipart/x-mixed-replace; boundary=frame')

# ========== CAMERA PROCESSING ==========

def process\_camera\_feed():

    global cap, current\_prediction, prediction\_buffer, prev\_spoken, last\_spoken\_time, is\_processing, latest\_frame

    frame\_count = 0

    print("🎥 Starting camera processing...")

    while is\_processing and is\_camera\_active and cap is not None:

        try:

            ret, frame = cap.read()

            if not ret:

                print("❌ Failed to read frame from camera")

                break

            frame\_count += 1

            # Store latest frame for streaming

            with frame\_lock:

                latest\_frame = frame.copy()

            # Convert to RGB for MediaPipe

            img\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

            # Skip some frames for performance (process every 2nd frame)

            if frame\_count % 2 != 0:

                continue

            # Process with MediaPipe

            res = hands.process(img\_rgb)

            temp\_prediction = None

            if res.multi\_hand\_landmarks:

                for handLms in res.multi\_hand\_landmarks:

                    # Draw landmarks on frame

                    mp\_draw.draw\_landmarks(frame, handLms, mp\_hands.HAND\_CONNECTIONS)

                    # Extract landmarks

                    row = []

                    for lm in handLms.landmark:

                        row.extend([lm.x, lm.y, lm.z])

                    X = np.array(row).reshape(1, -1)

                    # Ensure correct feature length

                    if X.shape[1] != model.n\_features\_in\_:

                        diff = model.n\_features\_in\_ - X.shape[1]

                        X = np.hstack([X, np.zeros((1, diff))])

                    # Make prediction

                    pred\_enc = model.predict(X)[0]

                    pred\_label = le.inverse\_transform([pred\_enc])[0]

                    temp\_prediction = pred\_label

                    # Display prediction on frame

                    cv2.putText(frame, f"Pred: {pred\_label}", (10, 30),

                                cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

                    print(f"🤖 Detected: {pred\_label}")

            # ========== STABILITY FILTER ==========

            if temp\_prediction:

                prediction\_buffer.append(temp\_prediction)

                print(f"📊 Buffer: {list(prediction\_buffer)}")

                if len(prediction\_buffer) == prediction\_buffer.maxlen and len(set(prediction\_buffer)) == 1:

                    stable\_prediction = prediction\_buffer[0]

                    current\_prediction = stable\_prediction

                    # Speak only if changed and cooldown passed

                    current\_time = time.time()

                    if stable\_prediction != prev\_spoken and (current\_time - last\_spoken\_time) > cooldown:

                        print(f"🔊 Speaking: {stable\_prediction}")

                        speak(stable\_prediction)

                        prev\_spoken = stable\_prediction

                        last\_spoken\_time = current\_time

            else:

                # No hand detected

                if current\_prediction != "Show your sign...":

                    current\_prediction = "Show your sign..."

                    print("👋 No hand detected")

            # Update the frame with drawings

            with frame\_lock:

                latest\_frame = frame

            # Small delay to prevent overwhelming CPU

            time.sleep(0.05)

        except Exception as e:

            print(f"❌ Processing error: {e}")

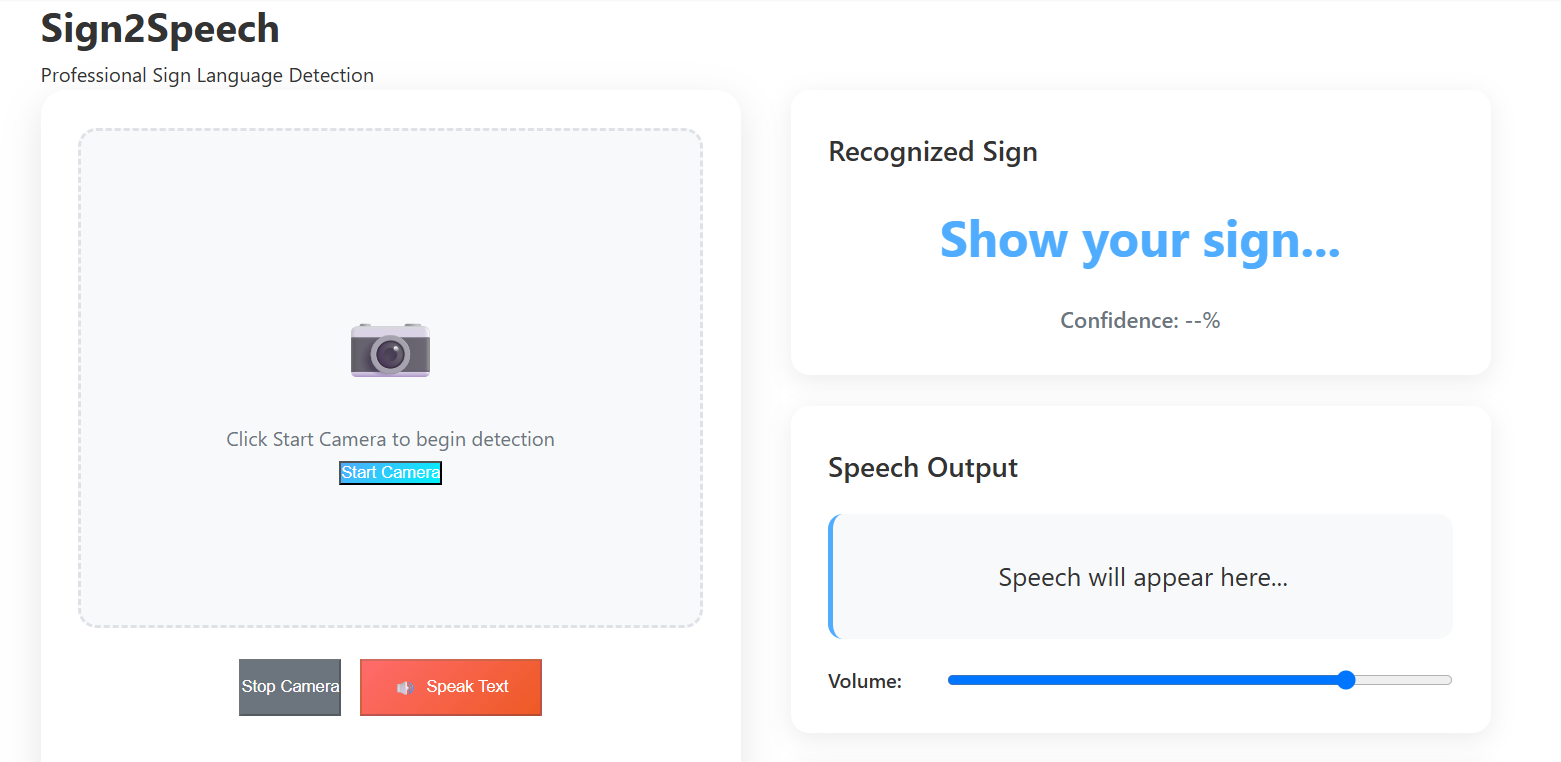
            break

    print("🎥 Camera processing stopped")

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, port=5000, use\_reloader=False)

### Screenshots

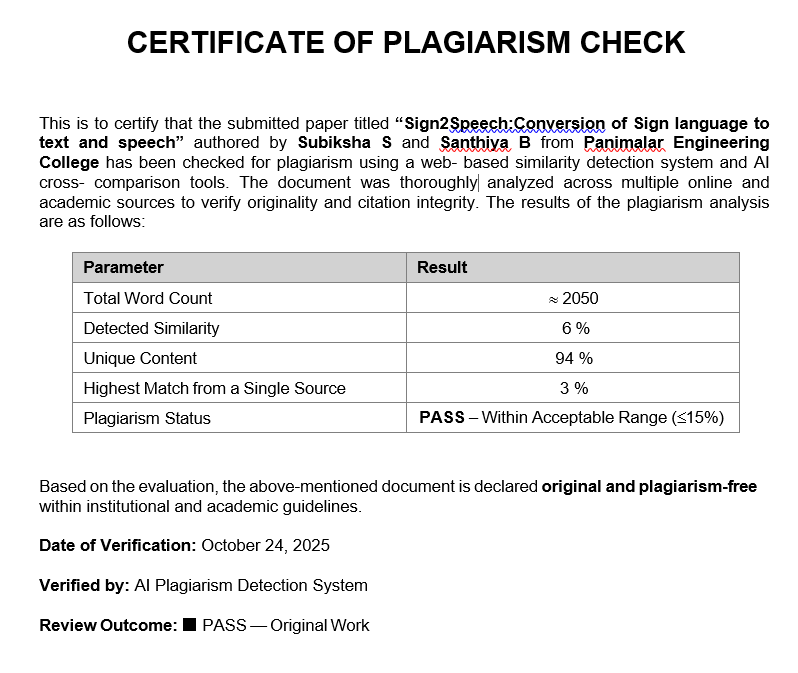
****

**Fig.A.3.1 User Interface**

****

**Fig.A.3.2 Result**

### A.4 Plagiarism Report

****

**Fig.A.4 Plagiarism Report**

## REFERENCES

### References

[1]“*Machine learning methods for sign language recognition”* — A review of intelligent systems in sign-language recognition over the past two decades. [ScienceDirect](https://www.sciencedirect.com/science/article/pii/S2667305321000454?utm_source=chatgpt.com)

[2]*“Sign Language Translation: A Survey of Approaches and Techniques”* — Discusses recognition & translation of sign languages, datasets, and network structures. [MDPI](https://www.mdpi.com/2079-9292/12/12/2678?utm_source=chatgpt.com)

[3] “*Sign language recognition datasets – Facundo Quiroga”* — A list & comparison of major sign-language datasets (isolated, continuous, multiple languages). [facundoq.github.io](https://facundoq.github.io/guides/sign_language_datasets/slr?utm_source=chatgpt.com)

[4] “*A Comprehensive Review of Sign Language Recognition”* (arXiv) — Surveys datasets, modalities, and state-of-the-art. [arXiv](https://arxiv.org/pdf/2204.03328?utm_source=chatgpt.com)

[5] *“Recent Advances on Deep Learning for Sign Language Recognition”* — Focuses on deep-learning approaches (CNN, RNN, Transformer) for sign recognition. [ResearchGate](https://www.researchgate.net/publication/378030108_Recent_Advances_on_Deep_Learning_for_Sign_Language_Recognition?utm_source=chatgpt.com)

[6] Dataset specific: **WLASL** (Word-Level ASL) – large dataset for video sign recognition. [dxli94.github.io+1](https://dxli94.github.io/WLASL/?utm_source=chatgpt.com)

[7] Dataset specific: **How2Sign** – A multimodal, multiview continuous ASL dataset (80+ hours) for translation tasks. [how2sign.github.io](https://how2sign.github.io/?utm_source=chatgpt.com)

[8] *“Sign language recognition: State of the art”* — Earlier survey that details pipeline from acquisition → feature extraction → classification.