# AI POWERED ACCESSIBILITY FOR ENABLING EFFECTIVE COMMUNICATION FOR HEARING AND SPEECH IMPAIRED IN VIRTUAL PLATFORMS

**A PROJECT REPORT**

***Submitted by***

| | |
|---|---|
| **M.ARTHI** | **Reg. No: 814421104001** |
| **S.DHARSHANI** | **Reg. No: 814421104008** |
| **S.KAMATCHI** | **Reg. No: 814421104013** |

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

***of***

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**SUDHARSAN ENGINEERING COLLEGE**

**SATHIYAMANGALAM**

**ANNA UNIVERSITY: CHENNAI 600 025**

**MAY 2025**

## BONAFIDE CERTIFICATE

Certified that this project report **"AI POWERED ACCESSABILITY FOR ENABLING THE EFFECTIVE COMMUNICATION FOR HEARING AND SPEECH IMPAIRED IN VIRTUAL PLATFORMS "** is the bonafide work of **"M.ARTHI (814421104001), S.DHARSHANI (814421104008) AND S.KAMATCHI (814421104013) "** who carried out the project work under my supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| Dr. P. SUJATHA M.E.,Ph.D. | Mrs. P.SIVAPRIYANGA M.E., |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| Associate Professor, | Assistant Professor, |
| Computer Science and Engineering, | Computer Science and Engineering, |
| Sudharsan Engineering College, | Sudharsan Engineering College, |
| Sathiyamangalam, | Sathiyamangalam, |
| Pudukkottai-622 501. | Pudukkottai-622 501. |

Submitted for the project Viva-Voce held on……………………………..

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

# ACKNOWLEDGMENT

First and Foremost, I thank **GOD** and **My Parents** for their blessings on me to complete this project successfully.

At this moment of accomplishment, first of all I pay homage to **Dr. M. ARULKUMARAN M.E., Ph.D. Principal, Sudharsan Engineering College, Sathiyamangalam**, for giving me an opportunity to do this project.

I express my whole hearted thanks to **Dr. P. SUJATHA M.E., Ph.D. Associate Professor and Head of the Department CSE, Sudharsan Engineering College, Sathiyamangalam**, for guided me in an inspiring manner to complete my project successfully.

I express my thanks to my project supervisor **Mrs.P.SIVAPRIYANGA M.E., Assistant Professor of the Department CSE, Sudharsan Engineering College, Sathiyamangalam**, and other staff members of CSE Department for their kind cooperation.

Last but not the least, I thank all **My Family Members** and **My Friends** for helping me to finish this project work successfully.

# ABSTRACT

Effective communication is vital for sharing information, ideas, and emotions. However, virtual meeting platforms like Zoom, Microsoft Teams, and Google Meet often fail to accommodate individuals with hearing and speech impairments, creating significant barriers to inclusivity. While sign language offers a means of communication for deaf individuals, its interpretation remains challenging for non- signers. Existing sign language recognition technologies are limited in accuracy and accessibility, making them unsuitable for seamless integration into virtual platforms. This project introduces an AI-driven system designed to bridge the communication gap between deaf and hearing participants in virtual meetings. The system employs advancements in deep learning, particularly Temporal Convolutional Networks (TCNs), to enable two-way communication in real-time. It includes three core modules: a Sign Recognition Module (SRM) that interprets signs using TCN, a Speech Recognition and Synthesis Module (SRSM) powered by Hidden Markov Models, which converts spoken language into text, and an Avatar Module (AM) that visually translates speech into corresponding signs. The Avatar Module is essential for visually representing spoken language in sign language format, ensuring non- signers can effectively communicate with sign language users in an intuitive and engaging way. Trained on Indian Sign Language, the system facilitates communication across diverse groups, including deaf, mute, hard-of-hearing, visually impaired, and non-signers. Its integration into popular virtual meeting platforms through a user-friendly web-based interface enhances accessibility and participation. This solution represents a significant advancement in fostering inclusivity and accessibility in virtual meeting environments.

# TABLE OF CONTENTS

| CHAPTER NO. | TITLE | PAGE NO. |
|---|---|---|
| | **ABSTRACT** | **i** |
| | **LIST OF ABBREVIATIONS** | **v** |
| | **LIST OF FIGURES** | **vi** |
| **1.** | **INTRODUCTION** | **1** |
| | 1.1 OVERVIEW | **1** |
| | 1.2 PROBLEM STATEMENT AND THE SHORTCOMINGS | **2** |
| | 1.3 AIM AND OBJECTIVE | **3** |
| | 1.4 SCOPE OF THE PROJECT AND ITS IMPLEMENTATION | **4** |
| **2.** | **LITERATURE SURVEY** | **5** |
| **3.** | **SYSTEM ANALYSIS** | **11** |
| | 3.1 EXISTING SYSTEM | **11** |
| | 3.2 PROPOSED SYSTEM | **13** |
| **4.** | **SYSTEM REQUIREMENTS** | **15** |
| | 4.1 HARDWARE REQUIREMENT | **15** |
| | 4.2 SOFTWARE REQUIREMENT | **15** |
| | 4.3 SOFTWARE DESCRIPTION | **16** |

| | | |
|---|---|---|
| **5.** | **SYSTEM DESIGN** | **20** |
| | 5.1 CONCEPT DIAGRAM | **20** |
| | 5.2 ARCHITECTURE DIAGRAM | **21** |
| | 5.3 DATAFLOW DIAGRAM | **22** |
| | 5.4 UML DIAGRAM | **25** |
| | 5.4.1 USECASE DIAGRAM | **25** |
| | 5.4.2 SEQUENCE DIAGRAM | **26** |
| | 5.4.3 ACTIVITY DIAGRAM | **27** |
| | 5.4.4 CLASS DIAGRAM | **28** |
| **6.** | **MODULE DESCRIPTION** | **30** |
| | 6.1 MODULES | **30** |
| | 6.2 MODULE DESCRIPTION | **30** |
| | 6.2.1 SIGN MEET WEB APP | **30** |
| | 6.2.2 END USER INTERFACE | **31** |
| | 6.2.3 SIGNET MODEL BUILD | **31** |
| | 6.2.4 SIGN LANGUAGE RECOGNITION | **31** |
| | 6.2.5 MULTILANGUAGE INTERPRETATION | **32** |
| | 6.2.6 SPEECH RECOGNITION | **32** |
| | 6.2.7 AVATAR GENERATION | **33** |
| | 6.2.8 VISUAL COMMUNICATION | **33** |

**7 SYSTEM TESTING 34**

7.1 SOFTWARE TESTING **34**

7.2 TYPES OF TESING **34**

7.3 TEST CASES **36**

7.4 TEST REPORT **37**

**8 CONCLUSION 39**

8.1 CONCLUSION **39**

8.2 FUTURE ENHANCEMENT **40**

**9 APPENDICES 41**

**9.1 SOURCE CODE 41**

**9.2 SCREENSHOTS 56**

**REFERENCES 64**

**9.**

## LIST OF ABBREVATION

| ABBREVIATION | EXPANSION |
|---|---|
| TCNS | Temporal Convolutional Networks |
| SRM | Sign Recognition Module |
| SRSM | Speech Recognition and Synthesis Module |
| AM | Avatar Module |
| SVM | Support Vector Machines |
| RPN | Region Proposal Network |
| HMM | Hidden Markov Models |

# LIST OF FIGURES

| FIGURE NO. | FIGURE NAME | PAGE NO. |
|---|---|---|
| 5.1 | CONCEPT DIAGRAM | **18** |
| 5.2 | ARCHITECTURE DIAGRAM | **19** |
| 5.3.1 | LEVEL 0 | **20** |
| 5.3.2 | LEVEL 1 | **21** |
| 5.3.3 | LEVEL 2 | **22** |
| 5.4.1 | USECASE DIAGRAM | **23** |
| 5.4.2 | SEQUENCE DIAGRAM | **24** |
| 5.4.3 | ACTIVITY DIAGRAM | **25** |
| 5.4.4 | CLASS DIAGRAM | **26** |
| 9.2 | SCREENSHOTS | **59** |

# CHAPTER 1

# INTRODUCTION

## 1.1. OVERVIEW

Sign language is manual communication commonly used by people who are deaf. Sign language is not universal; people who are deaf from different countries speak different sign languages. The gestures or symbols in sign language are organized in a linguistic way. Each individual gesture is called a sign. Each sign has three distinct parts: the handshape, the position of the hands, and the movement of the hands. American Sign Language (ASL) is the most commonly used sign language in the India.

Figure 1.1. Sign Language

A sign language (also signed language) is a language which uses manual communication, body language, and lip patterns instead of sound to convey meaning—simultaneously combining hand shapes, orientation and movement of the hands, arms or body, and facial expressions to fluidly express a speaker's thoughts. Signs often represent complete ideas, not only words. However, in addition to accepted gestures, mime, and hand signs, sign language often includes finger spelling, which involves the use of hand positions to represent the letters of the alphabet. Sign languages have developed in circumstances where groups of people with mutually unintelligible spoken languages found a common base and were able to develop signed forms of communication. Sign languages.

## 1.2. PROBLEM STATEMENT AND THE SHORTCOMINGS

In today's digital era, virtual meeting platforms like Zoom, Microsoft Teams, and Google Meet have become essential for communication and collaboration. However, these platforms often lack effective accessibility features for deaf and non-deaf users to interact seamlessly. Deaf individuals face significant challenges in understanding spoken content during virtual meetings, while non- deaf users struggle to communicate with their deaf counterparts due to the absence of real-time sign language interpretation. Traditional solutions such as sign language interpreters, captions, and assistive devices have limitations in terms of accuracy, availability, and real-time effectiveness. Relying on human interpreters is not always feasible.

## 1.3. DEEP LEARNING

Deep learning is a method in artificial intelligence (AI) that teaches computers to process data in a way that is inspired by the human brain. Deep learning models can recognize complex patterns in pictures, text, sounds, and other data to produce accurate insights and predictions.

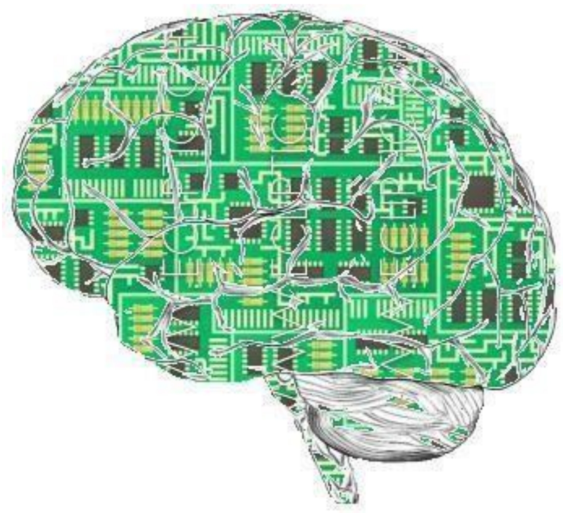


Figure 1.2: Deep Neuron

Deep learning algorithms are neural networks that are modeled after the human brain. For example, a human brain contains millions of interconnected neurons that work together to learn and process information. Similarly, deep learning neural networks, or artificial neural networks, are made of many layers of artificial neurons that work together inside the computer. Artificial neurons are software modules called nodes, which use mathematical calculations to process data. Artificial neural networks are deep learning algorithms that use these nodes to solve complex problems.

## 1.3. AIM AND OBJECTIVE

### Aim

The aim of the project is to develop an AI-powered system that facilitates seamless communication between hearing and speech-impaired individuals and hearing users by integrating sign language recognition, speech-to-text conversion, and visual representation in virtual platforms.

### Objectives

- To develop a Sign Recognition Module that translates sign language into text or speech.
- To create a Speech Recognition and Synthesis Module that converts spoken language into text.
- To implement an Avatar Module for visual sign language representation.
- To ensure real-time communication in virtual platforms like Zoom, Microsoft Teams, and Google Meet.
- To enhance accessibility and inclusivity for individuals with hearing and speech impairments.

## 1.4. SCOPE OF THE PROJECT AND ITS IMPLEMENTATION

This project focuses on enhancing communication accessibility by integrating AI-driven sign language recognition and speech-to-text conversion into popular virtual meeting platforms. It aims to support real-time, inclusive interactions for diverse users, including those with hearing and speech impairments.

- **Inclusive Communication**: Bridges the gap between hearing and speech-impaired individuals and hearing users through AI-driven sign language recognition and speech-to- text conversion.
- **Virtual Platform Integration**: Compatible with popular platforms like Zoom, Microsoft Teams, and Google Meet to ensure accessibility in virtual meetings.
- **Real-Time Translation**: Provides instant sign-to-text, speech-to-text, and text-to-sign conversion for smooth interaction.
- **Multi-User Support**: Beneficial for deaf, hard-of-hearing, mute individuals, and non- signers, promoting effective communication.
- **AI-Driven Accuracy**: Uses **Temporal Convolutional Networks (TCN)** and **Hidden Markov Models (HMM)** to improve recognition accuracy.
- **User-Friendly Interface**: A web-based interactive platform ensures ease of use and accessibility for diverse users.
- **Scalability**: Can be extended to support additional sign languages and integrate with more digital communication tools in the future.
- **Sensitivity to Background Conditions**: Accuracy may degrade under poor lighting.

# CHAPTER 2
# LITERATURE SURVEY

## 2.1 HAND GESTURE RECOGNITION FOR DEAF COMMUNICATION USING OPENCV

**Author**:Soma,Shrenika;,Myneni,Madhu,Bala

**Description**

Communication with deaf individuals remains a challenge for the general population, as most people are unfamiliar with sign language. This project addresses this gap by developing a system to translate hand gestures into readable text, thereby enabling seamless interaction. The system uses a camera to capture hand gestures and applies image processing techniques to interpret them. It begins with preprocessing of captured images using OpenCV-Python and various supporting libraries. The RGB color model is used with equal contribution from red, green, and blue channels to standardize image input.

The system employs Gaussian filtering to reduce image noise, followed by edge detection using first-order derivatives to analyze gradient intensities in horizontal and vertical directions. Template matching is achieved using the Sum of Absolute Difference (SAD) method, which calculates pixel-wise differences between input and dataset images to identify matching gestures. A dataset containing 70 samples for each of the 36 sign language symbols is utilized, ensuring a wide range of hand shapes and movements are covered. Each sample is uniquely associated with a corresponding alphabet or sign.

**Disadvantage**

• Limited to Static Gestures: The system currently supports only static hand gestures, limiting its usability for dynamic sign language.

## 2.2 REAL-TIME BANGLA SIGN LANGUAGE DETECTION WITH SENTENCE AND SPEECH GENERATION

**Author:**Dipon.Talukder.Fatima.Jahara

**Description**

Sign language serves as a vital communication bridge for individuals with hearing and speaking disabilities, such as the deaf and mute. However, its visual nature excludes people with visual impairments, highlighting the need for a system that can translate sign language into text and speech. This project proposes a real-time Bangla Sign Language (BdSL) detection system capable of recognizing hand gestures from video streams and converting them into complete sentences and speech outputs. The system utilizes the YOLOv4 object detection model for high-speed recognition and introduces three new signs to aid sentence generation: compound characters, space, and end of sentence.

The image processing pipeline begins with converting the captured video frames from RGB to HSV color space, followed by feature extraction using the Scale Invariant Feature Transform (SIFT) algorithm. Principal Component Analysis (PCA) is used to reduce dimensionality before inputting data into the model.

The dataset used includes 12,500 images across 49 classes, covering 39 Bangla alphabets, 10 Bangla digits, and 3 additional custom signs. Despite challenges such as noisy images, shadows, and complex backgrounds, the system achieved 82.06% mean average precision (mAP), demonstrating strong performance in real-time scenarios.

**Disadvantage**

• Limited Generalization to Other Languages: The system is specific to Bangla Sign Language and would require significant adaptation to support other sign languages.

## 2.3 SIGN LANGUAGES TO SPEECH CONVERSION PROTOTYPE USING THE SVM CLASSIFIER

**Author**: Malli Mahesh Chandra; Rajkumar S

**Description**

Effective communication for individuals with speech impairments remains a global challenge, affecting nearly 70 million people, including children with nonverbal autism. This project introduces a prototype that translates sign language gestures into speech using sensor-based gesture recognition. The goal is to bridge the communication gap between individuals with speech impairments and the general population. Real-time hand gestures are captured using sensor modules and transmitted via Bluetooth to a computer, where they are processed and classified using a Support Vector Machine (SVM) algorithm.

The system uses the MPU6050 sensor module, a compact device that integrates a 3-axis gyroscope, 3-axis accelerometer, and digital motion processor. Data from these sensors are collected using an Arduino Nano microcontroller and transmitted to a PC. Before classification, the incoming sensor data is preprocessed and structured to match a pre-trained dataset.

The dataset includes repeated gesture recordings at various positions to ensure robustness. A total of 22 ASL and 11 ISL gestures were trained and tested successfully. The system achieved 100% accuracy for the ISL dataset using a 75% training and 25% testing data split.

**Disadvantage**

• One-Hand Limitation: The prototype uses only a single glove, making it unsuitable for sign languages that require two-handed gestures.
• Limited Gesture Vocabulary: The system supports a limited set of gestures, which may not be sufficient for full conversational sign language.

## 2.4 WEARABLE SENSOR-BASED SIGN LANGUAGE RECOGNITION: A COMPREHENSIVE REVIEW

**Author:**Karly,Kudrinko,,Emile,Flavin,,Xiaodan,Zhu

**Description**

Sign language is a primary mode of communication for individuals who are Deaf, hard of hearing, or non-verbal. However, these individuals often face communication barriers when interacting with people who do not understand sign language. This paper presents a comprehensive literature review on wearable sensor-based sign language recognition (SLR) systems. It evaluates existing research by analyzing attributes such as sign language variation, sensor configuration, classification methods, study design, and performance metrics. The goal is to inform the development of user-centric and accurate wearable systems for effective sign language interpretation.

The methodology categorizes SLR techniques into computer vision-based models, wearable sensor-based systems, and hybrid systems. This review specifically focuses on wearable devices, filtering out non-English papers, non-peer-reviewed studies, and vision-based approaches using terms like "image", "camera", and "vision". One of the devices reviewed is the VPL Data Glove, which includes fiber optic sensors on each finger for joint flexion and a tracker on the palm for spatial orientation. Practical implementations of SLR require not only accuracy but also comfort and usability, especially for mobile applications.

**Disadvantage**

• Lack of Standardized Datasets: The wide variety of sensor setups and sign language dialects makes it difficult to create benchmark datasets for direct model..

## 2.5 MULTIMEDIA TECHNOLOGIES TO TEACH SIGN LANGUAGE IN A WRITTEN FORM

**Author**:Myasoedova,M.A;FarkhadovM.P

**Description**

This paper explores the use of multimedia technologies to teach Sign Language in a symbolic and written form. The authors focus on using notations to record sign elements as sequences of alphabetic, numeric, and graphical characters, enabling precise spatio-temporal representation of signs. The study supports the adoption of the SignWriting (SW) system for Russian Sign Language (RSL) and introduces a multimedia tool called SWiSL, an online platform aimed at enhancing recognition and literacy in written Sign Language.

The SW system assigns unique code chains to each sign element, allowing compact and precise representation. The research highlights communication breakdowns in spoken and signed communication due to poor articulation or unclear gestures and proposes the SW system as a solution to standardize and preserve sign meaning. The RSL corpus, covering subjects like family, people, and technology, is used as the dataset for this research and is presented in a written symbolic form using the SW system.

**Disadvantage**

• Limited User Adoption: The success of the system relies on its acceptance by native.

• Recognition Complexity: Automatically recognizing symbolic sign speech and accurately converting it into text remains a technical challenge, especially with unclear gestures or sign ambiguity.

## 2.6 RESEARCH ON DYNAMIC SIGN LANGUAGE ALGORITHM BASED ON SIGN LANGUAGE TRAJECTORY AND KEY FRAME EXTRACTION

**Author**: Yufei Yan; Chenyu Liu

**Description**

This paper addresses the challenge of building sign language recognition (SLR) systems using multilingual datasets, focusing on hand movement modelling—an area where existing cross-language resources are lacking. Unlike hand shapes (which can be shared across sign languages), hand movements are highly specific and difficult to model without dedicated data. The study proposes an approach where language-independent hand movement subunits are derived using light supervision, allowing SLR systems to benefit from shared datasets across different sign languages.

The researchers implement a Kullback-Leibler Hidden Markov Model (KL-HMM). In this model, feature observations are not single values but posterior distributions. Instead of using log-likelihood, KL divergence is calculated between the observed posterior distribution and a predefined state distribution. This method allows for better generalization across languages. The approach is validated on Swiss German Sign Language (DSGS), German Sign Language (DGS), and Turkish Sign Language (TSL).

**Disadvantage**

• Performance Gap: Language-independent hand movement modelling still shows a performance gap compared to language-specific models.

• Complex Implementation: Requires extensive posterior probability estimation and multilingual database access, which may not be feasible for all research setups.

# CHAPTER 3
# SYSTEM ANALYSIS

## 3.1. EXISTING SYSTEM

Most virtual communication platforms like Zoom, Microsoft Teams, and Google Meet do not offer effective real-time sign language interpretation. This creates barriers for individuals with hearing impairments, making it difficult for them to actively participate in virtual meetings, discussions, and conferences. The absence of integrated sign language recognition and translation features highlights the need for AI-driven solutions.

### Assistive Devices

Traditional assistive devices, such as hearing aids and cochlear implants, are designed to improve auditory perception for individuals with hearing loss. However, these devices do not address the communication challenges faced by those who rely on sign language, making them an incomplete solution for inclusive communication.

### Manual Communication Tools

Basic manual communication methods, such as pen and paper, are often used by individuals with hearing and speech impairments to convey messages. While effective for simple conversations, this approach is slow, inconvenient, and impractical for dynamic interactions in digital and professional environments.

### Sign Language Interpreters

In various settings, human sign language interpreters help bridge the communication gap between deaf and hearing individuals. However, interpreters are not always available, can be costly, and require prior arrangements, making them less practical for spontaneous or everyday.

**Support Vector Machines (SVMs) for Sign Language Recognition**

Support Vector Machines (SVMs) have been widely used for classification tasks in sign language recognition (SLR). They effectively handle high-dimensional feature vectors extracted from sign language images. However, their performance is limited when dealing with large datasets and continuous sign sequences, making them less efficient for real-time applications.

**Ensemble Learning for Improved Accuracy**

**Ensemble methods**, such as **Random Forests and Gradient Boosting**, enhance the accuracy and robustness of sign language recognition models by combining multiple classifiers. These techniques help in reducing errors and improving generalization but require high computational resources for training and implementation.

**Gesture Recognition by Learning from Poses (GRBP)**

**GRBP** is a gesture recognition method that focuses on learning spatial relationships between body joint poses for sign language recognition. By utilizing pose features extracted from skeletal data, this approach improves the accuracy of recognizing dynamic gestures. However, its reliance on high-quality skeletal data makes it challenging to implement in standard video-based sign language recognition systems.

### 3.1.1 DISADVANTAGES

- Virtual platforms lack built-in real-time sign language recognition and translation.
- Devices like cochlear implants do not support sign language use.
- Sign language interpreters are not always accessible and can be costly.

## 3.2 PROPOSED SYSTEM

The proposed system introduces an AI-driven sign language interpretation framework that enhances accessibility for individuals with hearing and speech impairments. By integrating deep learning and natural language processing techniques, the system enables seamless two-way communication between sign language users and non-signers. This ensures that virtual meetings, classrooms, and workplaces become more inclusive and accessible to all participants.

- **Sign Recognition Module (SRM)**

The Sign Recognition Module (SRM) utilizes Temporal Convolutional Networks (TCNs) to interpret sign language gestures accurately from live video input. This module processes hand movements, facial expressions, and body posture to ensure precise recognition of signs. Byleveraging TCNs, the system achieves high accuracy and real-time responsiveness, making virtual interactions more effective for sign language users.

- **Speech Recognition and Synthesis Module (SRSM)**

The Speech Recognition and Synthesis Module (SRSM) is responsible for converting spoken language into text and vice versa. This module employs Hidden Markov Models (HMMs) to process speech signals, transcribe them into text, and generate synthesized speech when needed. This feature allows users who rely on speech-based communication to effectively interact with sign language users, thereby bridging the communication gap.

- **Avatar Module (AM)**

The Avatar Module (AM) enhances accessibility by generating a real-time animated avatar that visually translates spoken language into sign language. This module ensures that deaf users who prefer sign language can receive information in an intuitive and interactive manner.

- **Integration with Virtual Platforms**

The system is designed to be seamlessly integrated with popular virtual meeting platforms such as Zoom, Microsoft Teams, and Google Meet. This integration ensures that individuals with hearing and speech impairments can participate fully in discussions without communication barriers. The system operates in real time, providing accurate translations and ensuring that sign language users are not excluded from professional or social interactions.

### 3.2.1 ADVANTAGES

- Real-time captioning converts spoken words into live text during virtual meetings, ensuring that hearing-impaired users can follow conversations.
- Speech-to-text transcription provides complete and accurate transcripts after meetings or events, allowing for review and understanding of missed content.
- Text-to-speech tools enable speech-impaired individuals to express themselves by converting typed text into natural-sounding audio in real time.
- AI-based sign language recognition systems can interpret gestures and convert them into text or speech, making virtual communication more inclusive.
- Multilingual translation capabilities allow captions and transcripts to be translated into various languages, bridging communication gaps for diverse users.
- Voice cloning technology helps create personalized digital voices, enabling speech-impaired individuals to communicate in a way that feels more natural.

# CHAPTER 4

# SYSTEM REQUIREMENT

## 4.1. HARDWARE REQUIREMENTS

- **Processor:** Intel Core i5 or higher
- **RAM:** 8GB or more
- **Storage:** 256GB SSD or higher
- **Camera & Microphone:** For real-time gesture and speech recognition
- **Monitor**:16 inch monitor
- **Keyboard**:Wired keyboard
- **Mouse**:optical
- **Graphics card**:Intel HD 4000 graphics family

## 4.2. SOFTWARE REQUIREMENTS

- **Operating System:** Windows 10 or 11 (for Windows-specific development)
- **Programming Language:** Python (version 3.6 or higher)
- **Web Framework:** Flask for implementing a web-based user interface,
- **Database Integration:** MySQL for storing and retrieving relevant data.
- **Integrated Development Environment (IDE):** IDLE
- **Web Technologies:** HTML, CSS, and JavaScript, Bootstrap
- **Libraries & Packages:** TensorFlow, OpenCV, NumPy, Pandas, Matplotlib, Seaborn, Pillow, Scikit-learn.
- **Video Calling Tools**: jitsi-For enabling real-time video calls.

## 4.3. SOFTWARE DESCRIPTION

### PYTHON 3.8

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

### Tensor Flow

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state- of-the-art in ML, and gives developers the ability to easily build and deploy ML-powered applications.it can used for the prediction of the models.

TensorFlow provides a collection of workflows with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages. Developers have the option to deploy models on a number of platforms such as on servers, in the cloud, on mobile and edgedevices, in browsers, and on many other JavaScript platforms. This enables developers to go frommodel building and training to deployment much more easily.

**Keras**

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation.

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc.
- This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

**Pandas**

pandas are a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. pandas are a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

Pandas is mainly used for data analysis and associated manipulation of tabular data in Data frames. Pandas allows importing data from various file formats such as comma-separated values, JSON, Parquet, SQL database tables or queries, and Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features. The development of pandas introduced into Python many comparable features of working with Data frames that were established in the R programming language.

**NumPy**

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

## Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

## MYSQL

MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. MySQL database that provides for how to manage database and to manipulate data wi MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications. It is developed, marketed, and supported by MySQL AB, a Swedish company, and written in C programming language and C++ programming language. With an intuitive interface, the application features numerous functionalities and makes it the preferred choice of developers from around the world. WampServer also has a “TrayIcon” allowing you .

# CHAPTER 5

# SYSTEM DESIGN

## 5.1 Concept diagram

This diagram shows interaction between Admin, Deaf User, Listener, and the Sign Meet Server. Each role performs specific actions like login, input (sign/speech), and receiving output (text/avatar). The server processes sign language recognition, interpretation, and avatar generation.

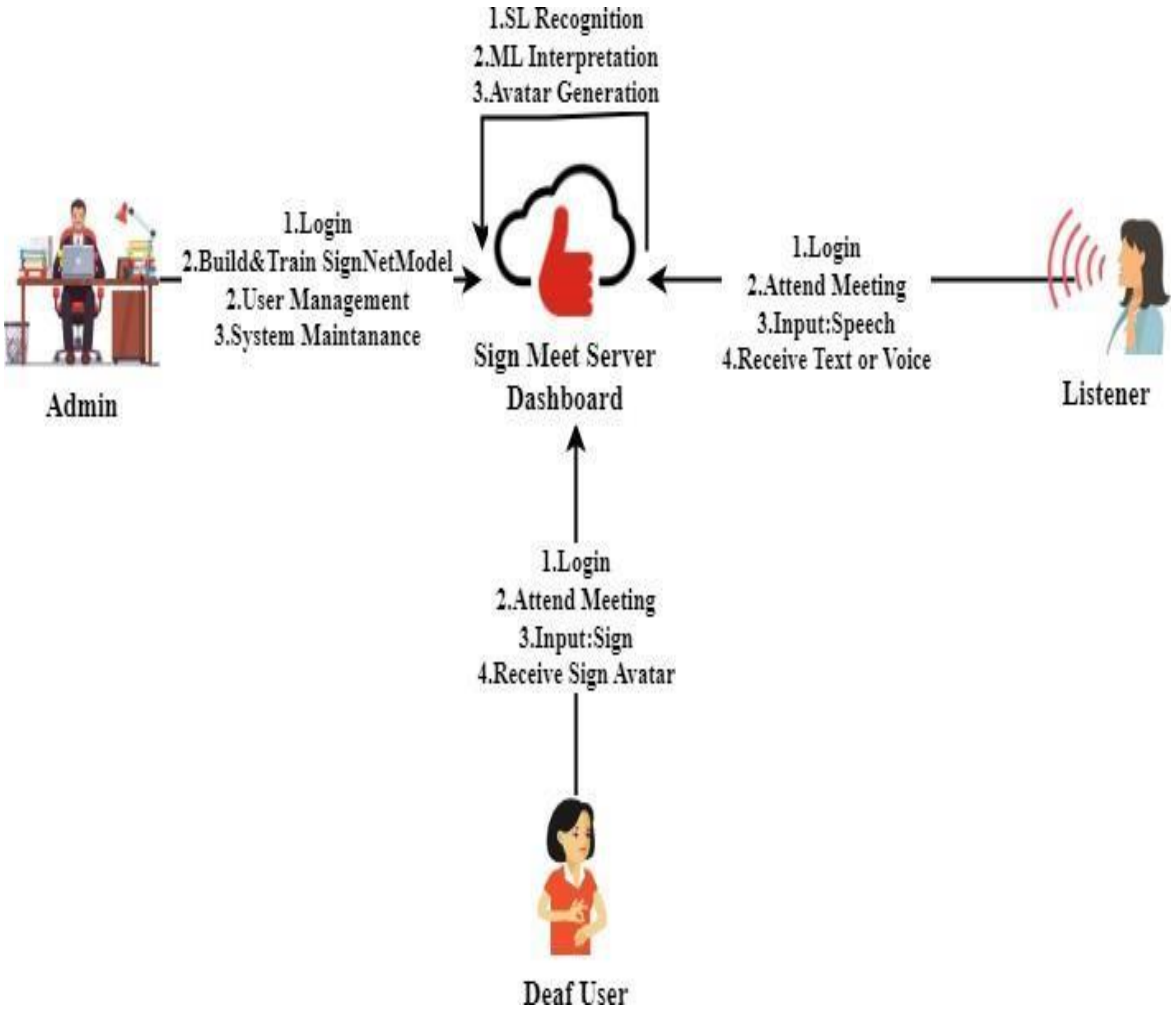


**FIG 5.1 CONCEPT DIAGRAM**

## 5.2 SYSTEM ARCHITECHTURE

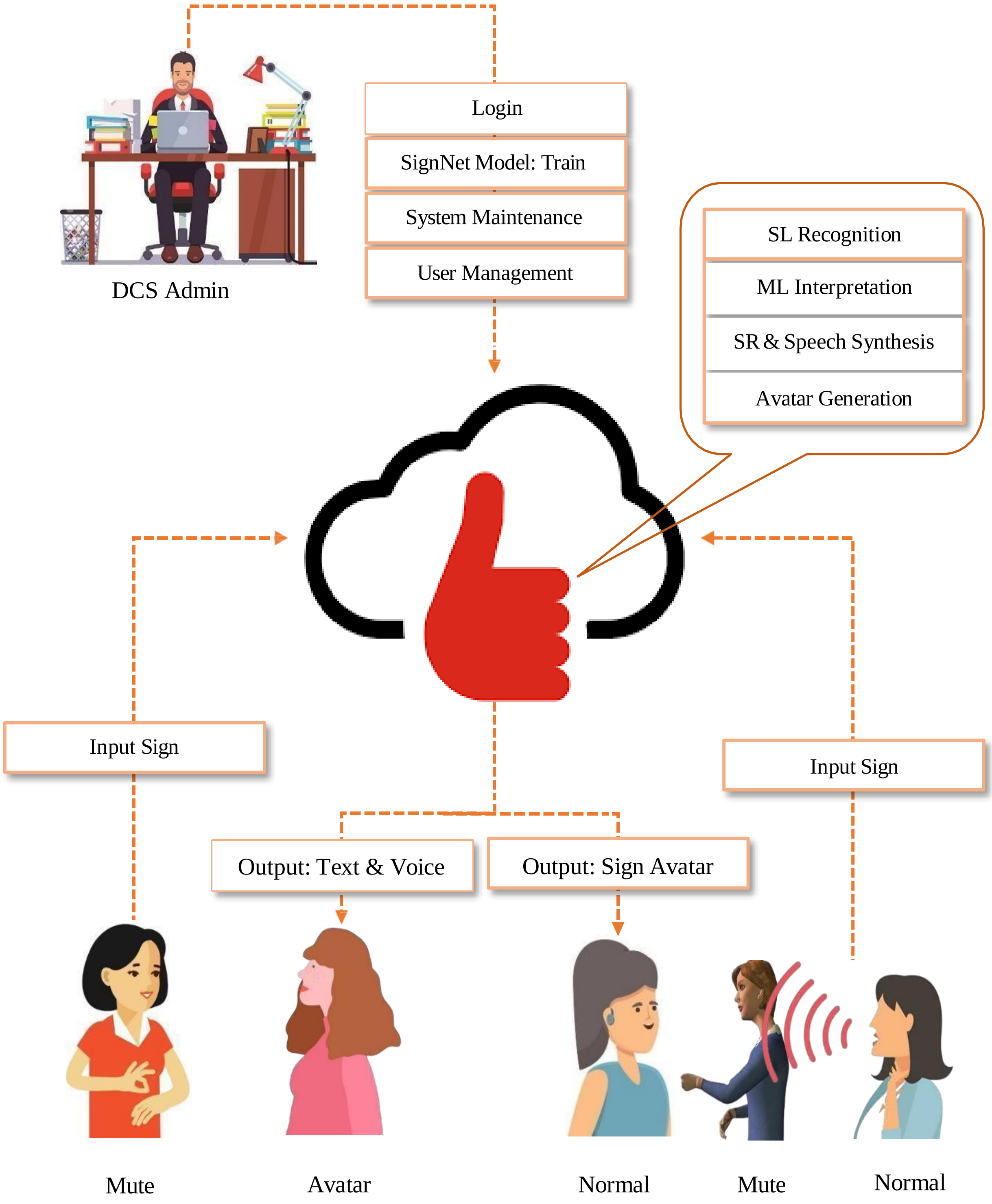


**FIG 5.2 SYSTEM ARCHITECHTURE**

## 5.3. DATA FLOW DIAGRAM

### LEVEL 0

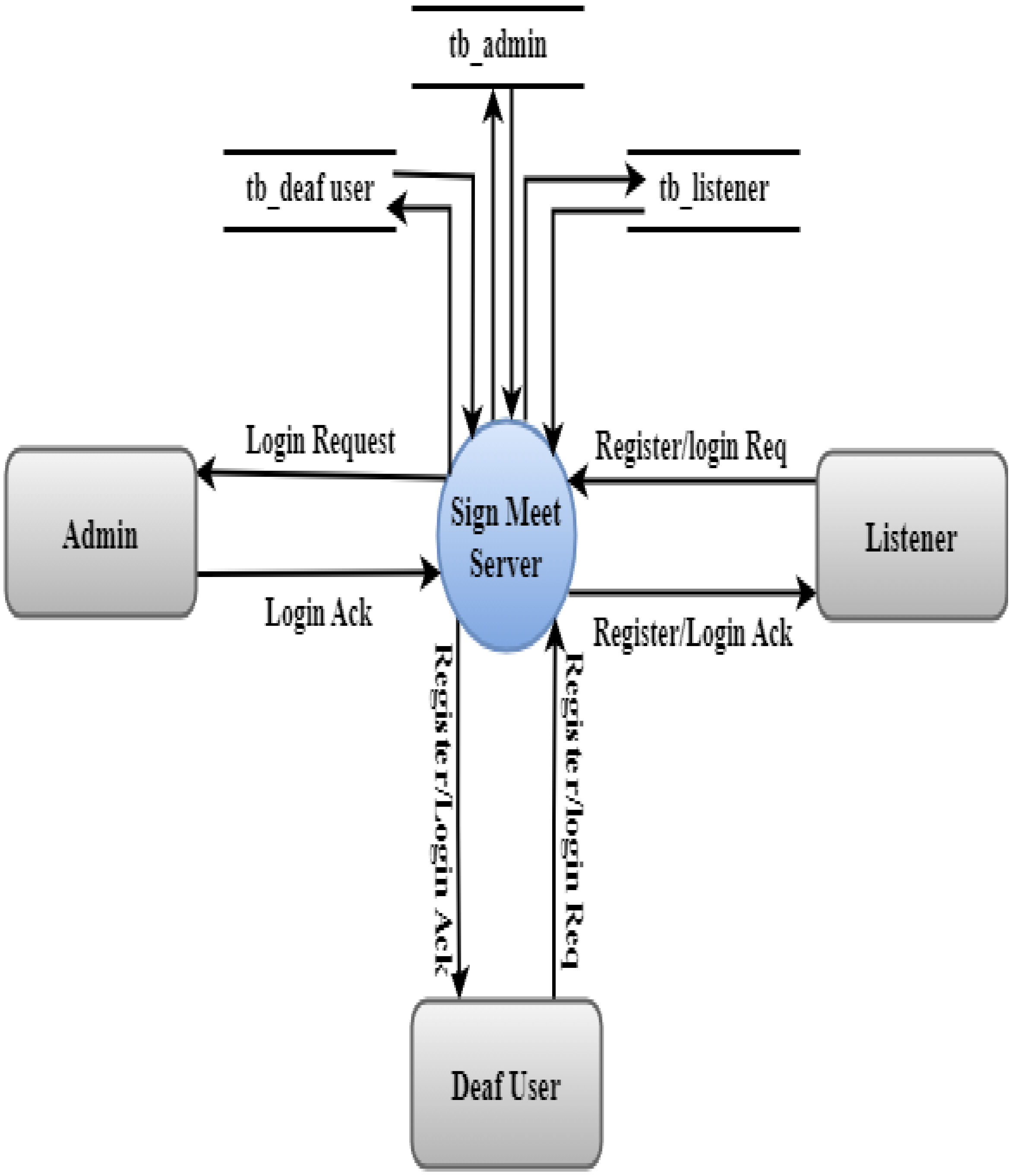


**FIG 5.3.1 LEVEL 0**

## LEVEL 1

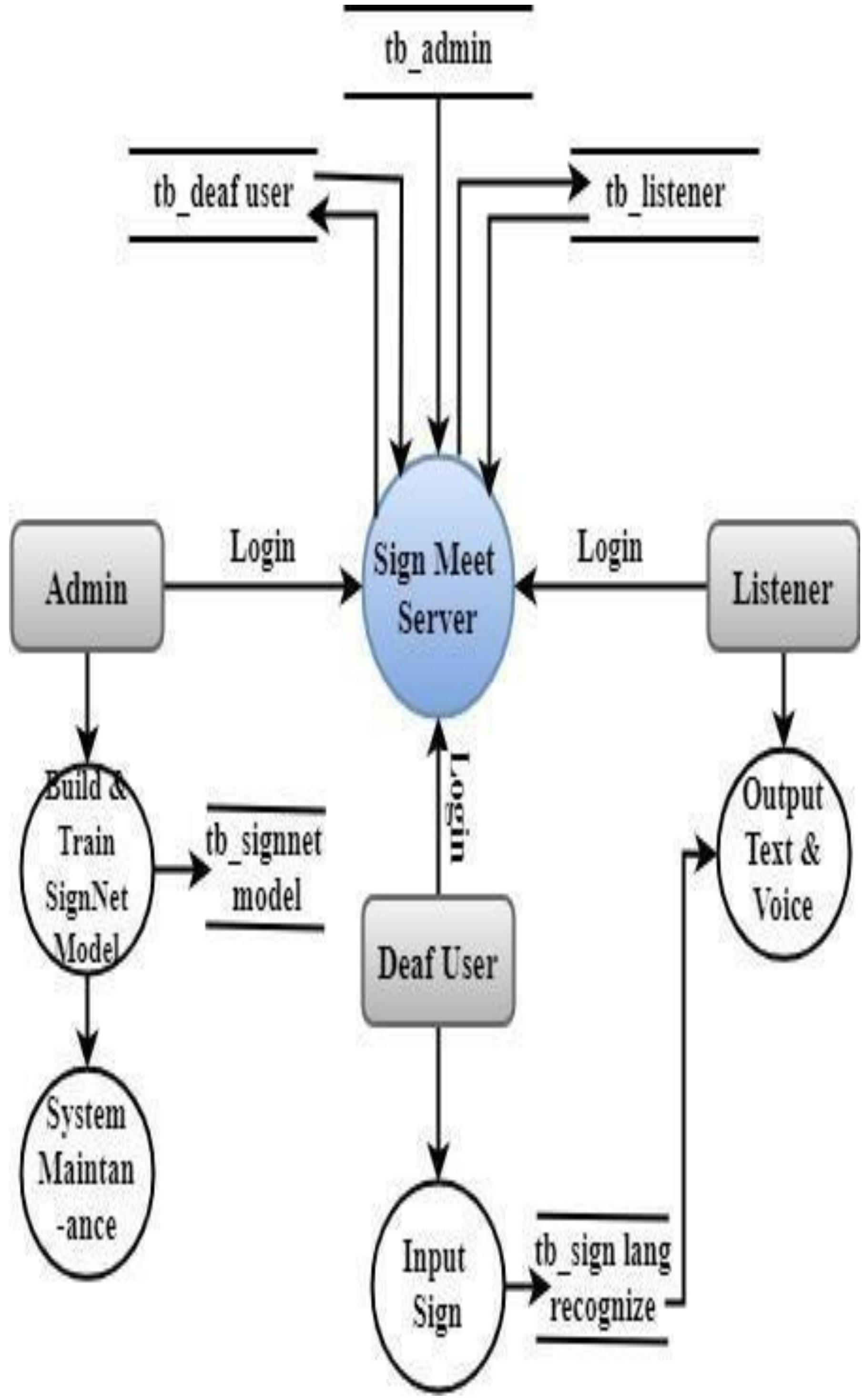


FIG 5.3.2 LEVEL 1 DFD

**LEVEL 2**

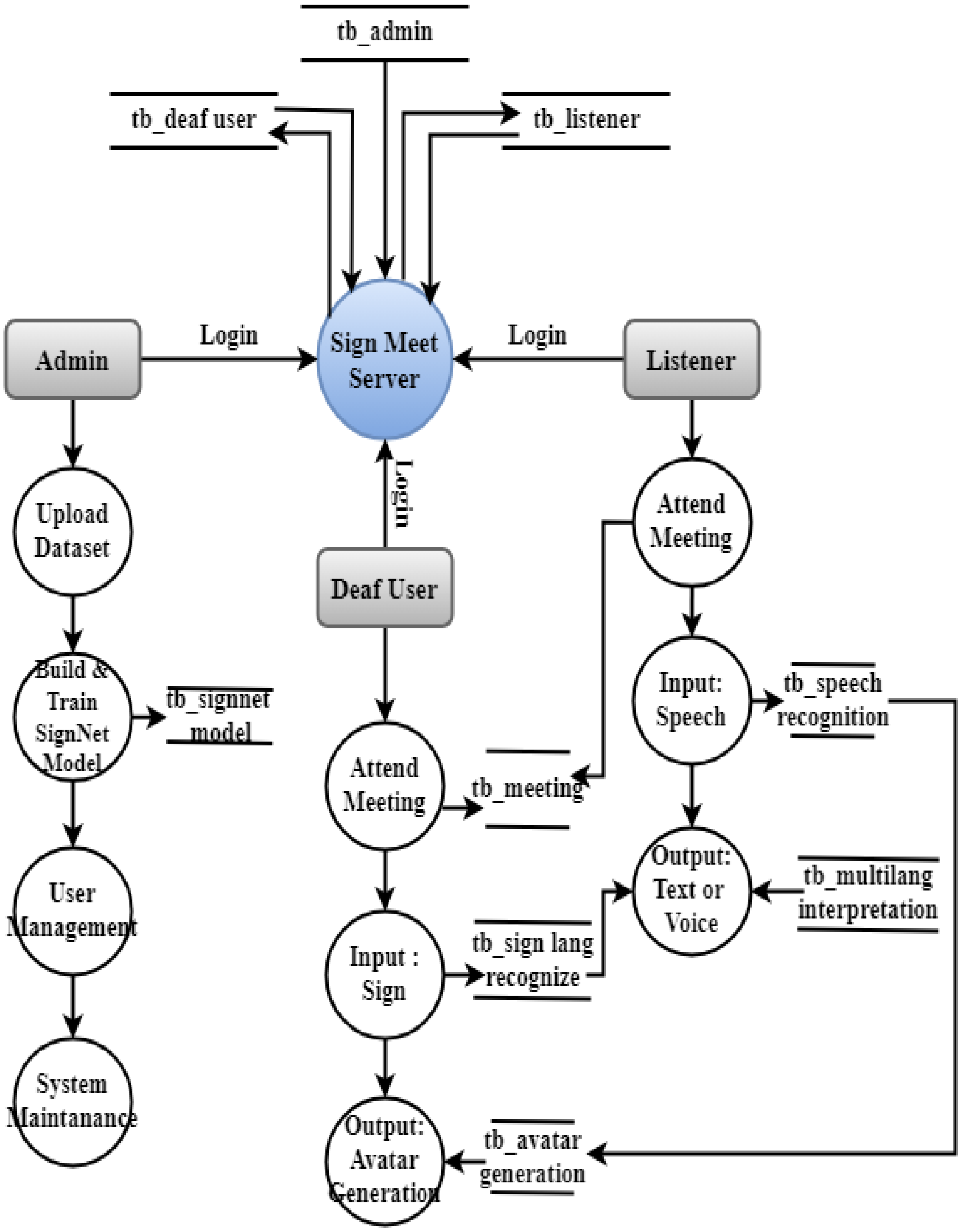


**FIG 5.3.3 LEVEL 2 DFD**

## 5.4UML DIAGRAM

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

- Use case diagram
- Sequence Diagram
- Class
- Activity Diagram

### 5.4.1 USE CASE DIAGRAM

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). In software and systems engineering, a use case is a list of actions or event steps typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system to achieve a goal. The actor can be a human or other external system. An actor in the Unified Modeling Language (UML) "specifies a role played by a user or any other system that interacts with the subject." "An Actor models a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data), but which is external to the subject." UML Use Case Include. Use case include is a directed relationship between two use cases which is used to show that behavior of the included use case (the addition) is inserted into the behavior of the including (the base) use case.The system can be enhanced by integrating features such as role-based access for government officers, dispute resolution modules, geolocation-based property validation, and integration with real-world data.

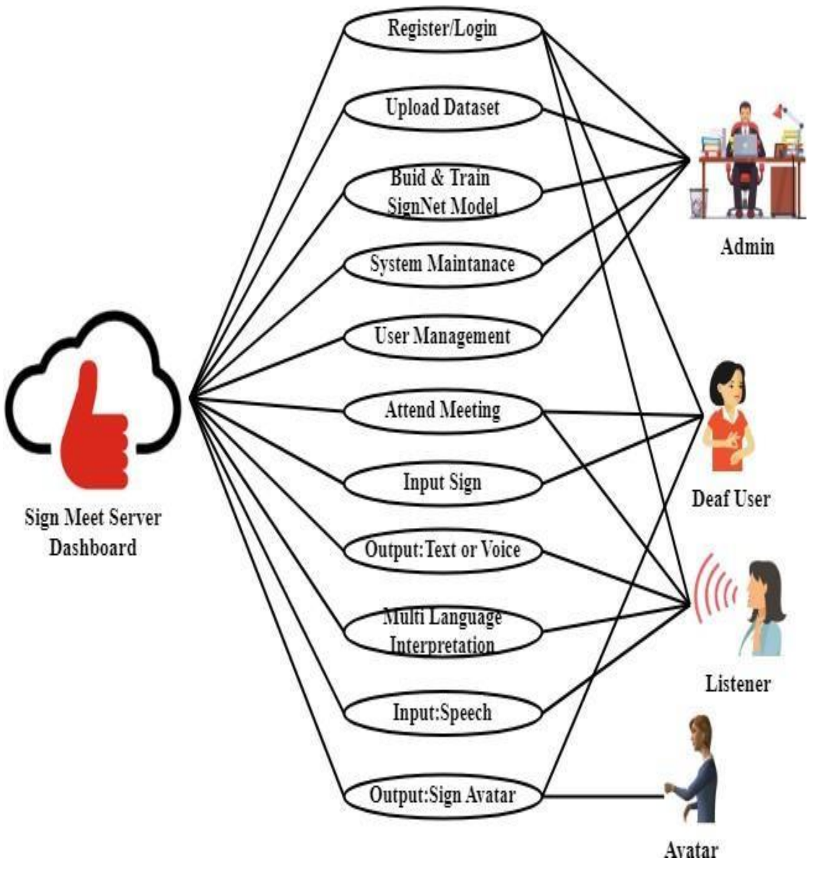


**FIG 5.4.1 USE CASE DIAGRAM**

### 5.4.2 SEQUENCE DIAGRAM

Sequence diagrams are sometimes called event diagrams or event scenarios. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

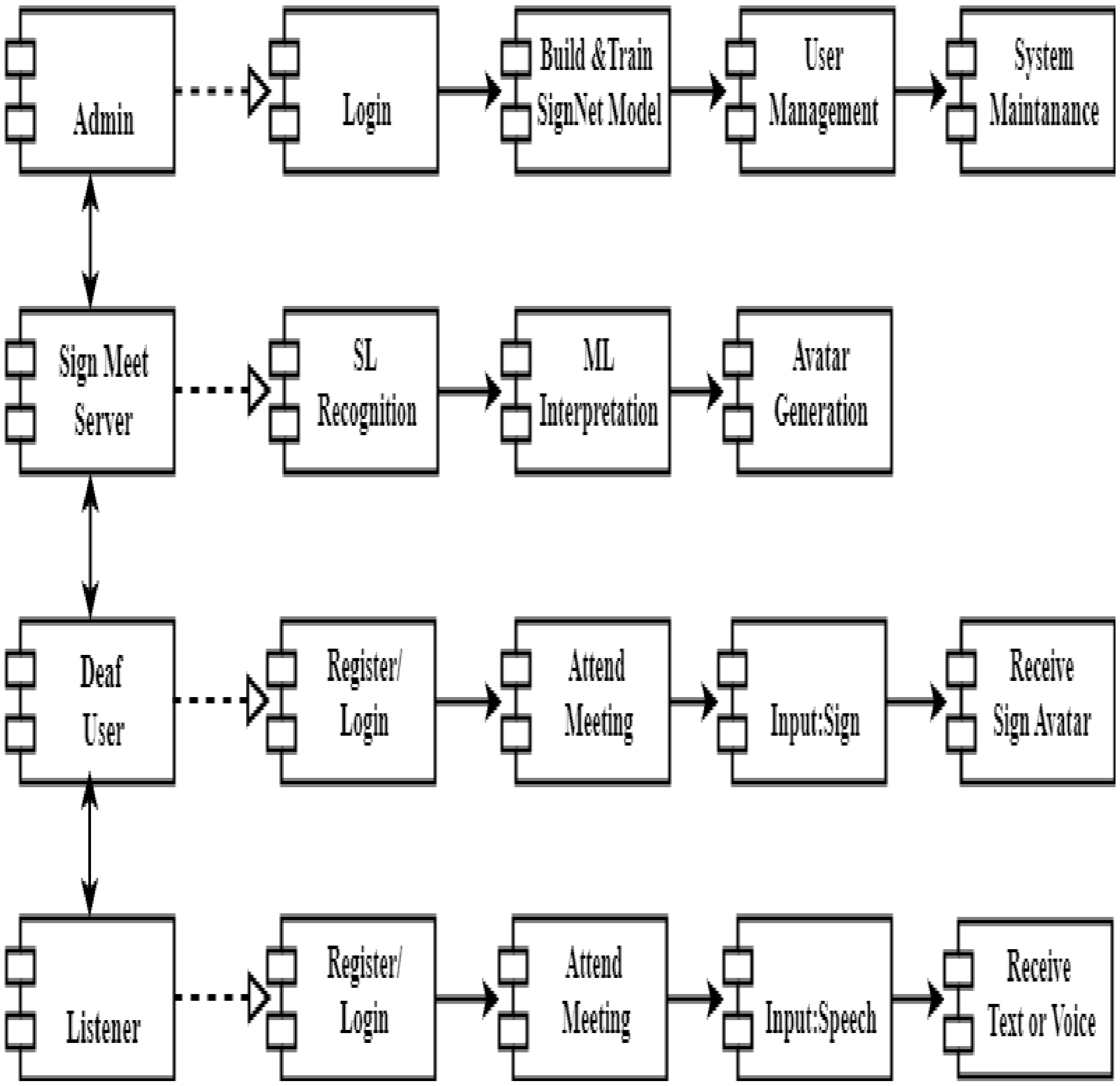


**FIG 5.4.2 SEQUENCE DIAGRAM**

### 5.4.3 ACTIVITY DIAGRAM

We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram. UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior.

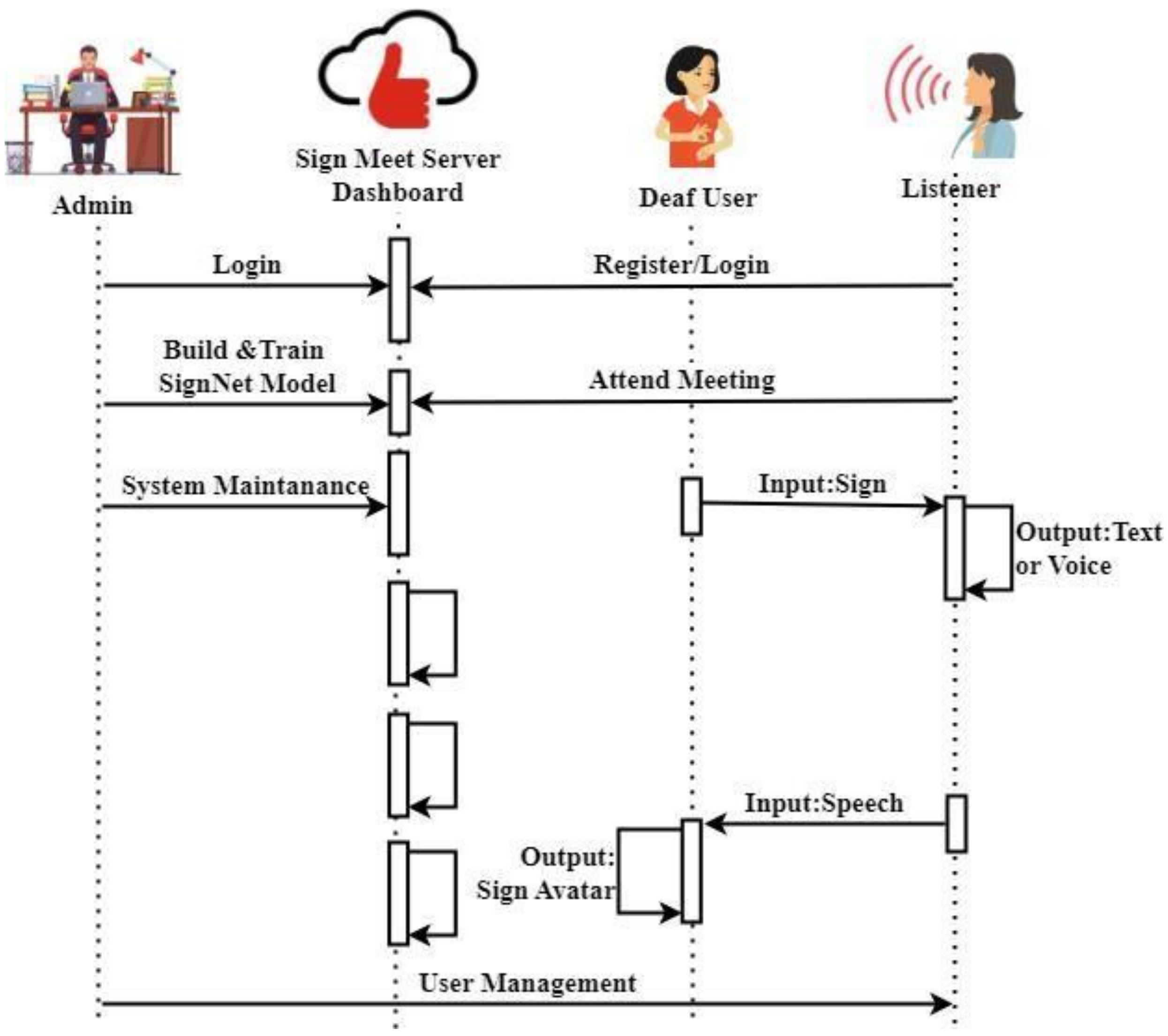


**FIG 5.4.3 ACTIVITY DIAGRAM**

### 5.4.4 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects. Each class represents a blueprint for creating objects (instances), and the diagram helps in understanding and designing the architecture of the system.

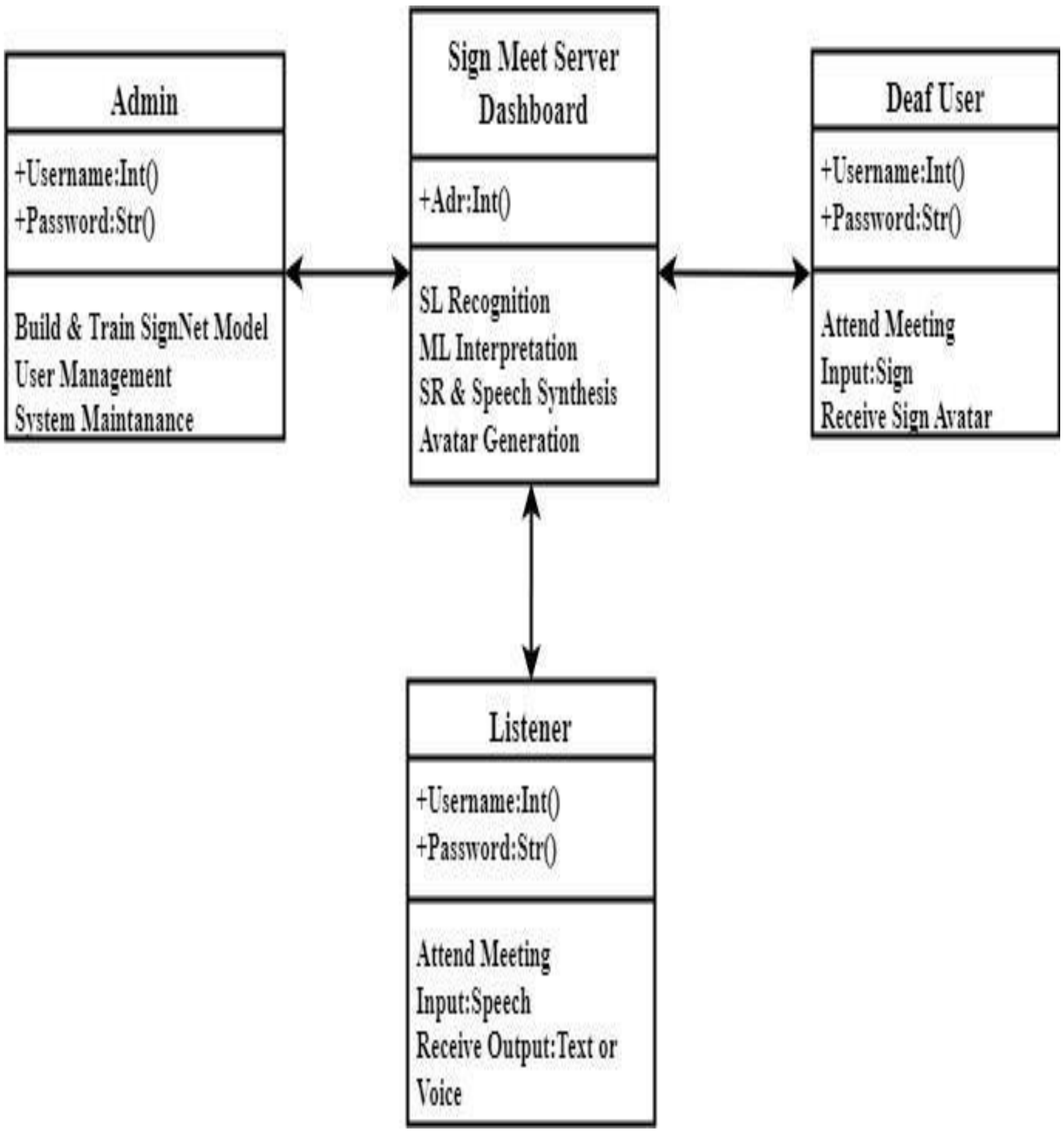


**FIG 5.4.4 CLASS DIAGRAM**

# CHAPTER 6
# MODULES DESCRIPTION

## 6.1. MODULES

- Sign Meet Web App
- End User Interface
- SignNet Model: Build and Train
- Sign Language Recognition
- Multilanguage Interpretation
- Speech Recognition
- Avatar Generation
- Visual Communication

## 6.2 DESCRIPTION

### 6.2.1 Sign Meet Web App

The Sign Meet Web App is a virtual meeting platform designed to bridge the communication gap between deaf and non-deaf users by integrating sign language recognition, speech synthesis, and avatar-based interpretation. The platform is developed using Python, Flask, MySQL, Bootstrap, WampServer, and other necessary libraries and is integrated with Jitsi, an open-source virtual meeting platform, for real-time video conferencing. This system ensures that users with hearing or speech impairments can actively participate in online meetings, lectures, and discussions without barriers. the Sign Meet Web App provides a comprehensive, AI-powered solution for enabling inclusive virtual communication. By integrating Jitsi for video conferencing and leveraging advanced AI techniques, the system effectively breaks communication barriers, ensuring a more accessible and connected virtual environment for everyone.

### 6.2.2 End User Interface

The End User Module defines the different roles within the Sign Meet Web App, ensuring that each user has specific functionalities tailored to their needs. The system supports three main types of users: Admin, Deaf Users, and Non-Deaf Users, each playing a crucial role in the platform's functionality.

### 6.2.3 SignNet Model: Build and Train

The SignNet Model is the core AI-powered sign language recognition system used in the Sign Meet Web App. This model is responsible for real-time gesture recognition, processing live video input, extracting key features, and classifying hand gestures into meaningful sign language representations. It is trained using advanced deep learning techniques to ensure high accuracy in interpreting sign language. The model undergoes multiple stages, including preprocessing, feature extraction, classification, and deployment, to provide seamless interaction between deaf and non- deaf users.

### 6.2.4 Sign Language Recognition

The Sign Language Recognition module is a critical component of the Sign Meet Web App, enabling seamless communication between deaf and non-deaf users. This module is designed to process live video input and recognize sign language gestures in real time, translating them into text or speech output. It utilizes Temporal Convolutional Networks (TCN) in conjunction with the trained SignNet Model to achieve high accuracy in sign language recognition. The system ensures that gestures are detected, processed, and classified efficiently, allowing for smooth and effective interaction in virtual meetings.

### 6.2.5 Multilanguage Interpretation

The Multilanguage Interpretation module enables real-time translation of sign language into multiple spoken and written languages, ensuring seamless communication for deaf and non-deaf users. It leverages Natural Language Processing (NLP) and deep learning to accurately translate recognized signs into text or speech across various languages. The system also supports speech-to-sign language conversion by using speech recognition techniques like MFCC and HMMs to transcribe spoken words into text, which is then converted into sign language and displayed via an AI-powered avatar. By integrating context-aware translation, the module ensures accurate and meaningful interpretations across different languages and sign language dialects. It adapts to regional variations and continuously improves through user interactions. Fully integrated into Jitsi Virtual Meet, this module allows users to set their preferred language, displaying translated text or avatar-based sign language alongside the meeting interface. This enhances accessibility and inclusivity, enabling effective communication in virtual platforms.

### 6.2.6 Speech Recognition

The Speech Recognition and Visual Communication module facilitates effective interaction between non-deaf and deaf users by converting spoken language into text and sign language. This enables non-deaf users to ask questions and receive responses in a visually accessible format.The speech recognition process begins with preprocessing using Wavelet Transform, which enhances speech clarity by filtering out noise. Next, Mel-Frequency Cepstral Coefficients (MFCC) are used for feature extraction, capturing essential speech characteristics to improve recognition accuracy.

### 6.2.7 Avatar Generation

The Avatar Generation module plays a crucial role in bridging the communication gap between deaf and non-deaf users by providing a real-time, animated visual representation of spoken language. This module is designed to convert recognized speech or text into sign language gestures using an AI-powered avatar, ensuring an inclusive and accessible communication experience. The avatar is dynamically generated using advanced motion synthesis techniques. When a non-deaf user speaks, the system first processes the speech through speech recognition models to convert it into text. The corresponding sign language gestures are then mapped based on predefined linguistic models. These gestures are performed by a realistic 3D animated avatar, ensuring accurate and expressive sign representation.

### 6.2.8 Visual Communication

For visual communication, an AI-driven avatar dynamically translates recognized speech into sign language, allowing deaf users to understand the conversation in real time. This avatar-based approach enhances accessibility and inclusivity in virtual interactions. Additionally, text-based responses from deaf users can be synthesized into speech, enabling a seamless two-way conversation. Integrated with virtual platforms Jitsi, this module ensures a more inclusive and interactive communication experience for all users. ensuring that each user has specific functionalities tailored to their needs. The system supports three main types of users: Admin, Deaf Users, and Non-Deaf Users, each playing a crucial role in the platform's functionality. ensuring an inclusive and accessible communication experience. The avatar is dynamically generated using advanced motion synthesis techniques.

# CHAPTER 7
# SYSTEM TESTING

## 7.1 SOFTWARE TESTING

Software testing is the process of evaluating and verifying that a software application functions correctly, meets specified requirements, and is free of defects. It ensures the reliability, security, and performance of the system. Testing is essential to identify bugs, improve functionality, and enhance user experience before deployment. For this project, testing is to ensure accurate sign language recognition, seamless virtual communication, proper speech-to-text conversion, avatar generation, and integration with virtual platforms like Jitsi.

## 7.2 TYPES OF TESTING

**Functional Testing**

Functional testing verifies that the application performs as expected according to the specified requirements. It checks whether the system's features, such as sign language recognition, speech- to-text conversion, avatar generation, and meeting functionalities, work correctly. This ensures that users can communicate without issues in virtual meetings.

**Unit Testing**

Unit testing focuses on testing individual components of the system to ensure they function correctly before integration.

**Integration Testing**

Integration testing checks the interaction between different modules of the system. For example, it verifies whether the SignNet Model integrates correctly with the virtual meeting platform (Jitsi), whether user authentication is done.

**Usability Testing**

Usability testing ensures that the system is user-friendly, intuitive, and accessible to both deaf and non-deaf users. It assesses whether users can easily navigate the platform, join meetings, recognize signs, and interact with the avatar-based sign language interpreter. Feedback from test users helps refine the interface for better accessibility.

**Security Testing**

Security testing focuses on protecting user data and ensuring the system is resistant to attacks. It verifies that sensitive information like user credentials, meeting links, and communication data is secure from unauthorized access. The project will implement encryption and authentication mechanisms to ensure user privacy.

**Compatibility Testing**

Compatibility testing ensures that the application functions correctly across different browsers (Chrome, Firefox, Edge), operating systems (Windows, macOS, Linux), and devices (PC, mobile, tablet). This ensures that users can access the platform without technical difficulties, regardless of their device or OS.

**System Testing**

System testing evaluates the entire system to check if all integrated components work together as intended. This ensures that all modules, from sign language recognition to virtual meeting integration, function as a single cohesive unit before deployment.

By applying these testing methodologies, the project will deliver accuracy, accessibility, and efficiency, creating an inclusive virtual communication platform for both deaf and non-deaf users.

## 7.3 TEST CASES

Test cases are a set of predefined conditions used to verify the functionality, reliability, and accuracy of the project. Each test case outlines specific inputs, expected outputs, and actual results to ensure the system meets its requirements. By executing these test cases, potential defects can be identified and resolved,

1. **Test Case ID:** TC001
   - **Input:** Admin enters valid credentials and clicks "Login"
   - **Expected Result:** Admin dashboard should be displayed
   - **Actual Result:** Admin dashboard displayed successfully
   - **Status:** Passed
2. **Test Case ID:** TC002
   - **Input:** Admin enters invalid credentials and clicks "Login"
   - **Expected Result:** Error message "Invalid username or password" should be displayed
   - **Actual Result:** Error message displayed
   - **Status:** Passed
3. **Test Case ID:** TC003
   - **Input:** Deaf user registers with all required details
   - **Expected Result:** Account should be created successfully
   - **Actual Result:** Account created successfully
   - **Status:** Passed
4. **Test Case ID:** TC004
   - **Input:** Deaf user registers with an already existing email
   - **Expected Result:** Error message "Email already exists" should be displayed
   - **Actual Result:** Error message displayed
   - **Status:** Passed

5. **Test Case ID:** TC005
    - **Input:** Deaf user logs in with valid credentials
    - **Expected Result:** User dashboard should be displayed
    - **Actual Result:** User dashboard displayed successfully
    - **Status:** Passed
6. **Test Case ID:** TC006
    - **Input:** Deaf user starts a virtual meeting
    - **Expected Result:** Meeting link should be generated and shared with participants
    - **Actual Result:** Meeting link generated and shared successfully
    - **Status:** Passed

## 7.4 TEST REPORT

### Introduction

The test report documents the testing process carried out for the project is to ensure its accuracy, efficiency, and seamless integration with virtual meeting platforms. It includes test objectives, scope, environment, results, and identified bugs.

### Test Objective

The objective of this testing is to validate the proper functionality of sign language recognition, speech synthesis, avatar generation, and multilingual interpretation within the virtual meeting environment.

### Test Scope

The testing scope covers the Sign Meet Web App modules, including user authentication, real- time sign language recognition, speech-to-text conversion, avatar display, and system integration with virtual platforms like Jitsi.

**Test Environment**

- **Operating System:** Windows / Linux
- **Development Framework:** Python, Flask, Bootstrap
- **Database:** MySQL, WampServer
- **Other Dependencies:** OpenCV, TensorFlow, Jitsi API

**Test Result**

The system has been tested for various functionalities, including user authentication, sign recognition accuracy, speech synthesis, and avatar generation. The test cases were executed successfully, and the system performed as expected, with a few minor bugs identified and reported.

**Bug Report**

A bug report records issues, defects, or errors encountered during testing. It includes details such as Bug ID, Test Case ID, Bug Description, Bug Status, and Output, ensuring developers can track and fix the issues efficiently.

| **Bug ID** | **TC ID** | **Bug Description** | **Bug Status** | **Output** |
|---|---|---|---|---|
| BUG001 | TC_05 | Sign recognition fails for certain gestures in dim lighting | Open | System misinterprets signs |
| BUG002 | TC_12 | Avatar animation lags during rapid speech conversion | Fixed | Avatar syncs with delay |

**Test Conclusion**

The project has undergone rigorous testing, and most functionalities performed as expected. Identified bugs are documented and will be addressed in the next development cycle to enhance system performance and usability.

# CHAPTER 8
# CONCLUSION

## 8.1 CONCLUSION

In conclusion, this project successfully bridges the communication gap between deaf and non-deaf users by integrating advanced Sign Language Recognition, Speech Recognition, and Avatar-Based Visual Communication within a virtual meeting platform. The module effectively recognizes sign language gestures using the SignNet Model trained with Temporal Convolutional Networks (TCNs) and converts speech into sign language using Hidden Markov Models (HMMs) and an animated avatar. By integrating with Jitsi virtual platforms, the module ensures seamless real-time communication in online meetings. Additionally, the multilanguage interpretation, speech synthesis, and notification system enhance accessibility and usability for a diverse group of users. This module provides a scalable and inclusive communication platform, significantly improving virtual interactions for the deaf community. This project provides a scalable and inclusive communication platform, significantly improving virtual interactions for the deaf community. Future enhancements could include expanded language support, AI-driven sign recognition improvements, and deeper integration with other conferencing platforms, further advancing accessibility and inclusivity.

## 8.2 FUTURE ENHANCEMENT

This project can be further improved with advanced features and optimizations to enhance accessibility, accuracy, and user experience. Some key future enhancements include:

**Expanded Language Support**: Incorporating additional sign languages, including regional and country-specific variations, to support a wider audience.

**Integration with More Virtual Platforms**: Extending compatibility beyond Jitsi to platforms like Zoom, Microsoft Teams, and Google Meet, allowing broader adoption.

**Enhanced Avatar Customization**: Improving the Avatar Module with realistic 3D animations, facial expressions, and lip-syncing for better sign language representation.

**Mobile App Development**: Creating a mobile version of the project to allow accessibility on smartphones and tablets for on-the-go communication.

**Automated Meeting Transcription and Summarization**: Implementing AI-based summarization tools to provide meeting notes and transcripts for improved accessibility and documentation.

By incorporating these enhancements, the proposed system can evolve into a comprehensive and highly accessible communication tool, fostering a truly inclusive virtual environment.

# CHAPTER 9
# APPENDICES

## 9.1 SOURCE CODE

**Packages**

```
from flask import Flask, render_template, Response, redirect, request, session, abort, url_for
from camera import VideoCamera
from camera1 import VideoCamera1
import mysql.connector
from random import randint
from urllib.request import urlopen
import cv2
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import shutil
import imagehash
from werkzeug.utils import secure_filename
from PIL import Image
import argparse
import urllib.request
import urllib.parse
import pyttsx3
from skimage import transform
from sklearn.model_selection import train_test_split
import joblib
import speech_recognition as sr
from googletrans import Translator
from gtts import gTTS
import mediapipe as mp
import tensorflow as tf
from PIL import Image
```

```
host="localhost",
user="root",
password="",
charset="utf8",
    database="sign_meet_new"
```

**Login**

```
    def login_user():
    msg=""
    if request.method=='POST':
    uname=request.form['uname']
    pwd=request.form['pass']
    cursor = mydb.cursor()
    cursor.execute('SELECT * FROM sign_user WHERE uname = %s AND pass = %s', (uname,
    pwd))
    account = cursor.fetchone()
    if account:
    session['username'] = uname
    ff=open("static/msg.txt","w")
    ff.write("")
    ff.close()
    cursor.execute('SELECT count(*) FROM sign_user WHERE status=1')
    dd = cursor.fetchone()[0]
    if dd==0:
    cursor.execute("update sign_user set status=1 where uname=%s",(uname,))
    mydb.commit()
    ff=open("static/deaf.txt","w")
    ff.write(uname)
    ff.close()
    return redirect(url_for('test_cam'))
    else:
    msg = 'Incorrect username/password!'
```

**User Registration**

```
def register():
msg=""
now = datetime.datetime.now()
rdate=now.strftime("%d-%m-%Y")
mycursor = mydb.cursor()
if request.method=='POST':
name=request.form['name']
mobile=request.form['mobile']
email=request.form['email']
uname=request.form['uname']
pass1=request.form['pass']
mycursor.execute('SELECT count(*) FROM sign_user WHERE uname = %s', (uname,))
cnt = mycursor.fetchone()[0]
if cnt==0:
mycursor.execute("SELECT max(id)+1 FROM sign_user")
maxid = mycursor.fetchone()[0]
if maxid is None:
maxid=1
sql = "INSERT INTO sign_user(id,name,mobile,email,uname,pass) VALUES (%s, %s, %s, %s,
%s, %s)"
val = (maxid,name,mobile,email,uname,pass1)
mycursor.execute(sql,val)
mydb.commit()
return redirect(url_for('login_user'))
else:
msg="fail"
```

**Speech Recognition**

```
def lg_translate(lg,output):
result=""
recognized_text=output
```

```
recognizer = sr.Recognizer()
translator = Translator()
try:
available_languages = {
'ta': 'Tamil',
'hi': 'Hindi',
'ml': 'Malayalam',
'kn': 'Kannada',
'te': 'Telugu',
'mr': 'Marathi',
'ur': 'Urdu',
'bn': 'Bengali',
'gu': 'Gujarati',
'fr': 'French'
}print("Available languages:")
for code, language in available_languages.items():
print(f"{code}: {language}")
#selected_languages = input("Enter the language codes (comma-separated) you want to translate
to: ").split(',')
selected_languages=lg.split(',')
for lang_code in selected_languages:
lang_code = lang_code.strip()
if lang_code in available_languages:
translated = translator.translate(recognized_text, dest=lang_code)
print(f"Translation in {available_languages[lang_code]} ({lang_code}): {translated.text}")
result=translated.text
else:
print(f"Language code {lang_code} not available.")
except Exception as e:
print("An error occurred during translation:", e)
return result
```

```
def translate_text(text, source_language, target_language):
api_key = 'AIzaSyDW9tvaQUsywmaILt73Go8Fy5mU6ILOixU' # Replace with your API key
url = f'https://translation.googleapis.com/language/translate/v2?key={api_key}'
payload = {
'q': text,
'source': source_language,
'target': target_language,
'format': 'text'
response = requests.post(url, json=payload)
translation_data = response.json()
translated_text = translation_data
#translation_data['data']['translations'][0]['translatedText']
return translated_text
def speak(audio):
engine = pyttsx3.init()
engine.say(audio)
engine.runAndWait()
def text_to_speech(text, language='en'):
# Create a gTTS object
tts = gTTS(text=text, lang=language, slow=False)
tts.save("static/output.mp3")
```

**Train Gesture**

```
def train_gesture():
msg=""
mycursor = mydb.cursor()
if request.method=='POST':
gname=request.form['gname']
mycursor.execute("SELECT count(*) FROM ga_gesture where gesture=%s",(gname,))
cnt = mycursor.fetchone()[0]
if cnt==0:
mycursor.execute("SELECT max(id)+1 FROM ga_gesture")
```

```
maxid = mycursor.fetchone()[0]
if maxid is None:
maxid=1
gf="f"+str(maxid)
gfile="f"+str(maxid)+".csv"
ff=open("static/label1.txt","w")
ff.write(gfile)
ff.close()
sql = "INSERT INTO ga_gesture(id,gesture,fname) VALUES (%s, %s, %s)"
val = (maxid,gname,gfile)
mycursor.execute(sql,val)
mydb.commit()
else:
mycursor.execute("SELECT * FROM ga_gesture where gesture=%s",(gname,))
gd = mycursor.fetchone()
gid=gd[0]
ff=open("static/label.txt","w")
ff.write(gname)
ff.close()
gf="f"+str(gid)
gfile="f"+str(gid)+".csv"
ff=open("static/label1.txt","w")
ff.write(gfile)
ff.close()
```

**Capture Hand Landmark**

```
def capture():
msg=""
gdata=[]
act=request.args.get("act")
st=request.args.get("st")
mycursor = mydb.cursor()
```

```
mycursor.execute("SELECT * FROM ga_gesture")
gdata = mycursor.fetchall()
if st=="del":
did=request.args.get("did")
mycursor.execute("SELECT * FROM ga_gesture where id=%s",(did,))
gd = mycursor.fetchone()
gfile=gd[2]
os.remove("static/hand_gesture_data/"+gfile)
mycursor.execute("delete from ga_gesture where id=%s",(did,))
mydb.commit()
return redirect(url_for('capture',act='1'))
def CNN():
#Lets start by loading the Cifar10 data
(X, y), (X_test, y_test) = cifar10.load_data()
#Keep in mind the images are in RGB
#So we can normalise the data by diving by 255
#The data is in integers therefore we need to convert them to float first
X, X_test = X.astype('float32')/255.0, X_test.astype('float32')/255.0
#Then we convert the y values into one-hot vectors
#The cifar10 has only 10 classes, thats is why we specify a one-hot
#vector of width/class 10
y, y_test = u.to_categorical(y, 10), u.to_categorical(y_test, 10)
#Now we can go ahead and create our Convolution model
model = Sequential()
#We want to output 32 features maps. The kernel size is going to be
#3x3 and we specify our input shape to be 32x32 with 3 channels
#Padding=same means we want the same dimensional output as input
#activation specifies the activation function
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same',
activation='relu'))
#20% of the nodes are set to 0
```

```
model.add(Dropout(0.2))
#now we add another convolution layer, again with a 3x3 kernel
#This time our padding=valid this means that the output dimension can
#take any form
model.add(Conv2D(32, (3, 3), activation='relu', padding='valid'))
#maxpool with a kernet of 2x2
model.add(MaxPooling2D(pool_size=(2, 2)))
#In a convolution NN, we neet to flatten our data before we can
#input it into the ouput/dense layer
model.add(Flatten())
#Dense layer with 512 hidden units
model.add(Dense(512, activation='relu'))
#this time we set 30% of the nodes to 0 to minimize overfitting
model.add(Dropout(0.3))
#Finally the output dense layer with 10 hidden units corresponding to
#our 10 classe
model.add(Dense(10, activation='softmax'))
#Few simple configurations
model.compile(loss='categorical_crossentropy',
optimizer=SGD(momentum=0.5, decay=0.0004), metrics=['accuracy'])
#Run the algorithm!
model.fit(X, y, validation_data=(X_test, y_test), epochs=25,
batch_size=512)
#Save the weights to use for later
model.save_weights("cifar10.hdf5")
#Finally print the accuracy of our model!
print("Accuracy:  &2.f%%"  %(model.evaluate(X_test,  y_test)[1]*100))
```

**#Classification**

```
def classify():
msg=""
mycursor = mydb.cursor()
```

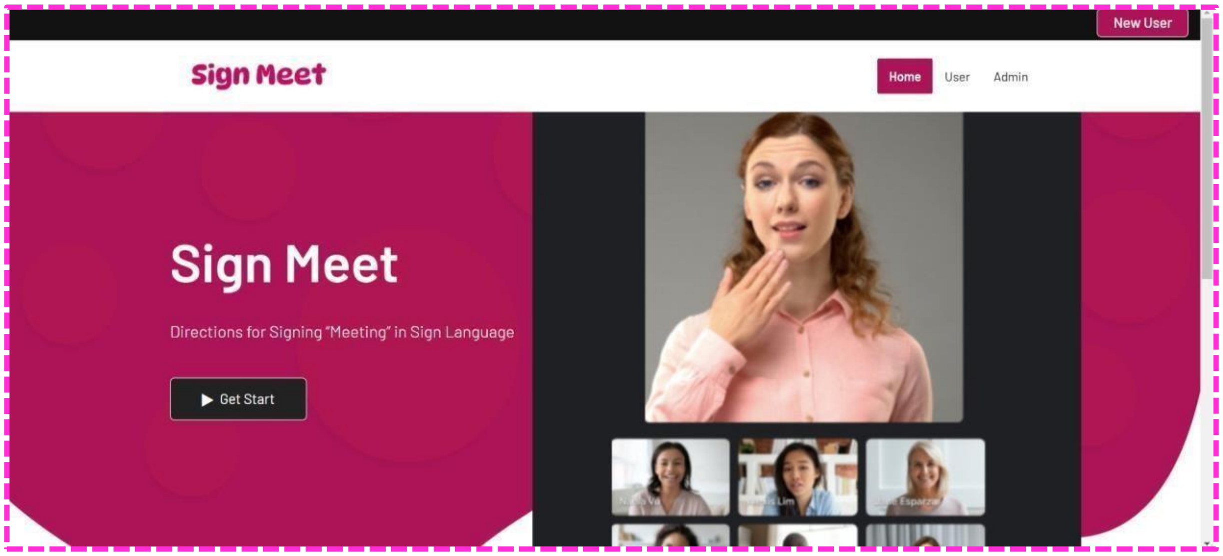
```
mycursor.execute("SELECT * FROM ga_gesture")
data = mycursor.fetchall()
dt=[]
dt2=[]
for dc in data:
dt.append(dc[1])
d1=dc[2].split(".")
dt2.append(d1[0])
#build model
DATA_DIR = "static/hand_gesture_data"
# Load data
data = []
labels = []
gesture_map = {}  # Label mapping
for idx, file in enumerate(os.listdir(DATA_DIR)):
gesture_name = file.split(".")[0]
gesture_map[idx] = gesture_name  # Store label mapping
file_path = os.path.join(DATA_DIR, file)
df = pd.read_csv(file_path, header=None)
data.extend(df.values)
labels.extend([idx] * len(df))
# Convert to numpy array
X = np.array(data)
y = np.array(labels)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Save model and gesture mapping
joblib.dump(model, "gesture_model.pkl")
joblib.dump(gesture_map,  "gesture_map.pkl")
print(f"Model trained with accuracy: {model.score(X_test, y_test) * 100:.2f}%")
mp_hands = mp.solutions.hands
mp_draw = mp.solutions.drawing_utils
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1,
min_detection_confidence=0.5)
# Create CSV file
DATA_DIR  =  "static/hand_gesture_data"
os.makedirs(DATA_DIR,  exist_ok=True)
class VideoCamera1(object):
def __init__(self):
self.video = cv2.VideoCapture(0)
def _del_(self):
self.video.release()
def get_frame(self):
_, frame = self.video.read()
frame  =  cv2.flip(frame,  1)
GESTURE_LABEL = glabel # Change for different gestures
FILE_NAME  =  os.path.join(DATA_DIR,  f"{gfile}")
with  open(FILE_NAME,  mode="a",  newline="")  as  f:
writer = csv.writer(f)
rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
result = hands.process(rgb_frame)
if result.multi_hand_landmarks:
for  hand_landmarks  in result.multi_hand_landmarks:
mp_draw.draw_landmarks(frame,  hand_landmarks,  mp_hands.HAND_CONNECTIONS)
# Extract hand landmark data
landmark_list  =  []
for lm in hand_landmarks.landmark:
landmark_list.extend([lm.x, lm.y, lm.z])
# Save to CSV
```

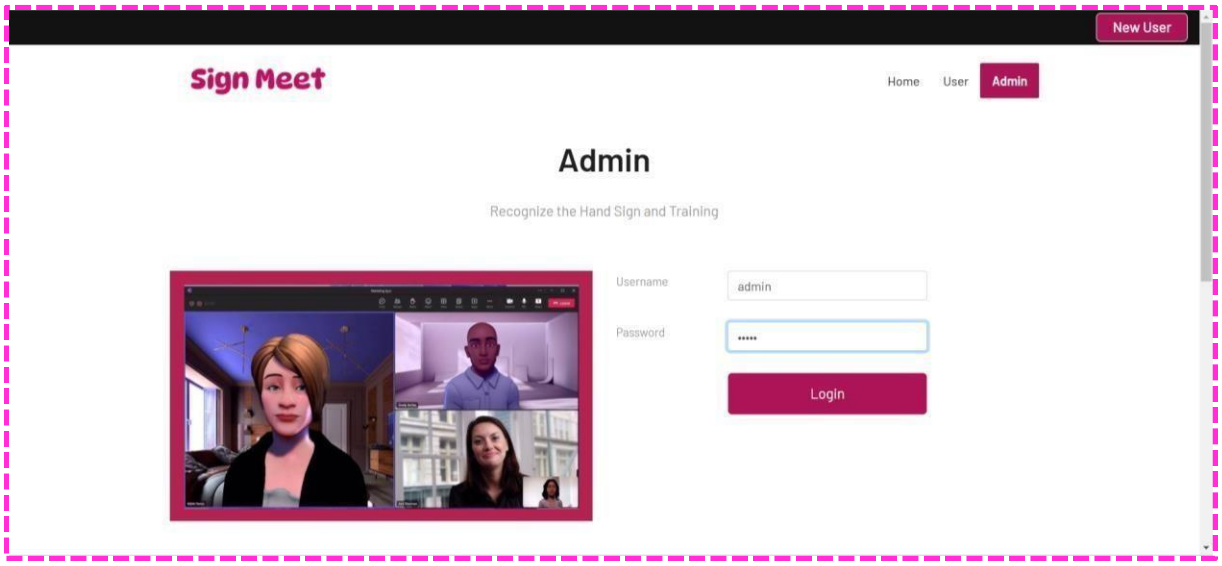
```
writer.writerow(landmark_list)
cv2.putText(frame, f"Recording: {GESTURE_LABEL}", (10, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
#TCN - Temporal Convolutional Network – Sign Language Recognition
def is_power_of_two(num: int):
return num != 0 and ((num & (num - 1)) == 0)
def adjust_dilations(dilations: list):
if all([is_power_of_two(i) for i in dilations]):
return dilations
else:
new_dilations = [2 ** i for i in dilations]
return new_dilations
def _init_(self,
dilation_rate: int,
nb_filters: int,
kernel_size: int,
padding: str,
activation: str = 'relu',
dropout_rate: float = 0,
kernel_initializer: str = 'he_normal',
use_batch_norm: bool = False,
use_layer_norm: bool = False,
use_weight_norm: bool = False,
**kwargs):
self.dilation_rate = dilation_rate
self.nb_filters = nb_filters
self.kernel_size = kernel_size
self.padding = padding
self.activation = activation
self.dropout_rate = dropout_rate
self.use_batch_norm = use_batch_norm
self.use_layer_norm = use_layer_norm
self.use_weight_norm = use_weight_norm
self.kernel_initializer = kernel_initializer
self.layers = []
self.shape_match_conv = None
self.res_output_shape = None
self.final_activation = None
super(ResidualBlock, self)._init_(**kwargs)
def tcn_full_summary(model: Model, expand_residual_blocks=True):
#import tensorflow as tf
# 2.6.0-rc1, 2.5.0...
versions = [int(v) for v in tf._version_.split('-')[0].split('.')]
if versions[0] <= 2 and versions[1] < 5:
layers = model._layers.copy()  # store existing layers
model._layers.clear()  # clear layers
for i in range(len(layers)):
if isinstance(layers[i], TCN):
for layer in layers[i]._layers:
if not isinstance(layer, ResidualBlock):
if not hasattr(layer, '_iter_'):
model._layers.append(layer)
else:
if expand_residual_blocks:
for lyr in layer._layers:
if not hasattr(lyr, '_iter_'):
model._layers.append(lyr)
else:
model._layers.append(layer)
else:
model._layers.append(layers[i])
model.summary() # print summary
```

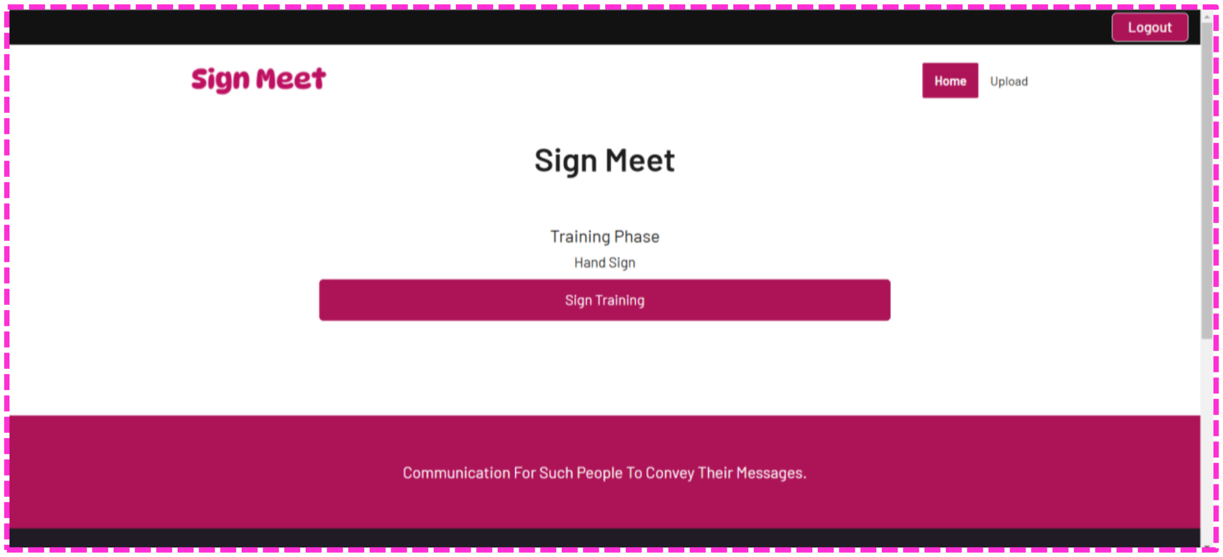
```
# restore original layers
model._layers.clear()
[model._layers.append(lyr) for lyr in layers]
def _build_layer(self, layer):
self.layers.append(layer)
self.layers[-1].build(self.res_output_shape)
self.res_output_shape = self.layers[-1].compute_output_shape(self.res_output_shape)
def build(self, input_shape):
with K.name_scope(self.name):  #  name scope used to make sure weights get unique names
self.layers = []
self.res_output_shape = input_shape
for k in range(2):  # dilated conv block.
name = 'conv1D_{}'.format(k)
with K.name_scope(name):  # name scope used to make sure weights get unique names
conv = Conv1D(
filters=self.nb_filters,
kernel_size=self.kernel_size,
dilation_rate=self.dilation_rate,
padding=self.padding,
name=name,
kernel_initializer=self.kernel_initializer
if self.use_weight_norm:
from tensorflow_addons.layers  import  WeightNormalization
# wrap it. WeightNormalization API is different than BatchNormalization or
LayerNormalization.
with K.name_scope('norm_{}'.format(k)):
conv = WeightNormalization(conv)
self._build_layer(conv)
with K.name_scope('norm_{}'.format(k)):
if self.use_batch_norm:
self._build_layer(BatchNormalization())
```

```
elif self.use_layer_norm:
self._build_layer(LayerNormalization())
elif self.use_weight_norm:
pass  # done above.
with K.name_scope('act_and_dropout_{}'.format(k)):
self._build_layer(Activation(self.activation,  name='Act_Conv1D_{}'.format(k)))
self._build_layer(SpatialDropout1D(rate=self.dropout_rate,  name='SDropout_{}'.format(k)))
if self.nb_filters != input_shape[-1]:
# 1x1 conv to match the shapes (channel dimension).
name = 'matching_conv1D'
with K.name_scope(name):
# make and build this layer separately because it directly uses input_shape.
# 1x1 conv.
self.shape_match_conv = Conv1D(
filters=self.nb_filters,
kernel_size=1,
padding='same',
name=name,
kernel_initializer=self.kernel_initializer
else:
name = 'matching_identity'
self.shape_match_conv = Lambda(lambda x: x, name=name)
with K.name_scope(name):
self.shape_match_conv.build(input_shape)
self.res_output_shape = self.shape_match_conv.compute_output_shape(input_shape)
self._build_layer(Activation(self.activation, name='Act_Conv_Blocks'))
self.final_activation = Activation(self.activation, name='Act_Res_Block')
self.final_activation.build(self.res_output_shape)   #  probably  isn't  necessary
# this is done to force Keras to add the layers in the list to self._layers
for layer in self.layers:
self.__setattr__(layer.name, layer)
self.__setattr__(self.final_activation.name,  self.final_activation)
super(ResidualBlock, self).build(input_shape)  # done to make sure self.built is set True
def call(self, inputs, training=None, **kwargs):
x1 = inputs
for layer in self.layers:
training_flag = 'training' in dict(inspect.signature(layer.call).parameters)
x1 = layer(x1, training=training) if training_flag else layer(x1)
x2 = self.shape_match_conv(inputs)
x1_x2 = self.final_activation(layers.add([x2, x1], name='Add_Res'))
return [x1_x2, x1]
def compute_output_shape(self, input_shape):
return [self.res_output_shape, self.res_output_shape]
```

## 9.2 SCREENSHOTS





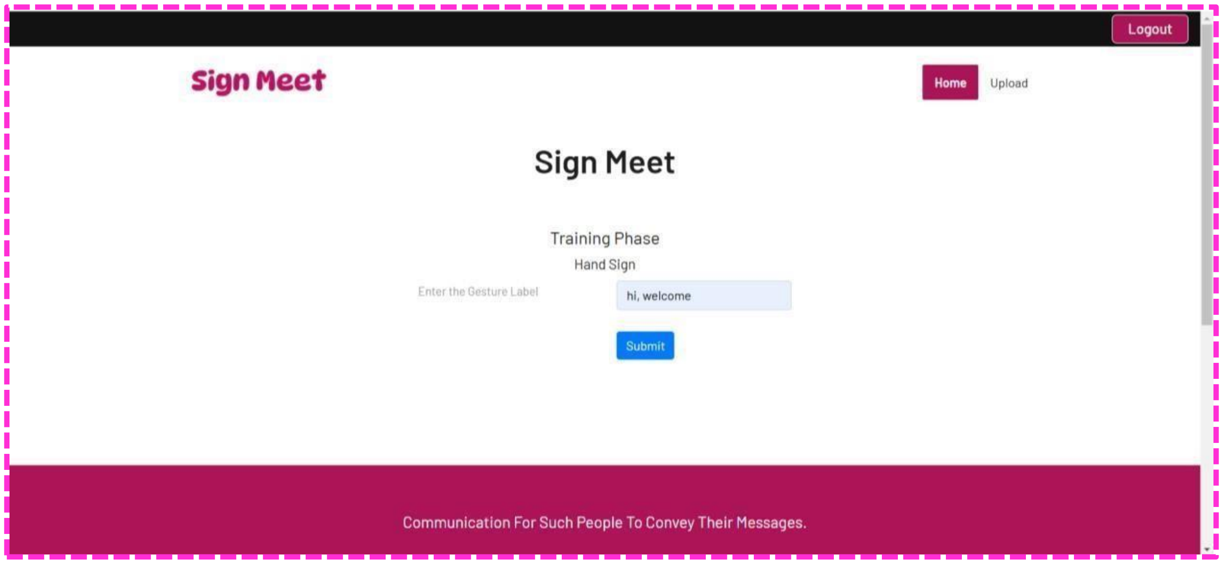


Sign Meet

Training Phase

Hand Sign

Communication For Such People To Convey Their Messages.



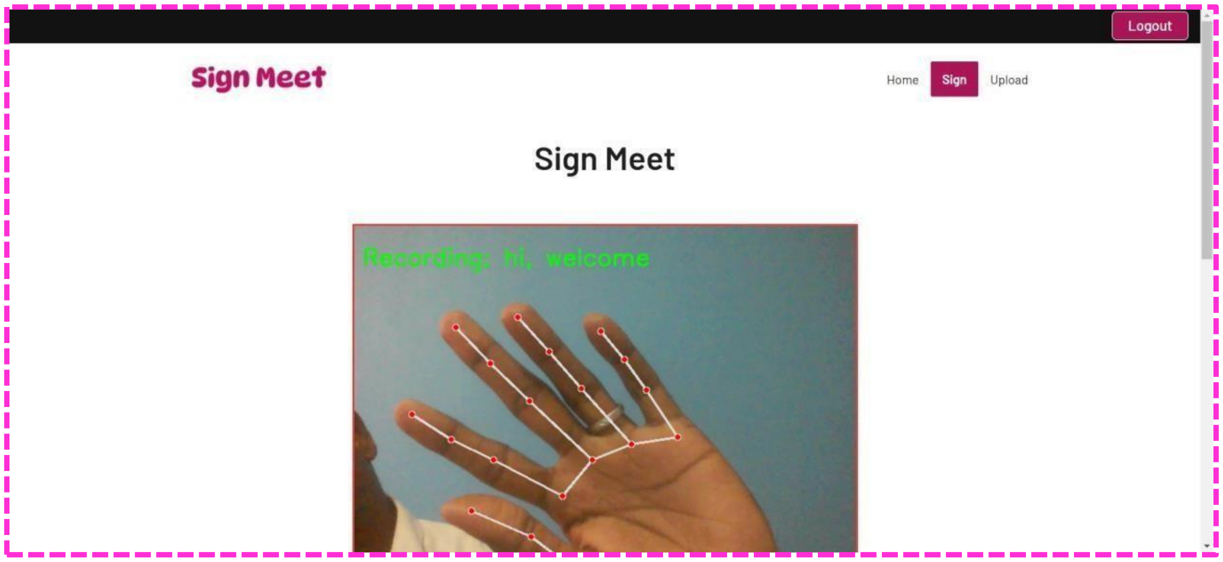
Sign Meet

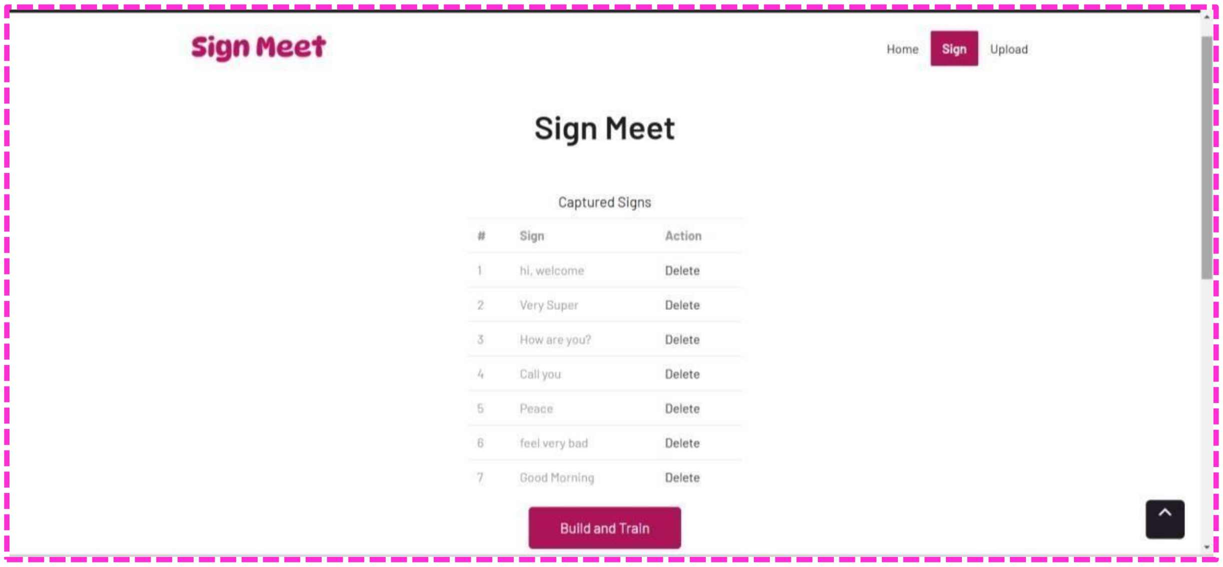
Training Phase

Hand Sign

Enter the Gesture Label: hi, welcome

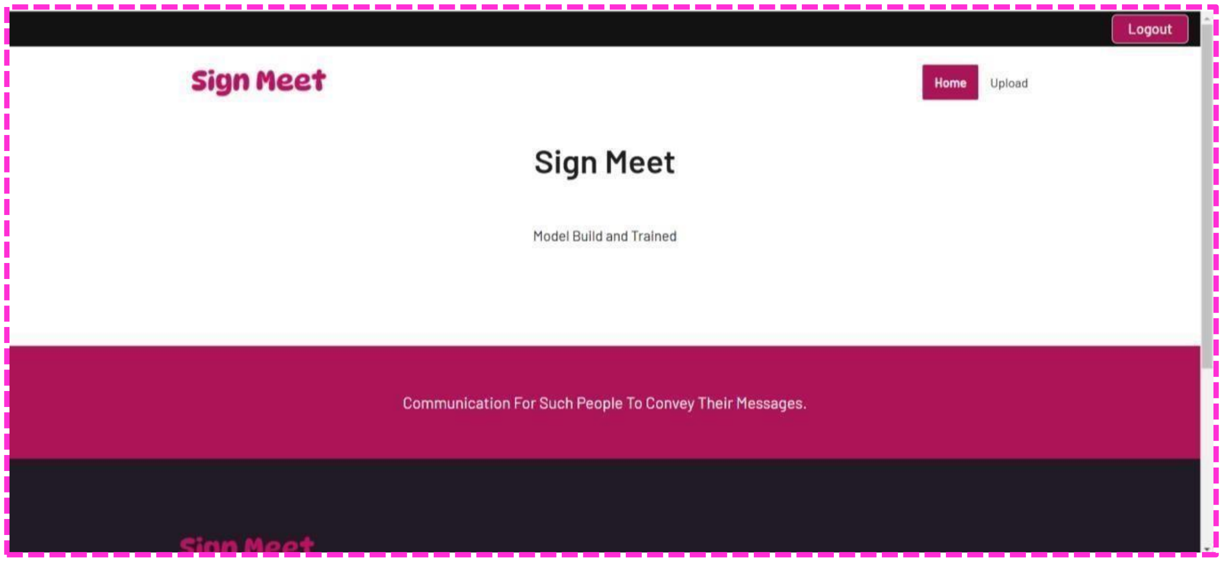
Communication For Such People To Convey Their Messages.

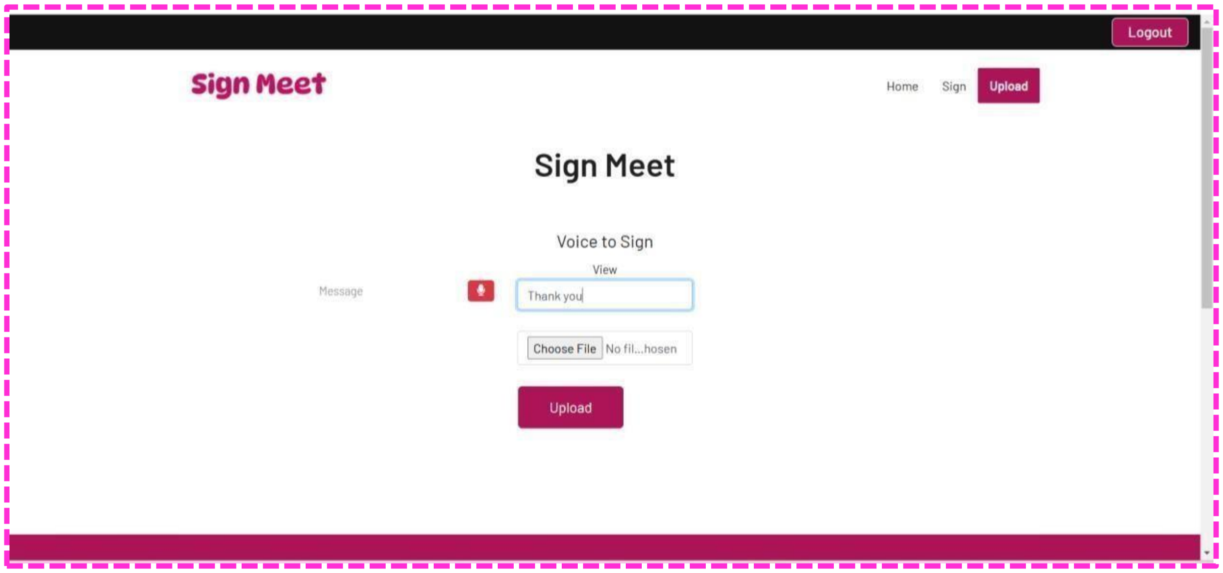


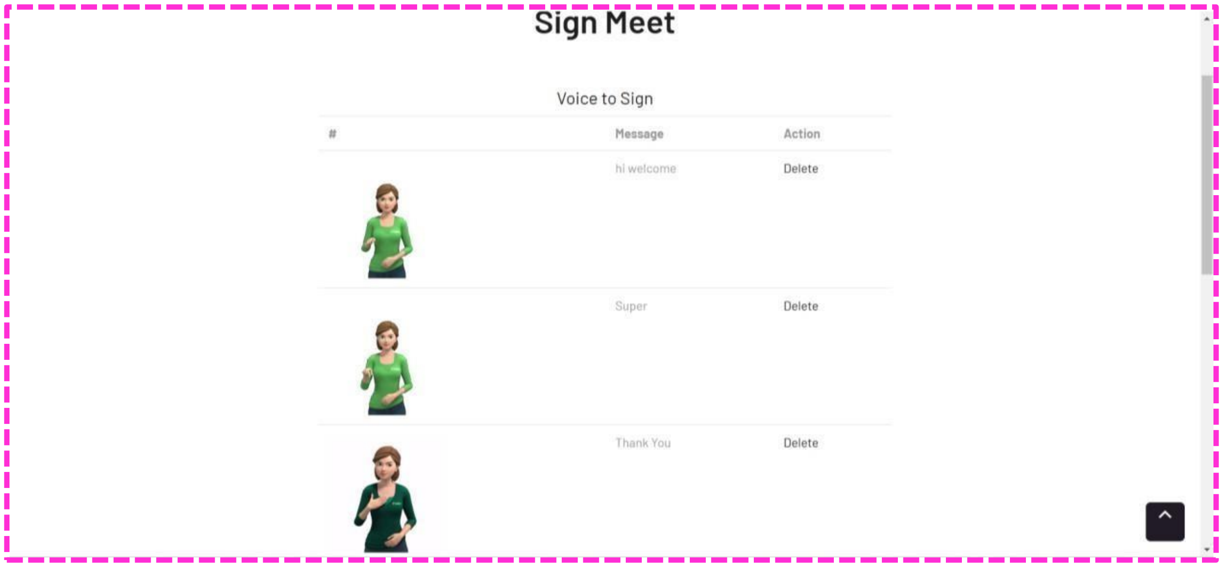


Captured Signs

| # | Sign | Action |
|---|---|---|
| 1 | hi, welcome | Delete |
| 2 | Very Super | Delete |
| 3 | How are you? | Delete |
| 4 | Call you | Delete |
| 5 | Peace | Delete |
| 6 | feel very bad | Delete |
| 7 | Good Morning | Delete |


Logout
Sign Meet
Home
Upload
Sign Meet
Model Build and Trained
Communication For Such People To Convey Their Messages.
Sign Meet

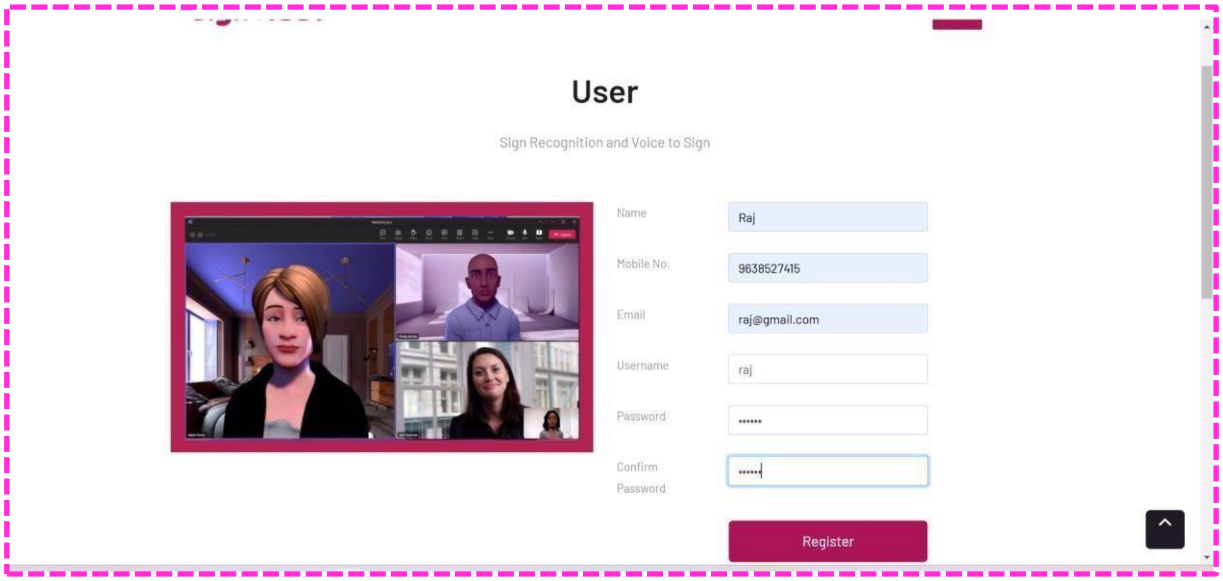

Logout
Sign Meet
Home
Sign
Upload
Sign Meet
Voice to Sign
View
Message
Thank you
Choose File No fil...hosen
Upload



Sign Meet

Voice to Sign

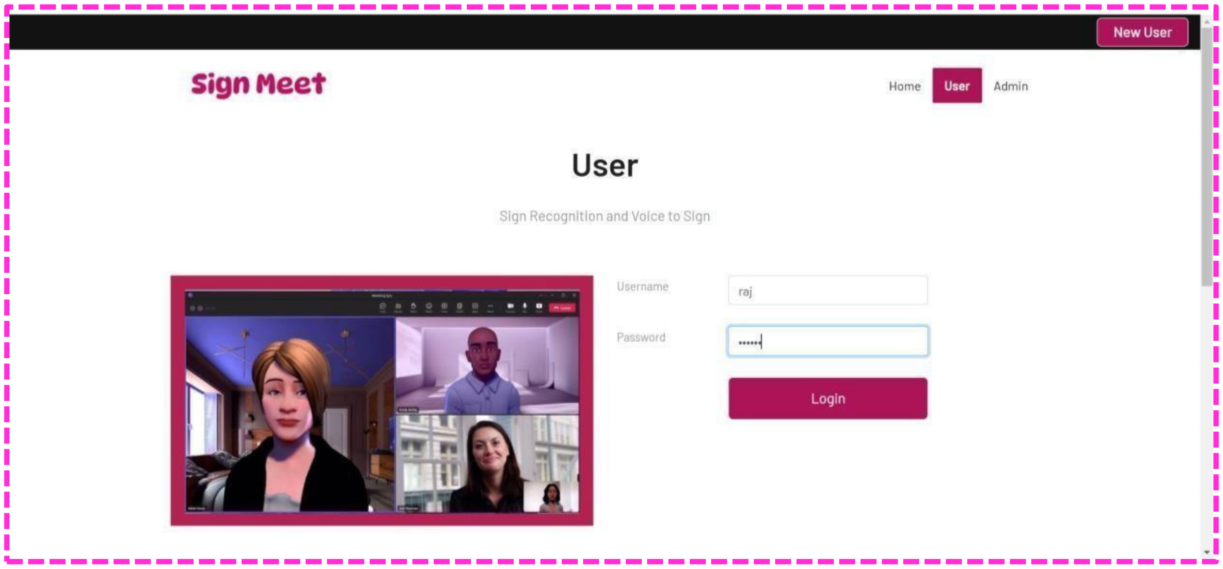
| # | Message | Action |
|---|---|---|
| | hi welcome | Delete |
| | Super | Delete |
| | Thank You | Delete |



User

Sign Recognition and Voice to Sign

| | |
|---|---|
| Name | Raj |
| Mobile No. | 9638527415 |
| Email | raj@gmail.com |
| Username | raj |
| Password | •••••• |
| Confirm Password | •••••• |


New User

Sign Meet

Home User Admin

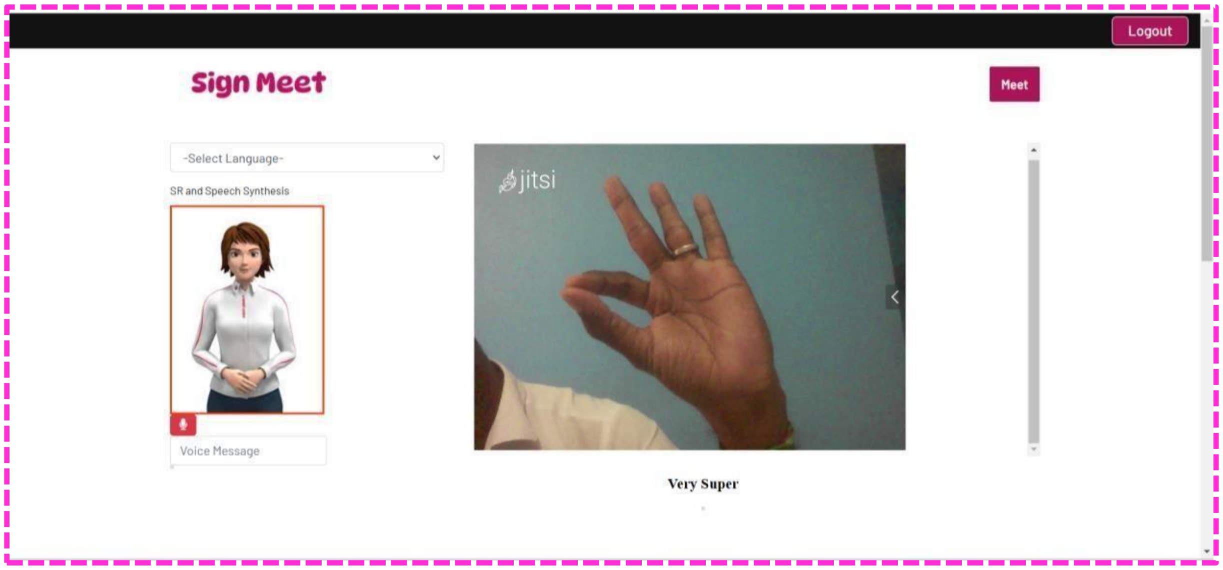
User

Sign Recognition and Voice to Sign

Username: raj

Password: ••••••

Login


Logout

Sign Meet

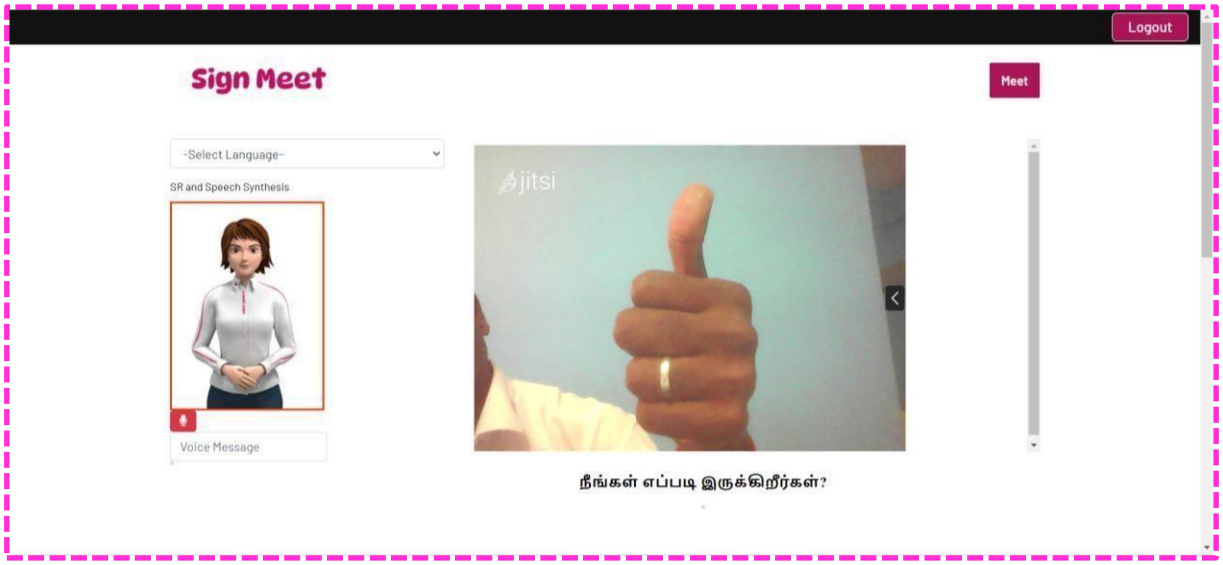
Meet

-Select Language-

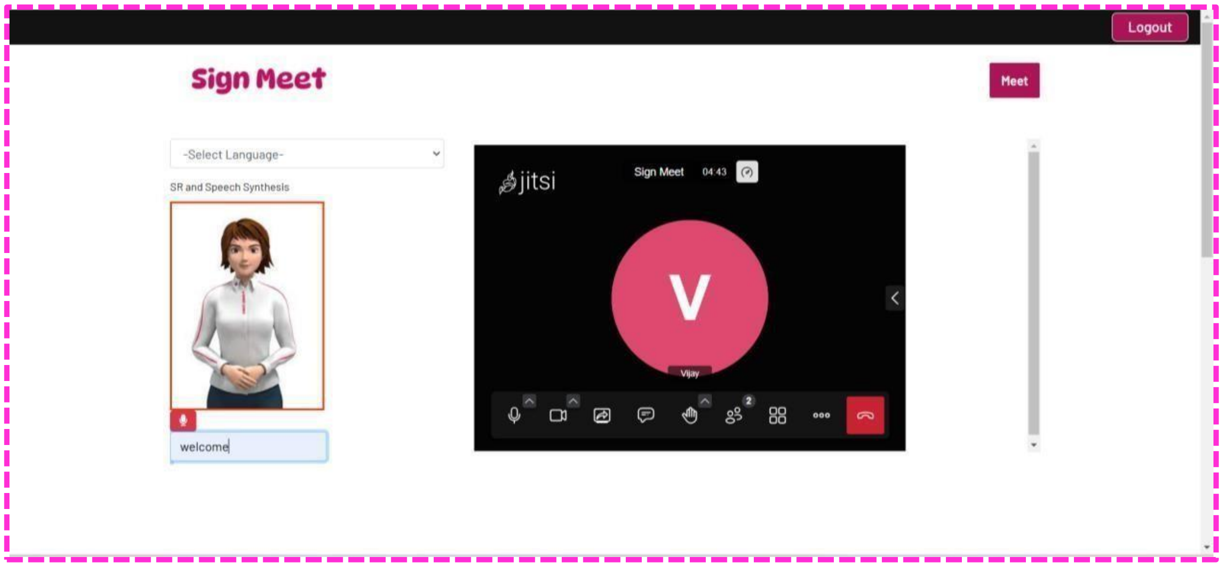
SR and Speech Synthesis

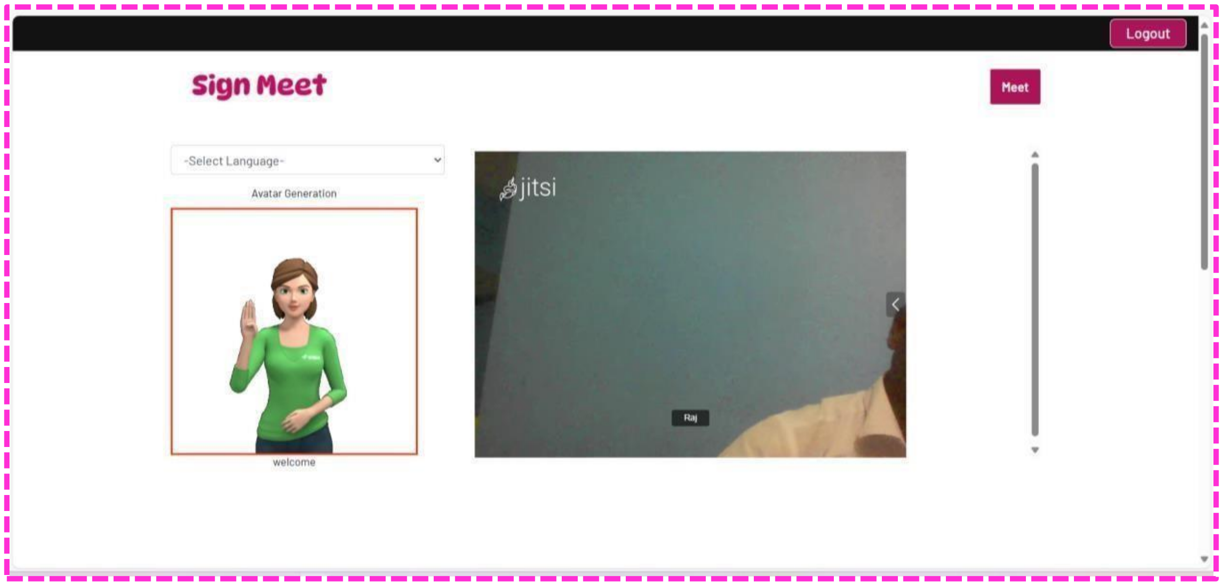
Voice Message

Very Super



நீங்கள் எப்படி இருக்கிறீர்கள்?




Logout
Sign Meet
Meet
-Select Language-
Avatar Generation
welcome
jitsi
Raj

## REFERENCES

1. T. Reddy Gadekallu, G. Srivastava, and M. Liyanage, “Hand gesture recognition based on a Harris hawks optimized convolution neural network,” Computers & Electrical Engineering, vol. 100, Article ID 107836, 2024
2. G. T. R. Chiranji Lal Chowdhary and B. D. Parameshachari, Computer Vision and Recognition Systems: Research Innovations and Trends, CRC Press, 2023.
3. M.M. Riaz and Z. Zhang, "Surface EMG Real-Time Chinese Language Recognition Using Artificial Neural Networks" in Intelligent Life System Modelling Image Processing and Analysis Communications in Computer and Information Science, Springer, vol. 1467, 2022.
4. G. Halvardsson, J. Peterson, C. Soto-Valero and B. Baudry, "Interpretation of Swedish sign language using convolutional neural networks and transfer learning", SN Computer Science, vol. 2, no. 3, pp. 1-15, 2021.
5. P. Likhar, N. K. Bhagat and R. G N, "Deep Learning Methods for Indian Sign Language Recognition", 2020 IEEE 10th International Conference on Consumer Electronics (ICCE- Berlin), pp. 1-6, 2020.
6. F. Li, K. Shirahama, M. A. Nisar, X. Huang and M. Grzegorzek, "Deep Transfer Learning for Time Series Data Based on Sensor Modality Classification", Sensors, vol. 31, no. 20(15), pp. 4271, Jul 2020.
7. J. J. Bird, A. Ekárt and D. R. Faria, "British sign language recognition via late fusion of computer vision and leap motion with transfer learning to american sign language", Sensors, vol. 20, no. 18, pp. 5151, 2020.
8. S. Sharma, R. Gupta and A. Kumar, "Trbaggboost: an ensemble-based transfer learning method applied to Indian Sign Language recognition", J Ambient Intell Human Comput Online First, 2020, [online] Available:

9. M. Oudah, A. Al-Naji and J. Chahl, "Hand Gesture Recognition Based on Computer Vision: A Review of Techniques", *J. Imaging*, vol. 6, no. 73, 2020.

10. Z. M. Shakeel, S. So, P. Lingga and J. P. Jeong, "MAST: Myo Armband Sign-Language Translator for Human Hand Activity Classification", IEEE International Conference on Information and Communication Technology Convergence, pp. 494-499, 2020.