

A Dual-Feature System: Gesture-to-word using CNN and Speech-to-text conversion

*A Project Report submitted
in partial fulfillment of the requirements
for the award of the degree of*

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

- | | |
|---------------------------------------------|----------------------------------------|
| 1. Shahanawaz Sulthana – 21B01A05G4 | 4. Vemavarapu Nag Sahithi – 21B01A05J0 |
| 2. Shaik Faheem Maharoor - 21B01A05G5 | 5. Vidiyala Rithu Sree – 21B01A05J1 |
| 3. Tadepalli Harika Naga Durga – 21B01A05H5 | |

Under the esteemed guidance of
Dr. P. Kiran Sree, M.Tech, Ph.D.
HOD & Professor



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)
(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202
2024 – 2025

SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)
(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



CERTIFICATE

*This is to certify that the project entitled "**A Dual-Feature System: Gesture-to-word using CNN and Speech-to-text conversion**", is being submitted by **Shahanawaz Sulthana, Shaik Faheem Maharoor, Tadepalli Harika Naga Durga, Vemavarapu Nag Sahithi, Vidiyala Rithu Sree**, bearing the **Regd. No. 21B01A05G4, 21B01A05G5, 21B01A05H5, 21B01A05J0, 21B01A05J1** in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology in Computer Science & Engineering**" is a record of Bonafide work carried out by her under my guidance and supervision during the academic year 2024–2025 and it has been found worthy of acceptance according to the requirements of the university.*

Internal Guide

Head of the Department

External Examiner

ACKNOWLEDGMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance has been a source of inspiration throughout the course of this seminar. We take this opportunity to express our gratitude to all those who have helped us in this seminar.

We wish to place our deep sense of gratitude to **Sri. K. V. Vishnu Raju, Chairman of SVES**, for his constant support on each and every progressive work of mine.

We wish to express our sincere thanks to **Dr. G. Srinivasa Rao, Principal of SVECW** for being a source of inspiration and constant encouragement.

We wish to express our sincere thanks to **Prof P. Venkata Rama Raju, Vice-Principal of SVECW** for being a source of inspirational and constant encouragement.

We wish to place our deep sense of gratitude to **Dr. P. Kiran Sree, Head of the Department of Computer Science & Engineering** for his valuable pieces of advice in completing this project successfully.

We are deeply indebted and sincere thanks to our PRC members **Dr. K. Ramachandra Rao, Dr. R. Anuj, Dr. N. Silpa** for their valuable advice in completing this project successfully.

We are deeply thankful to our project coordinator **Dr. P. R. Sudha Rani, Professor** for her indispensable guidance and unwavering support throughout our project's completion. Her expertise and dedication have been invaluable to our success.

Our deep sense of gratitude and sincere thanks to **our guide Dr. P. Kiran Sree, Head of the Department of Computer Science & Engineering** for his unflinching devotion and valuable suggestions throughout my project work.

Project Associates:

- 1.21B01A05G4 - Shahanawaz Sulthana
- 2.21B01A05G5 - Shaik Faheem Maharoor
- 3.21B01A05H5 - Tadepalli Harika Naga Durga
- 4.21B01A05J0 - Vemavarapu Nag Sahithi
- 5.21B01A05J1 - Vidiyala Rithu Sree

ABSTRACT

Effective communication is essential for social integration, yet individuals who are deaf or mute often face barriers in expressing themselves. Traditional communication methods, such as sign language, are not universally understood, limiting their ability to interact seamlessly. This project presents an innovative dual feature real-time system that bridges this gap by translating both hand gestures and spoken language into text, enhancing accessibility for individuals with speech and hearing impairments.

The proposed system employs **Convolutional Neural Networks (CNNs)** for accurate hand **gesture recognition**, enabling seamless translation of sign language into word. OpenCV is utilized to detect and track hand movements, ensuring real-time processing. Additionally, speech recognition is integrated using Python libraries such as **SpeechRecognition** and Google Web Speech API, allowing spoken words to be transcribed into text with high accuracy. By combining deep learning-based gesture recognition with speech-to-text conversion, this dual-feature system provides a robust, multimodal platform.

The utilization of CNNs, specifically the concept of transfer learning using **MobileNet architecture**, enhances the precision of gesture classification while maintaining computational efficiency, making it suitable for real-time applications. Additionally, the integration of speech processing technology enables accurate language interpretation. This **real-time approach** enhances user interaction by providing seamless and intuitive accessibility, reducing reliance on external assistance. The system's efficiency in recognizing diverse input modalities makes it a valuable tool for fostering inclusivity and accessibility.

In conclusion, this project demonstrates the potential of deep learning and computer vision to create an inclusive framework. By **integrating gesture and speech recognition**, it eliminates barriers for individuals with disabilities, promoting a more connected and accessible environment.

Table Of Contents

S. No	Topic	Page No
1.	Introduction	6 – 7
2.	System Analysis	8 – 10
	2.1 Existing System	
	2.2 Proposed System	
	2.3 Feasibility Study	
3.	System Requirements Specification	11 - 12
	3.1 Software Requirements	
	3.2 Hardware Requirements	
	3.3 Functional Requirements	
4.	System Design	13 - 20
	4.1 Introduction	
	4.2 Data flow diagrams (or) UML Diagrams	
5.	System Implementation	21 - 35
	5.1 Introduction	
	5.2 Project Modules	
	5.3 Algorithms	
	5.4 Screens	
6.	System Testing	36 – 57
	6.1 Introduction	
	6.2 Testing Methods	
	6.3 Test Cases	
7.	Conclusion	58 – 59
8.	Bibliography	60
9.	Appendix	61 – 68
	9.1 Appendix - A	
	9.2 Introduction to Python	
	9.3 Introduction to Convolutional Neural Network (CNN) Algorithm	
	9.4 Introduction to Introduction to MobileNet Architecture	
	9.5 Introduction to Introduction to HTML	
	9.6 Introduction to CSS	
	9.7 Introduction to JavaScript	
	9.8 Introduction to Flask Framework	
	9.9 Introduction to Transfer Learning Architecture	

1. INTRODUCTION

Hand gesture and speech recognition play a crucial role in assisting individuals with speech or hearing impairments, enabling them to communicate more effectively with their surroundings. Traditional communication methods, such as sign language, require knowledge and understanding from both the sender and the receiver, which may not always be feasible in diverse social and professional settings. This project aims to bridge this communication gap by developing a real-time, deep learning-based system that translates hand gestures and spoken language into text, thereby enhancing accessibility and inclusivity.

To achieve this, the system integrates computer vision techniques for gesture recognition and speech processing algorithms for voice-to-text conversion. The hand gesture recognition module leverages MobileNet, a lightweight convolutional neural network (CNN) known for its efficiency and high accuracy in resource-constrained environments. The model is trained using a dataset of American Sign Language (ASL) gestures, ensuring accurate classification of hand signs that correspond to specific letters, forming words and sentences. Transfer learning is employed to enhance model performance by utilizing pre-trained knowledge, enabling effective recognition even with a limited training dataset.

Alongside gesture recognition, the speech-to-text module utilizes Python libraries such as SpeechRecognition and the Google Web Speech API to transcribe spoken language into text. This dual-input approach allows users to communicate using either gestures or speech, making the system more versatile and user-friendly. Additionally, OpenCV framework is employed for efficient real-time hand tracking and feature extraction, ensuring smooth processing and accurate gesture detection.

One of the key challenges in gesture and speech recognition is adapting to diverse environments and user variations. To address this, the system incorporates data augmentation techniques to improve model generalization across different lighting conditions, backgrounds, and hand positions. Furthermore, model performance is evaluated using key metrics such as accuracy, precision, recall, and response time, ensuring robustness and reliability in real-world scenarios.

The integration of deep learning for gesture recognition and speech processing offers a practical, multimodal communication tool that significantly enhances accessibility for individuals with speech or hearing impairments. By combining efficiency, accuracy, and real-time processing, this project aims to contribute towards a more inclusive society where

communication barriers are minimized. The system's adaptability also makes it suitable for various applications, including education, assistive technology, human-computer interaction, and smart home automation.

Through this approach, the project not only addresses accessibility challenges but also demonstrates the potential of artificial intelligence in transforming assistive technologies. Future enhancements may include expanding the dataset for more comprehensive gesture recognition, integrating natural language processing (NLP) for contextual understanding, and deploying the system on edge devices for enhanced portability and real-world usability.

2. System Analysis

2.1 Existing System:

Existing systems designed to assist individuals with speech and hearing impairments primarily focus on either sign language translation or speech-to-text conversion. However, these systems have limitations that reduce their effectiveness and accessibility.

Sign language translation systems typically rely on wearable sensors to capture and analyze hand gestures, converting them into text or speech. While these systems help bridge the communication gap for individuals using sign language, they often suffer from real-time processing limitations, making interactions slower and less efficient. Furthermore, wearable sensor-based solutions can be expensive, requiring specialized gloves or motion sensors, which restrict their accessibility to a broader audience.

Additionally, these systems focus solely on gesture-based communication without incorporating speech recognition, which prevents them from supporting individuals with both speech and hearing impairments simultaneously.

On the other hand, speech-to-text systems, such as Google's Speech-to-Text API, provide real-time transcription of spoken words, helping individuals with hearing impairments understand conversations. However, these systems do not assist those who cannot speak, limiting their usability in a multi modal scenario. Another major limitation is that speech recognition systems struggle in noisy environments, making them less reliable in real-world applications where background noise can interfere with accuracy.

2.2 Proposed System:

The proposed system, "A Dual-Feature System: Gesture-to-Word using CNN and Speech-to-Text Conversion," is designed to bridge gaps for individuals with speech and hearing impairments by integrating real-time gesture recognition and speech-to-text conversion into a single, accessible platform.

For gesture recognition, the system utilizes OpenCV and Convolutional Neural Networks (CNNs), specifically employing the MobileNet architecture. MobileNet is chosen for its lightweight design and computational efficiency, enabling real-time processing of hand gestures with minimal resource consumption. This ensures smooth and accurate classification, making the system suitable for use on various devices, including mobile platforms.

For speech-to-text conversion, the system incorporates SpeechRecognition and the Google Web Speech API to transcribe spoken language into text. This feature enables individuals with hearing impairments to understand spoken communication in real time, ensuring effective interaction in different environments.

The system features a user-friendly interface, designed with accessibility in mind. It includes two clearly labeled buttons, one for gesture processing and another for speech processing allowing users to interact with the platform effortlessly. This ensures an intuitive and seamless experience, even for individuals with limited technical knowledge.

A key advantage of this system is its dual-feature integration, enabling users to switch between gesture-based and speech-based interaction as needed. The real-time processing capability ensures minimal latency, providing instant feedback for both gesture and speech recognition.

By combining deep learning-powered gesture recognition with efficient speech-to-text conversion, the proposed system offers a comprehensive assistive tool that enhances accessibility and promotes social inclusion.

2.3 Feasibility Study:

A feasibility study, sometimes called a feasibility analysis or feasibility report, is a way to evaluate whether or not a project plan could be successful. A feasibility study evaluates the practicality of your project plan to judge whether or not you're able to move forward with the project.

The different feasibilities that should be analyzed are:

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

Technical Feasibility:

Technical feasibility evaluates whether the proposed system can be effectively developed and implemented using the available technology and resources. The system leverages widely supported tools such as Python, TensorFlow, Keras, OpenCV, and SpeechRecognition, ensuring broad compatibility with modern computing platforms and seamless integration. Its scalable architecture supports future enhancements like additional languages, gestures, and improved accuracy, with cloud-based solutions like Google Web Speech API enabling flexible handling of large-scale data. Moreover, the use of open-source development tools backed by strong community support and comprehensive documentation ensures ease of development, implementation, and troubleshooting.

Economic Feasibility:

Economic feasibility evaluates whether the system is cost-effective to develop and maintain in the long run by considering factors such as development costs, operational costs, and return on investment (ROI). Since the project relies on open-source tools and libraries like Python, TensorFlow, OpenCV, and Google APIs, there are no licensing costs, and the hardware requirements are minimal, allowing it to run on a standard computer with a webcam and microphone. Operational costs are also low, as the speech-to-text module uses Google Web Speech API, which offers a free tier, while the gesture recognition module performs local processing, minimizing cloud dependency and reducing long-term expenses. The system delivers strong ROI by bridging communication gaps for the deaf and/or dumb community, with applications in education, healthcare, and customer service, thereby enhancing inclusivity and accessibility.

Operational Feasibility:

Operational feasibility determines whether the system can be successfully integrated and utilized by end users by evaluating factors such as user-friendliness, training and support, and system reliability. The system features an intuitive interface that allows users to interact through a webcam for gesture input and real-time speech transcription, ensuring ease of use even for individuals with minimal technical expertise. It requires minimal training due to comprehensive documentation and user guides will be provided to support smooth onboarding. Additionally, the use of a CNN-based gesture recognition model and the Google Web Speech API ensures high accuracy in real-world conditions, with continuous training and updates planned to maintain and enhance system reliability.

3. System Requirements Specification

3.1 Software Requirements

Software requirements for this project describe the essential tools, technologies, and platforms needed for its successful development and deployment.

Software requirements for this project are:

- **Operating System** - Windows 10/11 (64-bit)
- **Programming Language** - Python 3.10.7
- **Frameworks & Libraries** - TensorFlow/Keras, SpeechRecognition, Google Web Speech API, OpenCV(cv2), Flask
- **Front-End Technologies** - HTML5, CSS3, JavaScript, Bootstrap
- **Database** - MySQL in XAMPP
- **Web Server** - XAMPP with Apache server
- **Development Tools** - Visual Studio Code
- **Version Control** - GitHub

3.2 Hardware Requirements:

Hardware requires software to run correctly. Without the correct hardware, your software may not run efficiently or at all. It is important to consider both when making decisions about your IT systems, as this can affect the way you work, your productivity and your business bottom line.

Hardware requirements are:

- **Processor** - I5 and above
- **RAM** - 8 GB
- **Hard Disk** - 256 GB
- **Internet:** Stable connection for data access and updates
- **Peripherals:** Basic monitor, keyboard, and mouse for interaction

3.3 Functional Requirements:

Functional requirements are the desired operations of a program, or system as defined in software development and systems engineering. The systems in systems engineering can be either software electronic hardware or a combination of software-driven electronics. These are represented or stated in the form of input to be given to the system, the operation performed, and the output expected.

The functional requirements for this application are:

1. **Gesture Recognition in Real Time:** The system should detect and recognize hand gestures of alphabets instantly.
2. **Convert Recognized Gestures into Words:** Recognized hand gestures should be converted into corresponding alphabets forming word/words.
3. **Speech Recognition in Real Time:** The system should accurately recognize spoken words.
4. **Convert Recognized Speech into Text:** Recognized speech should be transcribed into text.
5. **Seamless Integration of Gesture and Speech Functionalities:** The system should effectively combine gesture and speech recognition capabilities.
6. **User-Friendly Interface:** The application should have an intuitive interface for users to view converted text.

3.4 Non-functional Requirements:

These requirements are defined as the quality constraints that the system must satisfy to complete the project. They are also called non-behavioral requirements or quality requirements/attributes. Nonfunctional requirements are more abstract.

The non-functional requirements of this project are:

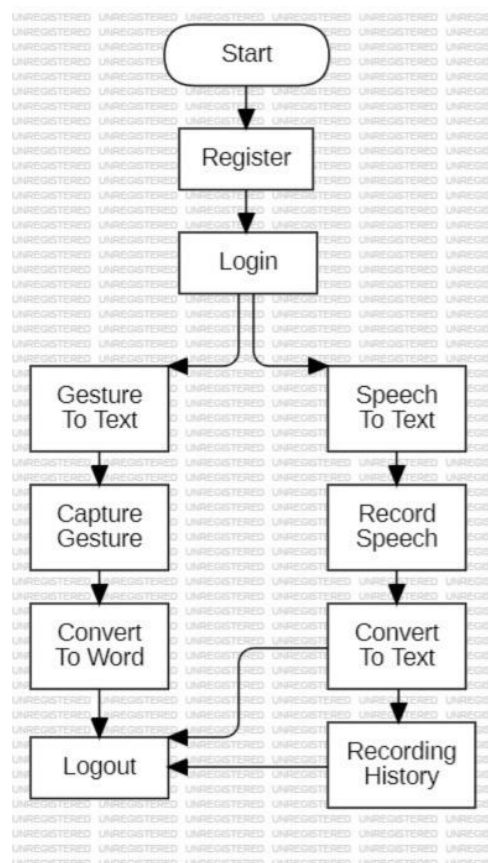
1. **Performance:** The system should demonstrate high performance with minimal latency in both gesture and speech processing requests.
2. **Accuracy:** The classification accuracy of the system should be high, ensuring reliable identification of different gestures and spoken words.
3. **Security:** The system should ensure the security and privacy of user data and voice recordings uploaded for processing.
4. **Accessibility:** The user interface should be accessible to users with disabilities, following relevant accessibility guidelines and standards.
5. **User Experience:** The interface should be user-friendly, easy to navigate, and provide a seamless experience.
6. **Personalization:** Users can access and manage their previous speech-to-text conversions, allowing them to review or reuse past interactions.

4. System Design

4.1 Introduction:

The software design for gesture-to-word and speech-to-text conversion employs a systematic approach to integrate deep learning techniques and speech processing techniques seamlessly. The Convolutional Neural Network (CNN) model, specifically MobileNet, serves as the backbone for recognizing hand gestures, while speech recognition libraries facilitate accurate speech-to-text conversion. The software encompasses modules for data preprocessing, model training, and real-time inference to ensure efficient operation.

For gesture recognition, the data preprocessing module involves image augmentation to improve the model's generalization across different hand shapes, lighting conditions, and backgrounds. The CNN model is fine-tuned using transfer learning with MobileNet to expedite training and enhance classification accuracy. In parallel, the speech-to-text module employs Google Web Speech API to transcribe spoken language into text with high precision.



The real-time inference module utilizes the trained MobileNet model to classify hand gestures from live camera input, providing instantaneous results. The speech recognition module

ensures seamless transcription, enabling multimodal accessibility. A user-friendly interface is designed to display recognized text output from both gesture and speech inputs. A robust backend, potentially implemented using Flask, handles user interactions, processes real-time inputs, and manages data for performance tracking.

Overall, the software design seamlessly integrates MobileNet-based deep learning techniques and speech recognition, ensuring an efficient, real-time system for assisting individuals with speech and hearing impairments. Its lightweight architecture facilitates quick and accurate processing, making it suitable for deployment on mobile platforms or edge devices.

4.2 Data flow diagrams (or) UML Diagrams:

A use-case diagram in the Unified Modeling Language (UML) is a behavioral diagram derived from use-case analysis, designed to provide a graphical overview of a system's functionality.

The Primary goals in the design of the UML are as follows:

- Provide users with a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modelling language.

UML encompasses various diagram types, including class diagrams for illustrating classes and relationships, sequence diagrams for depicting interactions over time, and use case diagrams for outlining system functionalities. Widely adopted in the software development lifecycle, UML diagrams offer a visual means to document, analyze, and design systems, fostering a common understanding among developers, designers, and project stakeholders.

1. Use Case Diagram:

The following are the use cases and their relationships in this project.

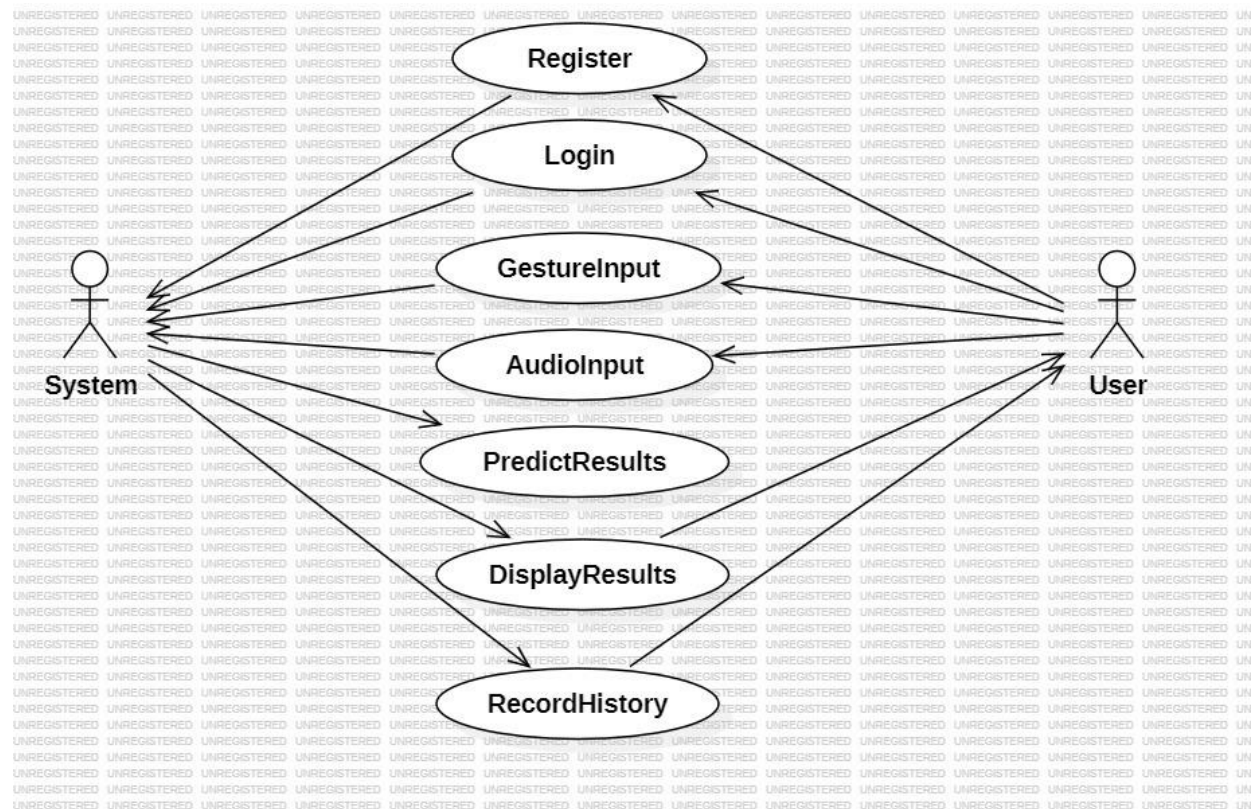
- **System:**

Represents the entire gesture and audio recognition system. This encompasses the Flask-based web application that facilitates user authentication, gesture and audio input processing, prediction generation, result display, and recording history management. The System interacts with all major components involved in these processes, serving as the central hub that connects the user with the application's core functionalities.

- **User:**

Represents the end-users interacting with the system. The User initiates various use cases, including Register and Login for authentication, GestureInput and AudioInput for providing

data, DisplayResults to view outcomes, and RecordHistory to manage past interactions. The User is the primary actor driving the system's functionalities through the web interface.



- Register:**
 Describes the user action of creating a new account within the system. This use case initiates the process of collecting user details (e.g., name, email, password) and storing them in the MySQL database via the /register route. It connects to the Login use case, enabling new users to authenticate after registration.
- Login:**
 Encompasses the process of user authentication. Users provide their email and password through the /login route, which verifies credentials against the users table in the database. Upon successful login, the session is established, connecting to subsequent use cases like GestureInput, AudioInput, PredictResults, DisplayResults, and RecordHistory, allowing authenticated access to the system's features.
- GestureInput:**
 Represents the user action of providing gesture data, likely captured via the webcam through the /open_webcam route, which triggers the execution of live.py. This use case initiates the PredictResults process by sending gesture data to the system for real-time analysis using machine learning models (e.g., TensorFlow/Keras with cvzone). It connects the user to the gesture recognition pipeline.

- **AudioInput:**

Describes the user action of uploading an audio file for transcription, handled via the /mic route. This use case involves processing the uploaded file using the SpeechRecognition library and storing the transcript in the recordings table of the database. It connects to PredictResults for potential audio-based predictions and RecordHistory for storing the input data.

- **PredictResults:**

Encompasses the core functionality of generating predictions based on GestureInput and AudioInput. This use case relies on trained machine learning models (e.g., loaded via Keras or TensorFlow) to analyze gesture or audio data in real-time, producing classification or transcription results. It connects to DisplayResults to forward the outcomes for user visualization.

- **DisplayResults:**

Allows users to visualize the outcomes of the PredictResults use case. This process renders the prediction or transcription results (e.g., gesture labels or audio transcripts) in templates like prediction.html, providing feedback to the user. It connects to PredictResults to display the processed data accurately.

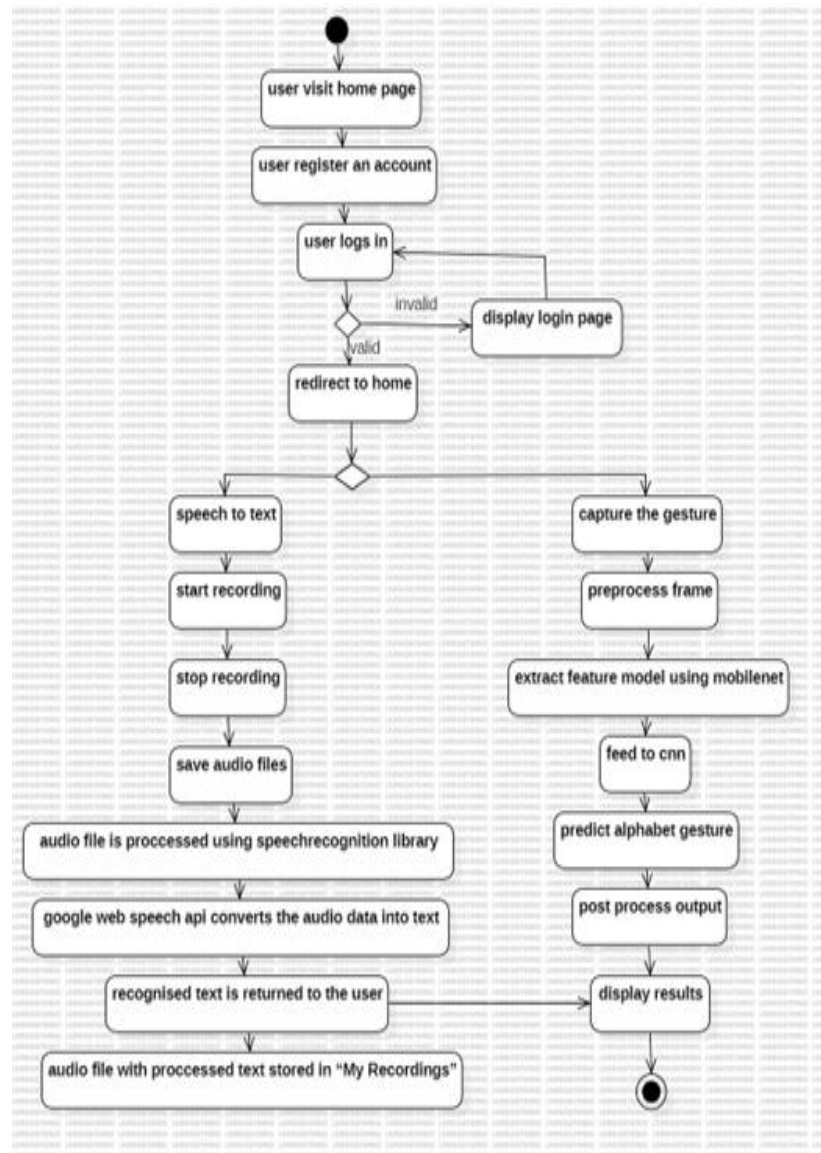
- **RecordHistory:**

Represents the functionality of viewing and managing the history of audio recordings and potentially gesture inputs. This use case retrieves data from the recordings table via the /my_recordings route and displays it in recordings.html, allowing users to who can then view the categorized outcomes, thus completing the activity loop

2.Activity Diagram:

Activity Diagrams are behavioral diagrams used to illustrate the flow of control in a system, particularly the steps involved in executing a use case. They provide a visual representation of the dynamic aspects of a system, focusing on the flow and sequence of activities, which helps stakeholders understand the order in which tasks are performed and how control transitions between them.

The activity diagram for our project illustrates the sequential flow of activities involved in user registration, login, speech-to-text conversion, and gesture recognition processes on the website. The process begins with the actor (user) initiating the "user visit home page" activity. This leads to the "user register an account" activity, where the user provides necessary details to create an account. Upon successful registration, the user proceeds to the "user logs in" activity



The login process involves the user entering their credentials, which the system validates. If the credentials are invalid, the system displays the login page again, allowing the user to retry. Upon successful validation, the user is redirected to the home page. From there, the diagram branches into two parallel processes: "speech to text" and "capture the gesture."

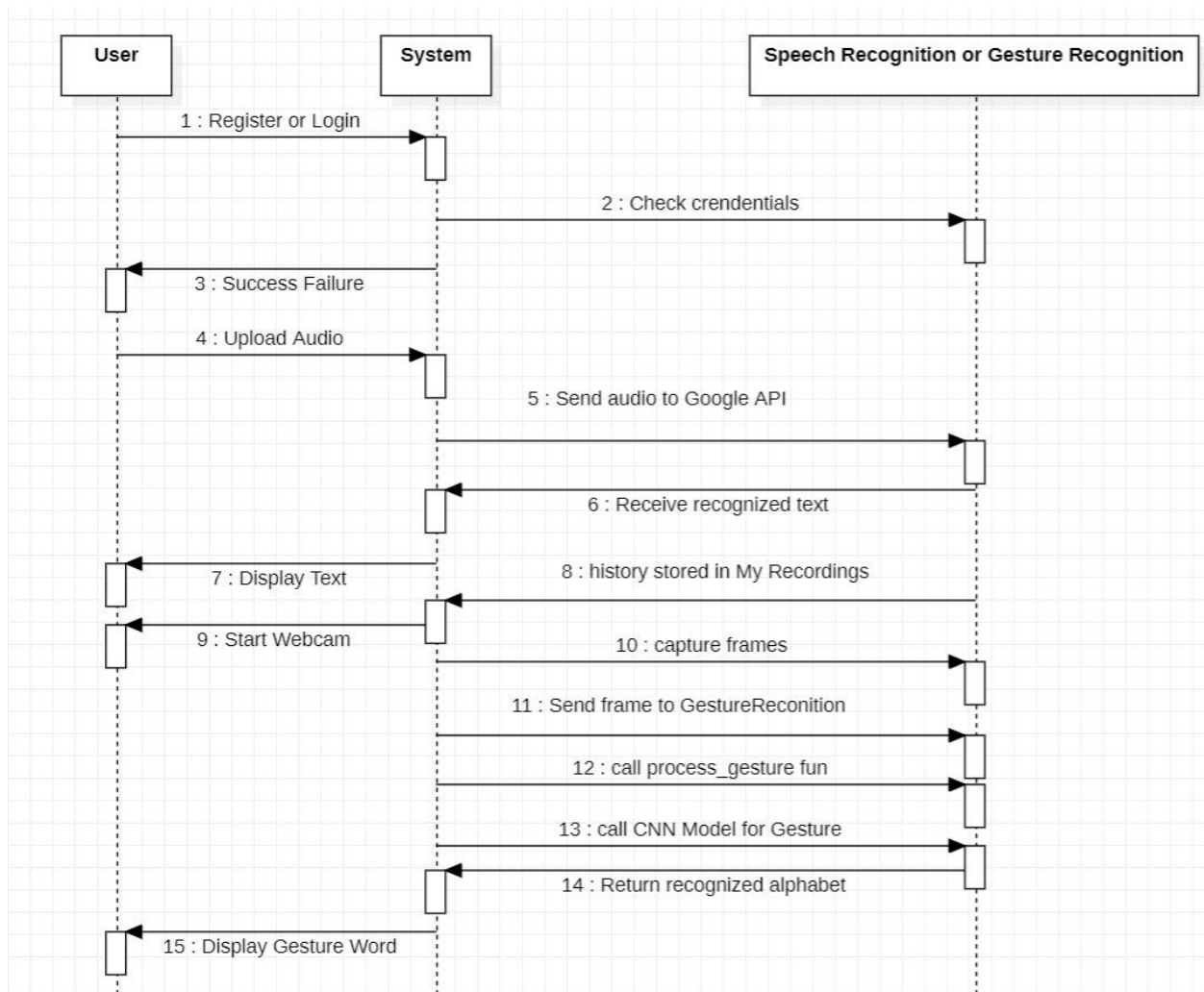
The "speech to text" process starts with the "start recording" activity, followed by "stop recording" and "save audio files." The audio file is then processed using a speech recognition library. Specifically, the Google Web Speech API converts the audio data into text. The recognized text is returned to the user, and the audio file with the processed text is stored in "My Recordings."

Simultaneously, the "capture the gesture" process begins with the "preprocess frame" activity, followed by "extract feature model using MobileNet." The extracted features are then fed to a

CNN (Convolutional Neural Network) to "predict alphabet gesture." The output is post-processed and the results are displayed to the user.

3.Sequence Diagram:

Sequence Diagrams depict the interaction between objects in a system in a sequential order, illustrating the order in which these interactions occur. They are also referred to as event diagrams or event scenarios. Sequence Diagrams describe how and in what order the objects in a system function. They are widely used by businessmen and software developers to document and understand requirements for new and existing systems. They provide a clear and concise representation of the sequence of events that occur during the execution of a use case or scenario.



The sequence diagram for our project's user registration, login, speech-to-text conversion, and gesture recognition processes illustrates the chronological order of interactions among the key actors: User, System, and Speech Recognition or Gesture Recognition modules.

The sequence begins with the User triggering the "Register or Login" message, prompting the System to collect and validate the user's credentials. The System performs a "Check credentials"

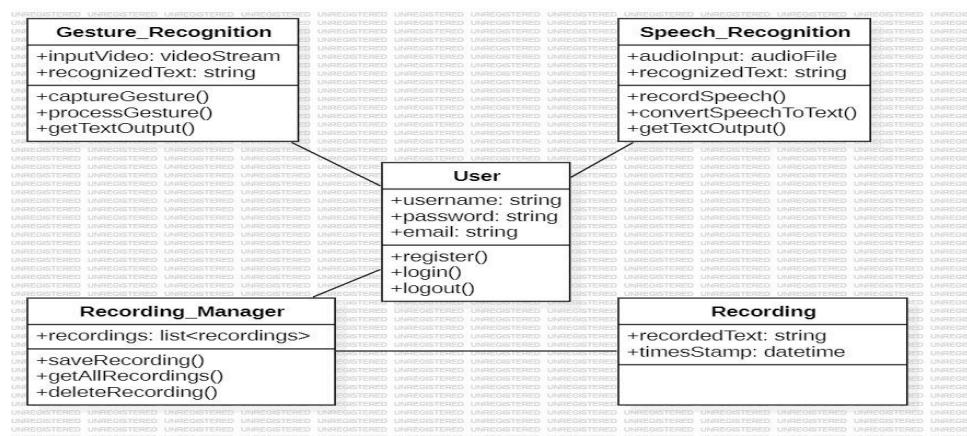
action internally to verify the input. If successful, the System sends a "Success Failure" message back to the User, confirming whether the registration or login attempt was successful. Upon successful authentication, the User gains access to the system's functionalities.

Following a successful login, the User initiates the "Upload Audio" message to start the speech-to-text process. The System communicates with the Speech Recognition module by sending a "Send audio to Google API" message. The Speech Recognition module processes the audio and returns a "Receive recognized text" message to the System. The System then stores the audio and its corresponding text with the "history stored in My Recordings" action and sends a "Display Text" message to the User, presenting the recognized text for review.

The User initiates the gesture recognition process by sending a "Start Webcam" message to the System, which begins capturing frames from the webcam. These frames are sent to the Gesture Recognition module, where they are processed using a CNN model to predict the gesture. The recognized alphabet is returned to the System, which then displays it to the User as a word or letter. The sequence diagram highlights the flow of messages between the User, System, and Recognition modules, showcasing the processing of gesture and speech inputs and the use of Google Web Speech API and CNN (MobileNet) for accurate recognition and user feedback.

4. Class Diagram:

Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems. Class diagrams provide a high-level overview of a system's design, helping to communicate and document the structure of the software. They are a fundamental tool in object oriented design and play a crucial role in the software development lifecycle.



The class diagram for our project outlines the core components involved in user management, speech and gesture recognition, and recording management, along with their interactions.

The User class includes attributes like username, email, and a private password. It provides methods such as register(), login(), and logout() to manage user sessions. Each user is associated one-to-one with both Speech_Recognition and Gesture_Recognition components.

The Speech_Recognition class handles audio input through attributes like audioInput and recognizedText. It includes methods such as recordSpeech(), convertSpeechToText(), and getTextOutput() for converting audio to text using APIs like Google Web Speech.

The Gesture_Recognition class works similarly but for video input. It has attributes inputVideo and recognizedText, and methods like captureGesture(), processGesture(), and getTextOutput() to recognize hand gestures using models like MobileNet.

The Recording class stores recognized text (recordedText) and its timeStamp. It is associated with Speech_Recognition and managed by Recording_Manager in a one-to-many relationship. The Recording_Manager class maintains a list of recordings and provides methods like saveRecording(), getAllRecordings(), and deleteRecording().

This structure ensures smooth integration of user interaction, recognition processes, and data management within the system.

5.System Implementation

5.1 Introduction:

System implementation is a crucial phase in the development lifecycle, where the conceptual designs and requirements are transformed into a working software or hardware system. This phase ensures the developed solution aligns with the intended objectives and is ready for operational use. It involves multiple coordinated steps to bridge the gap between development and deployment, requiring attention to both technical and user-centered considerations.

The initial step is setting up the necessary infrastructure, including hardware, software, and networking components, to support system deployment. This may involve procuring servers, configuring databases, and preparing the environment for installation. The infrastructure must be rigorously tested to ensure it meets performance, scalability, and reliability standards. Once ready, the system components are installed and configured based on design specifications. Depending on whether the system is standalone, distributed, or cloud-based, the deployment process is tailored to suit the architecture while minimizing disruptions to existing operations.

Following deployment, comprehensive testing is conducted to validate the system's functionality and performance. This includes unit testing, integration testing, system testing, and user acceptance testing. These tests simulate real-world conditions and stress scenarios to evaluate the system's reliability and ability to handle errors. Any defects discovered are addressed promptly to ensure a stable and high-quality implementation.

Beyond the technical setup, successful implementation also requires preparing end-users and stakeholders for adoption. This includes providing training, creating user manuals and tutorials, and offering support to ensure users can interact with the system confidently. Effective communication and change management are key to managing expectations, addressing concerns, and ensuring a smooth transition. Overall, system implementation demands thorough planning, collaboration, and execution to deliver a system that meets user needs and organizational goals.

5.1.1 Data Selection:

In selecting data for gesture-to-text conversion using MobileNet and deep learning techniques, several factors are considered. The ASL dataset is used to train the model to recognize various hand gestures corresponding to different alphabets. The dataset includes diverse images covering different angles, lighting conditions, and hand positions to ensure the model learns to classify gestures effectively in real-world scenarios.

Each image in the dataset is accurately labeled with the corresponding letter, allowing the model to associate visual features with specific gestures. Proper labeling is crucial for the deep learning model to correctly map hand movements to textual output.

To ensure fairness and robustness, the dataset is curated to maintain a balanced representation of different hand gestures. This helps prevent bias toward certain signs and ensures accurate recognition across all included gestures.

For the speech-to-text module, we directly integrate the Google Web Speech API, eliminating the need for a separate dataset. This API processes spoken language and transcribes it into text in real time, providing a seamless experience for users.

5.1.2 Data preprocessing:

Data preprocessing is essential for building an efficient deep learning model in the real-time dual-feature system using hand gestures. Since the project uses a CNN-based approach for American Sign Language (ASL) recognition, preprocessing helps ensure accuracy and consistency. The dataset includes images of hand gestures representing ASL alphabets, which may vary in resolution, lighting, background, and orientation. To standardize inputs, images are resized to 1024x1024, optionally converted to grayscale to reduce complexity, and normalized to a [0,1] range for faster convergence. To increase dataset diversity and prevent overfitting, data augmentation techniques such as random rotations, flipping, brightness/contrast adjustments, cropping, and zooming are applied. The data is then split into 90% training and 10% testing sets. Class labels for each ASL gesture are converted using one-hot encoding, and Global Average Pooling (GAP) is used to reduce spatial dimensions and model complexity, enhancing generalization.

For the speech-to-text module, no additional preprocessing like noise reduction or spectrogram generation is performed. Instead, the system utilizes the Google Web Speech API along with the SpeechRecognition library to directly convert spoken words into text. These tools handle the complete speech processing pipeline efficiently, ensuring accurate recognition without the need for extra steps. By combining optimized preprocessing for gesture recognition and direct API-based speech processing, the system achieves robust and real-time performance for assistive communication.

5.1.3 Model Selection:

In selecting a model for gesture-to-text conversion using MobileNet and deep learning techniques like CNN, several factors are considered. MobileNet is chosen for its efficiency and lightweight architecture, making it well-suited for real-time applications with limited computational resources. Since the system needs to accurately recognize ASL (American Sign Language) gestures, the model should be robust to variations in hand position, lighting conditions, and background clutter.

Additionally, a CNN-based model is used alongside MobileNet to enhance feature extraction and classification accuracy. The CNN model is trained specifically on ASL dataset images, enabling better adaptation to sign language recognition. The training process ensures that the model learns meaningful representations of gestures, improving its ability to classify hand movements effectively.

For the speech-to-text module, google-web-speech API along with SpeechRecognition library is used due to its high accuracy in transcribing spoken words into text. This approach eliminates the need for heavy computational models while ensuring reliable speech processing.

Considering real-world constraints such as latency, computational efficiency, and accuracy, MobileNet's balance between performance and lightweight deployment makes it an optimal choice for gesture recognition. Similarly, leveraging pre-trained speech-to-text APIs ensures seamless and efficient speech processing. Together, these models create a robust and accessible communication system for individuals with hearing and speech impairments.

5.1.4 Model Transfer:

In the realm of gesture-to-text conversion using MobileNet and deep learning technique like CNN, transfer learning plays a crucial role. Transfer learning involves leveraging knowledge gained from one task or domain and applying it to another. Instead of training MobileNet from scratch, a pre-trained MobileNet model, which has already learned to recognize general image features from a vast dataset is used as a feature extractor for gesture recognition.

The pre-trained MobileNet model's feature extraction capabilities are then fine-tuned or adapted to recognize ASL (American Sign Language) gestures. This fine-tuning process saves significant training time and computational resources while enhancing the model's ability to differentiate between various hand signs. By training the model with a dataset of ASL images, MobileNet learns to recognize gesture-specific patterns effectively. The extracted features are then fed into a custom CNN classifier, which refines the predictions and maps them to corresponding text outputs.

For the speech-to-text module, transfer learning is indirectly applied by utilizing google-web-speech API, which is trained on large-scale speech datasets. This eliminates the need for extensive training while ensuring high accuracy in real-world speech transcription.

Ultimately, transfer learning enables efficient adaptation of MobileNet for gesture recognition, ensuring high accuracy with limited labeled data. Similarly, leveraging pre-trained speech models streamlines speech-to-text conversion, creating a robust and resource-efficient communication system for individuals with hearing and speech impairments.

5.1.5 Model Training:

Model training for gesture-to-text conversion using MobileNet and CNN involves several steps to enable accurate recognition of hand gestures. Initially, a large dataset of American Sign Language (ASL) gestures is collected and labeled. These images serve as training samples for the model. During training, the pre-trained MobileNet model acts as a feature extractor, identifying key visual patterns in the gesture images. The extracted features are then passed through a custom CNN classifier, which maps them to corresponding text labels.

The training process involves multiple iterations, where the model adjusts its internal parameters to minimize the difference between predictions and actual labels. This optimization is achieved using techniques like Adam optimizer. Data augmentation (such as rotation, scaling, and flipping) may also be applied to improve the model's ability to generalize to different hand positions and lighting conditions. Training continues until the model achieves a high accuracy in recognizing gestures.

For speech-to-text conversion, the system utilizes google-web-speech API, which is already trained on large-scale speech datasets. Instead of training a speech model from scratch, this API processes spoken input and converts it into text with high accuracy.

5.1.6 Evaluation:

Evaluation in the context of gesture-to-text and speech-to-text conversion involves assessing the model's accuracy, reliability, and effectiveness in real-world applications. For the gesture-to-text module, the evaluation process includes testing the CNN-based model on a dataset of ASL gesture images. The model's predictions are compared with the correct labels, and performance metrics such as accuracy, precision, recall, and F1-score are calculated. The system's ability to correctly classify gestures under varying lighting conditions, backgrounds, and hand orientations is also assessed.

For the speech-to-text module, evaluation involves analyzing how accurately the model transcribes spoken words into text. This includes testing the system with different speakers, accents, speech speeds, and background noise levels. Metrics such as word error rate (WER) and phoneme error rate (PER) are used to measure transcription accuracy.

Additionally, cross-validation can be applied to ensure that the model generalizes well across different datasets. The final evaluation determines how well the system performs in real-world conditions, guiding potential improvements and optimizations for enhanced usability and robustness.

5.1.7 Testing and Validation:

In the context of gesture-to-text and speech-to-text conversion using deep learning and api, testing and validation are essential for evaluating the system's accuracy, robustness, and real-world applicability.

Validation occurs during training, where a portion of the dataset is set aside to assess the model's learning progress. For the gesture-to-text module, this involves validating the CNN-based classifier using unseen ASL gesture images to ensure it generalizes well and does not overfit the training data. For the speech-to-text module, validation helps fine-tune the speech recognition system's accuracy under different noise levels and accents which happens internally.

Testing is performed after training, using completely new data that the model has never seen before. In the gesture module, test images of hand signs are fed into the system to verify whether the model correctly translates them into corresponding words. In the speech module, spoken words from different users are processed by the google-web-speech API to evaluate transcription accuracy.

By systematically validating and testing both modules, the system's performance, accuracy, and robustness are measured, ensuring it can reliably assist individuals with speech or hearing impairments in real-world scenarios.

5.1.8 Continuous improvement:

Continuous improvement in the context of gesture-to-text and speech-to-text conversion involves refining the system over time to enhance its accuracy, adaptability, and user experience.

For the gesture-to-text module, improvements can be made by expanding the dataset to include a more diverse range of gestures, hand orientations, lighting conditions, and backgrounds. Fine-tuning the CNN model, using advanced architectures like Transformers or Vision-based models, and applying data augmentation techniques can further boost accuracy. Regular model retraining with newly collected gesture data ensures the system remains robust.

For the speech-to-text module, improvements involve incorporating more speech samples from diverse speakers, accents, and noisy environments. Enhancements in language models using techniques like transformer-based models (e.g., Whisper, Wav2Vec 2.0) can improve transcription accuracy. Additionally, fine-tuning the model based on real-time user feedback and incorporating error correction mechanisms can enhance performance.

5.2 Project Modules:

This project consists of multiple modules that work together to enable real-time gesture and speech recognition into text conversion. Below is a detailed breakdown of the key modules:

5.2.1. Data Collection and Preprocessing

The gesture dataset consists of American Sign Language (ASL) images representing different hand gestures. To enhance model performance, the dataset ensures diversity by including variations in lighting conditions, hand orientations, and backgrounds. Additionally, data augmentation techniques such as rotation, scaling, and flipping are applied to improve

generalization and robustness. For the speech-to-text module, instead of using a pre-collected speech dataset, the system integrates the Google-Web-Speech API for real-time speech recognition. This eliminates the need for extensive preprocessing, as the API handles noise reduction and normalizes input audio for improved accuracy in speech-to-text conversion.

5.2.2. Gesture Recognition Module

A pre-trained MobileNet model is utilized as a feature extractor by removing its top classification layers and fine-tuning it specifically for ASL gesture recognition. This allows the model to extract relevant features from input hand gesture images, which are then used for further classification. On top of these extracted features, a custom CNN-based classifier is designed to recognize different ASL gestures and map them to corresponding alphabets. The model is trained to improve recognition accuracy, and optimization techniques such as dropout and batch normalization are applied to enhance its performance and generalization.

5.2.3. Speech-to-Text Module

For real-time speech-to-text conversion, the SpeechRecognition library is first used to capture speech input from the user through a microphone while applying initial noise reduction to enhance clarity. The processed audio is then sent to the Google Web Speech API, which performs additional noise reduction and speech processing to improve recognition accuracy. Finally, the API converts the spoken words into text output with good precision. To further refine the results, text correction techniques are applied, ensuring improved transcription accuracy and overall system efficiency.

5.2.4. User Interface (UI) Module:

The User Interface (UI) Module is designed to provide a seamless and interactive experience for users. The frontend development focuses on creating a simple yet effective graphical user interface (GUI) to display recognized gestures and speech outputs in real-time. Frameworks like Flask are utilized for UI implementation, ensuring accessibility and ease of use. Additionally, a "My Recordings" tab is included to store previously recognized gestures and speech inputs along with their corresponding text. This feature allows users to review and manage enhancing usability and accessibility.

5.3 Algorithm:

The project "**A Dual-Feature System: Gesture-to-Word Using CNN and Speech-to-Text Conversion**" aims to assist individuals with speech and hearing impairments through two independent yet complementary modules. The first module uses **Convolutional Neural Networks (CNNs)** for recognizing hand gestures (specifically ASL), while the second module

uses the **Google Web Speech API** to convert speech into text. Together, these modules enable seamless, inclusive communication depending on user needs.

In the **Gesture-to-Word Module**, the system employs **MobileNet** as a feature extractor through transfer learning, combined with a custom CNN for classifying ASL gestures. The model is trained on a diverse dataset of hand gestures representing the alphabet, with preprocessing steps like image resizing, normalization, and grayscale conversion applied. Data augmentation techniques such as flipping and scaling help improve the model's generalization to new inputs, making it more robust in real-world conditions.

During model training, CNN layers extract features like edges and shapes from the gesture images. Pooling layers reduce dimensionality, improving efficiency, while the final softmax layer predicts the most likely gesture class. The **Adam optimizer** and **cross-entropy loss function** are used for model tuning. Once trained, the system can detect live gestures through a webcam and convert them into corresponding letters or words on the screen.

The **Speech-to-Text Module** captures audio input using a microphone and processes it using the **SpeechRecognition library** integrated with Google's Web Speech API. The audio undergoes noise reduction and is then transcribed into text using phonetic recognition. Post-processing techniques are used to correct minor errors and enhance readability, ensuring the output text is accurate and easy to understand.

Together, these modules form a flexible and accessible solution for communication. A deaf user can rely on the speech-to-text feature to follow conversations, while a mute user can communicate using gestures. The modular design ensures adaptability, and the system can evolve over time with more training data and user feedback. By combining deep learning with speech technologies, the project offers a practical and inclusive tool for breaking communication barriers.

Architecture:

In this project the architecture consists of two distinct yet interconnected modules: Gesture-to-Word using CNN and Speech-to-Text Conversion using the Google-Web-Speech API. This dual-feature system is designed to facilitate seamless connection for individuals with speech and hearing impairments by enabling both visual and auditory input processing.

The Gesture-to-Word Module is powered by a Convolutional Neural Network (CNN), which is specifically designed to recognize and classify American Sign Language (ASL) gestures. The CNN architecture is chosen due to its effectiveness in feature extraction from images and its ability to

generalize across different hand orientations and lighting conditions. The model comprises multiple convolutional layers followed by activation functions, pooling layers, and fully connected layers. The convolutional layers are responsible for extracting spatial features such as edges, shapes, and patterns in hand gestures, while pooling layers reduce dimensionality, ensuring computational efficiency. The final classification layer utilizes a softmax activation function to determine the most probable word corresponding to a recognized gesture.

To improve the model's generalization, transfer learning is employed using pre-trained CNN architectures such as MobileNet. These models, trained on large-scale image datasets, provide robust feature extraction capabilities that are fine-tuned for gesture recognition. Data augmentation techniques such as rotation, flipping, and contrast adjustments are applied to improve the model's robustness and adaptability to diverse hand gestures. The trained model is deployed for real-time recognition, where a webcam captures the user's hand gestures, processes the image through the CNN model, and outputs the corresponding text.

The Speech-to-Text Module leverages automatic speech recognition (ASR) technology to transcribe spoken language into text. The architecture of this module is built around the Google-Web-Speech API, which provides powerful speech recognition capabilities with support for multiple languages and accents. The process begins with capturing audio input from a microphone, followed by preprocessing techniques such as noise reduction and feature extraction. The speech signal is then processed by the Google-Web-Speech API, which converts it into a structured textual format.

This module is optimized to work in real-time, allowing individuals with hearing impairments to instantly view the spoken words on their screen.

The integration of both modules provides a versatile dual-feature system that enhances accessibility for individuals with disabilities. A mute person can use the gesture-to-word module to convey messages to non-sign language users, while a deaf person can rely on the speech-to-text module to understand spoken words. The architecture is designed to function independently or in combination, allowing users to switch between modules based on their needs.

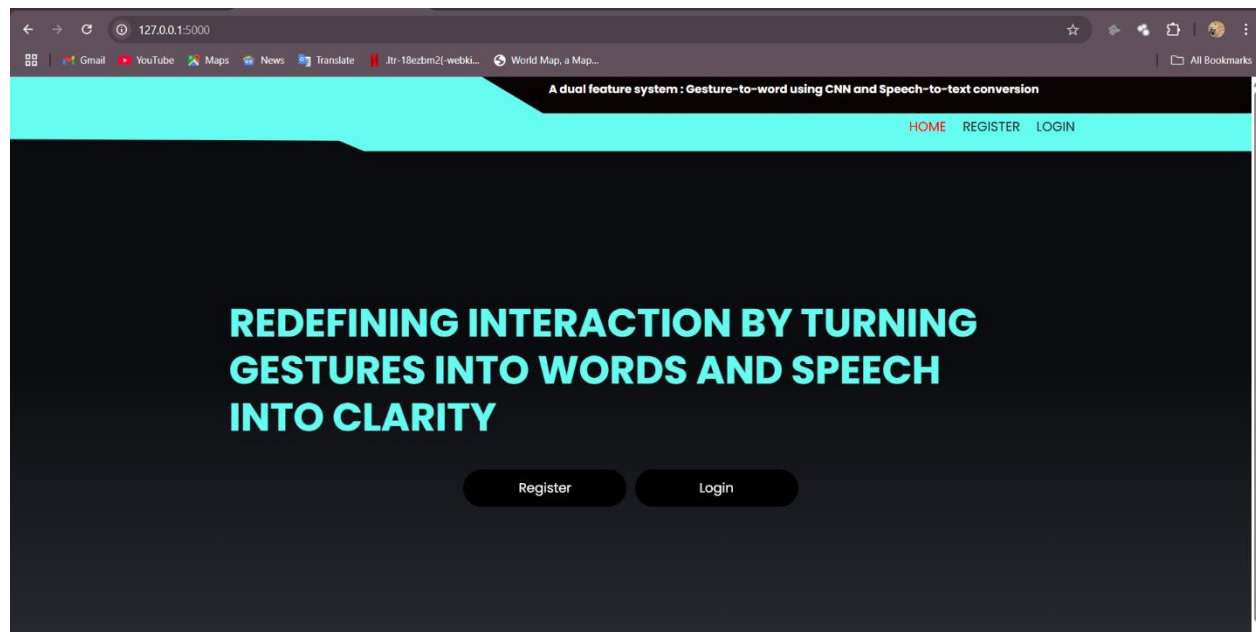
One of the key advantages of this architecture is its real-time processing capability. The gesture recognition model is optimized for low-latency predictions, enabling smooth interactions, while the speech-to-text module processes spoken language efficiently through cloud-based APIs.

In summary, the architecture of "A Dual-Feature System: Gesture-to-Word Using CNN and Speech-to-Text Conversion" combines deep learning-based visual recognition and speech processing technologies to create an inclusive communication platform. By leveraging CNNs for

gesture recognition and ASR for speech transcription, the project provides an effective, and user-friendly solution that connects individuals with speech and hearing impairments. Through continuous improvements in model accuracy, data augmentation, and system integration, this architecture contributes to advancing assistive technologies and promoting accessibility.

5.4 Screens:

Screen 1: Home Page (Landing Screen)



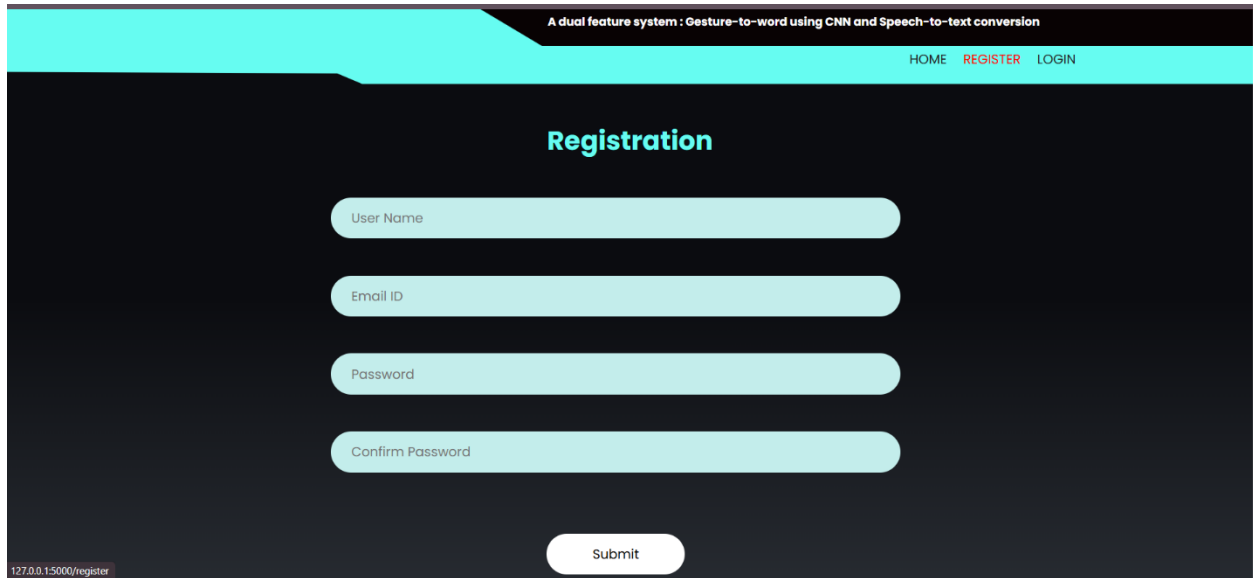
Description:

This screen serves as the landing page for a web application focused on gesture-to-word and speech-to-text conversion. At the top, displaying a tagline in white text: "A dual feature system: Gesture-to-word using CNN and Speech-to-text conversion." This tagline highlights the core functionalities of the project, indicating that it leverages Convolutional Neural Networks (CNN) for gesture recognition and provides speech-to-text conversion capabilities.

Below the tagline, three navigation links are aligned on the right side of the header in red text: "HOME," "REGISTER," and "LOGIN." The "HOME" link is likely active, as this is the landing page, while "REGISTER" and "LOGIN" provide options for users to create an account or sign in.

The browser tabs visible at the top suggest that this page is being viewed on a local server (IP address: 127.0.0.1:5000), indicating that the project is likely in a development or testing phase.

Screen 2: Registration Page

A screenshot of a web browser displaying a registration page. The browser's address bar shows '127.0.0.1:5000/register'. The page has a dark blue header with a white navigation bar containing 'HOME', 'REGISTER', and 'LOGIN'. Below the header, the word 'Registration' is centered in a large, bold, light blue font. Underneath, there are four light blue rounded rectangular input fields with placeholder text: 'User Name', 'Email ID', 'Password', and 'Confirm Password'. At the bottom center, there is a white rounded rectangular button with the text 'Submit'.

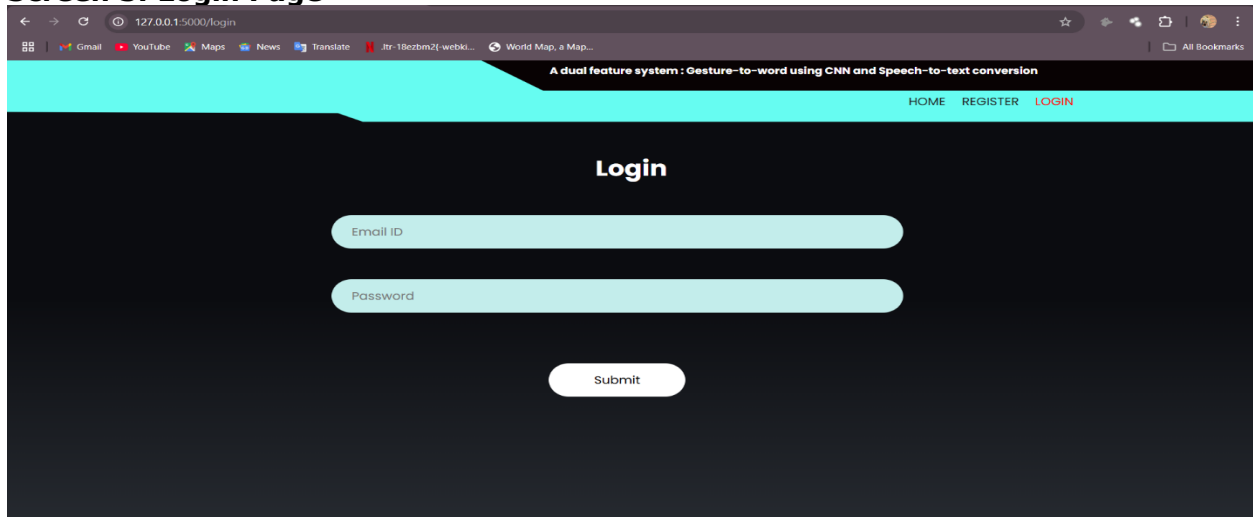
Description:

The main content is centered and begins with the word "Registration" in large, cyan text, serving as the page's title. Below the title, there is a simple form with four input fields, each enclosed in a light cyan, rounded rectangle with a placeholder label in gray text:

1. "User Name": A field for the user to enter their desired username.
2. "Email ID": A field for the user to input their email address.
3. "Password": A field for the user to enter a password.
4. "Confirm Password": A field to re-enter the password for verification.

Below the form, a prominent "Submit" button is centered, styled as a white, rounded rectangle with black text. This button allows users to submit their registration details once all fields are filled.

Screen 3: Login Page

A screenshot of a web browser displaying a login page. The browser's address bar shows '127.0.0.1:5000/login'. The page has a dark blue header with a white navigation bar containing 'HOME', 'REGISTER', and 'LOGIN'. Below the header, the word 'Login' is centered in a large, bold, light blue font. Underneath, there are two light blue rounded rectangular input fields with placeholder text: 'Email ID' and 'Password'. At the bottom center, there is a white rounded rectangular button with the text 'Submit'.

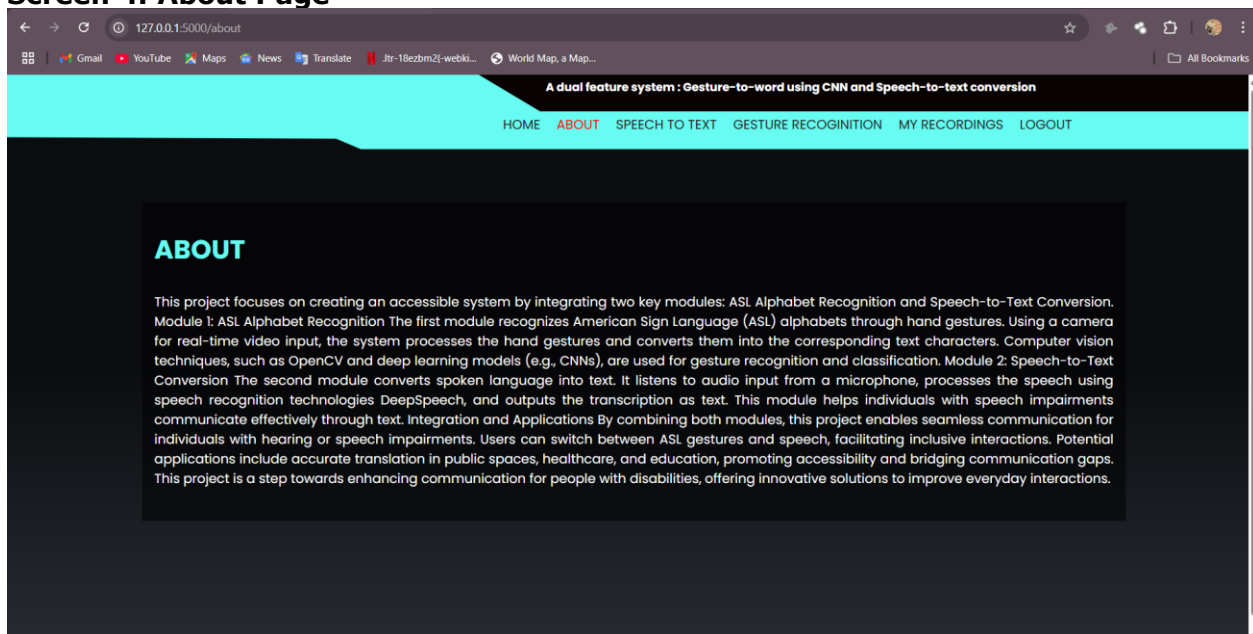
Description:

This screen is the login page of the web application, designed to allow existing users to access their accounts. Three navigation links are displayed on the right side: "HOME," "REGISTER," and "LOGIN."

The main content is centered and begins with the word "Login" in large, white text, serving as the page's title. Below the title, there is a simple form with two input fields, each enclosed in a light cyan, rounded rectangle with a placeholder label in gray text:

1. "Email ID": A field for the user to enter their registered email address.
2. "Password": A field for the user to input their password.

Screen 4: About Page

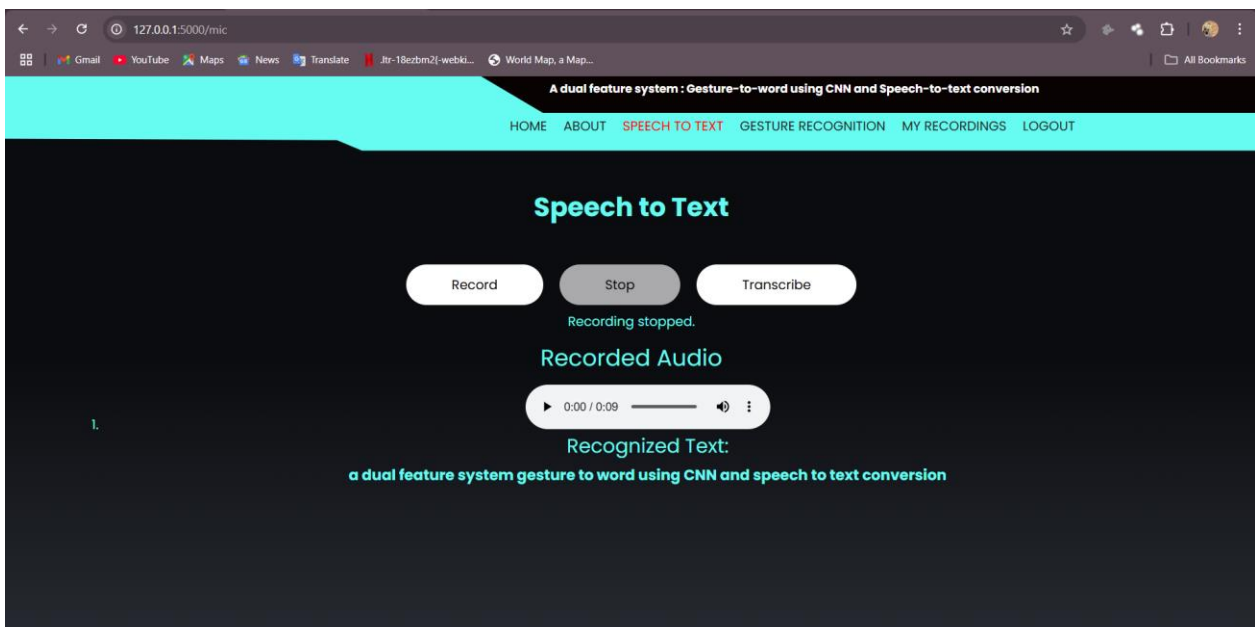
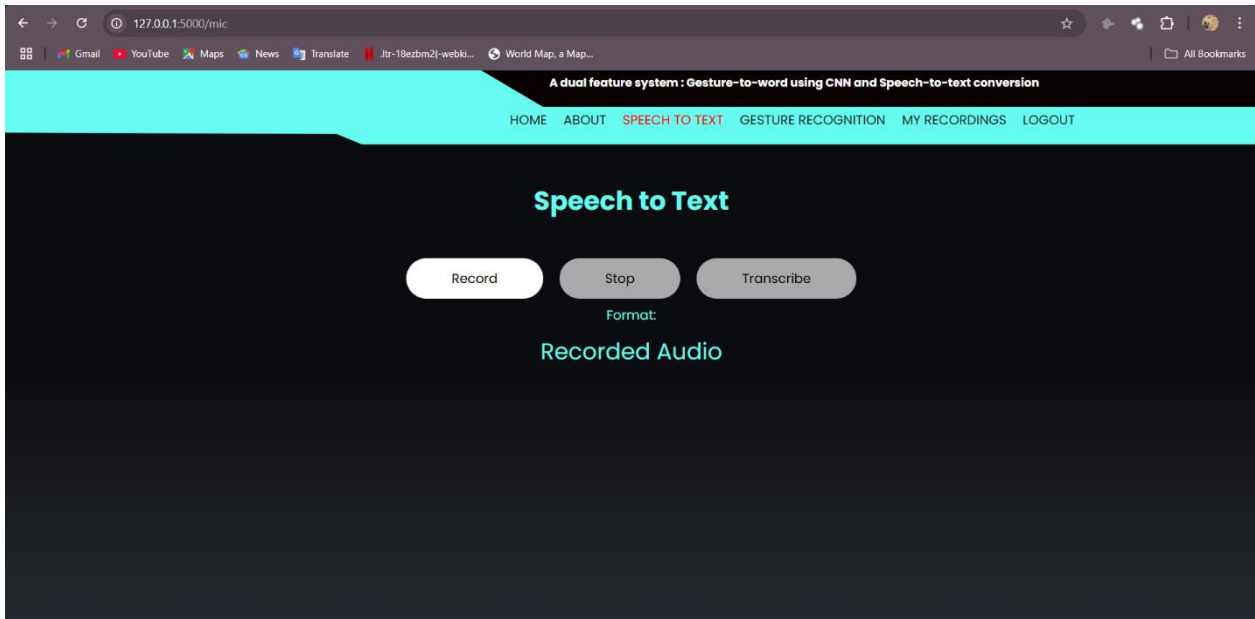


Description:

This screen is the "About" page of the web application, providing detailed information about the project's purpose and functionalities

The main content is centered and begins with the word "ABOUT" in large, bold, white text, serving as the page's title. The text is presented within a slightly darker rectangular section, making it stand out against the black background while maintaining readability.

Screen 5: Speech to Text Page



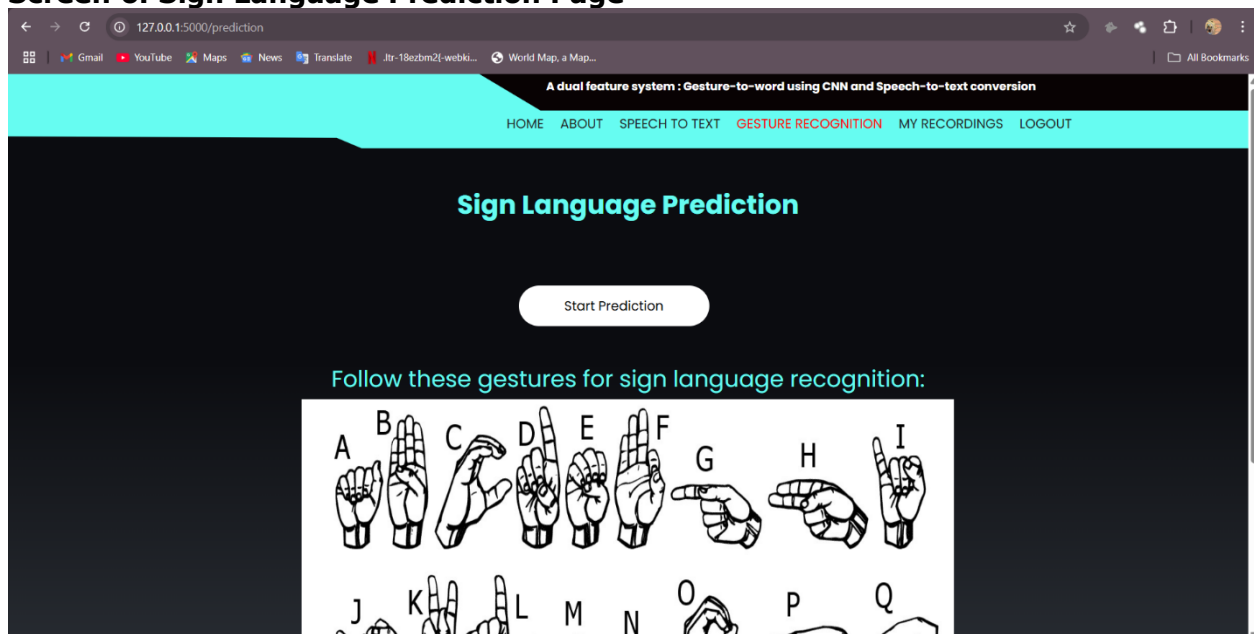
Description:

This screen is the "Speech to Text" page of the web application, designed to facilitate audio recording and transcription functionalities. A navigation menu with six links is aligned on the right side: "HOME," "ABOUT," "SPEECH TO TEXT," "GESTURE RECOGNITION," "MY RECORDINGS," and "LOGOUT." The "SPEECH TO TEXT" link is likely active, indicating that this is the current page, and the presence of "LOGOUT" suggests that the user is logged in.

The main content is centered and begins with the phrase "Speech to Text" in large, bold, white text, serving as the page's title. Below the title, there are three circular buttons labeled "Record," "Stop," and "Transcribe," arranged horizontally. The "Record" button is white, while the "Stop" and "Transcribe" buttons are gray, indicating that the recording has not yet started. Beneath

these buttons, a section labeled "Recorded Audio" is displayed in white text, intended to show the recorded audio files or transcribed text once the process is initiated (shown in second figure),

Screen 6: Sign Language Prediction Page



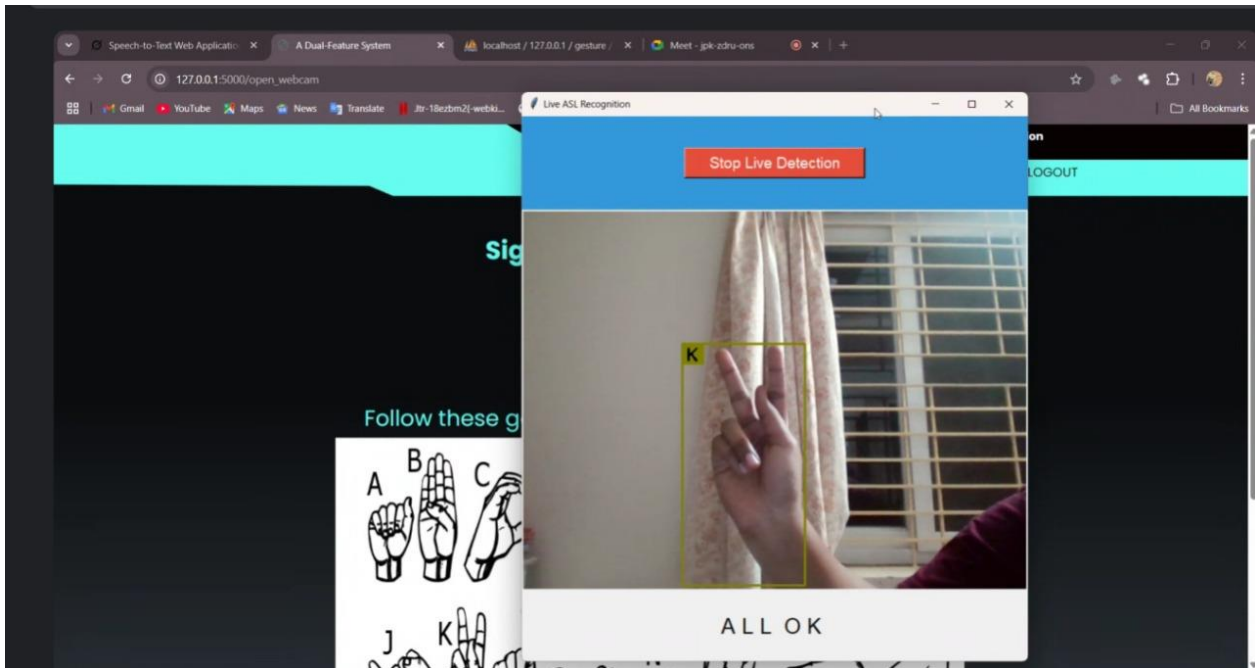
Description:

The main content is centered and begins with the phrase "Sign Language Prediction" in large, bold, cyan text, serving as the page's title. Below the title, there is a single white circular button labeled "Start Prediction" in black text, which is likely used to initiate the gesture recognition process. Underneath the button, the text "Follow these gestures for sign language recognition:" is displayed in cyan, followed by a grid of hand gesture illustrations representing letters of the alphabet (A through Z). The gestures are depicted in black and white line drawings.

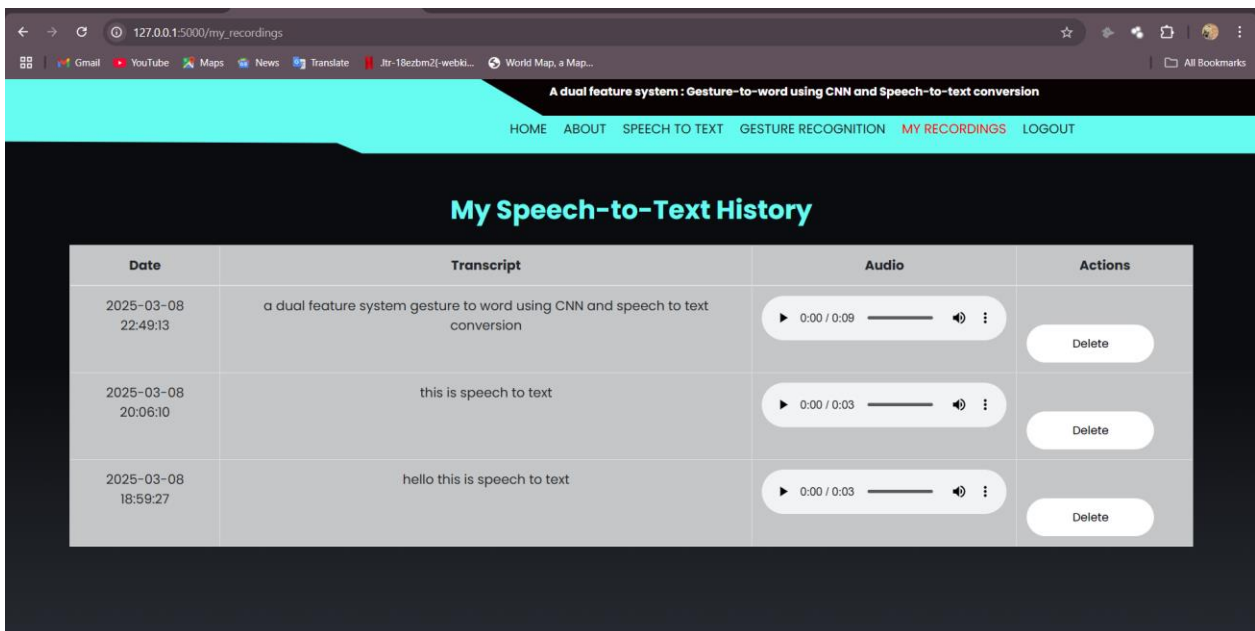
Screen 7: Sign Language Detection Page

Description:

This screen is the "Sign Language Detection" page of the web application, designed to enable real-time gesture recognition for sign language interpretation using a webcam. Below this, a live video feed from the webcam is displayed, showing a hand gesture (the letter "K" in sign language) within a green bounding box, indicating live detection for letter K, similarly other colours for other alphabets. A red button labelled "Stop Live Detection" is prominently placed in the center of the video feed, allowing the user to halt the detection process. Additionally, a white pop-up box at the bottom of the feed displays "ALL OK," likely indicating stacked letters from user sign detections.



Screen 8: My Speech-to-Text History Page



Description:

This screen is the "My Speech-to-Text History" page of the web application, designed to display a user's past speech-to-text recordings and their transcriptions. A navigation menu with six links is aligned on the right side: "HOME," "ABOUT," "SPEECH TO TEXT," "GESTURE RECOGNITION," "MY RECORDINGS," and "LOGOUT." The "MY RECORDINGS" link is likely active, indicating that this is the current page, and the presence of "LOGOUT" suggests that the user is logged in.

The main content is centered and begins with the phrase "My Speech-to-Text History" in large, bold, cyan text, serving as the page's title. Below the title, there is a table listing the user's recording history with four columns: "Date," "Transcript," "Audio," and "Actions."

Additionally, we have a logout button after the Home page will be displayed.

6.System Testing

6.1 Introduction:

Testing is a crucial phase in software development aimed at identifying faults or weaknesses in a product. It ensures that the software meets user requirements and functions correctly without unacceptable failures. The main purpose of testing is not to prove the software is flawless, but to uncover errors and issues so they can be addressed before deployment.

Software testing includes different types, each targeting specific needs—such as unit testing for individual components and integration testing to check how modules work together. Testing is carried out at various levels of the software development life cycle and involves both verification (checking if the product is built right) and validation (checking if the right product is built).

Additionally, testing involves both examining the code and executing it under different environments and conditions. Independent testing teams often handle this to ensure objectivity. Tools such as comparison with previous versions, industry standards, or similar products help assess the system's behavior.

In machine learning, testing includes using validation sets to tune the model and test sets to evaluate real-world performance. The validation set helps prevent overfitting, while the test set, chosen randomly and used only once, ensures unbiased assessment and generalization of the model.

Cross-validation:

Cross-validation is a model evaluation technique that can be performed even on a limited dataset. The training set is divided into small subsets, and the model is trained and validated on each of these samples.

Validation:

In this method, we perform training on the 50% of the given data-set and rest 50% is used for the testing purpose. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e higher bias.

Leave One Out Cross Validation :

In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point. It has some advantages as well as disadvantages also. An advantage of using this method is that we make use of all data points and hence it is low bias. The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points times.

k-fold cross-validation:

The most common cross-validation method is called k-fold cross-validation. To use it, you need to divide the dataset into k subsets (also called folds) and use them k times. For example, by breaking the dataset into 10 subsets, you will perform a 10-fold crossvalidation. Each subset must be used as the validation set at least once. In Machine Learning model development, the performance of the system is measured in terms of accuracy, precision and recall of the model. It's common to also use validation accuracy and test accuracy as evaluation metrics, which provide a better indication of the model's performance on unseen data.

Accuracy:

Accuracy is a measure of how many model predictions are correct. While high accuracy is desirable, it's not the only metric to consider when evaluating performance.

Testing involves evaluating a system to ensure it meets specified requirements and helps identify errors or missing functionalities. Depending on experience, roles such as Software Tester or QA Analyst are assigned for this task. Testing should ideally begin early in the SDLC—from the requirements phase—and continue through to deployment. The timing and extent of testing often depend on the development model used (e.g., Waterfall, Incremental).

Errors can occur at any stage of development, and not all are caught immediately. **Static analysis** identifies defects without executing code (e.g., inspections or static tools), while **dynamic analysis**—mainly through testing—involves executing the code to uncover defects.

The goal of testing is to ensure the **software under test (SUT)** behaves as expected through a set of defined test cases. The **effectiveness of testing** relies on how well test cases are selected. Concepts such as errors, faults, test suites, and test harnesses are central to this process.

Ultimately, testing aims to produce high-quality, error-free software. The three main stages of the testing process are **test planning, test case design, and test execution**, supported by techniques like black-box and white-box testing, and evaluated using metrics like coverage and reliability.

Basic concepts and definitions relating to testing, like error, fault, failure, test case, test suite, test harness, etc.

- The testing process—how testing is planned and how testing of a unit is done.
- Test case selection using black-box testing approaches.
- Test case selection using white-box approaches.
- Some metrics like coverage and reliability that can be employed during testing.

The basic goal of the software development process is to produce software that has no errors or very few errors. We have seen that different levels of testing are needed to detect the defects injected during the various tasks in the project. The testing process for a project consists of three high-level tasks test planning, test case design, and test execution. We will discuss these in the rest of this section.

1.Test Plan:

In a project, testing commences with a test plan and terminates with successful execution of acceptance testing. A test plan is a general document for the entire project that defines the scope, approach to be taken, and the schedule of testing, as well as identifies the test items for testing and the personnel responsible for the different activities of testing.

The test planning can be done well before the actual testing commences and can be done in parallel with the coding and design activities.

The inputs for forming the test plan are:

- (1) Project plan
- (2) Requirements document
- (3) Architecture or design document.

2.Test Case Design:

The test plan focuses on how the testing for the project will proceed, which units will be tested, and what approaches (and tools) are to be used during the various stages of testing. However, it does not deal with the details of testing a unit, nor does it specify which test cases are to be used.

3.Test Case Execution:

The test case specifications only specify the set of test cases for the unit to be tested. However, executing the test cases may require construction of driver modules or stubs. It may also require modules to set up the environment as stated in the test plan and test case specifications.

6.2 Testing methods :

System Testing includes testing of a fully integrated software system.

Generally, a computer system is made with the integration of software (any software is only a single element of a computer system).

The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different types of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as System testing.

In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system. It is end-to-end testing where the testing environment is similar to the production environment.

There are mainly two widely used methods for software testing, one is White box testing which uses internal coding to design test cases and another is black box testing which uses GUI or user perspective to develop test cases.

The following are the Testing Methodologies:

- Unit Testing.
- Integration Testing.
- Validation Testing.
- White box testing.
- Black box testing.

Unit Testing :

Unit testing is a software testing method that verifies the smallest testable parts of an application, such as functions, methods, or modules, in isolation. It ensures each unit of code performs as expected and adheres to design specifications. Conducted during the development phase, typically by developers, unit testing involves checking module interfaces, control paths, and expected outputs. It is the first level of testing in models like SDLC, STLC, and V-Model, and is generally implemented using white-box testing techniques. While developers usually perform unit testing, QA engineers may also contribute due to time constraints. Unit tests help detect bugs early, reduce development costs, improve understanding of the codebase, speed up modifications, and act as useful documentation. They also support code reusability and maintainability.

To execute unit tests, developers write specific code to rigorously test individual functions, often using frameworks for automation. This isolation helps identify and eliminate unnecessary dependencies, making the codebase more robust. Unit testing can be done manually or automatically, with automation being the preferred and more efficient approach in most modern software engineering practices.

Integration Testing :

Integration Testing is a type of software testing where individual modules are combined and tested as a group to ensure they work together as intended. It addresses both verification and

construction by checking how modules interact and communicate, especially since they are often developed by different programmers. This testing identifies defects in interfaces, data flow, and integration points between modules. It is necessary when requirements change during development, or when issues may arise in database or hardware interfaces, or due to poor exception handling. Unlike unit testing, which focuses on individual functions, integration testing emphasizes testing the links between modules. It is also known as Integration & Testing (I&T), String Testing, or Thread Testing. Common approaches include Big-Bang, Bottom-Up, Top-Down, and Mixed Integration Testing.

Big-Bang Integration Testing:

It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing is very expensive to fix. The Big-bang integration workflow diagram is a process diagram that shows how different parts of a system are integrated. It is typically used to show how different software components are integrated. It is a graphical representation of the software development process that is used in many organizations. It is a process that starts with the idea for a new software project and ends with the delivery of the software to the customer.

Bottom-Up Integration Testing:

Bottom-up Testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving upward from bottom to top through control flow of architecture structure. In these, low-level modules are tested first, and then high-level modules are tested. This type of testing or approach is also known as inductive reasoning and is used as a synthesis synonym in many cases. Bottom-up testing is user-friendly testing and results in an increase in overall software development. This testing results in high success rates with long-lasting results.

Top-Down Integration Testing:

Top-down testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through control flow of architecture structure. In these, high-level modules are tested first, and then low-level modules are tested. Then, finally, integration is done to ensure that the system is working properly. Stubs and drivers are used to carry out this project. This technique is used to increase or stimulate behavior of Modules that are not integrated into a lower level.

Mixed Integration Testing:

A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In a top-down approach, testing can start only after the top-level module has been coded and unit tested. In the bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. It is also called the hybrid integration testing. also, stubs and drivers are used in mixed integration testing.

Validating Testing :

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements. Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment. Validation checks are performed on the following fields.

Text Field:

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error message. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. The testing should be planned so that all the requirements are individually tested successful test is one that gives out the defects for the inappropriate data and produces an output revealing the errors in the system.

Preparation of Test Data :

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

Using Live Test Data:

Live test data refers to real data extracted from an organization's files, used to test a system during or after partial development. Users may input data from their regular activities, or analysts may enter it themselves. While this data reflects realistic scenarios and typical processing, it is often limited in quantity and may not cover all possible input combinations or edge cases. As a result, live data tends to test only common situations, which may not reveal critical issues or potential system failures, limiting the effectiveness of the overall system testing.

Using Artificial Test Data:

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program. The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications. The package "Virtual Private Network" has satisfied all the requirements specified as per software requirement specification and was accepted.

White Box Testing:

White Box Testing, also known as structural or glass box testing, is a technique that evaluates the internal structure and code of a program. It ensures that internal operations perform as specified and all code components are properly tested. The main objectives include verifying all independent paths within modules, logical decisions on both true and false outcomes, loop boundaries, and internal data structures. This method helps detect logical errors, design flaws, and typographical or syntax issues.

To perform white box testing, testers must understand the application's source code, making it essential for them to be familiar with the programming language and tools used. Tests are then created and executed by programmers. Common techniques include control-flow testing, which ensures all statements and branch conditions are tested, and data-flow testing, which checks variable declaration, usage, and modification throughout the code.

Black Box Testing:

Black Box Testing is a software testing method where the internal structure or code is not known to the tester. It focuses solely on testing the software's functionality by providing inputs and observing outputs, ensuring that the system behaves as expected. Also known as behavioral or specification-based testing, it is mostly performed by testers or end users.

This method is useful for detecting functionality issues without delving into the code.

Techniques include Equivalence Partitioning (testing representative input groups), Boundary Value Analysis (testing edge inputs), Cause-Effect Graphing (mapping inputs to outputs), Pairwise Testing (testing combinations of input pairs), and State-Based Testing (testing behavior based on input-driven state changes).

The generic process involves understanding requirements, designing both valid and invalid test scenarios, creating test cases, executing them, comparing actual vs expected results, and retesting if flaws are found.

6.3 Test Cases:

User Registration:

- **Description:** Verify that user details are successfully stored in the MySQL database after registration and that form validations are enforced correctly.
- **Test Steps:**
 1. Access the registration page of the gesture and audio recognition system by navigating to the /register route (e.g., <http://localhost:5000/register>).
Enter the required credentials in the registration form (name, email, password, and confirm password).
 2. Click the "Register" or "Submit" button.
- **Expected Results:** User details (name, email, password) should be successfully stored in the users table of the MySQL database, and the user should be redirected to the login page (/login) with a success message (e.g., "Successfully Registered!"). If validations fail, an appropriate error message should be displayed on the register.html page.
- **Validations:**
 - Email Validation: The email must be in a valid format (e.g., user@example.com) using a regex pattern like `^[a-zA-Z0-9.%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`. It should also check for uniqueness against existing emails in the users table (case-insensitive).
 - Password Validation: The password must contain at least 8 characters and be strong, including:
 - At least one uppercase alphabet (e.g., A-Z).
 - At least one lowercase alphabet (e.g., a-z).
 - At least one numeric character (e.g., 0-9).
 - At least one special symbol (e.g., @, \$, !, %, , &, ?).
 - Spaces are not allowed.
 - Confirm Password Validation: The confirm password field must match the password field exactly, ensuring consistency during registration.

- Empty Field Validation: All fields (name, email, password, confirm password) must be non-empty to proceed with registration.

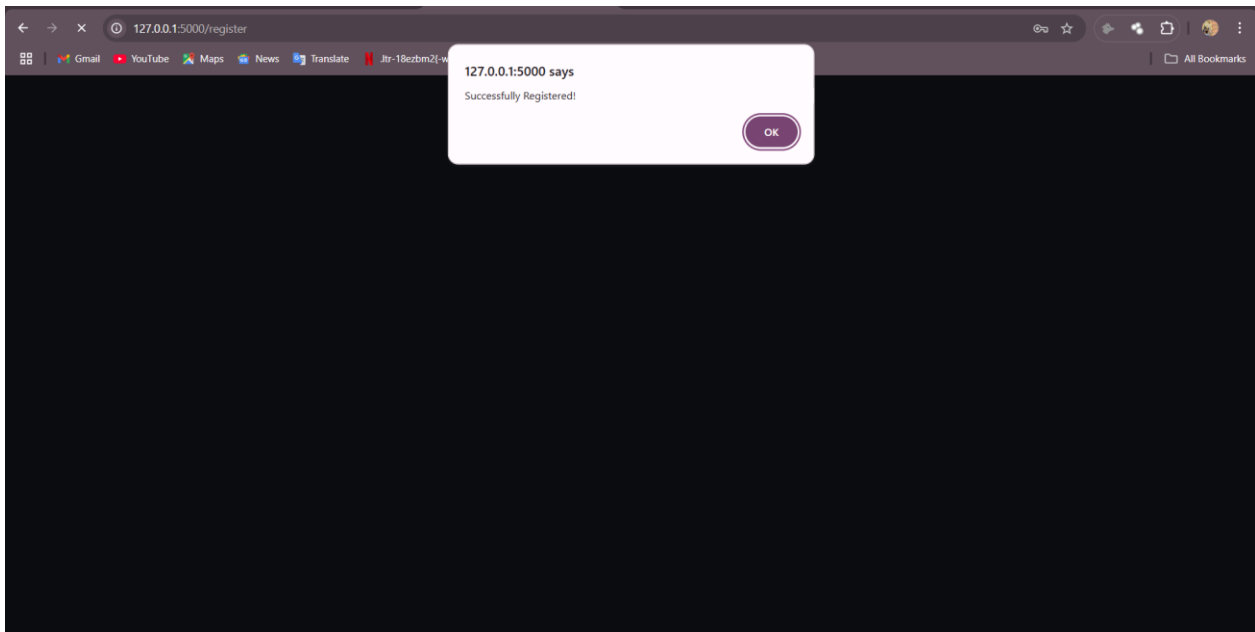
Test Case 1a: email validation(failure).

The screenshot shows a web browser at the URL 127.0.0.1:5000/register. The page has a dark blue header with the text "A dual feature system : Gesture-to-word using CNN and Speech-to-text conversion" and navigation links for HOME, REGISTER, and LOGIN. The main content area is titled "Registration" in red. It contains four input fields: a name field with "Shaik Faheem", an email field with "shaik", a password field with masked characters, and a confirm password field with masked characters. A red error message box is displayed over the email field, stating: "Please include an '@' in the email address. 'shaik' is missing an '@'". A "Submit" button is located at the bottom of the form.

Result: Please include @ in the email address.

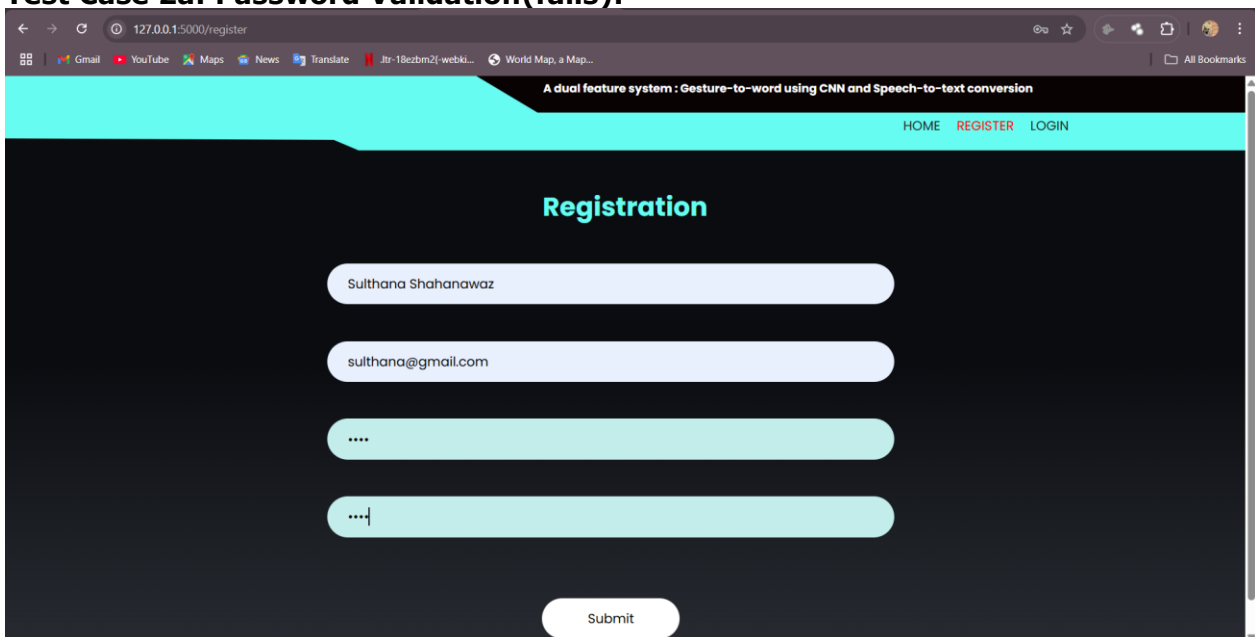
Test Case 1b: email validation pass.

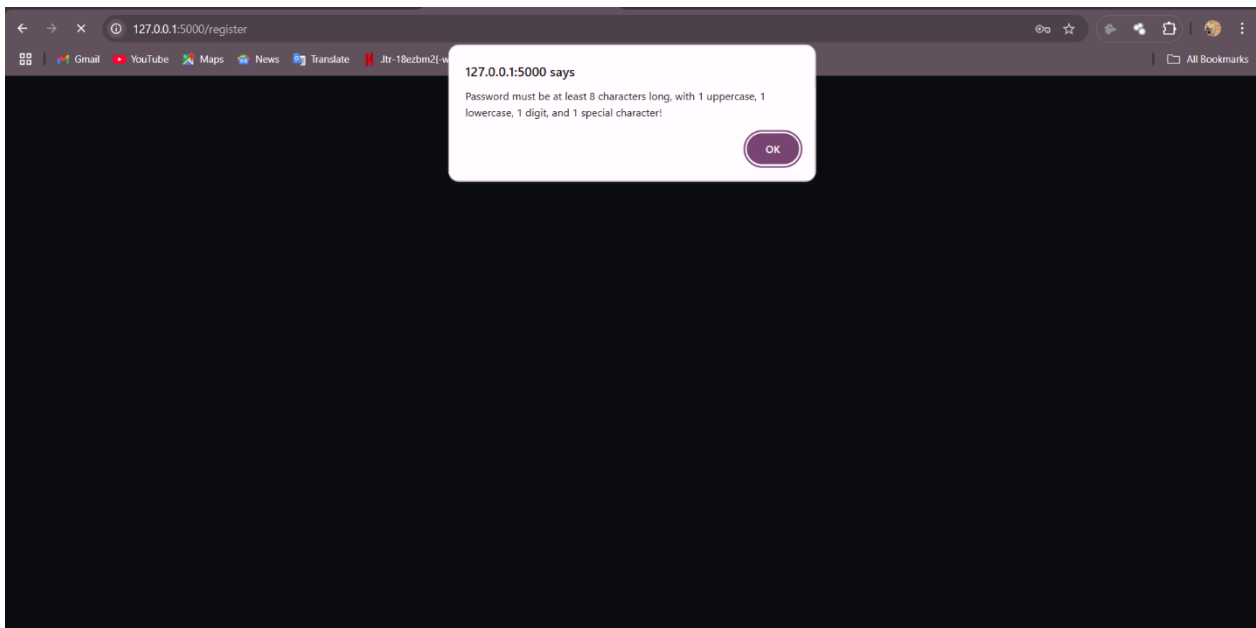
The screenshot shows the same web browser and registration form as in Test Case 1a. The email field now contains the valid email address "shaikfaheem@gmail.com". The error message box is no longer present. The "Submit" button remains at the bottom of the form.



Result: User registered

Test Case 2a: Password Validation(fails).

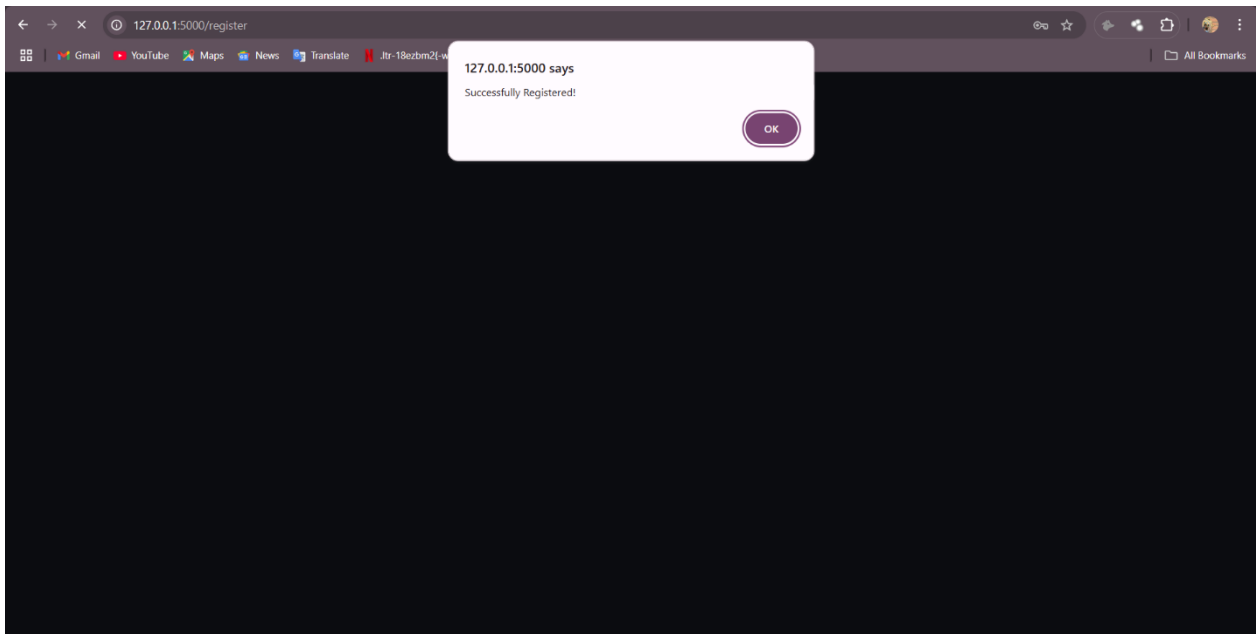




Result: Password validation message.

Test Case 2b: Password Validation pass

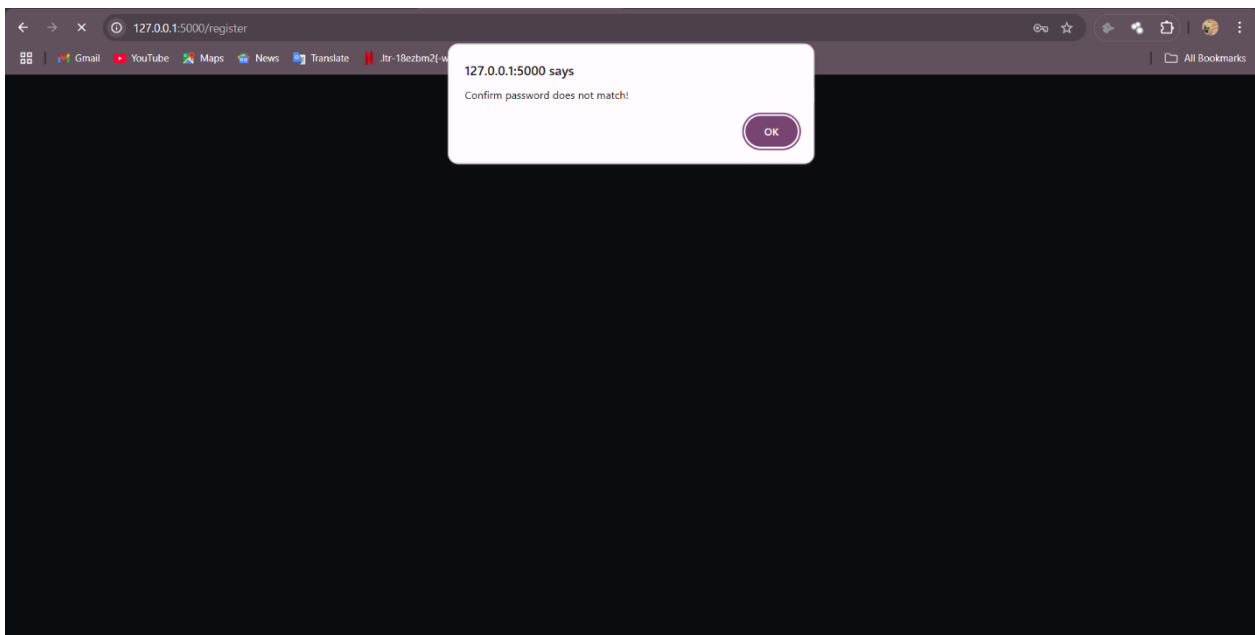
A screenshot of a web application interface. At the top, there is a dark blue header with the text "A dual feature system : Gesture-to-word using CNN and Speech-to-text conversion". Below the header is a navigation bar with links "HOME", "REGISTER", and "LOGIN". The main content area has a dark blue background. In the center, the word "Registration" is displayed in a light blue font. Below it, there are four input fields: the first contains "Sulthana Shahanawaz", the second contains "sulthana@gmail.com", and the next two are empty and masked with dots. At the bottom, there is a "Submit" button.



Result: User registered

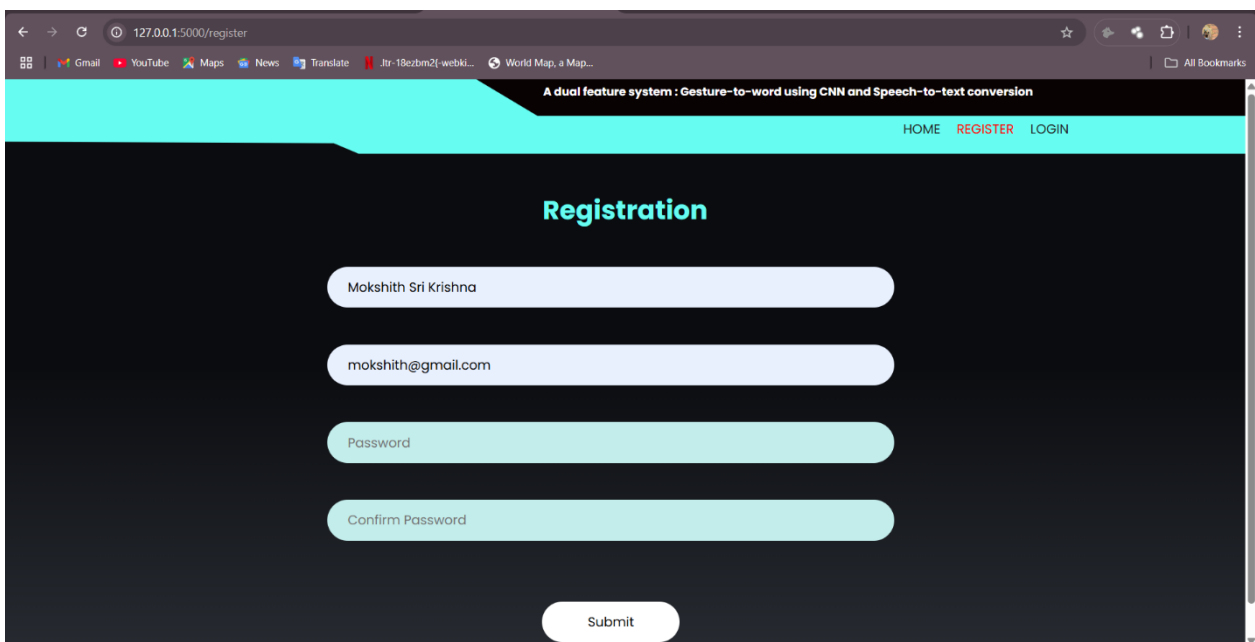
Test case 3: Confirm Password doesn't match

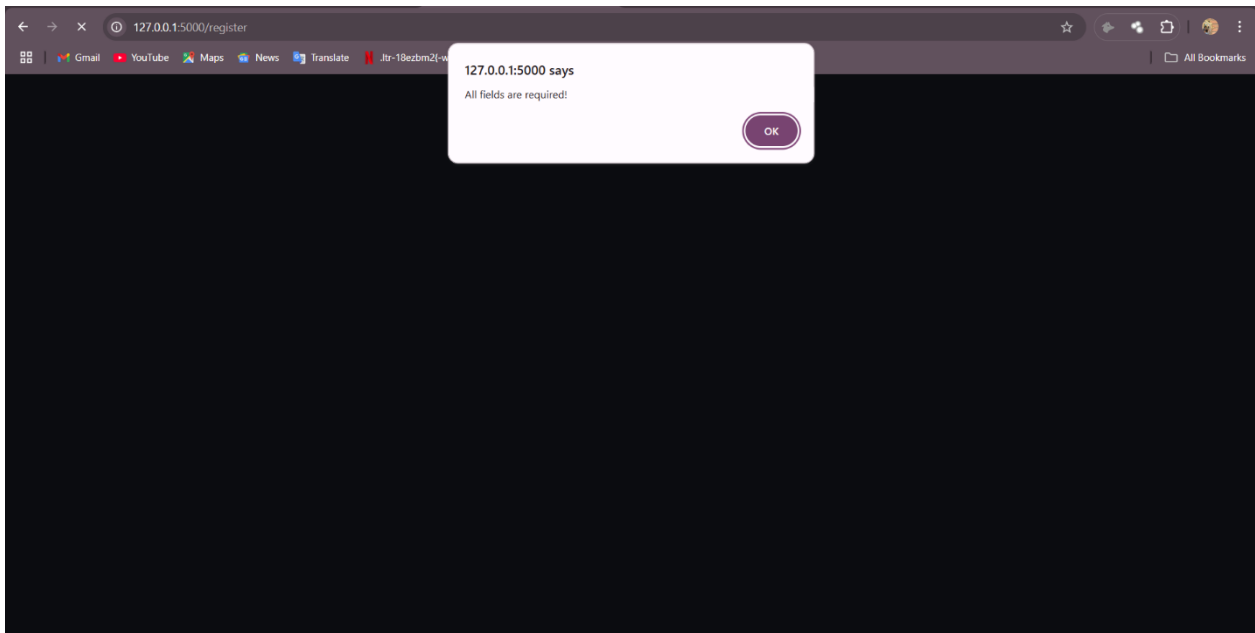
A screenshot of a web application interface. At the top, a dark blue header contains the text 'A dual feature system : Gesture-to-word using CNN and Speech-to-text conversion' and navigation links 'HOME', 'REGISTER', and 'LOGIN'. Below the header, the word 'Registration' is displayed in a large, bold, dark blue font. The registration form consists of four input fields: a text field for the name 'Mokshith Sri Krishna', a text field for the email 'mokshith@gmail.com', a password field with a light blue background and masked characters '*****', and a confirm password field with a light blue background and masked characters '****'. A 'Submit' button is located at the bottom of the form. The background of the page is dark.



Result: Confirm password doesn't match

Test Case 4: Empty field validation



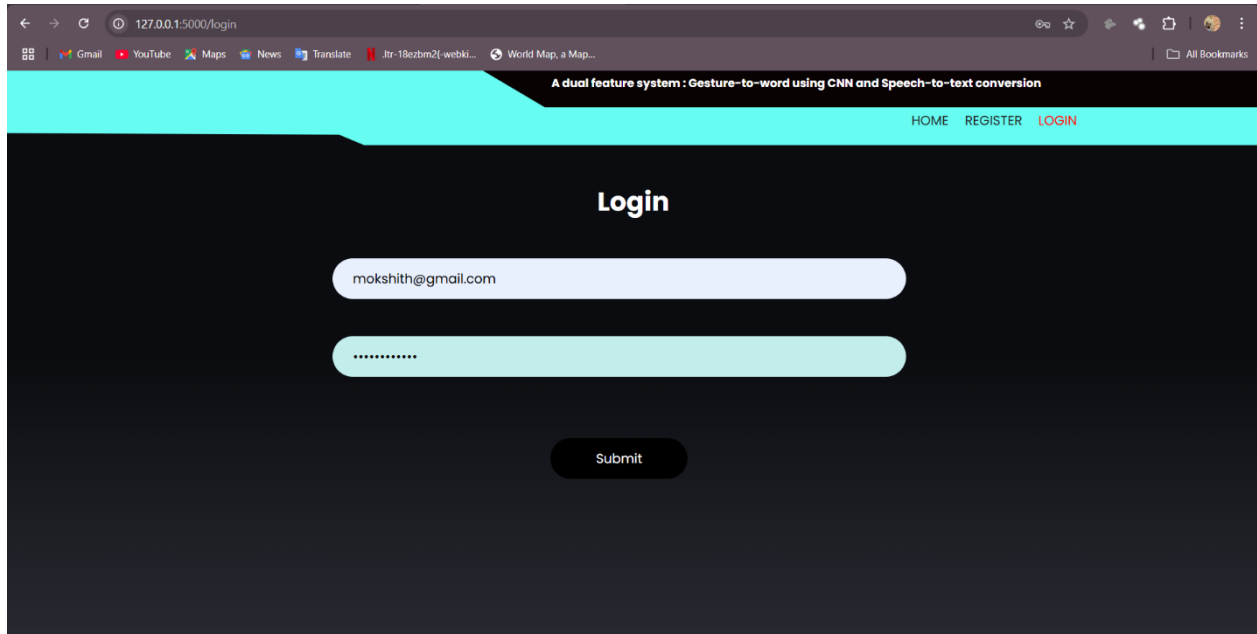


Result: All fields are required.

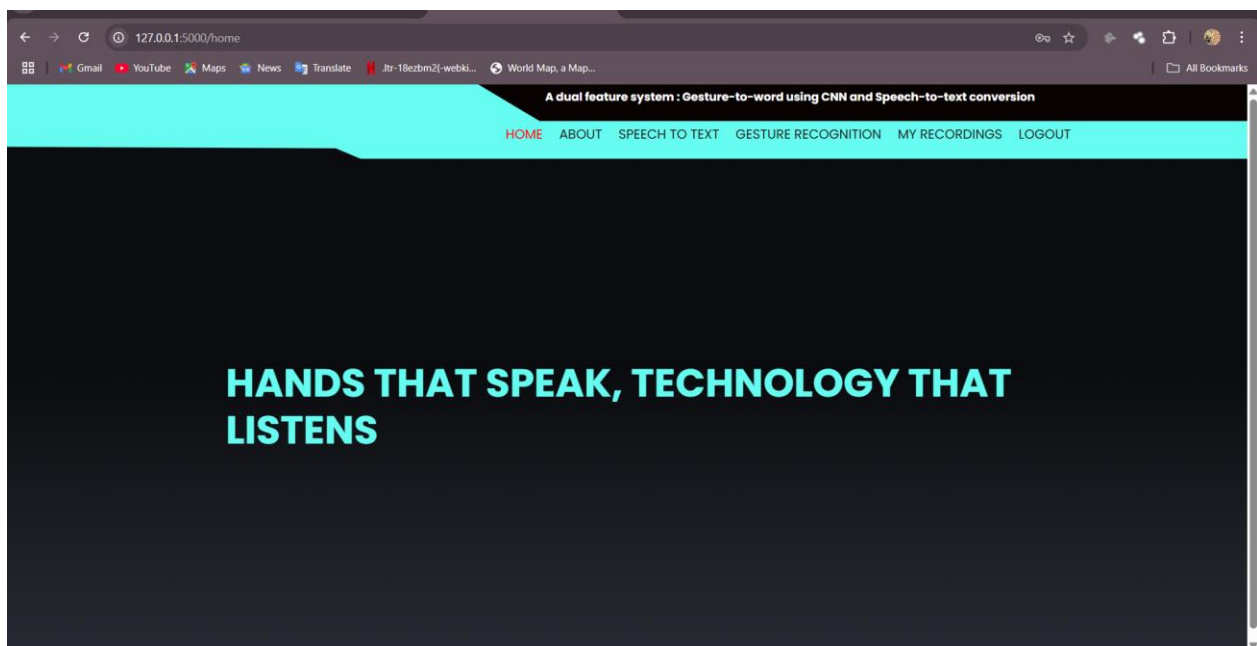
User Authentication:

- **Description:** Verify that a user can authenticate and log in to the speech-to-text and gesture recognition system.
- **Test Steps:**
 1. Access the system's login page hosted on the local server using XAMPP (Apache and MySQL).
 2. Enter valid credentials (username and password) registered previously.
 3. Click on the "Login" button.
 4. Expected Results: The system should authenticate the user and redirect them to the home page, where they can access speech-to-text and gesture recognition functionalities.
 5. Actual Results: User "testuser" successfully signed into the website using XAMPP-hosted MySQL database. The credentials entered (username: testuser, password: testPass@123) matched the stored credentials in the MySQL database. The system authenticated the user correctly, and the user was redirected to the home page, enabling access to upload audio and start the webcam for gesture recognition.

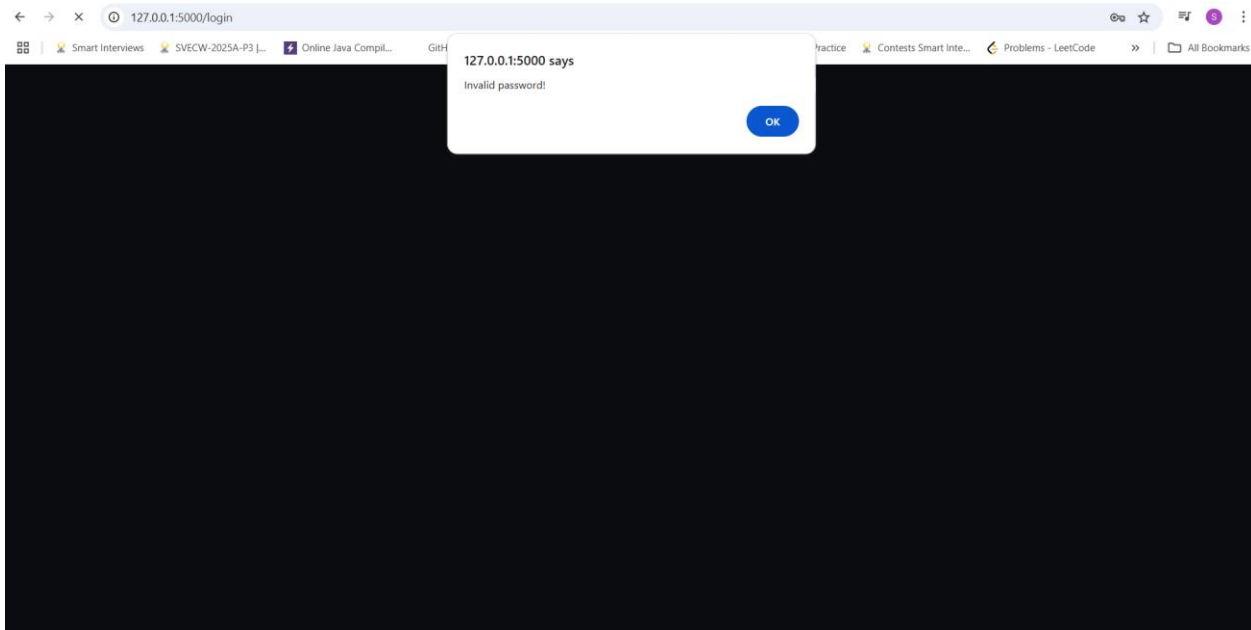
Test Case1: Registered user is logged in successfully.



Result: Home page is opened.



Test Case 2: Invalid Credentials



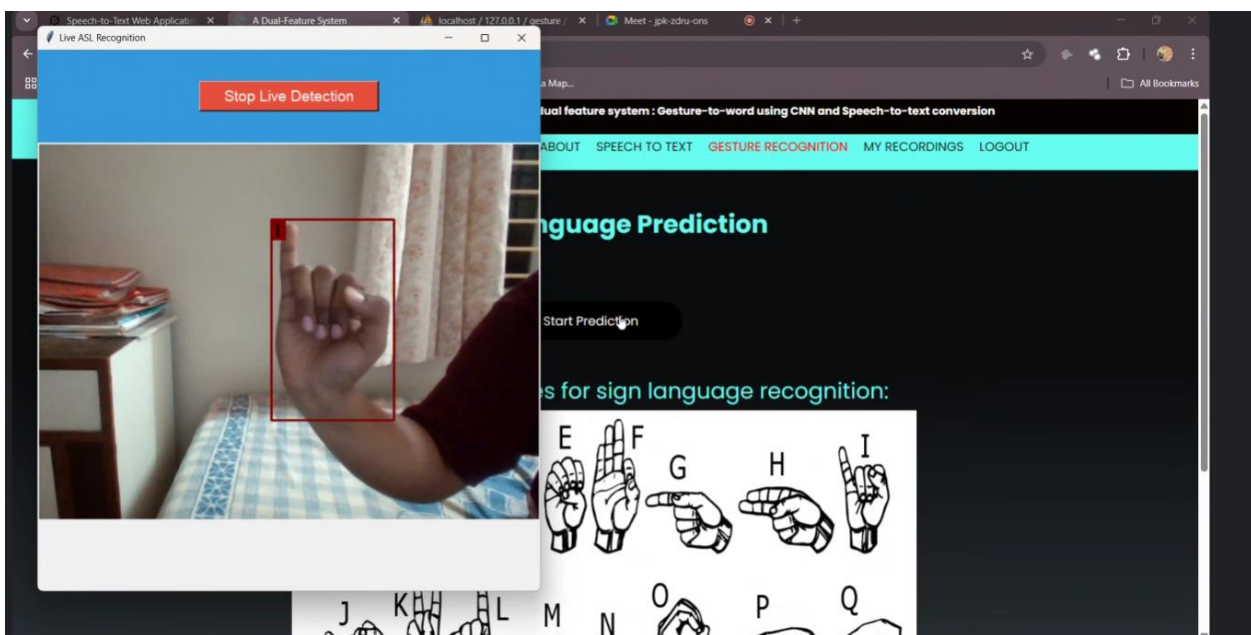
Result: Invalid Password prompt is displayed

Gesture Input Functionality:

- **Description:** Verify that the gesture input system correctly captures user gestures via the webcam, processes them using the prediction engine, and displays the results accurately, including confirming predicted letters and handling spaces between words.
- **Test Steps:**
 1. Access the prediction page of the gesture and audio recognition system by navigating to the /prediction route (e.g., <http://localhost:5000/prediction>).
 2. Click the "Start Recording" button to initiate the /open_webcam route, which triggers the live.py script.
 3. Ensure a webcam is connected and active, then perform a predefined gesture (e.g., a hand sign representing a letter like "A") in front of the webcam.
 4. Observe the real-time output on the window, where the system predicts an alphabet based on the gesture.
 5. Press the Enter key to confirm the predicted letter (e.g., "A") and add it to the output text.
 6. Perform another gesture (e.g., a space gesture or a specific sign) and press the Space key to insert a space between words.
 7. Repeat steps 3-6 for additional letters (e.g., "B") and spaces to form a word or phrase (e.g., "A B").
 8. Stop the webcam feed by clicking a stop button (if implemented).
- **Expected Results:**
 - The webcam feed should start successfully, displaying the user's gestures in real-time.

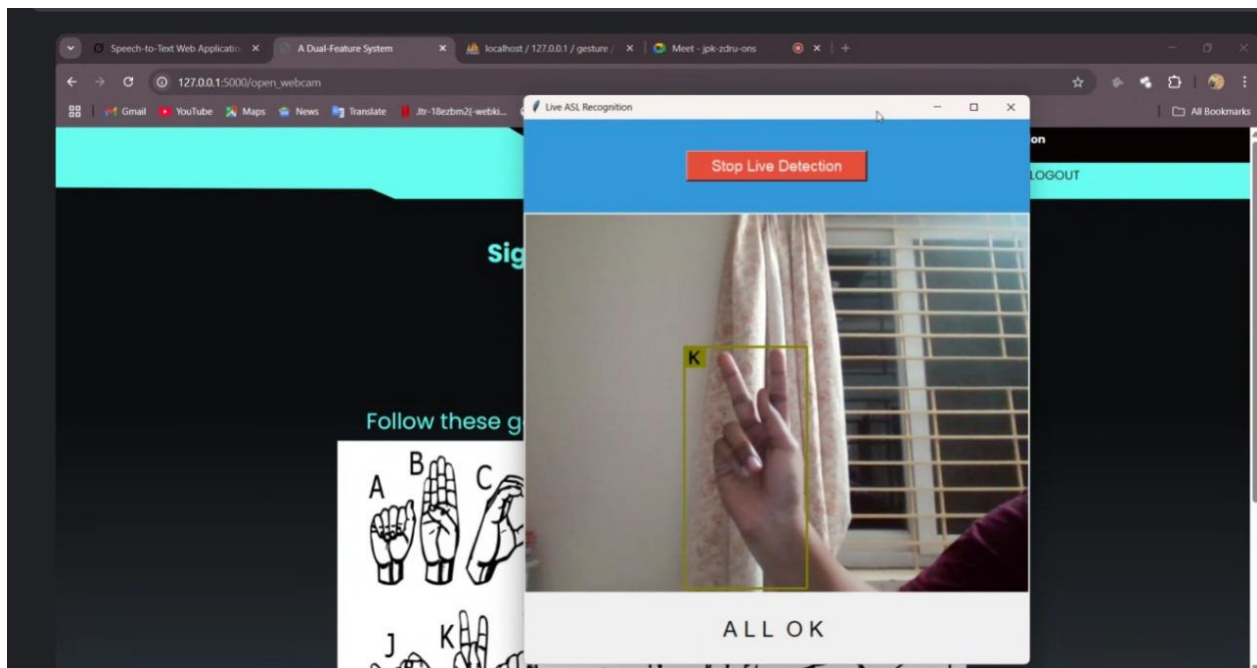
- The system should process the gesture input using the prediction engine (TensorFlow/Keras model with cvzone) and generate a prediction of an alphabet.
 - The PredictResults should be accurately computed and displayed on the prediction.html page or a related interface, matching the performed gesture (e.g., gesture for "A" predicts "A").
 - Pressing the Enter key should confirm the predicted letter and append it to the output text (e.g., typing "A" after confirmation).
 - Pressing the Space key should insert a space between confirmed letters or words (e.g., "A B" after confirming "A", spacing, and "B").
 - The webcam feed should stop cleanly without errors when terminated.
- **Validations:**
 - Webcam Activation: The system must detect and activate the webcam without errors (e.g., no "Camera not found" messages).
 - Gesture Detection: The prediction engine must recognize the gesture within a reasonable time frame (e.g., <2 seconds) and with acceptable accuracy (e.g., >80% for predefined alphabet gestures).
 - Result Display: The displayed result must correspond to the performed gesture (e.g., gesture for "A" displays "A" after Enter is pressed).
 - Letter Confirmation: Pressing the Enter key must correctly confirm the predicted letter and update the output text without duplication or errors.
 - Space Handling: Pressing the Space key must insert a space between confirmed letters or words accurately, ensuring proper word separation.

Testcase1: Prediction of a single alphabet one at a time



Result: Corresponding single alphabet is predicted.

Testcase 2: Formation of a word by confirming alphabet by clicking 'Enter' key and 'Spacebar' for spaces.



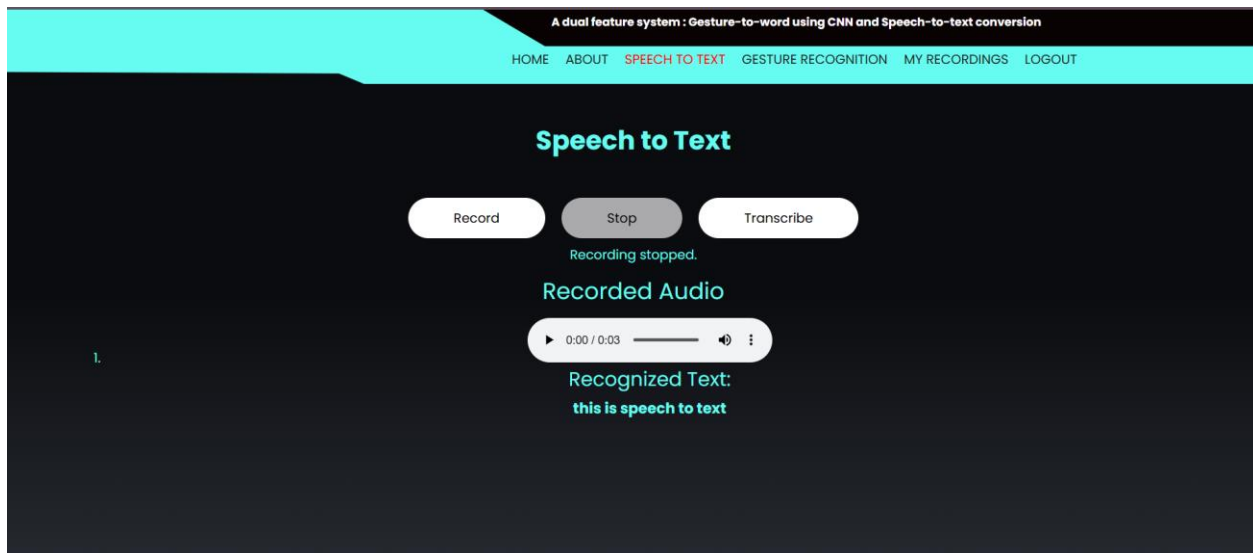
Result: A word is formed from the corresponding alphabets by confirming alphabet by clicking 'Enter' key and 'Spacebar' for spaces.

Speech to Text Module:

- **Description:** Verify that the user can successfully use the speech-to-text module to record audio, stop recording, and convert spoken words into text using the transcribe functionality.
- **Test Steps:**
 1. Access the home page after logging in as a user on the local server using XAMPP (Apache and MySQL).
 2. Navigate to the speech-to-text module from the home page.
 3. Click the "Start Recording" button to begin audio capture.
 4. Speak a short phrase (e.g., "Hello, this is a test").
 5. Click the "Stop Recording" button to end the audio capture.
 6. Click the "Transcribe" button to convert the recorded audio into text.
- **Expected Results:** The system should successfully record the spoken phrase, stop the recording, and upon clicking the "Transcribe" button, convert the audio into text (e.g., "Hello, this is a test") using the Google Web Speech API. The recognized text should be displayed to the user, and the audio file with the processed text should be saved in "My Recordings."
- **Actual Results:** User navigated to the speech-to-text module after logging in. The user clicked "Start Recording," spoke "Hello, this is a test," and then clicked "Stop Recording."

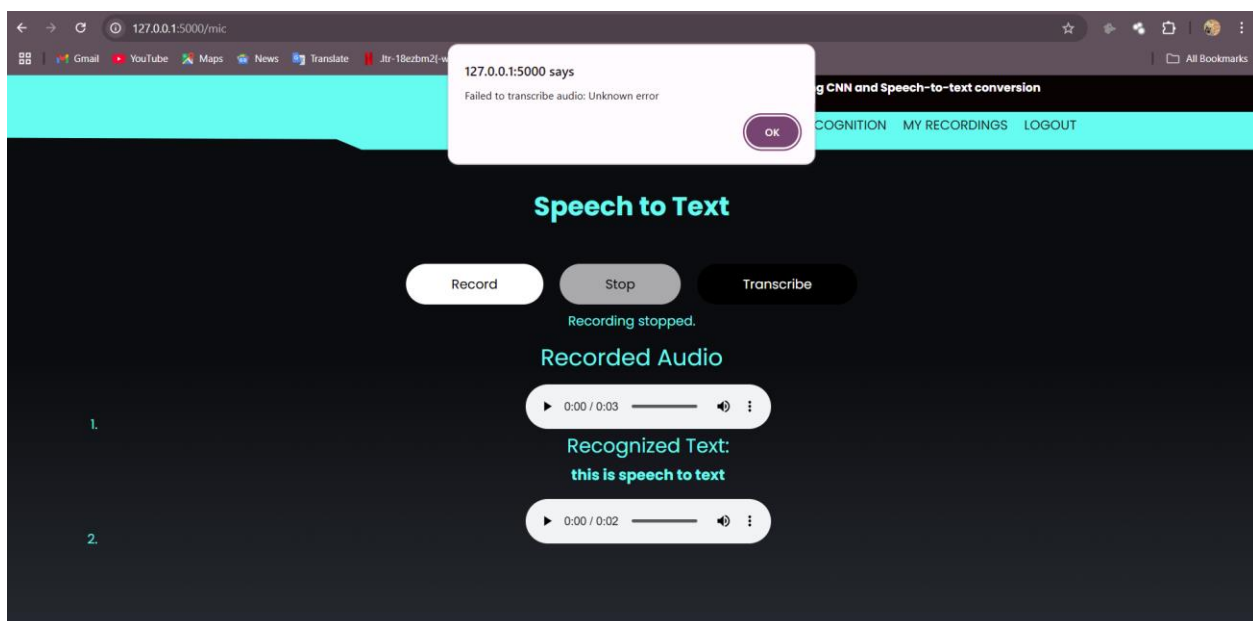
Upon clicking the "Transcribe" button, the system processed the audio using the Google Web Speech API and displayed the text "Hello, this is a test" correctly. The audio file with the processed text was successfully saved in "My Recordings" within the XAMPP-hosted environment.

Test case 1: User records speech



Result: User spoken speech is recorded as a mp3 file and able to be played.

Test Case 2: User doesn't speak while recording



Result: error pops up

My Recordings Storage and Deletion:

- **Description:** Verify that the user's spoken audio file and its corresponding transcribed text are stored in the "My Recordings" section with the recording date, and that the user can delete a recording using the delete option.

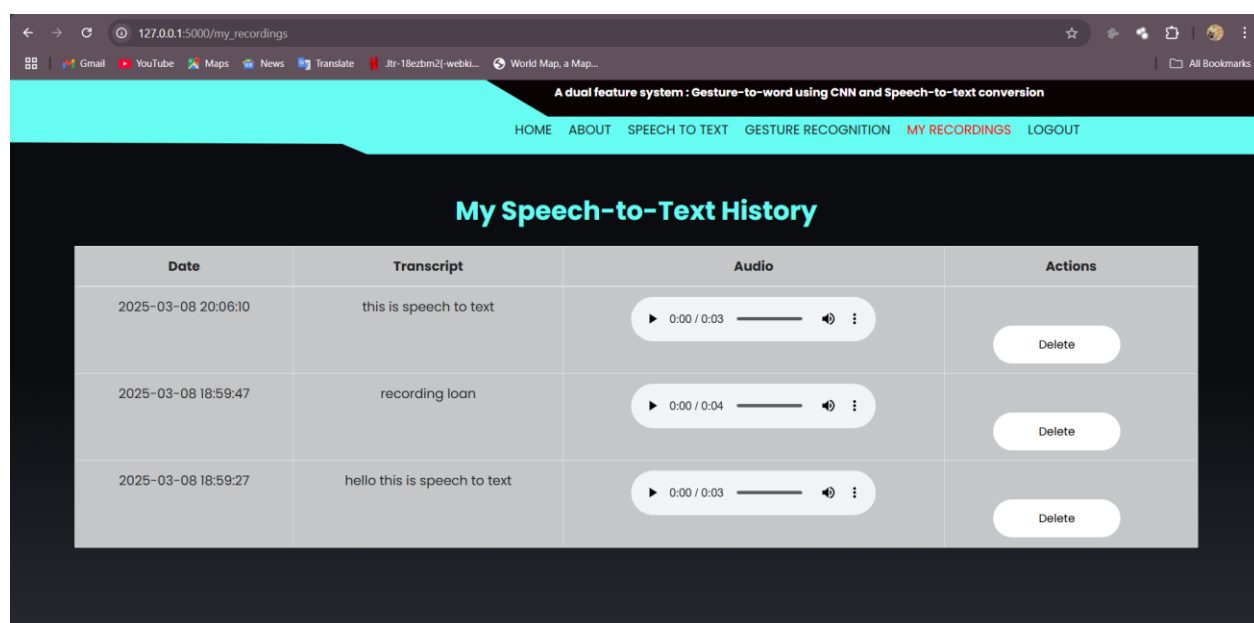
- **Test Steps:**

1. Access the home page after logging in as a user on the local server using XAMPP (Apache and MySQL).
2. Navigate to the speech-to-text module and record a short audio (e.g., speak "This is a test recording").
3. Stop the recording and click the "Transcribe" button to convert the audio into text.
4. Navigate to the "My Recordings" section to view the stored audio file and its transcribed text.
5. Verify the date of the recording is displayed alongside the audio file and transcribed text.
6. Select the recorded audio file and click the "Delete Recording" option to remove it.

- **Expected Results:** The system should store the recorded audio file and its transcribed text (e.g., "This is a test recording") in the "My Recordings" section, along with the correct recording date (e.g., March 08, 2025). The user should be able to delete the recording, and the audio file and its corresponding text should no longer appear in the "My Recordings" section after deletion.

Actual Results: User recorded an audio saying "This is a test recording" in the speech-to-text module. After transcription, the audio file and its transcribed text "This is a test recording" were successfully stored in the "My Recordings" section in the XAMPP-hosted MySQL database. The recording date was displayed as "March 08, 2025." The user then selected the recording and clicked "Delete Recording," which removed the audio file and its transcribed text from the "My Recordings" section as expected, with no trace of the recording remaining in the database.

Test case: History of recordings

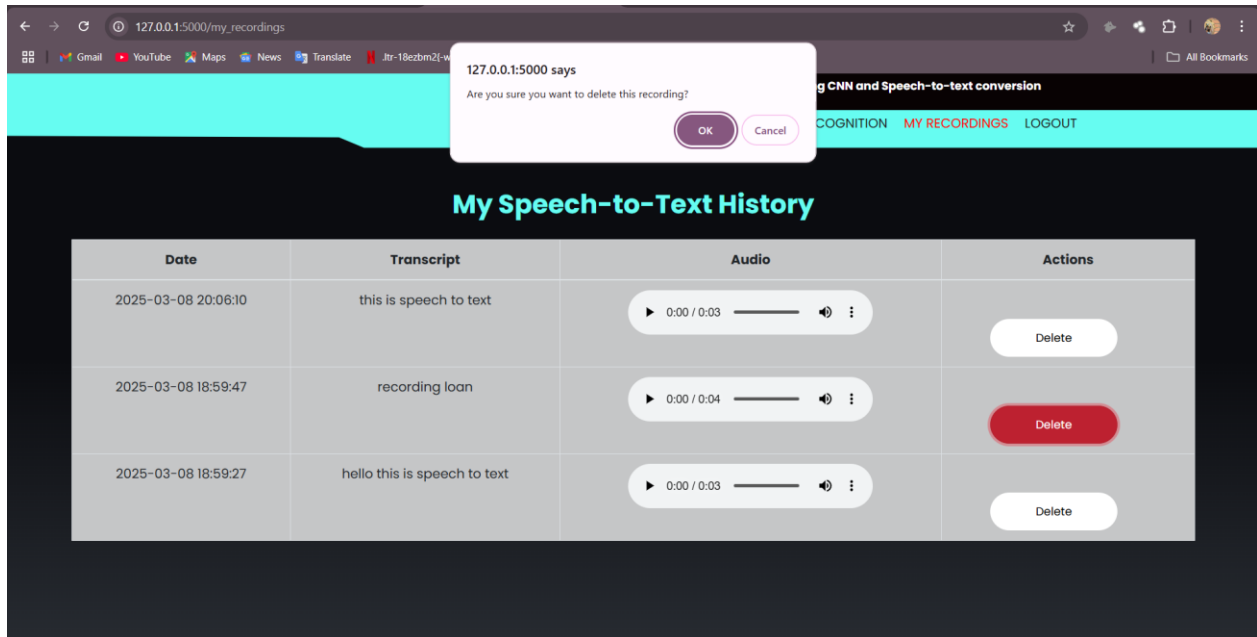


The screenshot displays the 'My Speech-to-Text History' page of a web application. The page has a dark background with a light blue header bar. The header bar contains the text 'A dual feature system : Gesture-to-word using CNN and Speech-to-text conversion' and navigation links: HOME, ABOUT, SPEECH TO TEXT, GESTURE RECOGNITION, MY RECORDINGS (highlighted), and LOGOUT. Below the header, the title 'My Speech-to-Text History' is centered. A table with four columns is shown: Date, Transcript, Audio, and Actions. The table contains three rows of recorded audio files. Each row has a 'Delete' button in the 'Actions' column.

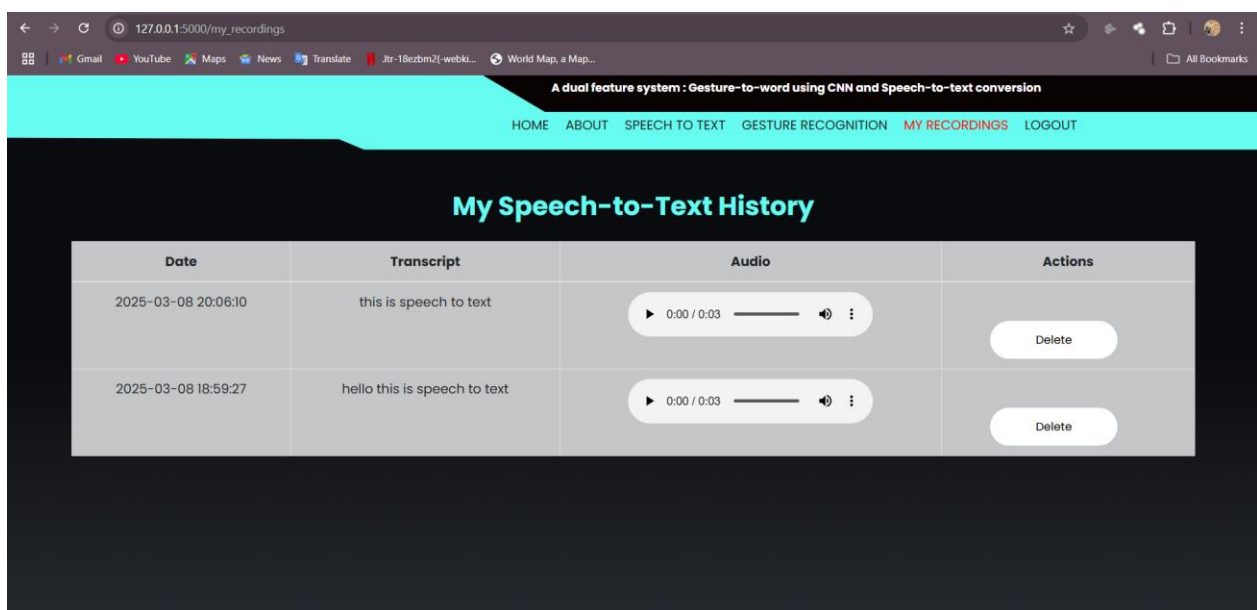
Date	Transcript	Audio	Actions
2025-03-08 20:06:10	this is speech to text	0:00 / 0:03	Delete
2025-03-08 18:59:47	recording loan	0:00 / 0:04	Delete
2025-03-08 18:59:27	hello this is speech to text	0:00 / 0:03	Delete

Result: A page with all recording history is displayed

Test Case: Deleting a recording.



after confirming that particular recording will be deleted.



Result: Selected recording is deleted

User Logout:

- **Description:** Verify that a user can successfully log out of the speech-to-text and gesture recognition system.
- **Test Steps:**
 1. Access the system's home page after logging in as a user on the local server using XAMPP (Apache and MySQL).
 2. Locate and click on the "Logout" button or link on the home page or user dashboard.

3. Confirm the logout action if prompted by the system.
- **Expected Results:** The system should terminate the user's session, log the user out, and redirect them to the login page.
 - **Actual Results:** User "testuser123" was logged into the website using the XAMPP-hosted MySQL database. After clicking the "Logout" button on the home page, the system terminated the session successfully. The user was redirected to the login page, and attempts to access the home page again required re-authentication with valid credentials.

7.CONCLUSION

In conclusion, the integration of deep learning and speech recognition techniques in Real-Time demonstrates the potential of technology in enhancing accessibility for individuals with speech and hearing impairments. This project successfully employs MobileNet as a feature extractor for American Sign Language (ASL) gesture recognition and leverages the Google Web Speech API for real-time speech-to-text conversion, creating an effective and inclusive dual-feature system.

Our work began with an extensive study of assistive technologies, emphasizing the need for real-time solutions that connects the deaf and mute community. Recognizing the advancements in convolutional neural networks (CNNs) and pre-trained models, we adopted MobileNet to extract meaningful features from ASL gesture images while ensuring computational efficiency. Additionally, we incorporated SpeechRecognition and the Google-Web-Speech API to process speech input, enabling seamless real-time transcription.

The implementation phase involved dataset collection, preprocessing, and model fine-tuning to optimize performance.

Data augmentation techniques, such as rotation, scaling, and flipping, enhanced the diversity and robustness of the gesture recognition model. By leveraging Global Average Pooling instead of traditional oversampling techniques, we improved model efficiency while avoiding biases introduced by imbalanced datasets. Furthermore, the speech module was designed to handle real-time inputs efficiently, ensuring minimal latency and high transcription accuracy.

Evaluation results indicate that the system effectively classifies ASL gestures and converts speech into text with good precision. The gesture recognition model performed well across varying lighting conditions, hand orientations, and backgrounds, showcasing its adaptability to real-world scenarios. Similarly, the speech recognition system, supported by Google's API, provided reliable transcriptions, even in noisy environments. The integration of both modalities into a user-friendly interface further enhances the usability and practicality of the system.

Beyond technical contributions, this project highlights the broader impact of AI-driven assistive technologies in fostering inclusivity and accessibility. By providing an intuitive and efficient communication tool, we contribute to a more inclusive society where technology empowers individuals with disabilities. The implementation of real-time speech and gesture recognition paves the way for further research in multimodal systems, sign language translation, and human-computer interaction.

Moving forward, future work may explore the integration of natural language processing (NLP) for contextual understanding, expanding the ASL vocabulary for more comprehensive communication, and developing mobile-friendly applications for wider accessibility. Collaborations with linguists, speech therapists, and accessibility researchers could further enhance the system's effectiveness and usability.

In conclusion, this project successfully demonstrates the potential of deep learning and AI in assistive communication, bridging communication barriers for individuals with hearing and speech impairments. By harnessing the power of CNN-based gesture recognition and speech-to-text conversion, we take a significant step toward creating a more inclusive and accessible world where technology serves as a vital enabler of human interaction.

8. Bibliography

- [1] Abhishek B, Kanya Krishi, Meghana M, Mohammed Daaniyaal, Anupama H S. "Hand Gesture Recognition Using Machine Learning Algorithms."
- [2] Syed Raquib Shareef, Mohammed Mannan Hussain. "Hand Gesture Recognition System for Deaf and Dumb."
- [3] Shanthini, E., Akshaya, N., & Haritha, J. (2023, June). Hand Gesture Vocalizer using MobileNetV2 for Deaf Mutes. In 2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN) (pp. 1481-1485). IEEE.
- [4] Tiwari, A., Gangrade, A., Tiwari, A., & Bandhu, K. C. (2022). Design and Implementation of Conversion of Gesture to Voice Using OpenCV and Convolution Neural Network. In Information and Communication Technology for Competitive Strategies (ICTCS 2020) ICT: Applications and Social Interfaces (pp. 921-930). Springer Singapore.
- [5] Kim, S. Y., Urm, S. J., Yoo, S. Y., Kim, S. J., & Lee, K. M. (2023). Application of Sign Language Gesture Recognition Using Mediapipe and LSTM. *Journal of Digital Contents Society*, 24(1), 111-119.
- [6] Bogdan Iancu. "Evaluating Google Speech-to-Text API's Performance for Romanian e-Learning Resources"
- [7] Dr. Kavitha R., Nachammai N., Ranjani R., Shifali J. "Speech Based Voice Recognition System for Natural Language Processing"
- [8] Reddy, V. M., Vaishnavi, T., & Kumar, K. P. (2023, July). Speech-to-Text and Text-to-Speech Recognition Using Deep Learning. In 2023 2nd International Conference on Edge Computing and Applications (ICECAA) (pp. 657-666). IEEE.
- [9] Amrutha, K., & Prabu, P. (2023). Evaluating the pertinence of pose estimation model for sign language translation. *International Journal of Computational Intelligence and Applications*, 22(01), 2341009.
- [10] Elakkiya, A., Surya, K. J., Venkatesh, K., & Aakash, S. (2022, December). Implementation of speech to text conversion using hidden markov model. In 2022 6th International Conference on Electronics, Communication and Aerospace Technology (pp. 359-363). IEEE.
- [11] Raju, M. D. N. G., Reddy, A. R., Pranith, P. S., Nikhil, L. S., Pasha, S., & Bhat, P. V. (2022). Hand Gesture Recognition And Voice Conversion For Deaf And Dumb. *Journal of Positive School Psychology*, 6(7), 4953-4960.
- [12] Aasofwala, N., Verma, S., & Patel, K. (2023, July). NLP based model to convert English speech to Gujarati text for deaf & dumb people. In 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE.

9.Appendix

9.1 Appendix-A Project Repository Details

Project Title: A Dual-Feature System: Gesture-to-Word using CNN and Speech-to-Text Conversion

Batch: C8

Batch Members:

1. Shahanawaz Sulthana – 21B01A05G4
2. Shaik Faheem Maharoor – 21B01A05G5
3. Tadepalli Harika Naga Durga – 21B01A05H5
4. Vemavarapu Nag Sahithi – 21B01A05J0
5. Vidiyala Rithu Sree – 21B01A05J1

Department: Computer Science and Engineering

Institution: Shri Vishnu Engineering College for Women

Guide: Dr. P. Kiran Sree

Submission Date: 10/03/2025

Project Repository Link

The complete project files, including source code, documentation, and additional resources, are available at the following GitHub repository:

https://github.com/RithuSreeVidiyala/a_dual_feature_system_gesture_to_word_using_cnn_and_speech_to_text_conversion

GitHub Repository:

https://github.com/RithuSreeVidiyala/a_dual_feature_system_gesture_to_word_using_cnn_and_speech_to_text_conversion.git

For quick access, scan the QR code below



9.2 Introduction to Python :

Python has become the dominant language for deep learning due to its simplicity, versatility, and a rich ecosystem of libraries. In the "Gesture-to-Word Using CNN and Speech-to-Text Conversion" system, Python plays a central role in building both gesture recognition and speech processing modules. For gesture recognition, Python libraries like TensorFlow and Keras enable the creation of CNN models, while OpenCV is used for preprocessing images—extracting hand contours and applying transformations to enhance accuracy. These tools collectively support the development of real-time, responsive gesture-to-text conversion systems.

For the speech-to-text module, Python offers powerful libraries such as SpeechRecognition and the Google Web Speech API for converting spoken language into text. Audio processing tools like pydub and librosa are employed for tasks such as noise reduction and feature extraction. For more robust transcription in noisy environments, advanced deep learning models like Wav2Vec2 from Hugging Face can be fine-tuned. These tools provide flexibility and high accuracy, making them ideal for building accessible voice interfaces.

Python's broader ecosystem—featuring libraries like NumPy, Pandas, Matplotlib, and Seaborn—supports data handling and visualization, aiding in model evaluation and debugging. Jupyter Notebooks further enhance productivity by offering an interactive development environment for experimenting with CNN architectures and fine-tuning hyperparameters. With support for transfer learning using models like MobileNetV2 and ResNet50, Python enables efficient training and deployment. Overall, Python's comprehensive toolkit makes it a powerful backbone for developing innovative AI-based assistive technologies.

9.3 Introduction to Convolutional Neural Network (CNN) Algorithm

Convolutional Neural Networks (CNNs) are a specialized class of deep learning models designed for processing structured grid data, primarily images. They have significantly transformed the fields of computer vision, natural language processing, and speech recognition by effectively capturing spatial and hierarchical patterns within data. CNNs utilize convolutional operations to automatically extract relevant features from input data, eliminating the need for manual feature engineering. Their ability to learn spatial hierarchies and recognize patterns makes them highly effective in tasks like object recognition, image classification, and gesture recognition.

Architecture of CNNs

CNNs are composed of multiple layers that work in tandem to process and analyze image data. The core building blocks include convolutional layers, pooling layers, activation functions, and fully connected layers. Convolutional layers apply learnable filters (kernels) to

the input image, capturing essential features such as edges, textures, and patterns. These feature maps serve as the foundation for higher-level pattern recognition. Pooling layers, such as max pooling and average pooling, reduce the spatial dimensions of feature maps, thereby improving computational efficiency and ensuring translation invariance. Activation functions like ReLU (Rectified Linear Unit) introduce non-linearity, allowing CNNs to learn complex relationships within the data. The final layers are fully connected layers, which map the extracted features to the desired output, such as classification labels in an image recognition task.

Training Process

The training process of CNNs involves backpropagation and gradient descent optimization to fine-tune model parameters. CNNs minimize a predefined loss function by adjusting weights and biases through iterative learning. During forward propagation, the input passes through the network, and the output is compared with the expected result to compute the loss. In backward propagation, gradients of the loss function with respect to network parameters are computed using techniques like stochastic gradient descent (SGD), Adam, or RMSprop. The model updates its parameters based on these gradients to improve accuracy over multiple iterations. With each training cycle, CNNs become more adept at recognizing patterns and making accurate predictions.

Key Components and Techniques

Several advanced techniques enhance the efficiency and performance of CNNs. Batch normalization standardizes the activations of each layer, accelerating training and improving convergence. Dropout regularization prevents overfitting by randomly deactivating certain neurons during training, ensuring the network generalizes well to unseen data. Data augmentation techniques such as image rotation, flipping, scaling, and cropping artificially expand the training dataset, improving the model's ability to handle variations in real-world scenarios. Additionally, Transfer Learning allows CNNs to leverage pre-trained models like MobileNet, ResNet, and VGG for tasks with limited training data, significantly reducing computational costs while maintaining high accuracy.

Applications of CNNs

CNNs have found extensive applications across diverse domains. In image classification, CNNs have set new benchmarks in competitions like ImageNet. Object detection techniques powered by CNNs are widely used in surveillance, autonomous driving, and medical imaging. Facial recognition systems employ CNNs for identity verification in security applications. In medical image analysis, CNNs assist in diagnosing diseases from X-rays, MRIs, and CT scans. Additionally, CNNs are now being integrated into natural language processing (NLP), enabling

advancements in text classification, sentiment analysis, and speech recognition. In gesture-based communication, CNNs play a crucial role in recognizing sign language, enabling accessibility solutions for individuals with hearing or speech impairments.

In conclusion, CNNs have revolutionized deep learning applications by providing an efficient mechanism for learning hierarchical features from image and video data. Their ability to automatically extract and refine features has made them indispensable in fields like computer vision, healthcare, security, and human-computer interaction. As research in deep learning progresses, CNN architectures continue to evolve, unlocking new possibilities for real-time AI-driven applications.

9.4 Introduction to MobileNet Architecture :

MobileNet is a lightweight CNN architecture designed for efficient deep learning on mobile and embedded devices. Its core innovation is the depthwise separable convolution, which splits standard convolution into two steps—depthwise convolution (processing each input channel separately) and pointwise convolution (1x1 convolution to combine features across channels). This significantly reduces the model's parameters and computational cost.

MobileNet includes two tunable hyperparameters: the width multiplier, which adjusts the number of channels, and the resolution multiplier, which reduces input image size—both helping control the model's complexity. Variants like MobileNetV2 and V3 introduce optimizations such as inverted residuals and better feature flow for improved speed and accuracy.

Thanks to its lightweight design, MobileNet is ideal for real-time tasks like image classification, object detection, and segmentation on devices with limited resources, such as smartphones, IoT devices, and drones.

9.5 Introduction to HTML:

In the realm of deep learning for gesture-to-word and speech-to-text conversion, integrating an intuitive and user-friendly interface is essential for seamless interaction between users and the underlying model. HTML, as the standard markup language for creating web pages, plays a crucial role in designing the user interface (UI) for processing live-cam feed and recording speech input. This section explores the significance of HTML in this context, emphasizing its role in improving accessibility and engagement in gesture and speech recognition systems.

HTML, or HyperText Markup Language, serves as the fundamental building block of the World Wide Web. It structures content on web pages by defining elements such as headings, paragraphs, images, forms, and buttons. In the context of this project, HTML provides the foundation for the UI, facilitating users in processing live-cam feed for recognition and

recording speech for transcription. By structuring the interface with HTML, developers ensure that the platform remains intuitive and user-friendly.

The first step in engaging users in gesture-to-word and speech-to-text conversion is to create an intuitive input interface. The `<button>` element can be utilized to trigger the live-cam and speech recording process, ensuring users can interact effortlessly with the system. This simple yet effective setup encourages users to engage with the application and leverage its functionalities with ease.

HTML5 introduces advanced features that further enrich the user experience. The `accept` attribute in file upload inputs can be used to restrict submissions to specific formats, ensuring compatibility with the recognition model. Additionally, HTML5's `<audio>` and `<video>` elements enhance the interface by allowing users to preview their recorded speech or uploaded gestures before submission. Features like drag-and-drop functionality for images improve convenience, making the process more interactive.

In conclusion, HTML serves as a fundamental component in developing user interfaces for gesture-to-word and speech-to-text conversion systems. Its simplicity, compatibility with other technologies, and ability to enhance the user experience make it an indispensable tool for creating intuitive input interfaces. By integrating HTML with backend technologies and incorporating responsive design principles, developers can build an accessible and engaging platform that fosters user interaction with deep learning-powered communication tools.

9.6 Introduction to CSS:

In the dynamic landscape of deep learning applications for gesture-to-word and speech-to-text conversion, the user interface (UI) plays a vital role in shaping user experience. CSS, or Cascading Style Sheets, emerges as a powerful companion to HTML, enabling developers to enhance the aesthetics and usability of the UI. This section explores the significance of CSS in designing a visually appealing and functional interface for uploading gestures and recording speech.

CSS serves as the stylistic backbone of web development, allowing developers to define the presentation and layout of HTML elements. In this project, CSS is instrumental in ensuring that the UI is not only functional but also visually appealing. By leveraging CSS, developers can apply colors, fonts, and design elements that improve readability and user engagement, ensuring a seamless experience for users interacting with the system.

CSS operates on the principle of cascading styles, ensuring a systematic and organized approach to styling HTML elements. Additionally, it provides cross-browser compatibility, ensuring that the interface appears consistent across different web browsers. This is particularly important for gesture and speech recognition applications, as users may access the system from a variety of browsers.

In conclusion, CSS is a fundamental element in the development of user interfaces for gesture-to-word and speech-to-text conversion systems. Its role extends beyond aesthetics, encompassing responsive design, animations, and branding customization. By integrating CSS with HTML and JavaScript, developers can create visually appealing, user-friendly interfaces that enhance engagement and accessibility, ultimately improving the overall usability of the deep learning-powered application.

9.7 Introduction to JavaScript:

JavaScript, a versatile and dynamic programming language, plays a crucial role in enhancing the gesture-to-word and speech-to-text conversion system's user interface. It provides interactivity, real-time feedback, and seamless communication between the frontend and backend, ensuring an engaging and responsive user experience. This section explores JavaScript's role in improving the UI and user interaction within this project.

JavaScript enables real-time feedback mechanisms, improving user experience during live-cam feed and speech recording. By implementing event listeners, JavaScript can monitor user actions, validate file formats, and dynamically update the UI. For example, if a user is giving the gesture as live-cam feed, JavaScript can display a preview, ensuring that the correct file is selected. Similarly, during speech recording, a visual indicator can inform users that the system is actively capturing audio.

By handling events such as clicks, hovers, and key presses, JavaScript enhances user engagement. For instance, JavaScript can enable a "Start Recording" button to change dynamically when the speech recognition process begins and stops. These interactive elements make the interface more intuitive and user-friendly.

In conclusion, JavaScript is an essential component in the development of user interfaces for gesture-to-word and speech-to-text conversion applications. Its ability to enhance interactivity, provide real-time feedback, and enable seamless backend communication makes it indispensable for creating an intuitive and engaging user experience. By integrating JavaScript with HTML and CSS, developers can craft a powerful, interactive interface that bridges the gap between users and deep learning models.

9.8 Introduction to Flask Framework:

Flask, a lightweight and modular web framework in Python, is ideal for deploying deep learning applications like the "Gesture-to-Word Using CNN and Speech-to-Text Conversion" system. Its microservices architecture allows the gesture recognition and speech-to-text modules to be developed as independent, scalable components. Flask provides clean routing mechanisms using decorators like `@app.route()` to handle HTTP requests—essential for processing gesture image uploads and audio inputs in real time. With its minimal setup and flexibility, Flask easily integrates with deep learning frameworks like TensorFlow, Keras, OpenCV, and speech libraries, enabling quick deployment of trained models.

To handle data management and user input, Flask leverages tools like Flask-SQLAlchemy for seamless interaction with databases, storing classification results and transcripts efficiently. It supports Flask-WTF forms and POST requests for capturing user-uploaded gesture images and voice recordings, validating them before processing. The project's functionality is exposed via RESTful APIs using Flask-RESTful, with endpoints such as `/predict_gesture`, `/transcribe_speech`, and `/speak_text`, enabling integration with web clients or mobile apps. Security is ensured through Flask-Login and Flask-JWT-Extended, providing user authentication and protection against threats using CORS, CSRF tokens, and encryption.

To ensure real-time responsiveness, Flask is deployed using production-ready WSGI servers like Gunicorn and platforms like Heroku, AWS, or Google Cloud. It also integrates Celery for asynchronous execution of resource-intensive tasks, such as gesture inference or speech transcription, ensuring smooth multitasking. In conclusion, Flask's flexibility, scalability, and compatibility with AI tools make it the perfect backbone for delivering an interactive, accessible web application for assistive communication.

9.9 Introduction to Transfer Learning Architecture:

Transfer Learning is a powerful deep learning technique that allows a model trained on a large dataset to be adapted for a new but related task. In the context of "A Dual-Feature System: Gesture-to-Word Using CNN and Speech-to-Text Conversion," Transfer Learning significantly enhances the accuracy and efficiency of the gesture recognition module by leveraging existing deep learning models trained on extensive image datasets. Instead of training a model from scratch, Transfer Learning enables the reuse of pre-trained models like MobileNetV2, ResNet50, or VGG16, which have already learned essential features from large-scale datasets such as ImageNet.

A Transfer Learning model typically consists of two main components: the feature extractor and the task-specific classifier. The lower layers of a deep neural network, known as the feature extractor, capture fundamental visual patterns such as edges, textures, and shapes.

These learned representations remain useful across different image recognition tasks, including gesture recognition. The upper layers, which form the classifier, are modified and fine-tuned to classify specific hand gestures corresponding to American Sign Language (ASL) alphabets or words. This approach reduces the computational effort required to train the model while improving its performance on the new dataset.

The implementation of Transfer Learning for gesture recognition follows a structured process. First, a suitable pre-trained model is selected based on its ability to balance accuracy and computational efficiency. Models such as MobileNetV2 are preferred due to their lightweight architecture, making them well-suited for real-time applications. The initial layers of the model are frozen, preserving the general visual features learned from large-scale datasets. A new fully connected classifier is then added to the model, replacing the original layers to tailor the network specifically for ASL gesture classification. To further improve accuracy, selective layers of the model are fine-tuned by gradually unfreezing them and retraining on the gesture dataset. The model is then optimized using techniques such as categorical cross-entropy loss and the Adam optimizer, ensuring it achieves high classification performance.

In this project, Transfer Learning plays a key role in the Gesture-to-Word Module by enabling efficient ASL gesture classification. The pre-trained MobileNetV2 model is fine-tuned on an ASL dataset, allowing real-time recognition of hand gestures captured through a webcam. The recognized gestures are then mapped to corresponding words, facilitating seamless communication for individuals with speech and hearing impairments. Additionally, while the Speech-to-Text Module primarily relies on Google's Web Speech API, Transfer Learning techniques can also be applied to fine-tune pre-trained speech recognition models such as Wav2Vec2 or DeepSpeech, improving their transcription accuracy.

Overall, Transfer Learning proves to be a crucial component in this project, ensuring high accuracy, efficiency, and scalability for real-time gesture recognition. By integrating a pre-trained MobileNetV2 model and fine-tuning it for ASL classification, the system achieves reliable performance while minimizing computational costs.