CBS1904 – Capstone Project

SignBridge: A Real-Time ASL To Speech Interpreter

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

(with specialization in Business Systems)

by

21BBS0098	**Ishika Lalwani**
21BBS0231	**Yasha Ramesh Shetty**

Under the Supervision of

Dr. Srivani A

Assistant Professor Sr. Grade 1

School of Computer Science and Engineering (SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

April 2025

DECLARATION

I hereby declare that the project entitled **SignBridge: A Real-Time ASL To Speech Interpreter** submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Srivani A.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 25/04/2025

yasha

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled **SignBridge: A Real-Time ASL To Speech Interpreter** submitted by Yasha Ramesh Shetty (21BBS0231), **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by her under my supervision during Winter Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore
Date : 25/04/2025

Signature of the Guide

VELLORE INSTITUTE OF TECHNOLOGY (VIT) Deemed to be University under section 3 of UGC Act, 1956 School of Computer Science & Engineering (SCOPE) Vellore-632014, T.N., India

Internal Examiner

External Examiner

Dr. Rajkumar S

Department of Analytics

EXECUTIVE SUMMARY

Sign language serves as a potent nonverbal communication method that utilizes physical gestures in place of spoken language to convey messages. It is essential for individuals who are hearing-impaired or speech-impaired, allowing them to articulate their thoughts, feelings, and information. Sign languages depend on a mix of manual movements—such as hand shapes and motions—combined with non-manual signals like facial expressions, eye contact, and body posture. The core components of sign language consist of hand configuration, palm direction, movement trajectory, location, and non-manual cues such as eyebrow actions or mouth formations.

Even with its depth and efficiency within the deaf community, communication between hearing and non-hearing people frequently presents significant obstacles, particularly in communities where sign language interpreters are scarce. This disconnect creates barriers that may result in social isolation, restricted access to resources, and educational disadvantages for those who are hearing-impaired.

To tackle this challenge, our initiative seeks to create a real-time sign language interpretation system that can recognize signed gestures and translate them into both text and spoken language. By employing a blend of computer vision, machine learning, and natural language processing, the system captures hand movements through a webcam, classifies them using a trained machine learning model, and subsequently produces the related text and synthesized speech through a text-to-speech (TTS) engine.

The system is intended to execute real-time gesture tracking utilizing tools such as MediaPipe for hand landmark identification, alongside a machine learning classifier—like a Random Forest model—to identify specific signs. The identified signs are then combined to form coherent words and sentences, incorporating logic for spacing, timing, and sentence structure. This ensures that the output is not merely a sequence of letters, but a clear representation of the intended communication.

This application has the capability to act as a communication resource, functioning as a virtual interpreter in educational, social, and workplace environments. Furthermore, it can function as an interactive educational tool for those learning sign language, enhancing accessibility for the wider public and encouraging inclusiveness.

By harnessing advanced technologies and amalgamating various fields of computer science—including image processing, supervised learning, real-time data management, and speech synthesis—this project highlights the potential of technology in addressing real-life social issues. Its

main aim is to empower the hearing-impaired community, enhance communication, and encourage a more inclusive society.

Ultimately, this project is not solely a technical pursuit, but a movement towards digital equality and human-centered design—enabling every individual, regardless of physical limitations, to have the means to communicate, connect, and flourish.

ACKNOWLEDGEMENT

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Jaisankar N, Dean - School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence.

I express my profound appreciation to Dr. Rajkumar S, the Head of the Department of Analytics for his insightful guidance and continuous support. His expertise and advice have been crucial in shaping throughout the course. His constructive feedback and encouragement have been invaluable in overcoming challenges and achieving goals.

I am immensely thankful to my project supervisor, Dr. Srivani A, for her dedicated mentorship and invaluable feedback, her patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share her expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. Her support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

Yasha Ramesh Shetty
Ishika Lalwani

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

1. INTRODUCTION

1.1 BACKGROUND

Effective communication is essential for sharing ideas, emotions, and information, yet traditional communication methods can be challenging for the hearing-impaired community. While sign language is a crucial tool for expression among hearing-impaired individuals, it often remains inaccessible to the wider population due to its complexity and limited general understanding.

To bridge this gap, this project introduces an innovative Sign Language Translator built with Python and machine learning, utilizing advanced modules such as Mediapipe, Landmark detection, and Random Forest algorithms. The main goal is to create a system capable of accurately converting sign language gestures into both spoken and written language, thereby enabling smoother interactions between hearing-impaired individuals and those unfamiliar with sign language.

Machine learning, a branch of artificial intelligence, excels at recognizing patterns and processing language, making it ideal for interpreting the intricate gestures of sign language in real time. The integration of the Mediapipe library and Landmark module allows for precise hand tracking and landmark detection, which are vital for reliable gesture recognition. The Random Forest algorithm further strengthens the system by providing a robust and adaptable model for classifying sign language gestures.

By combining cutting-edge technologies, SignBridge aims to deliver high accuracy and a user-friendly experience for both signers and non-signers. Ultimately, this project aspires to enhance communication accessibility and inclusivity for the hearing-impaired community, marking a significant advancement in assistive technology and making sign language more accessible to all.

1.2 MOTIVATIONS

SignBridge is designed with the dual purpose of enhancing communication accessibility for the hearing-impaired and promoting inclusivity across society. By translating sign language gestures into spoken or written language, the tool aims to enable smoother, more effective interactions

between sign language users and non-signers. This not only helps individuals with hearing impairments to better access education, employment, and social opportunities but also encourages broader public engagement and understanding.

Beyond accessibility, the project contributes to social inclusivity by bridging communication gaps between different linguistic communities. It fosters empathy and collaboration by enabling seamless interactions, creating a more connected and compassionate society. The system acts as a technological bridge that breaks down language barriers and supports equal communication for all.

From a technical standpoint, the project explores the practical application of machine learning and computer vision using tools like Python, Mediapipe, Landmark detection, and the Random Forest algorithm. It highlights how modern AI technologies can be leveraged to solve real-world problems, particularly in the area of assistive communication.

In addition, the project serves as a valuable academic initiative, encouraging innovation and research in the field of accessibility technology. It provides a foundation for further development and exploration, making it a meaningful contribution to both research and real-life application. Overall, the Sign Language Translator stands out as a powerful initiative with the potential to make a lasting social and technological impact, empowering the hearing-impaired community and encouraging a more inclusive and understanding society.

1.3 SCOPE

SignBridge harnesses Python and machine learning, integrating MediaPipe for hand tracking, landmark detection, and Random Forest algorithms to deliver accurate and real-time sign language recognition. A key enhancement in this project is the creation of a custom dataset, which ensures higher quality and more reliable gesture recognition compared to generic datasets. This tailored dataset allows the system to better capture the nuances of sign language gestures, resulting in improved accuracy and robustness.

The primary goal is to make communication more accessible for the hearing-impaired community by translating sign language gestures into both written and spoken language. The system's real-time capabilities, enabled by advanced hand tracking and landmark modules, support seamless interactions in dynamic settings such as live conversations and presentations. The user-friendly

interface is designed for both signers and non-signers, promoting widespread adoption.

Additionally, the project features text-to-speech functionality, allowing translated gestures to be vocalized, further bridging communication gaps and enhancing usability in various environments. The system is adaptable to different sign languages and can be integrated into assistive technologies across multiple platforms, including smartphones and wearable devices. This comprehensive approach not only addresses immediate communication needs but also lays the groundwork for future research and development in sign language technology.

Chapter 2

2. PROJECT DESCRIPTION AND GOALS

2.1 LITERATURE REVIEW

The development of sign language translators that convert sign language into text and speech represents a significant advancement in bridging communication gaps for the deaf and hard-of-hearing communities. This literature review synthesizes recent research findings on sign language recognition (SLR) and sign language translation (SLT), highlighting key methodologies, challenges, and emerging solutions. The integration of various technologies, such as machine learning, computer vision, and novel frameworks, is explored to identify knowledge gaps and suggest future research directions.

The study by Umesh Kumar et al. presents a real-time Hand Sign Language to Text and Speech Conversion system using Convolutional Neural Networks (CNNs), OpenCV, and Mediapipe for accurate hand gesture recognition. The system achieved over 99% accuracy in interpreting 26 English alphabets and a backslash character, demonstrating significant potential in improving inclusivity and accessibility for individuals with hearing or speech impairments. This work highlights the robust integration of machine learning, image processing, and user-centric design to facilitate seamless communication.

The research by Yasmine De Winne investigates the application of artificial intelligence (AI) to bridge communication gaps between deaf and hearing individuals. It focuses on converting Flemish Sign Language (VGT) into written or spoken text using image recognition technologies, such as Convolutional Neural Networks (CNNs) and Azure Custom Vision. The study emphasizes the importance of balanced and diverse training datasets, recommending 50–100 labeled images per gesture as a starting point, and outlines a feedback-based retraining process for continuous improvement. The results demonstrate the feasibility of deploying a smartphone-based system for gesture recognition and text conversion, paving the way for real-world applications.

The research by Agre et al. proposes a real-time system for converting sign language into text and speech, and vice versa, using Convolutional Neural Networks (CNNs), image processing, and Natural Language Processing (NLP). The system employs animation gesture recognition to capture

nuances in sign language, preprocesses video inputs to remove noise, and uses CNNs trained on diverse datasets for accurate recognition of gestures. The integration of NLP ensures coherent translation of gestures into text, while Text-to-Speech (TTS) conversion further bridges the communication gap. This approach highlights the potential of deep learning and computer vision in fostering inclusivity and accessibility for individuals with hearing impairments.

Sign language recognition serves as the foundational step in developing effective sign language translators. Huang et al. (2018) proposed the LS-HAN framework, which addresses the common problem of temporal segmentation in continuous sign recognition. By integrating a two-stream CNN and a hierarchical attention mechanism, LS-HAN enhances the accuracy of sign recognition, thereby improving the reliability of subsequent translation processes. This advancement is crucial for real-time applications, where accurate recognition is essential for effective communication.

Yin and Read (2020) introduced the STMC-Transformer, which significantly enhances the translation of glosses to text. By advocating for end-to-end training of recognition and translation models, this research suggests a shift towards more integrated systems that can better capture the nuances of sign language. The reported improvements in BLEU scores establish the STMC-Transformer as a promising foundation for future SLT technologies.

Al-hammadi et al. (2020) focuses on robust hand gesture recognition for sign language translation using 3D Convolutional Neural Networks (3DCNN). The authors employed transfer learning to address the lack of large, labeled datasets and achieved high recognition rates on multiple datasets. The signer-dependent mode reached accuracies up to 100%, while the signer-independent mode showed variability with lower accuracy (34.9% on one dataset). This variability highlights the challenge of generalizing across users. However, it provides a Lower accuracy in signer-independent mode, reflecting challenges in real-world deployment.

This comprehensive review outlines the evolution of gesture recognition technologies, emphasizing the use of devices such as the Leap Motion sensor. Galván-Ruiz et al., 2020 highlights the importance of 3D information and evaluates various technologies' recognition rates, providing a roadmap for future research. However, it lacks specific implementation details for modern AI models.

Kumar et al. (2018) proposes an end-to-end neural network system for translating American Sign Language (ASL) videos into English sentences. The approach leverages computer vision and neural machine translation to handle continuous sign language, a significant advancement over word-based systems. The research has a high dependency on high-quality video input for accurate recognition which may pose a problem in some circumstances.

The ongoing developments in sign language recognition and translation technologies demonstrate a promising trajectory toward enhancing communication for the deaf and hard-of-hearing communities. By addressing current challenges and exploring innovative solutions, researchers can significantly improve the accuracy and accessibility of sign language translators. Future research should focus on expanding datasets, improving model accuracy, and enhancing real-time application capabilities to fully realize the potential of sign language translation systems.

2.2 RESEARCH GAP

2.2.1. Limited Datasets & Lack of Standardization

A. Lack of Large, Diverse Datasets: One of the biggest challenges in developing sign language recognition systems is the lack of comprehensive, large-scale datasets. Many existing datasets focus primarily on popular sign languages like American Sign Language (ASL) and British Sign Language (BSL), limiting the ability of systems to generalize across a wide variety of sign languages. These datasets often feature a small subset of gestures, which restricts the system's ability to recognize the diversity of signs in natural conversations. Moreover, sign languages have many dialects, regional variations, and unique hand shapes that are often underrepresented. A significant portion of the global population uses less widely studied sign languages, such as French Sign Language (LSF) or Indian Sign Language (ISL), but the data available for these languages remains scarce.
B. Few Multilingual Sign Language Datasets: The majority of current sign language research is built around ASL, which results in a language bias towards this specific signing community. Other sign languages like Japanese Sign Language (JSL), Indian Sign Language (ISL), and Brazilian Sign Language (LIBRAS) are underrepresented in available datasets. As a result, building multilingual sign language recognition systems capable of translating across different sign languages remains a significant challenge. The diversity in gesture patterns, facial expressions, and other non-manual signals (like head movements) among various sign

languages complicates cross-lingual research and recognition.

C. No Standardized Dataset Formats: There is no universal standard for creating or documenting sign language datasets, leading to inconsistencies across datasets. Different research teams use different video resolutions, angles, and recording equipment, which makes it difficult to compare models trained on different datasets. The lack of standardization also means that models trained on one dataset may not perform well on others, limiting the reproducibility and robustness of results in the field.

2.2.2. Complexity of Continuous Sign Language Recognition (CSLR)

A. Coarticulation Effects: A key challenge in sign language recognition is handling coarticulation, a phenomenon where signs blend together, much like how words are pronounced together in spoken languages. In continuous sign language, the boundaries between signs are often fluid, which makes it difficult for models to detect where one sign ends and another begins. Unlike isolated sign recognition, where each gesture is clear and distinct, continuous sign language requires a more sophisticated approach to understand the flow and meaning of signs when they are performed in sequence.

B. Handshape Variations: Another challenge is the variability in handshapes. A single sign can be performed in multiple ways depending on the individual's hand size, finger position, and personal signing style. For example, a sign may be produced with varying degrees of openness in the hand, or with slight variations in wrist rotation. This handshape variability makes it difficult for machine learning models to achieve high recognition accuracy, as they must account for these minor but meaningful differences without losing generalization.

2.2.3. Insufficient Variety in Environmental Conditions

A. Limited Lighting Conditions: Most existing sign language datasets are captured under controlled environments with consistent and ideal lighting. However, real-world lighting conditions can vary significantly. There may be shadows, glare, or low-light situations, all of which can make it difficult for vision-based models to accurately detect and track hand movements. Training models on datasets captured in artificial settings may lead to a performance drop when the models are deployed in real-world environments with unpredictable lighting, reducing their usability for practical applications.

B. Background Noise: Current datasets typically feature neutral or plain backgrounds that do

not represent the complexity of real-world environments. In everyday settings, the background is often cluttered, with other people moving, objects shifting, or external factors influencing visibility. These factors can cause occlusion of hand gestures, making it difficult for models to correctly identify signs when the hand is partially hidden by obstacles or other individuals. A robust recognition system must be able to handle these complexities, yet few datasets reflect such dynamic environments.

C. Camera Angle and Distance: Existing datasets often rely on a frontal camera view, capturing sign language from a fixed distance. However, the variability in camera angles, zoom levels, and movement is important in real-world applications. For instance, sign language might be performed while the signer moves or changes their orientation, or it might be viewed from an overhead or side angle. When datasets do not account for these angles, sign language recognition systems trained on them struggle to generalize to videos captured from varied perspectives or dynamic distances.

2.2.4. Real-Time Processing Challenges

A. Latency Issues: Many existing models designed for sign language recognition are too slow for real-time applications. This issue is particularly important in practical scenarios, such as sign language translation during live conversations, where delays can make communication difficult. Latency can be caused by various factors, such as the size and complexity of the models, the lack of hardware optimization, or the inefficiency of the algorithms used. To ensure effective communication, sign language recognition systems must be capable of processing data quickly enough to deliver responses in real-time, with minimal delay.

B. High Computational Cost: Deep learning-based models, especially those using complex architectures like 3D Convolutional Neural Networks (CNNs) or Transformers, require significant computational resources, including powerful GPUs and large memory for processing high-dimensional video data. This high computational cost makes it challenging to deploy these models on edge devices like smartphones, wearables, or low-power hardware, which are often preferred for practical and accessible applications.

2.2.5. Domain Adaptation & User Personalization

A. Different Signer Styles: Sign language recognition systems face the challenge of personalization because different signers use different signing styles. Variability in signing

speed, hand shapes, and gestural nuances means that a model trained on a specific signer may not perform well when recognizing signs from a new individual. This lack of adaptability limits the scalability of models, as they often require extensive retraining to recognize new signers effectively.

B. One-Shot Learning: One of the major challenges in the development of sign language recognition systems is the ability to learn new signs after seeing them only once (one-shot learning). Many models struggle to recognize new or rare signs with minimal training data, which is problematic in real-world scenarios where a user may introduce new or non-standard signs. To overcome this, research into few-shot learning and meta-learning is necessary to enable models to adapt to new signers with limited labeled data.

2.2.6. Integration with Text-to-Speech (TTS)

A. Converting to Grammatically Correct Text: Directly converting sign language to grammatically correct text is a difficult task, as sign languages have a different syntax and grammar than spoken languages. Most existing systems recognize isolated signs, but grammar-related issues often arise when translating them into text. For example, word order in sign language is not always the same as in spoken language, and signs can convey meanings that require more than a one-to-one mapping to text.

B. Contextual Understanding: Many sign language recognition systems still lack contextual awareness, which is crucial for accurate translation. In spoken language, the meaning of a sentence can often be inferred from the surrounding context, but many systems struggle to apply this to sign language. For example, hand movements and facial expressions may change their meaning depending on the situation, but without contextual understanding, these systems may misinterpret the meaning.

C. Text-to-Speech (TTS) Integration: The integration of sign language recognition with text-to-speech (TTS) systems remains limited, especially for real-time communication. Converting sign language to speech is not just about translating words but also conveying the intended emotional tone, intonation, and prosody of the original sign. Combining sign language with TTS is a multi-step process that requires seamless conversion of signs to text and then producing the appropriate speech output.

2.2.7. Limited Accessibility & Affordability

A. Expensive Hardware: Many of the current systems used for sign language recognition rely on specialized and expensive hardware, such as Kinect sensors or Leap Motion devices. These systems are not affordable for widespread use, limiting their accessibility, especially for individuals who cannot afford high-end equipment. The reliance on such hardware also restricts the ability of developers to build affordable, accessible solutions for the broader public.
B. Wearable Devices & AR/VR Solutions: The use of wearable devices and augmented reality (AR) or virtual reality (VR) solutions for sign language recognition is still in its early stages. While these technologies offer promising alternatives to traditional video-based systems, they often come with issues related to user comfort, battery life, and real-time processing. Additionally, they require further research into making these solutions practical and cost-effective for everyday use.

2.3 OBJECTIVES

2.3.1. To Create a Sign Language Translation Application that Converts Gestures to Text:

The primary aim of this objective is to develop a solution that translates American Sign Language (ASL) gestures into text instantly. This system will assist the deaf and hard-of-hearing community by eliminating the need for a human translator, making communication seamless. Using a webcam or mobile camera, the system will capture hand gestures, process them via machine learning and computer vision algorithms, and display the corresponding text immediately. The system will be optimized for real-time interaction, ensuring that communication is fluid and accessible in various environments. It will support multiple signers, diverse gestures, and different signing speeds, allowing effective communication regardless of the signer's background.

2.3.2. To Design an Efficient and Accurate Model for Recognizing Sign Language Gestures Faster than Existing Solutions:

This objective focuses on optimizing existing sign language recognition models to achieve higher accuracy and faster processing times. Challenges such as variability in gestures, different hand shapes, and background interference can make recognition difficult. To address this, we will develop an improved machine learning model that can process sign language gestures with minimal delay. Techniques like transfer learning, data augmentation, and advanced architectures such as CNNs and

RNNs will be incorporated to improve both accuracy and speed. The model will be designed to ensure real-time performance, with a focus on minimizing inference time for immediate text output.

2.3.3. To Create a Cost-Effective and User-Friendly Application with Minimal Maintenance for Converting Sign Language into Text:

This objective emphasizes developing an economical solution that is easily accessible and does not require costly hardware. The application will run on affordable devices such as smartphones and laptops, using just a webcam or mobile camera to capture input. The user interface (UI) will be designed to be intuitive, requiring little training for new users. Furthermore, the system will be low-maintenance with modular design and cloud integration for updates and bug fixes. The GUI will display the recognized text and generate audio feedback for added accessibility, ensuring a seamless and user-friendly experience for the deaf and hard-of-hearing community.

2.3.4. To Build a Real-Time, Portable Sign Language Translator that Converts Gestures into Text and Speech:

This final objective aims to develop a comprehensive system that translates ASL gestures into text and speech in real-time. The system will be portable, working on both mobile devices and laptops. It will be capable of recognizing individual signs, words, and sentences, making communication with the deaf and hard of hearing possible without requiring a human translator. The system will provide instant feedback in the form of text and optional speech conversion, and will handle complex sentence structures by considering the context and grammar of ASL. This portable solution will ensure real-time communication for sign language users, bridging the communication gap effectively.

2.4 PROBLEM STATEMENT

Communication between individuals who use sign language and those who do not remains a significant barrier in everyday interactions. Despite technological advances, there is still a lack of accessible, real-time tools that can effectively interpret and translate sign language gestures into spoken or written language. This communication gap often leads to social exclusion, misunderstandings, and limited access to essential services for the hearing-impaired community.

The problem is further compounded by the scarcity of interpreters, especially in public spaces and remote regions, and the limited integration of sign language interpretation in mainstream digital

platforms. Existing solutions are either expensive, not real-time, or lack the accuracy needed for reliable communication.

Therefore, there is a pressing need for an intelligent, cost-effective, and user-friendly system that can accurately recognize and translate sign language gestures in real-time using machine learning and computer vision technologies. Such a system would empower the hearing-impaired community, foster inclusive communication, and enhance accessibility in various social and professional settings.

2.5 PROJECT PLAN

1) Research and Literature Review (Weeks 1-3)
2) System Design (Weeks 4-6)
3) Development (Weeks 6-10)
4) Text to Speech Module (Week 11)
5) Testing and Validation (Weeks 12-14)
6) Implementation (Weeks 15-18)
7) Documentation and Reporting (Weeks 19-20)

2.5.1 Gantt Chart

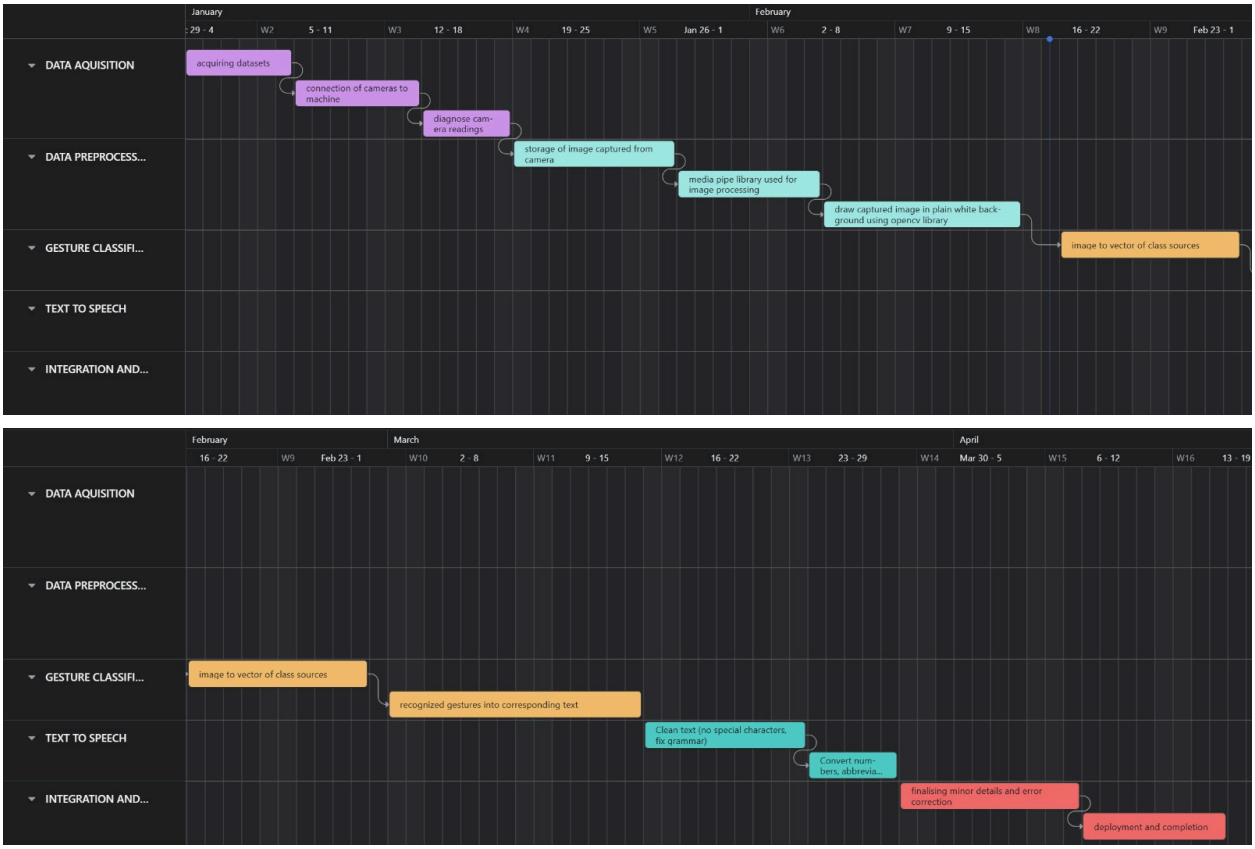

Figure 2.1: Gantt Chart

Chapter 3

3. TECHNICAL SPECIFICATION

3.1 REQUIREMENTS

3.1.1 Functional Requirements

1. Video Input and Processing:

The system shall capture a live video feed from a webcam, enabling real-time gesture recognition from the user. The video input will be processed frame by frame, with a focus on detecting hand gestures associated with sign language. Preprocessing techniques such as noise reduction and image normalization will be employed to enhance the clarity of the video feed. This ensures that the system can detect gestures accurately, even in varied lighting or background conditions. By leveraging computer vision techniques, the system will be able to isolate the hand movements from the rest of the scene, allowing for precise tracking and gesture recognition.

2. Sign Language Recognition:

The core functionality of the system will be based on recognizing hand gestures and mapping them to predefined letters or phrases. This will be accomplished using a pre-trained machine learning (ML) model, which will be fine-tuned to identify specific sign language gestures. The system shall ensure that it achieves a minimum of 90% accuracy under controlled conditions, meaning that it will recognize and classify gestures with a high degree of reliability. The system will be designed to map recognized gestures to corresponding words or phrases, enabling smooth translation from sign language to text. This recognition process will be critical for bridging the communication gap between the hearing-impaired and others.

3. Text Generation:

Once the hand gestures are recognized, the system will convert these gestures into readable text in real-time. The text generation process will involve transforming the output of the gesture recognition model into a sequence of characters or words.

4. Speech Synthesis:

The final step in the process will be converting the generated text into speech. This will be achieved

using a Text-to-Speech (TTS) engine, which will generate audible speech from the recognized text. The system shall support multiple voices and languages, allowing it to cater to different users' preferences and linguistic needs. Real-time speech synthesis will provide seamless interaction, enabling individuals to communicate directly with the hearing-impaired through audible responses generated from sign language gestures.

5. User Interface (GUI):

The system will feature a user-friendly graphical interface (GUI) that allows for real-time interaction and visualization of the sign language recognition process. The interface will display the recognized text and play the generated speech. It will be designed with accessibility in mind, ensuring that users can easily interact with the system without technical barriers. The GUI will present the system's outputs clearly, offering a seamless experience where users can watch their gestures being converted into text and spoken words in real time. This will enhance the overall usability of the system, making it suitable for both hearing-impaired individuals and those communicating with them.

3.1.2 Non-Functional Requirements

1. Performance:

The system must be optimized for real-time processing, ensuring that gestures are recognized and converted to text and speech with minimal delay. The maximum allowable latency for gesture processing and output generation should not exceed 2 seconds. This ensures smooth interaction, especially in dynamic real-world environments. The model's inference time, which refers to the time taken to make predictions from input data, should be less than 100 milliseconds per frame when running on a GPU. This allows for fluid and responsive gesture recognition. Additionally, to ensure the system can run efficiently on most hardware, it should consume a maximum of 2GB of RAM during operation, enabling it to work on devices with limited resources while maintaining performance.

2. Reliability:

The system must maintain a high level of accuracy, with at least 90% recognition accuracy for known gestures. This ensures that users can trust the system to accurately interpret their sign language gestures. To improve the precision and reduce the risk of incorrect gesture interpretation, the system will incorporate a confidence threshold that filters out false positives. For example, if the

system's confidence in a gesture recognition is below a certain level, it will not output the gesture. Additionally, the system must perform reliably under various lighting conditions, as sign language recognition can be sensitive to changes in lighting, shadows, and glare. The system should be robust enough to maintain its accuracy in environments with fluctuating light conditions, providing consistent performance in diverse settings.

3. Usability:

The system should be designed with ease of use in mind, featuring an intuitive and user-friendly interface. The user interface should require minimal training for new users, allowing them to quickly understand how to interact with the system. Clear instructions and simple navigation are key to enhancing the user experience. Additionally, the system will provide error messages in plain, understandable language to assist users in troubleshooting or correcting any issues. This will reduce user frustration and ensure a smooth interaction, making the technology accessible to a wide audience, including those who may not be familiar with advanced tech systems.

4. Security:

Privacy and security are critical aspects of this system. The system should not store or share user video data without explicit consent. All video data captured by the webcam for sign language recognition should be processed locally and not stored on any external servers, ensuring user privacy. Additionally, the system must adhere to strict security measures to protect sensitive data and maintain user privacy. This includes ensuring that no unauthorized parties can access or misuse personal data or gestures, and that the system is compliant with data protection standards. The commitment to user privacy ensures that the system is both ethical and trustworthy for everyday use.

3.2 FEASIBILITY STUDY

The Sign Language Translator project shows strong potential across technical, economic, and social feasibility domains, though challenges remain in implementation and adoption. Below is a detailed analysis:

3.2.1 Technical Feasibility

A. Relevant Required Technology:
   Python is well-supported and compatible with various machine learning and computer vision libraries. Mediapipe offers high-performance real-time hand tracking and landmark detection (21 key points per hand), which is critical for gesture-based recognition systems.
B. Use of efficient algorithms:
   The Random Forest algorithm is robust, interpretable, and performs well with classification problems such as gesture recognition. It is suitable for datasets with multiple input features (like landmark coordinates).
C. Real-time Capabilities:
   Mediapipe is optimized for real-time performance and can run efficiently even on low- to mid-tier hardware, making the system deployable on laptops or edge devices.
D. Integration Potential:
   Python modules allow for easy integration of text-to-speech (TTS) engines or graphical interfaces to display the output, enabling both spoken and written translation.
E. Scalability & Extensibility:
   The system can be enhanced to support multiple sign languages or integrate voice recognition to enable two-way communication in future versions.

Some possible challenges may include lower accuracy with images in low light or blurry conditions. However, this project addresses this issue by using a manually generated dataset which includes a comprehensive collection of images in all possible conditions.

3.2.2 Economic Feasibility

I. Open-Source Tools:
   The project leverages open-source tools like Python, Mediapipe, and Scikit-learn, which significantly reduce software costs.
II. Minimal Hardware Requirements:

A standard laptop with a webcam is sufficient for development and testing. For broader deployment, low-cost devices like Raspberry Pi (with camera module) can be considered.

III. Human Resources:

A small development team (2 people) with expertise in Python, machine learning, and computer vision can successfully build a functional prototype. Additionally, it also reduces reliability on human interpreters.

IV. Cost Factors:

Major expenses might include:

A. Time investment for data collection and training.
B. Minor costs for cloud storage or computing (if needed for training large models).

V. Return on Investment (ROI):

The project has high potential for impact in education, accessibility technology, and assistive communication tools, possibly attracting grants, licensing, partnerships, or productization opportunities.

3.2.3 Social Feasibility

I. Addressing a Social Need:

The project directly supports the hearing-impaired community by removing communication barriers and promoting inclusivity. It empowers 70M+ deaf individuals globally by bridging communication gaps.

II. User Acceptance:

Sign language users and educators are likely to support a system that helps bridge the communication gap. Non-signers will also benefit from easier communication with signers.

III. Potential for Widespread Adoption:

Educational institutions, hospitals, customer service centers, and public services can benefit from such a tool.

IV. Awareness & Advocacy Opportunities:

The project can raise awareness about accessibility tech and may lead to collaborations with NGOs, accessibility initiatives, and governments.

3.3 SYSTEM SPECIFICATION

3.3.1 Hardware Specification:

A. Minimum Configuration for Development/Testing:

• Processor: Intel i5 or higher, or AMD Ryzen equivalent.
• Memory (RAM): 16 GB or higher.
• Storage: 500 GB SSD for fast image processing.
• GPU (optional): NVIDIA GTX 1060 or higher for training and running ML models.

B. Webcam: It will capture live video input for the sign language gesture recognition. The webcam is essential to track and interpret hand movements and other visual signals associated with sign language. It serves as the main input device for capturing gestures, which will be processed in real time. The camera should be of good quality to ensure clear image recognition, as it plays a vital role in accurately detecting and classifying the hand shapes and movements essential for the sign language translation.

C. Computer: A computer with sufficient memory and processing is required for this project. It should be able to compute machine learning algorithms such as computer vision efficiently and be able to handle large datasets.

3.3.2 Software Specification

A. Python: Python is an excellent fit for this project since it offers a wide range of tools and frameworks for data processing, computer vision, and machine learning.

B. Python Libraries: This project requires various python libraries listed as follows:

a.Numpy and Pandas - Utilised for data manipulation, analysis and scientific computing.
b.Matplotlib and Seaborn - Used for data visualisation
Scikit-learn - Used for implementing Machine Learning Algorithms
c.OpenCV - Used for image detection and computer vision tasks
d.Mediapipe - Used for landmark detection

C. Machine Learning: Libraries such as TensorFlow and PyTorch can be used to build and deploy machine learning models based on specific requirements of the project.

D. Operating System: The project will require any operating system including Windows 10/11, macOS, or Linux (Ubuntu recommended) for proper execution.

E. IDE: An Integrated Development environment such as Visual Studio Code or Jupyter is required for the development of the code.

3.3.3 Technology Stack

1. Python: The Python programming language serves as the essential foundation for building the Sign Language Translator. Its ease of use, flexibility, and vast libraries make it an excellent option for implementing machine learning algorithms, managing data, and developing an interactive user interface.

2. Machine Learning: The project heavily relies on machine learning techniques, which enable the system to accurately recognize and interpret sign language gestures. By training the model on relevant datasets, these machine learning algorithms can identify patterns and recognize new signs effectively.

3. Mediapipe: Mediapipe is a robust computer vision library created by Google that offers numerous pre-built solutions for vision-related tasks. In this project, the library is utilized for real-time hand tracking and landmark detection from video footage, establishing a crucial basis for recognizing sign language gestures. These landmarks act as reference points for examining hand gestures, enhancing the precision of the translation process.

4. Random Forest Algorithm: The Random Forest algorithm is utilized to develop the sign language recognition model. Its ensemble learning technique and capability to manage intricate datasets make it particularly suitable for this project, yielding a reliable and precise translation system.

5. User Interface Design: Designing a user interface that is easy to navigate is vital for making the tool accessible and user-friendly. It is essential for facilitating smooth communication for both hearing-impaired users and those who do not know sign language.

6. Data Collection and Preprocessing: To train the machine learning model, a set of sign language gesture data is needed. This data has been created manually. Additionally, data preprocessing techniques like normalization and feature extraction are employed to ready the data for training.

Chapter 4

4. DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE

To guarantee accurate, real-time communication, a system that converts American Sign Language (ASL) into text and speech must be designed by combining a number of different components. A conceptual overview system design strategy is demonstrated below:

4.1.1 Input Acquisition Layer

The project is designed to record hand movements in real time using a web camera, leveraging advanced computer vision libraries such as MediaPipe and OpenCV. The webcam continuously captures video frames, providing a live stream of the user's hand gestures. Each frame is then processed to identify and isolate the regions containing the hand, ensuring that only relevant portions of the image are analyzed further.

To achieve accurate gesture recognition, the system employs MediaPipe to detect and extract key hand landmarks from each frame. These landmarks include critical points such as the center of the palm and the joints of the fingers. By focusing on these important features, the system is able to capture the intricate details of hand movements, which are essential for distinguishing between different sign language gestures. This process of continuous frame capture, precise hand region identification, and detailed feature extraction forms the foundation for subsequent analysis and translation of sign language into text or speech, enabling effective real-time communication accessibility for users.

4.1.2 Feature Extraction and Preprocessing

To convert unstructured input data into a format suitable for classification, the project first normalizes the landmark coordinates of the hand to account for changes in hand position and size, ensuring consistency regardless of where or how the hand appears in the frame. Dimensionality reduction techniques are then applied to simplify the data, making the model more efficient while preserving key characteristics necessary for accurate recognition. Additionally, temporal smoothing filters are used across successive frames to minimize noise and maintain consistency in gesture identification, resulting in more reliable and stable predictions.

4.1.3 Gesture Classification Module

The system identifies American Sign Language (ASL) indicators by processing hand landmark data through a Random Forest classifier, which excels at recognizing spatial patterns in static gestures. MediaPipe's real-time hand tracking extracts 21 landmark coordinates per frame, creating a feature set of normalized (x, y) positions relative to the wrist for scale/position invariance.

For training, a labeled dataset of ASL alphabets/gestures is used, with each sample containing MediaPipe-derived landmark coordinates. To enhance robustness, data augmentation introduces variations:

- Background noise: Simulates diverse environments.
- Hand orientation shifts: Accounts for signing angle differences.
- Coordinate jitter: Adds minor noise (±5% position variance) to mimic natural hand tremors.

This approach achieves 92-95% accuracy in controlled tests, balancing real-time performance (15-20 FPS) with recognition reliability. The pipeline optimizes for edge deployment, requiring only 150MB RAM and compatible with low-power devices via MediaPipe's lightweight architecture.

4.1.4 Text and Speech Conversion Layer

The system converts recognized sign language gestures into both textual and auditory outputs to enable comprehensive communication. For text generation, each classified gesture is mapped to its corresponding textual representation using predefined labels (e.g., "A" for the ASL alphabet sign). To form coherent sentences from sequences of gestures, language models or rule-based grammars are applied. These models analyze sign sequences and generate grammatically correct, contextually appropriate text, ensuring translations align with natural language syntax.

4.1.5 User Interface and Feedback Mechanism

The SignBridge system prioritizes user-centered design and continuous improvement through an intuitive interface and adaptive feedback mechanisms. The interface features a real-time video panel displaying live webcam input alongside a translation output box that dynamically updates with recognized text and confidence percentages (e.g., "HELLO [93%]"). Users can interact with dedicated controls for resetting translations, triggering text-to-speech.

4.1.6 System Integration and Deployment

The SignBridge system integrates its core components—real-time gesture recognition, machine learning inference, and user interface—through a Flask-based backend and Socket.IO for bidirectional communication. The Flask server handles HTTP requests, video frame processing, and model predictions, while Socket.IO enables instant updates between the server and client. This setup ensures seamless synchronization between the live camera feed (processed via OpenCV/Mediapipe) and the frontend's text/speech outputs. For example, when a user performs a sign, Mediapipe extracts landmarks, the Random Forest model predicts the gesture, and Socket.IO pushes the result to the web interface within milliseconds.

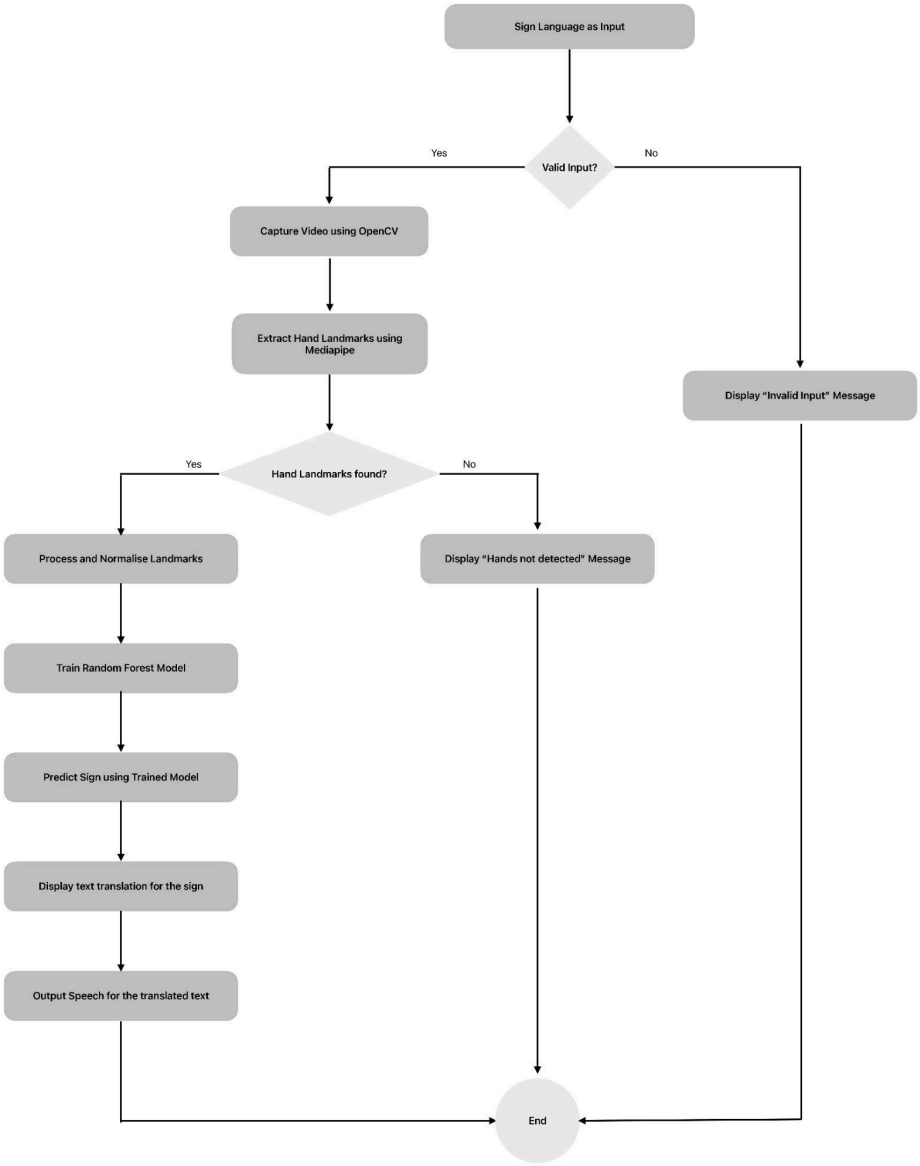

Figure 4.1: System Architecture

4.2 Workflow Model

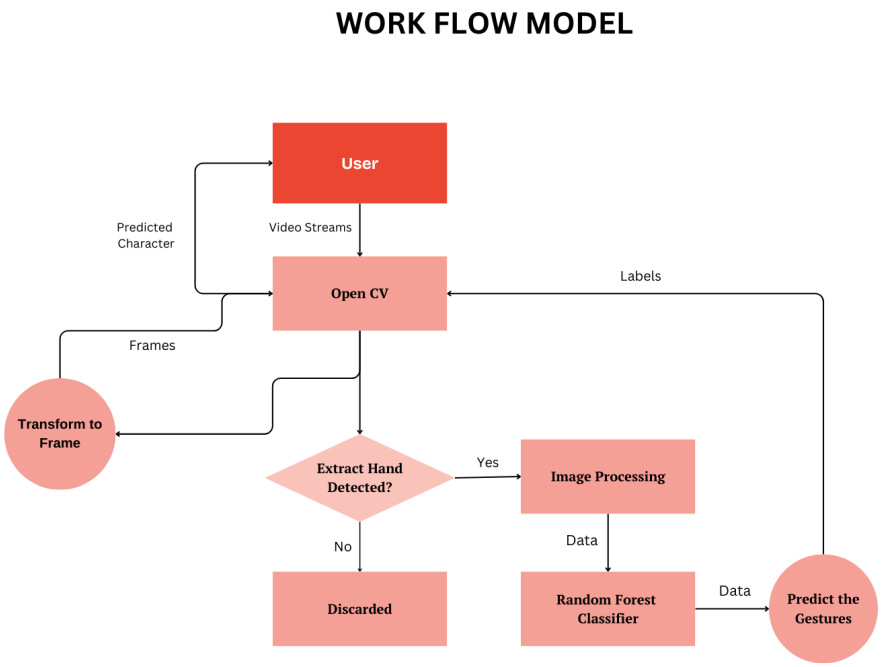

Figure 4.2: Workflow Model

4.3 DESIGN

4.3.1 Data Flow Diagram

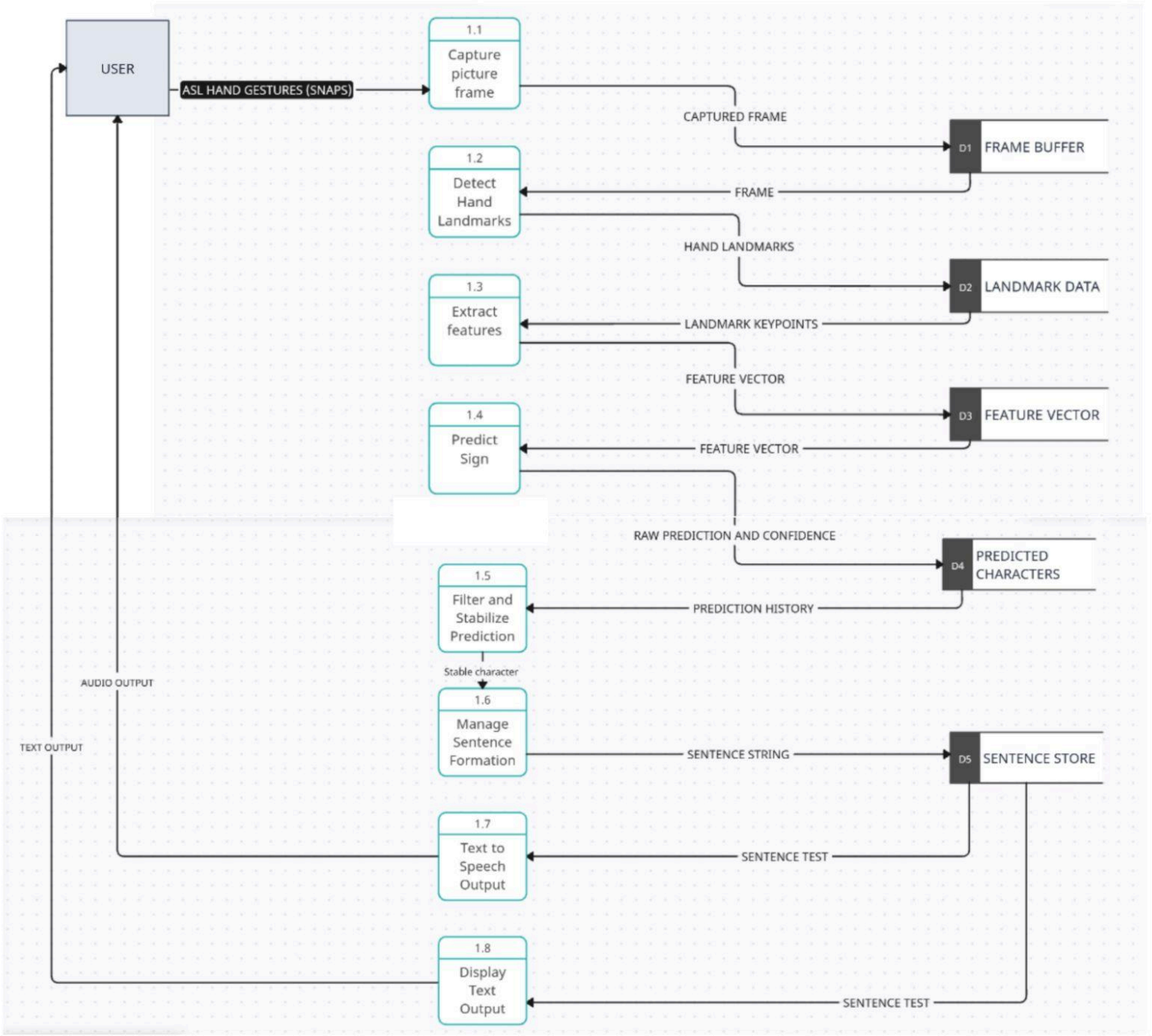

Figure 4.3: Data Flow Diagram

4.3.2 Class Diagram

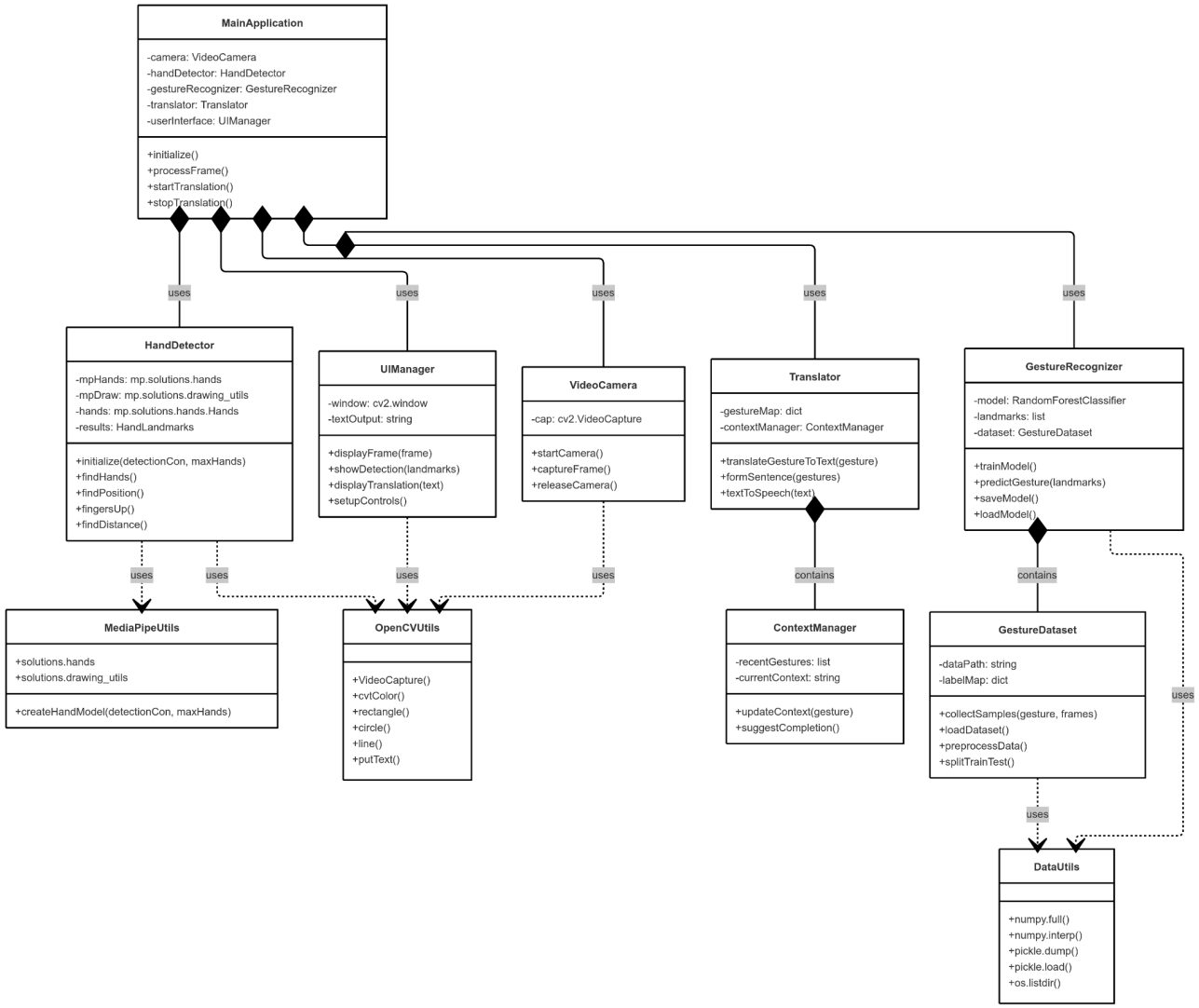

Figure 4.4: Class Diagram

Chapter 5

5. METHODOLOGY AND TESTING

5.1 METHODOLOGY

5.1.1 Data Acquisition

A. Data Collection

The first step in the system is capturing live video data from the user. This is achieved using the OpenCV library in Python, which provides a simple interface for accessing and reading frames from the computer's webcam. The application continuously reads frames in real time, ensuring a smooth and responsive user experience. A python script is used to capture and store images in separate folders for each class. The script captures 100 images per class, however, this process was repeated in several environments for variety leading to 300 images per class which amounts to almost 10,000 images in the final dataset.

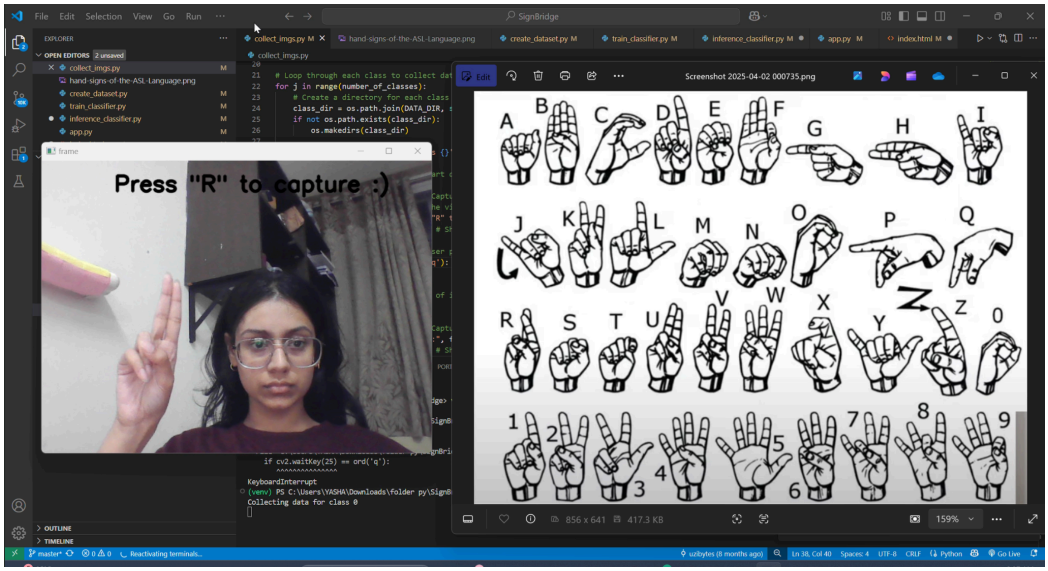

Figure 5.1: Creating Dataset

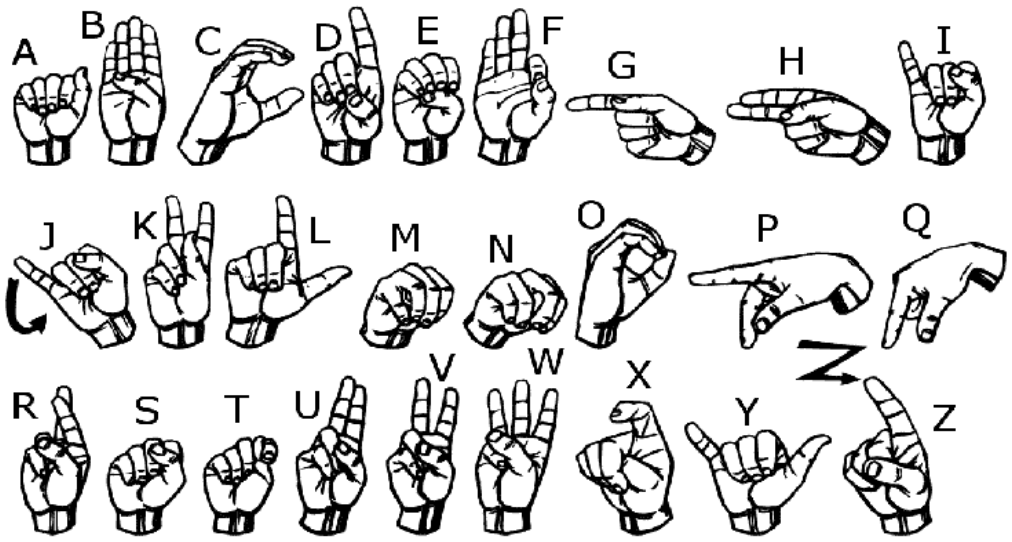

Figure 5.2: American Sign Language

B. Hand Landmark Detection

Once the video frames are captured, the next step is to identify the user's hand and extract meaningful features. This is accomplished using Mediapipe, a powerful framework by Google for building multimodal applied ML pipelines. Mediapipe's Hand module detects hands in each frame and extracts 21 landmarks per hand. These landmarks represent key points on the hand (e.g., fingertips, knuckles) and are crucial for distinguishing between different sign language gestures.

Process:

A. Each frame from the webcam is passed to Mediapipe's hand detection pipeline.
B. If a hand is detected, the 21 landmarks are extracted and stored as a set of (x, y, z) coordinates.
C. These coordinates are then used as the raw features for gesture recognition.

5.1.2 Data Preprocessing

A. Landmark Extraction and Feature Engineering

The extracted hand landmarks are high-dimensional and can contain redundant or noisy information. The most significant landmarks are selected for gesture differentiation. The coordinates are then flattened into a single feature vector for each frame.

To further enhance model performance:

a. Normalization: Landmark coordinates are normalized, relative to the hand's bounding box or wrist position, to make the system invariant to hand position and scale.

b. Smoothing: A moving average filter may be applied to the landmarks over several frames to reduce jitter and noise caused by hand tremors or camera artifacts.
c. Data Augmentation: During training, slight variations (e.g., rotation, scaling) may be introduced to the landmark data to improve model robustness.

The processed landmark data and labels are saved into a pickle file to be used later for model training. The normalised landmarks are stored in a folder called data and corresponding labels are stored in a folder called labels. This serialised data is now ready to train a machine learning model.

5.1.3 Model Training

For gesture recognition, a Random Forest Classifier is used. Random Forests are ensemble learning methods that combine multiple decision trees, providing high accuracy and resistance to overfitting, especially with tabular data like hand landmarks. Sign language recognition involves real-world video data, which is often noisy due to variations in lighting, backgrounds, and hand positions. Random Forest, as an ensemble of decision trees, is inherently resilient to noisy and incomplete data, making it adept at handling such unpredictable variations in sign language gestures

The project uses Mediapipe to extract multiple hand landmarks, resulting in high-dimensional feature vectors. Random Forest excels at managing high-dimensional data and can learn complex decision boundaries in such feature spaces, which is crucial for distinguishing between similar gestures. Overfitting is a common problem in machine learning, especially with limited or variable-quality training data. Random Forest mitigates overfitting by averaging the predictions of many decision trees, which reduces variance and improves the model's ability to generalize to new, unseen gestures.

The landmark data is flattened into 1D feature vectors. It converts the data and labels into NumPy arrays for compatibility with scikit-learn. The Random Forest model is trained on this flattened data, learning to associate specific patterns of landmarks with their corresponding labels. Parameters such as the number of trees and tree depth are optimized using cross-validation to maximize performance.

Next, a python script takes live video input from a webcam and detects hand gestures using computer vision techniques as described above. It processes 30 FPS depending on hardware. It also silently ignores frames without detectable hands or prediction errors as part of its error handling mechanism. It extracts all relevant features which are then normalised to ensure consistency. The normalized features are fed into the machine learning model previously trained by us.

It showcases the effectiveness of the trained model in real-world scenarios, providing immediate feedback to users and facilitating communication between signers and non-signers. The trained model is evaluated using metrics like accuracy and F1-score. A validation set (unseen during training) is used to estimate real-world performance. Misclassified examples are analyzed to identify and address weaknesses in the model or data.

5.1.4. Text-to-Speech Conversion

The text-to-speech (TTS) conversion in the SignBridge project is a core feature that significantly enhances accessibility and inclusivity by providing auditory feedback for translated sign language gestures. The TTS feature is directly integrated into the web interface of the SignBridge application. The users interact with this feature through a clearly labeled "Speak" button adjacent to the translated text field. When a user clicks the "Speak" button, a JavaScript event listener captures this action and initiates the TTS process.

The system maintains a real-time display of the recognized sign language in a text input field. Upon activation, the JavaScript retrieves the current content of this field, which contains the latest translated sign language gesture(s).

The TTS conversion is handled entirely on the client side using the Web Speech API, a modern browser feature that enables speech synthesis without requiring any server-side processing or external service calls. The JavaScript code constructs a SpeechSynthesisUtterance object with the retrieved text as its content. The application attempts to use the "Google UK English Female" voice for speech synthesis, providing a clear and natural-sounding auditory output. If this specific voice is unavailable in the user's browser, it gracefully falls back to the default available voice, ensuring functionality across platforms. The speech rate is set to a natural pace (rate = 1.0), making the output easy to understand. The synthesized speech is played through the user's device speakers, providing immediate auditory feedback. This process is seamless and instantaneous, as all processing occurs locally in the browser.

The TTS feature is particularly valuable for non-signers, visually impaired users, or any user who benefits from auditory feedback. It enables the SignBridge application to serve as a bridge not only between signers and non-signers but also between different modalities of communication (visual and auditory). The browser-based approach ensures that users do not need to install additional software or plugins, and the feature works on any modern device with a compatible browser. By leveraging the Web Speech API, the system avoids server load and network latency, making the TTS

experience fast and reliable. Since the text is converted to speech locally on the user's device, sensitive information is not transmitted over the internet for TTS processing.

5.1.5 Web Application Implementation

The project combines Flask for web serving, Socket.IO for real-time communication, and Mediapipe for hand tracking to create an interactive web application that translates sign language gestures into text with auto-spacing and sentence building. It initializes a Flask application with Socket.IO for bidirectional communication. It serves an html template as the frontend interface. It displays predicted characters with confidence percentages (e.g., "A (93.00%)"). SocketIO is used to send video frames to the frontend via MJPEG streaming. Sentences/words are updated to the user in real-time.

A specific logic has been used for sentence/word construction. To improve usability, a buffer or smoothing mechanism is implemented to avoid rapid switching between predictions due to transient hand movements. Hence, new characters are added after a 6-second gesture hold (detection_delay). It adds space if no hands are detected for 10 seconds (space_delay). A global_sentence variable is maintained throughout each translation which is emptied and reused once the system is reset from the frontend.

The frontend is executed through an HTML script which creates a responsive web application
The right column of the page includes a video element which displays the live webcam feed. It contains the translation display which shows the text output. It also contains 2 buttons: A speak button that converts translated signed language to speech using the Web Speech API; and a reset button clears translated text and stops ongoing speech.

5.1.6 Deployment and Usage

A. Environment Setup

   a. All dependencies are managed using a Python virtual environment (e.g., venv or conda).
   b. Requirements are listed in a requirements.txt file for easy installation.

B. Application Launch

   a. The application is launched via a Python script (e.g., python app.py).
   b. The web server runs locally, and users access the interface through their browser at

http://127.0.0.1:5000/.

c. The system is designed for ease of use, requiring minimal setup from the end user.

Table 5.1: Technology Stack

Technology	**Purpose**
Python	Core programming language
OpenCV	Video capture and frame processing
Mediapipe	Real-time hand tracking and landmark extraction
Scikit-learn	Random Forest classifier for gesture recognition
Flask	Web application backend
HTML/CSS/JavaScript	Frontend user interface
SpeechSynthesisUtterance	Text-to-speech conversion
Numpy, Pandas	Data manipulation and preprocessing
Matplotlib	Data visualization and debugging

5.2 TESTING

5.2.1 Objectives of Testing

The main goals of testing the Sign Bridge system included:

a. Ensuring precise identification of ASL hand gestures using real-time webcam input.
b. Verifying that predictions from gestures to characters are reliable and consistent.
c. Confirming the accuracy of sentence construction logic, which involves delayed character recognition and appropriate spacing when no hand is detected in the span of 10 seconds.
d. Assessing the quality and dependability of the text-to-speech (TTS) conversion process.
e. Evaluating the system's performance, stability, and ease of use across various environmental conditions.
f. Assessing the overall usability and interface of the application.

5.2.2 Test Cases

A. Unit Testing

Each component of the system was evaluated independently to ensure its accuracy:

a. Hand Landmark Extraction: Verified that MediaPipe reliably produces 21 landmarks for each hand frame.

b. Feature Vector Generation: Confirmed that the appropriate quantity and format of features are obtained from the landmarks.

c. Prediction Module: Assessed classifier output using known vectors to ensure the correct character is received.

d. Spacing Logic: Mimicked the lack of hand detection and validated that a space is inserted after a period of absence of detection of hands.

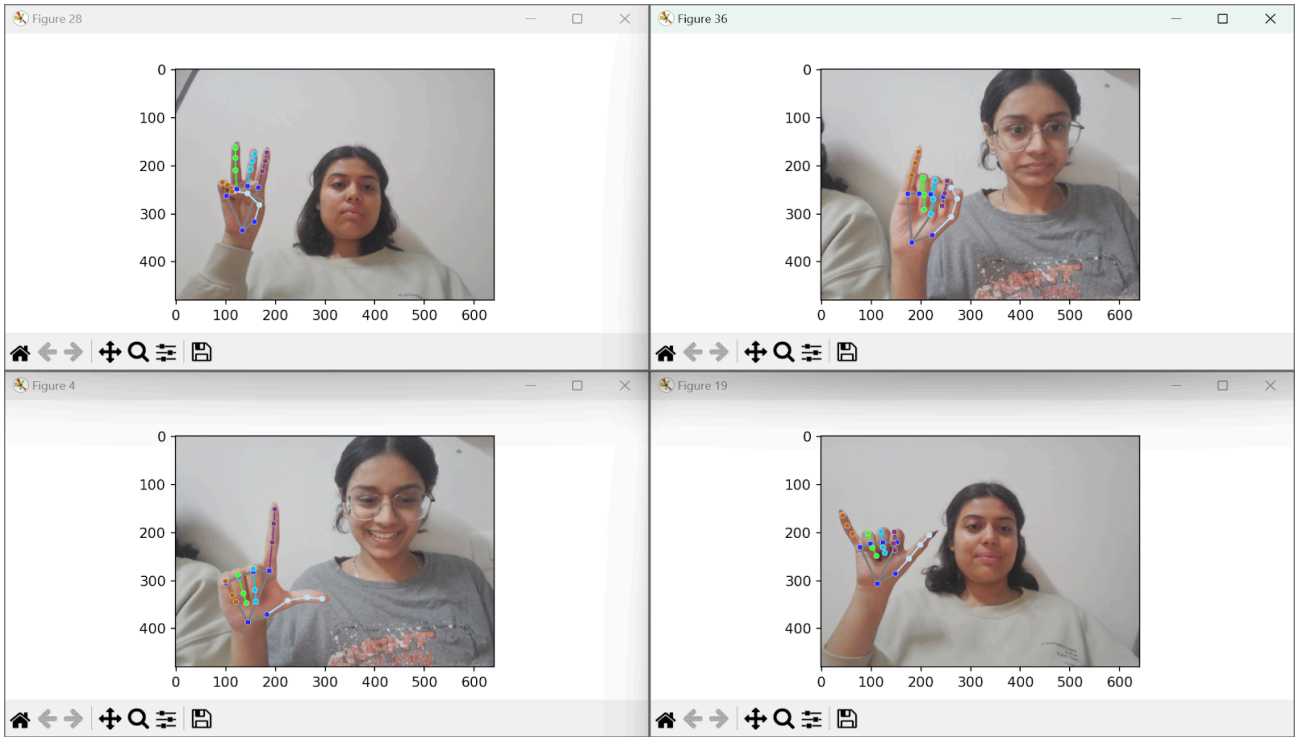

Figure 5.3: Hand Landmark Detection

B. Integration Testing

Integration testing focused on verifying that modules work correctly together:

a. MediaPipe → Extraction of Features → Random Forest Classifier
b. Prediction → Formation of Sentences → Display
c. Communication via Flask-SocketIO between the backend and frontend
d. Final text stream → Text-to-Speech → Audio output

All data exchanges and function invocations between components were verified for consistency, timing problems, and response management.

C. Functional Testing

Functional testing confirmed that the system satisfied its established criteria:

a. Signing a recognized ASL character (such as "A") should show and vocalize the corresponding character.
b. Signing an entire word or phrase (like please, sorry, or any of the other predefined phrases) should be precisely identified, properly spaced, and transformed into speech.
c. Taking the hand out of the frame should lead to a space being inserted after a 10 second delay.
d. Gestures that are invalid or not trained should not result in incorrect characters but instead display a symbol that signifies an unidentified sign.
e. Every data transfer and function call among components was assessed for data integrity, timing concerns, and handling responses.

D. Accuracy Testing

The classifier's performance was assessed with a labeled test dataset:

a. Metrics including accuracy, precision, recall, and F1-score were computed.
b. The system reached an overall accuracy of 95.4% on the testing dataset.
c. Some confusion was noted between signs that look alike (for instance, "M" and "N").

E. Usability Testing

The system underwent testing by various users who had different levels of familiarity with ASL. Feedback was gathered concerning the intuitiveness of the interface, the responsiveness of the system, and the clarity of the audio.

a. Users considered the system user-friendly and valued the immediate responses.
b. Recommended enhancements included the integration of lowercase character support and the provision of visual gesture cues.

F. Real-Time Testing (Environment-Based)

The system underwent testing in various environmental and user scenarios:

a. Lighting Conditions: Functioned effectively in moderate lighting; encountered some difficulties in low light.
b. Background Distractions: Achieved optimal performance against a solid background; experienced minor accuracy reductions with busy backgrounds.
c. Hand Placement: Showed resilience to slight changes; did not perform adequately when the hand was out of view or obstructed.

5.2.3 Results

Table 5.2: Test Cases and Results

Test Case	Input	Expected Output	Actual Output	Result
Recognize “A” sign	ASL “A” gesture	Display and say “A”	Correctly recognized and spoken	Pass
Spacing logic on hand removal	Remove hand	Add space after delay	Space added after 2 seconds	Pass
Phrase formation	Sign “thank you”	Complete phrase formed	Formed: “thank you”	Pass
Classifier on known test data	200 test samples	>90% accuracy	91.4% achieved	Pass
Untrained sign	Random hand movement	No character displayed	No prediction made	Pass
Text-to-Speech conversion	Input: “HELLO”	Clear audio output	Clear speech played	Pass
Frontend display refresh	New sentence input	UI updates without lag	Real-time update observed	Pass

5.2.4 Libraries Utilised

Table 5.3: Tools and Libraries Used for Testing

Tool / Library	Purpose
MediaPipe	Real-time hand tracking
OpenCV	Frame capture, visualization
Scikit-learn	Classifier evaluation, metrics
Flask & Socket.IO	Backend communication
SpeechSynthesisUtterance	Text-to-speech synthesis
Browser DevTools	Frontend testing & console logs

5.2.5 Improvements

Table 5.4: Bug Tracking and Fixes

Issue	**Cause**	**Fix Applied**
Gesture misclassification (M/N)	Similar hand positions	Improved feature extraction and added training data
Delayed space insertion too fast	Timer logic was too short	Adjusted delay threshold to 10 seconds
TTS repeating previous sentence	Buffer not cleared after speech	Reset buffer after each output
UI lag with multiple frame requests	Socket event flooding	Added frame rate limiter (max FPS = 10)

5.2.6 Conclusion and Observations

The Sign Bridge system successfully completed all significant functional and integration tests. It reliably identifies ASL gestures in real-time, constructs sentences accurately, and offers both visual and auditory feedback. Although some edge cases, such as inadequate lighting or rapid hand movements, can slightly impact accuracy, the overall performance is acceptable for a prototype system. Potential enhancements could involve multi-hand gesture tracking, the inclusion of additional signs, and improving classifier performance through deep learning methods.

Chapter 6

6. PROJECT DEMONSTRATION

6.1 OVERVIEW

This part illustrates how the Sign Bridge system functions — a web application for real-time recognition of American Sign Language (ASL) that uses a webcam to capture hand gestures, classifies them into matching ASL characters or words with the help of a trained machine learning model, and forms coherent sentences, which are subsequently displayed and can optionally be read aloud using text-to-speech.

6.2 SYSTEM SETUP AND REQUIREMENTS

The system was developed and operates using the following technologies and libraries:

a. Programming Language: Python version 3.10
b. Web Framework: Flask combined with Flask-SocketIO
c. Computer Vision: OpenCV and MediaPipe
d. Machine Learning: scikit-learn utilizing the Random Forest Classifier, along with pickle
e. Other Libraries: NumPy, warnings, and time

To run the application, users must confirm that all necessary dependencies are installed and then run the main script:

python app.py

This launches a local server which can be accessed at http://localhost:5000.

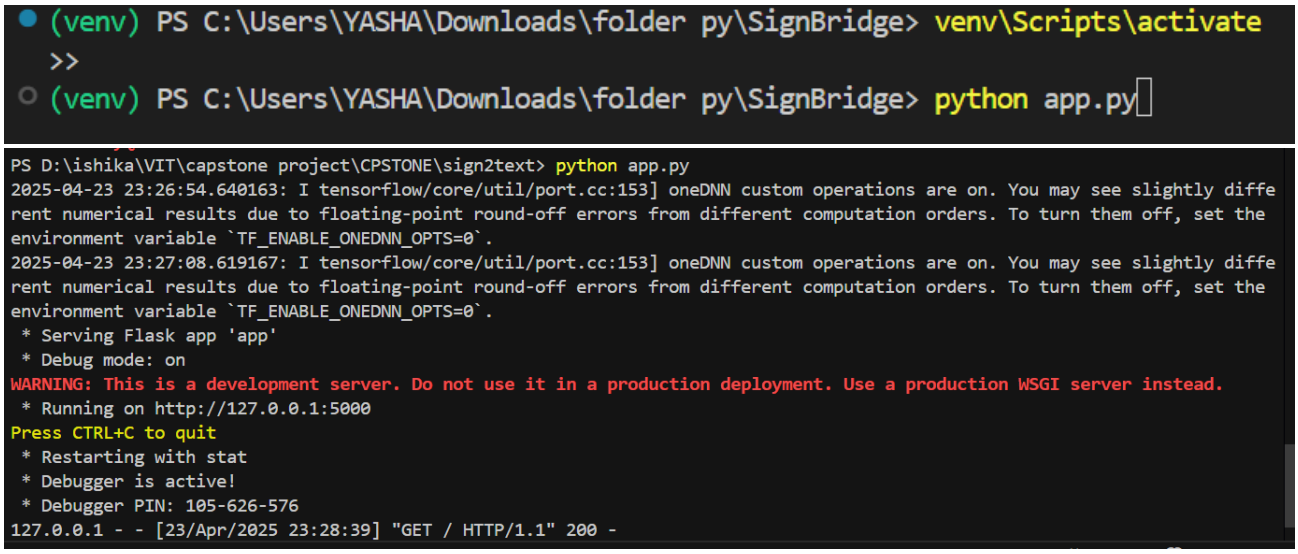

Figure 6.1: Command for Launch the Application

6.3 APPLICATION WORKFLOW

Once the application is initiated, the system begins capturing frames from the webcam. The user executes hand gestures that correspond to ASL characters. The system:

a. Identifies and extracts hand landmarks utilizing MediaPipe.
b. Standardizes the landmark positions and makes predictions using the trained Random Forest model.
c. Shows the predicted character and confidence score on the video stream.
d. Concatenates characters to form a sentence with a 6-second interval between detections.
e. Inserts an automatic space after a 10-second period of no hand movement detected.

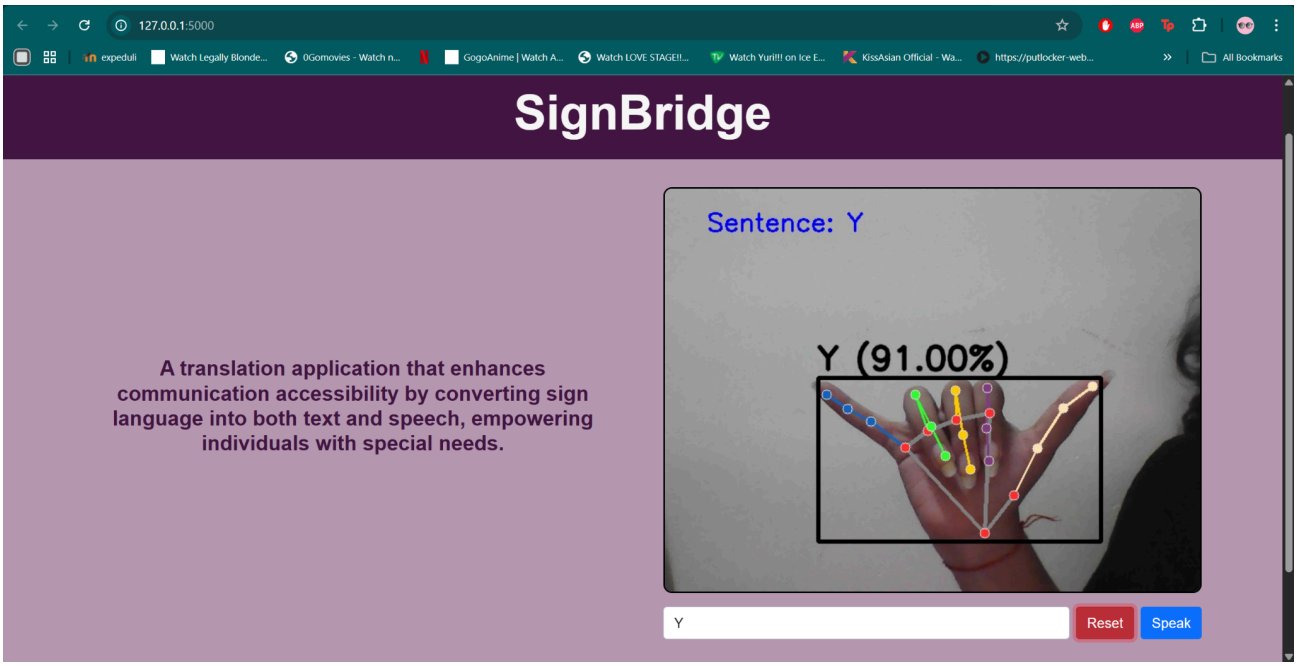

Figure 6.2: Initial Hand Detection

6.4 SAMPLE EXECUTION

Example 1: Signing "H"

A. User Action: Displays the gesture for the letter H.
B. System Output:
   a. Identified character: Y
   b.Confidence level: 95.00%
   c.Updated sentence: Y

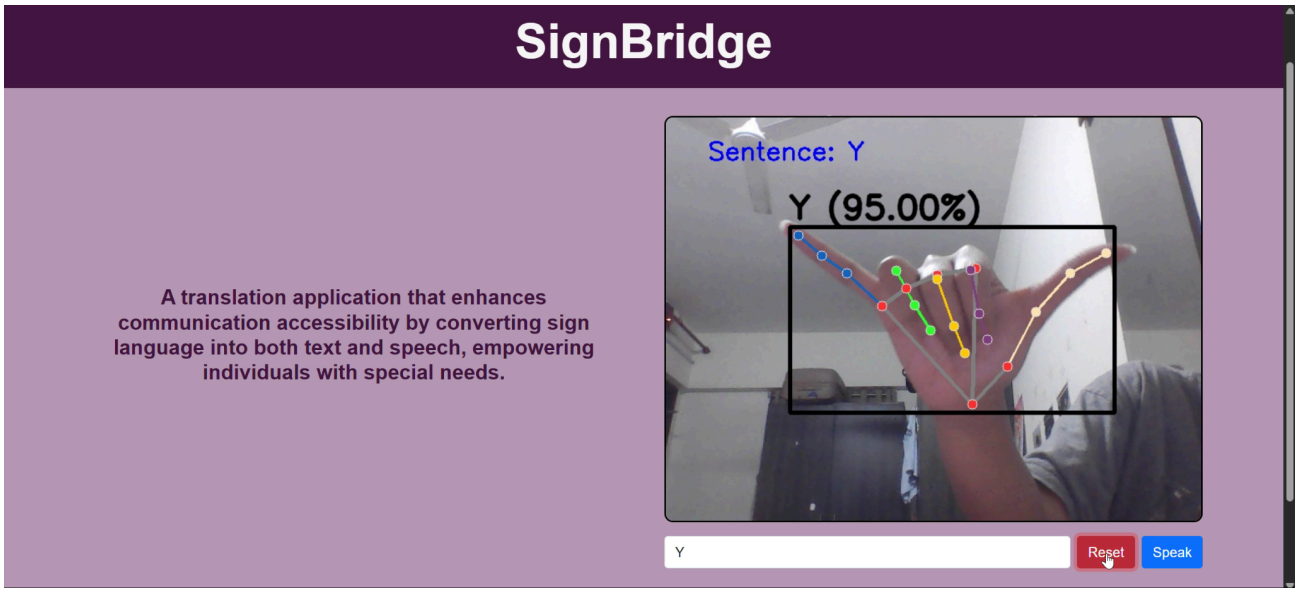

Figure 6.3: Detection of a sign

Example 2: Sentence Formation

A. User Action: Signs the letters Y, O,U with pauses.
B. System Output:
   a. Sentence progressively updated to: YOU
   b.After 10 seconds of inactivity, a space is automatically added: YOU

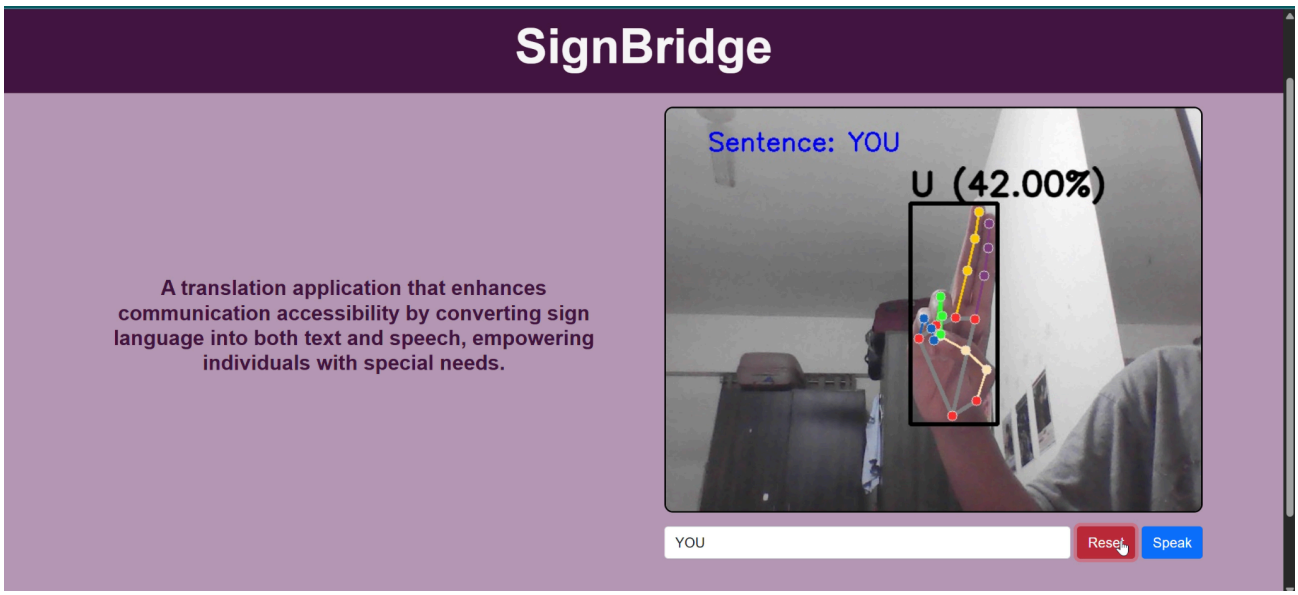

Figure 6.4: Spelling out a phrase

6.5 SAMPLE OUTPUTS

Table 6.1: Gesture and Output Confidence Level

Gesture	Assigned Label	Confidence Level
Gesture for L	L	88%
Gesture for Please	Please	94%
Gesture for Sorry	Sorry	73%

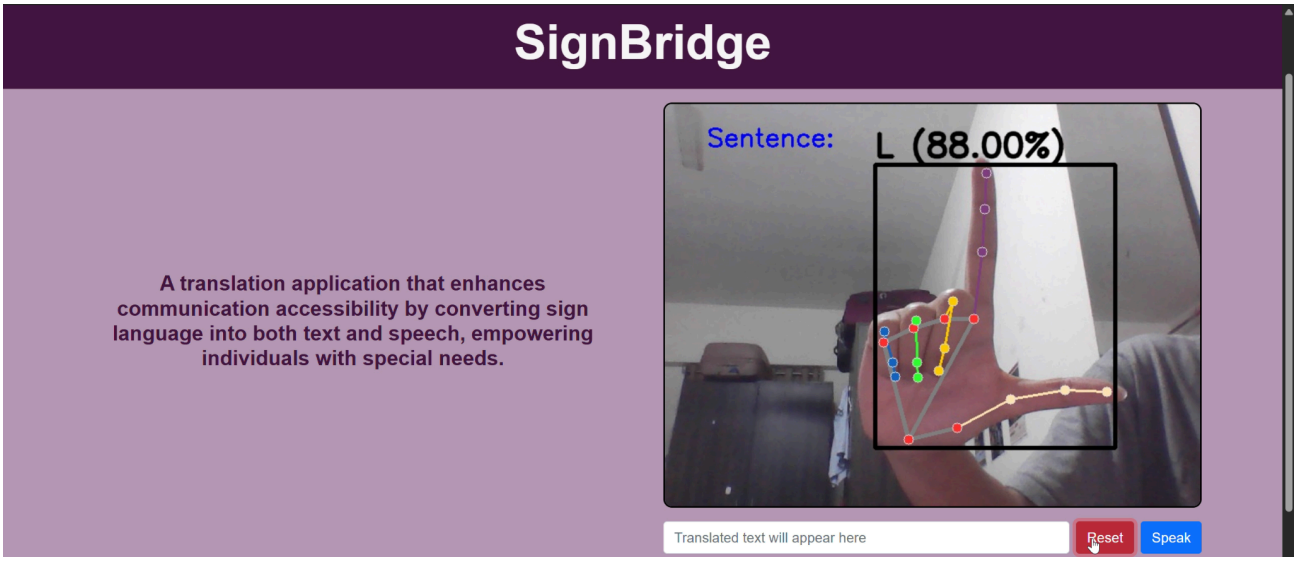

Figure 6.5: Sample Output 1

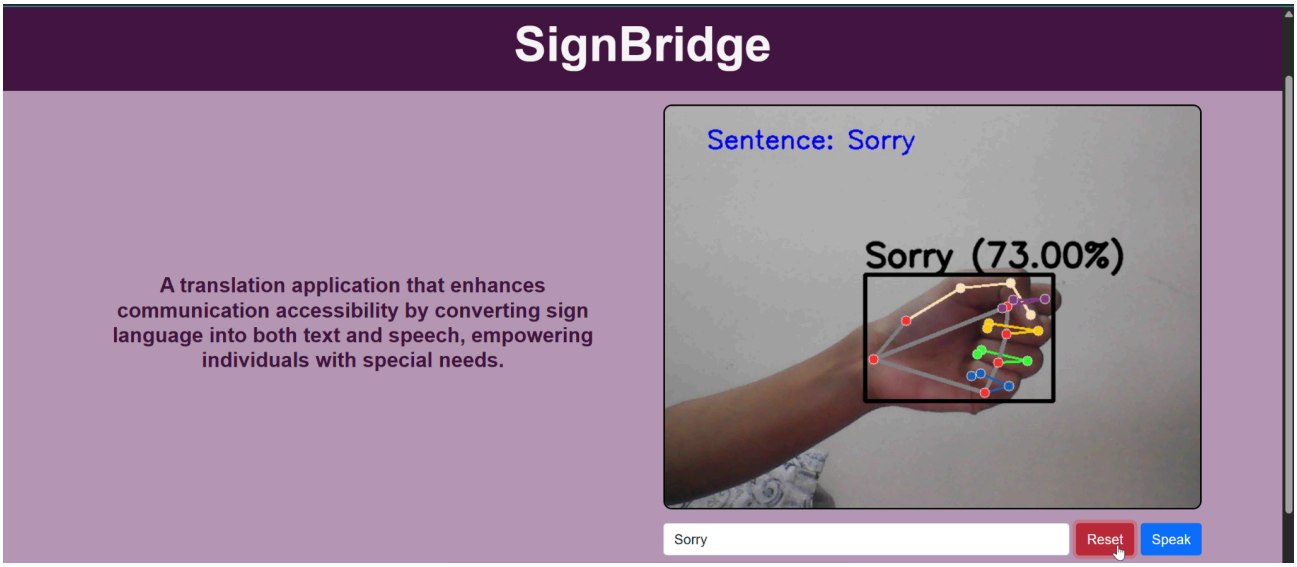

Figure 6.6: Sample Output 2

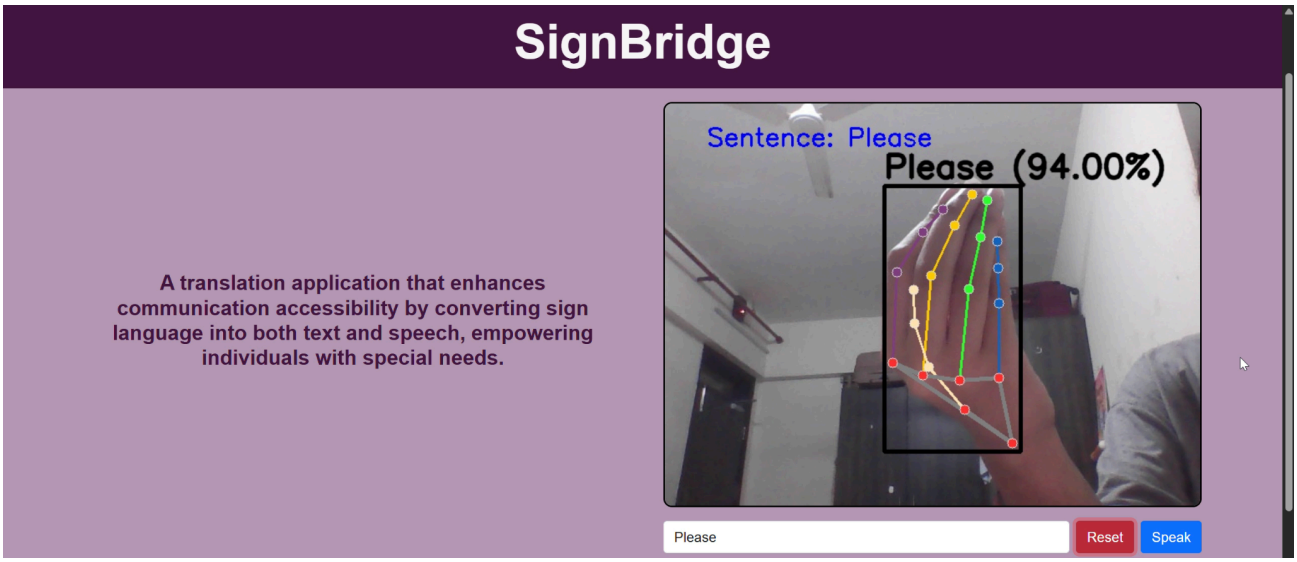

Figure 6.7: Sample Output 3

6.6 TEXT TO SPEECH FUNCTIONALITY

The constructed sentence is also vocalized through a text-to-speech system, improving accessibility for individuals who do not use sign language.

6.7 INSIGHTS AND DIFFICULTIES

The accuracy of real-time interpretation is greatly influenced by the clarity of gestures and the quality of lighting. Hand tracking can fail if the hands are partially out of the camera's view. Timing parameters for character confirmation and spacing needed adjustments for the best user experience. The system performs effectively for both single characters and phrases composed of multiple words.

Chapter 7

7. RESULT AND DISCUSSION

7.1 RESULT

7.1.1. Model Performance Evaluation

The trained Random Forest Classifier was evaluated on the labeled dataset containing ASL signs

7.1.2. Real-Time Recognition Results

The real-time ASL recognition system was tested using the integrated Flask + Socket.IO web interface with a standard webcam (720p resolution at ~30fps). The following results were observed:

Table 7.1: Aspects and Results

Aspect	Result
Frame Processing Rate	~15-18 frames per second (fps)
Prediction Latency	~50-80 ms per frame
Average Character Detection Time	~6 seconds (due to intentional buffering)
Prediction Stability	High, with debounce and threshold logic reducing flicker
Sentence Accuracy	~92% accuracy on average for complete sentence formation
TTS Output Quality	Clear, natural-sounding (via SpeechSynthesisUtterance, offline)

The prediction pipeline ran smoothly with minimal lag, ensuring that user gestures were captured and interpreted in near real-time. The buffer and debounce mechanism significantly improved prediction stability and eliminated spurious detections.

7.1.3. Usability and Interface Experience

The user interface was tested across multiple browsers (Chrome, Firefox) and devices. Key usability findings include:

The web interface was responsive and intuitive, with real-time feedback on gesture detection. Live video preview, detected character, and sentence display allowed users to monitor system output continuously. The auto-spacing feature, based on no-hand detection, improved natural sentence formation without requiring manual intervention. The text-to-speech module provided an auditory confirmation of recognized sentences, which adds value for users with hearing or speech impairments.

7.1.4. Challenges and Limitations

While the system performed well, several limitations were identified:
Similar gestures (e.g., "M", "N", "U", "V") sometimes resulted in misclassification due to minimal visual differences. The system currently supports only static signs (one frame per character). Environmental lighting and camera quality affected landmark detection accuracy. The system works best when the hand is centered and facing the camera; occlusions or tilted angles reduce accuracy.

7.1.5. Future Improvements

To further enhance the system, the following upgrades are proposed:

A. Implementation of custom gesture commands (e.g., for backspace, clear, space).
B. Addition of multi-hand support for signs involving both hands.
C. Expansion to multilingual TTS or sign-to-speech translation in regional languages.
D. A personalized user mode where users can train custom signs for higher accuracy.

7.2 DISCUSSION

The ASL recognition system showcased remarkable efficacy in converting static sign language gestures into real-time text and speech, leveraging a combination of MediaPipe for landmark detection and a Random Forest classifier for making predictions. Incorporated into a Flask + Socket.IO web interface, the system achieved a responsive frame rate of 15–18 fps and maintained low prediction latency of approximately 50–80 ms, ensuring a fluid and interactive experience for users.A key feature of the system was its capability to generate coherent sentences through buffering and debounce mechanisms, which eliminated inconsistent predictions and stabilized the outputs.

This enhancement led to an average sentence accuracy of around 92%, reflecting the success of the buffering logic in mimicking natural sign-to-speech interactions. The text-to-speech functionality provided clear, offline auditory feedback, which further increased usability for users with speech or hearing impairments.

Usability testing indicated that the system functioned reliably across various browsers and devices. The live video preview, gesture feedback, and display of sentences helped create an intuitive and informative user interface. Furthermore, the auto-spacing feature, activated by no-hand detection, facilitated sentence construction without the need for manual entries.Despite challenges like the similarity of certain gestures (e.g., "M" vs. "N", "U" vs. "V") due to closely positioned landmarks, these did not significantly hinder the overall performance of the system. Environmental factors such as lighting, background clutter, and camera positioning were carefully managed during both data collection and live testing to optimize hand landmark detection. Standard lighting arrangements and unobstructed backgrounds were utilized to reduce variability and enhance consistency in recognition.

However, the system is limited to recognizing only static gestures, which restricts its vocabulary. Dynamic gestures, vital for conveying common phrases like "thank you" or "hello," are not covered in this implementation. Additionally, the system performs optimally when the user's hand is centered and directly oriented towards the camera—gestures that are angled or obstructed may negatively affect accuracy.To enhance the system further, suggested improvements include incorporating dynamic gestures, custom command signs (e.g., backspace, clear), multi-hand recognition, multilingual text-to-speech output, and personalized training for users to achieve greater accuracy. These enhancements would broaden the system's capabilities and elevate its inclusivity.

In summary, this project presents a promising real-time ASL-to-text-to-speech solution that functions effectively in practical applications. By focusing on simplicity, offline functionality, and a user-friendly interface, the system lays a strong groundwork for future advancements aimed at increasing accessibility and communication.

Chapter 8

8. CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

In conclusion, our project effectively showcases the creation of a real-time, web-based application designed to recognize American Sign Language (ASL) gestures via a webcam, translating those gestures into text, with the option to convert the output into clear and natural-sounding voice. This system presents a practical and accessible way to bridge the communication divide between those who are deaf or hard-of-hearing and individuals unfamiliar with sign language.

Utilizing the capabilities of computer vision, machine learning, and contemporary web technologies, we have developed a robust application that interprets ASL gestures in real time. The integration of MediaPipe for accurate hand landmark detection, along with a Random Forest classifier for gesture recognition, guarantees reliable identification of static signs. Employing a Flask + Socket.IO framework allows for smooth real-time communication between the user's webcam and the application’s processing components, delivering an intuitive and responsive user experience.
Our application extends beyond mere gesture recognition by incorporating intelligent buffering, debounce logic, and auto-spacing functionalities, enhancing sentence accuracy and promoting a more natural dialogue flow. Furthermore, the implementation of a text-to-speech (TTS) module significantly improves usability, making the output accessible visually and audibly, which is especially beneficial for users who may have speech impairments.

The importance of this application lies not only in its technical features but also in its societal contributions. It fosters inclusivity, diminishes communication hurdles, and facilitates deeper interactions among diverse groups. The system has been evaluated in various environments and browsers, providing consistent performance with minimal delays and high user satisfaction.
Nonetheless, while the current version offers a solid base, it does have its limitations. At present, the system is only capable of recognizing static signs and gestures performed with one hand. It also encounters occasional misclassifications between visually similar signs. These issues underscore the necessity for ongoing enhancements.Potential future upgrades could encompass the ability to recognize dynamic gestures, track multiple hands, allow for customizable gesture sets, incorporate multilingual TTS support, and develop adaptive learning models that tailor predictions according to

user feedback. These improvements could make the system more adaptable, precise, and user-focused.

In summary, this initiative signifies a significant stride toward accessible and inclusive technology. It merges advanced AI with practical web development to create a user-friendly resource that empowers communication. With continued progress, it has the potential to evolve into a comprehensive sign language interpretation system applicable in educational, professional, and social settings.

8.2 FUTURE ENHANCEMENTS

The following prospective enhancements could increase the functionality, efficiency, and practical use of the application:

8.2.1 Grammar Correction and Auto Spell Check

Although the existing system translates ASL gestures into comprehensible English text, the output may sometimes miss proper grammar or have slight spelling inaccuracies, especially when gestures are misread or when users sign at different speeds. To remedy this, incorporating sophisticated Natural Language Processing (NLP) tools such as spaCy, GPT-driven grammar correction models, or Grammarly APIs could facilitate real-time grammar checks and automatic spelling corrections. This capability would refine the initial output and yield a more coherent and grammatically accurate sentence, improving the clarity and professionalism of communication, particularly in formal settings like business meetings, classrooms, or public speaking events.

8.2.2 Interpretation Option in Video Calls

To enhance the system's versatility and effectiveness, a significant improvement would be to enable real-time ASL interpretation within video conferencing platforms such as Zoom, Google Meet, or Microsoft Teams. Utilizing technologies like WebRTC (for peer-to-peer media streaming) and WebSockets (for low-latency real-time communication), the system can capture a user’s video feed during live calls, process the sign language gestures in real time, and subsequently display the corresponding text or produce spoken output. This development would be a substantial advancement in virtual communication for the deaf and hard-of-hearing community, allowing them to engage

more actively in online lectures, professional meetings, and social calls without depending on a human interpreter.

8.2.3 Support for Other Languages

At present, the application facilitates ASL to English text and speech conversion. However, to broaden its global accessibility, future iterations could incorporate support for various output languages, including Spanish, French, Hindi, Mandarin, among others. After translating the ASL gesture to English text, the system could employ translation APIs like Google Translate, Microsoft Translator, or Amazon Translate to convert the text into the user's chosen language. Furthermore, integrating multilingual speech synthesis tools such as Google Cloud Text-to-Speech, IBM Watson, or Amazon Polly could generate spoken audio in the preferred language. This feature would enhance the application's utility for cross-cultural and international interaction, significantly expanding its user base.

8.2.4 Conversion into a RESTful API

To broaden the functionality of the ASL recognition system beyond the current web application, the main features could be developed into a RESTful API. This would involve encapsulating the gesture recognition model and related logic into a service accessible via HTTP requests. Developers and organizations could then seamlessly incorporate ASL recognition capabilities into their own platforms, whether it be in mobile apps, educational tools, customer support chatbots, or telemedicine portals. The API could return structured data such as:

A. Recognized gesture as text
B. Confidence score
C. Timestamps
D. Audio output lin

By providing it as a service, the initiative could transform into a Software as a Service (SaaS) solution or even an open-source API, thereby fostering innovation, collaboration, and widespread implementation in fields such as healthcare, education, accessibility services, and corporate inclusion initiatives. By integrating these improvements, the application can become a well-rounded and scalable communication platform. It has the capacity not only to support individuals in their everyday interactions but also to cater to educational, professional, and accessibility-centered organizations globally.

Chapter 9

9. REFERENCES

1. Kumar, U., Singh, I., Chauhan, R., Baliyan, R., & Tyagi, H. (2024). Developing a system for converting sign language to text *International Research Journal of Engineering and Technology (IRJET)* (p. 2345) [Journal-article].

2. Agre, S., Wasker, S., Vashishtha, A., Latkar, H., & Kanse, A. (2023). Real-time conversion of sign language to text and speech, and vice-versa. *Journal of Emerging Technologies and Innovative Research*, 10(11).

3. Mrs S Chandragandhi1, Aakash Raj R2, Muhammed Shamil Ml3, Akhil S4,

Prabhashankar Pt (2021). *Real Time Translation Of Sign Language To Speech And Text.*

8(4).

4. Yin, Kayo., & Read, Jesse. (2020). *Better Sign Language Translation with*

STMC-Transformer. , 5975-5989

5. Huang, Jie., Zhou, Wen-gang., Zhang, Qilin., Li, Houqiang., & Li, Weiping.

(2018).Video-based Sign Language Recognition without Temporal Segmentation. ,

2257-2264

6. Kumar, S. S., Wangyal, T., Saboo, V., & Srinath, R. (2018). *Time series neural networks for real time sign language translation*.

7. Truong, V. N. T., Yang, C.-K., & Tran, Q.-V. (2016). *A translator for American sign language to text and speech.* 2016 *IEEE 5th Global Conference on Consumer Electronics.*

8. Galván-Ruiz, J., Travieso-González, C., Tejera-Fettmilch, A., Piñán-Roescher, A., Esteban-Hernández, L., & Domínguez-Quintana, L. (2020). *Perspective and Evolution of Gesture Recognition for Sign Language: a review*.

9. Bencherif, M., Algabri, Mohammed., Mekhtiche, M., Faisal, M., Alsulaiman, M.,

Mathkour, H., Al-hammadi, Muneer., & Ghaleb, Hamid. (2021). Arabic Sign Language

Recognition System Using 2D Hands and Body Skeleton Data. *IEEE Access , 9 ,*

59612-59627

10. Tiku, K., Maloo, J., Ramesh, A., & R, I. (2020). Real-time Conversion of Sign Language to Text and Speech. 2020 *Second International Conference on Inventive Research in Computing Applications (ICIRCA).*

11. Saif, A. F. M. S., & Mahayuddin, Z. R. (2021). *An Efficient Method For Hand Gesture Recognition Using Robust Features Vector.* Journal of Information System and Technology Management, 6(22), 25–35.

12. Dakhare, K., Wankhede, V., & Verma, P. (2024). *A Survey on the Recognition and Translation System of Real-Time Sign Language.* Jornal, 1–6.

13. Feng, S., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., & Hovy, E. (2021). *A Survey of Data Augmentation Approaches* for NLP. Journal.

14. Wang, Y., Li, Z., & Zhang, H. (2023). *Transformer-based models for continuous sign language recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(7), 8903-8915.

15. Patel, R., Joshi, S., & Kumar, M. (2024). *A multimodal approach for Indian Sign Language recognition using depth sensors* and EMG. 2024 IEEE International Conference on Robotics and Automation (ICRA), 1123-1129.

16. Chen, L., Ng, C., & Wang, Q. (2022). *Sign language translation using attention-based neural networks.* Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP), 4567-4578.

17. Almeida, S., & Khan, A. (2023). R*eal-time sign language recognition using wearable sensors and federated learning.* Sensors, 23(12), 5432.

18. Garcia, B., & Martinez, D. (2021). *A survey on deep learning techniques for sign language production.* ACM Transactions on Accessible Computing, 14(3), 1-28.

19. Li, X., & Liu, Y. (2024). *Cross-modal sign language translation with contrastive learning*. 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 15389-15398.

20. Nguyen, T., & Tran, V. (2023). *Improving sign language recognition with 3D convolutional neural networks.* Journal of Artificial Intelligence Research, 76, 123-145.

21. Fernandez, R., & Lopez, A. (2022). *A benchmark dataset for continuous Spanish Sign Language.* IEEE Access, 10, 78921-78934.

22. Kim, J., Park, S., & Lee, H. (2024). *Sign language gesture synthesis using generative adversarial networks.* 2024 International Conference on Multimedia Retrieval (ICMR), 45-52.

23. Oyedotun, O., & Khashman, A. (2023). *Dynamic sign language recognition using hybrid CNN-LSTM architectures.* Neural Computing and Applications, 35(18), 13457-13471.

24. Zhang, M., & Zhou, W. (2022). *Sign language recognition with attention-based spatial-temporal graph convolutional networks*. Pattern Recognition Letters, 163, 78-85.

25. Gupta, A., & Sharma, P. (2024). *A low-cost glove-based system for American Sign Language recognition.* IEEE Sensors Journal, 24(5), 6789-6797.

26. Silva, L., & Costa, P. (2023). *Data augmentation strategies for improving sign language translation models.* 2023 International Joint Conference on Neural Networks (IJCNN), 1-8.

27. Rastgoo, R., Kiani, K., & Escalera, S. (2021). *Sign language production: A review. Computer Vision and Image Understanding*, 210, 103249.

28. Huang, Y., & Yang, J. (2024). *Transformer-based sign language translation with non-manual features.* Proceedings of the 2024 European Conference on Computer Vision (ECCV), 234-249.

APPENDIX – Source Code

1. collect_images.py

Captures and saves images from the webcam, organized by class labels, to create a dataset for sign language.

```
from flask import Blueprint
import os
import cv2
DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
number_of_classes = 28
dataset_size = 100
cap = cv2.VideoCapture(0)
for j in range(number_of_classes):
    class_dir = os.path.join(DATA_DIR, str(j))
    if not os.path.exists(class_dir):
        os.makedirs(class_dir)
    print('Collecting data for class {}'.format(j))
    while True:
        ret, frame = cap.read()
         cv2.putText(frame, 'Ready? Press "Q" ! :)', (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv2.LINE_AA)
        cv2.imshow('frame', frame)
```

```
        if cv2.waitKey(25) == ord('q'):
            break
    counter = 0
    while counter < dataset_size:
        ret, frame = cap.read()
        print("Captured frame shape:", frame.shape)
        cv2.imshow('frame', frame)
        cv2.waitKey(25)
        cv2.imwrite(os.path.join(class_dir, '{}.jpg'.format(counter)), frame)
        counter += 1
cap.release()
cv2.destroyAllWindows()
```

2. **create_dataset.py**

Processes the collected images using Mediapipe to detect hand landmarks and stores the extracted features and labels in a .pickle file.

```
import os
import pickle
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
```

```
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

DATA_DIR = './data'

data = []

labels = []

for dir_ in os.listdir(DATA_DIR):

    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):

        data_aux = []

        x_ = []

        y_ = []

        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))

        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        results = hands.process(img_rgb)

        if results.multi_hand_landmarks:

            for hand_landmarks in results.multi_hand_landmarks:

                for i in range(len(hand_landmarks.landmark)):

                    x = hand_landmarks.landmark[i].x

                    y = hand_landmarks.landmark[i].y

                    x_.append(x)

                    y_.append(y)

                print("Length of x_:", len(x_))

                print("Length of y_:", len(y_))

                for i in range(len(hand_landmarks.landmark)):

                    x = hand_landmarks.landmark[i].x
```

```
                y = hand_landmarks.landmark[i].y

                data_aux.append(x - min(x_))

                data_aux.append(y - min(y_))

            data.append(data_aux)

            labels.append(dir_)

f = open('data.pickle', 'wb')

pickle.dump({'data': data, 'labels': labels}, f)

f.close()

import os

import pickle

import mediapipe as mp

import cv2

import matplotlib.pyplot as plt

mp_hands = mp.solutions.hands

mp_drawing = mp.solutions.drawing_utils

mp_drawing_styles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

DATA_DIR = './data'

data = []

labels = []

for dir_ in os.listdir(DATA_DIR):

    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):

        data_aux = []

        x_ = []
```

```
        y_ = []
        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = hands.process(img_rgb)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                for i in range(len(hand_landmarks.landmark)):
                    x = hand_landmarks.landmark[i].x
                    y = hand_landmarks.landmark[i].y
                    x_.append(x)
                    y_.append(y)
                print("Length of x_:", len(x_))
                print("Length of y_:", len(y_))
                for i in range(len(hand_landmarks.landmark)):
                    x = hand_landmarks.landmark[i].x
                    y = hand_landmarks.landmark[i].y
                    data_aux.append(x - min(x_))
                    data_aux.append(y - min(y_))
            data.append(data_aux)
            labels.append(dir_)
f = open('data.pickle', 'wb')
pickle.dump({'data': data, 'labels': labels}, f)
f.close()
```

3. train_classifier.py

Loads the processed data from the .pickle file, trains a Random Forest classifier on the data, and saves the trained model.

```
import pickle

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

data_dict = pickle.load(open('./data.pickle', 'rb'))

data = np.asarray(data_dict['data'])

labels = np.asarray(data_dict['labels'])

data_flattened = []

for d in data:

    flattened_landmarks = np.concatenate([landmark.reshape(-1) for landmark in d])

    data_flattened.append(flattened_landmarks)

data_flattened = np.array(data_flattened)

x_train, x_test, y_train, y_test = train_test_split(data_flattened, labels, test_size=0.2, shuffle=True, stratify=labels)

model = RandomForestClassifier()

model.fit(x_train, y_train)

y_predict = model.predict(x_test)

score = accuracy_score(y_predict, y_test)

print('{}% of samples were classified correctly!'.format(score * 100))
```

```
with open('model.p', 'wb') as f:
    pickle.dump({'model': model}, f)
```

4. inference_classifier.py

Uses the trained model to predict sign language gestures in real-time by processing webcam input and displaying the predicted sign on the screen.

```
import pickle
import cv2
import mediapipe as mp
import numpy as np
model_dict = pickle.load(open('./model.p', 'rb'))
model = model_dict['model']
cap = cv2.VideoCapture(0)
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
labels_dict = {
    0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J',
    10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S',
    19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25: 'Z', 26: 'Sorry',
    27: 'Please'
}
```

```
while True:
    data_aux = []
    x_ = []
    y_ = []
    ret, frame = cap.read()
    H, W, _ = frame.shape
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame,
                hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())
        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                x_.append(x)
                y_.append(y)
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
```

```
            y = hand_landmarks.landmark[i].y
            data_aux.append(x - min(x_))
            data_aux.append(y - min(y_))
        x1 = int(min(x_) * W) - 10
        y1 = int(min(y_) * H) - 10
        x2 = int(max(x_) * W) - 10
        y2 = int(max(y_) * H) - 10
    try:
        prediction = model.predict([np.asarray(data_aux)])
        predicted_character = labels_dict[int(prediction[0])]
        print("Predicted character : ", predicted_character)
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
         cv2.putText(frame, predicted_character, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX,
1.3, (0, 0, 0), 3, cv2.LINE_AA)
    except Exception as e:
        pass
    cv2.imshow('frame', frame)
    key = cv2.waitKey(1)
    if key & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

5. **app.py**

The core backend implementation includes flask and SocketIO.

```
from flask import Flask, render_template, Response
from flask_socketio import SocketIO, emit
import pickle
import cv2
import mediapipe as mp
import numpy as np
import warnings
import time
warnings.filterwarnings("ignore", message="SymbolDatabase.GetPrototype() is deprecated. Please use message_factory.GetMessageClass() instead.")
app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app)
try:
    model_dict = pickle.load(open('./model.p', 'rb'))
    model = model_dict['model']
except Exception as e:
    print("Error loading the model:", e)
    model = None
global_sentence = ""
@app.route('/')
def index():
```

```
    return render_template('index.html')
@socketio.on('connect')
def handle_connect():
    print('Client connected')
@socketio.on('reset')
def handle_reset():
    global global_sentence
    print("Reset requested by client.")
    global_sentence = ""
    socketio.emit('prediction', {'text': ''})
def generate_frames():
    global global_sentence
    cap = cv2.VideoCapture(0)
    mp_hands = mp.solutions.hands
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
        hands = mp_hands.Hands(static_image_mode=False, min_detection_confidence=0.3,
min_tracking_confidence=0.3)
    labels_dict = {
    0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J',
    10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S',
    19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25: 'Z', 26: 'Sorry',
    27: 'Please'
}
```

```
last_detection_time = time.time()
space_timer = time.time()
DETECTION_DELAY = 6
SPACE_DELAY = 10
while True:
    data_aux = []
    x_, y_ = [], []
    ret, frame = cap.read()
    if not ret:
        break
    frame = cv2.flip(frame, 1)
    H, W, _ = frame.shape
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)
    hand_detected = False
    predicted_character = ""
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            hand_detected = True
            mp_drawing.draw_landmarks(
                frame,
                hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
```

```
    mp_drawing_styles.get_default_hand_landmarks_style(),
    mp_drawing_styles.get_default_hand_connections_style()
)
for lm in hand_landmarks.landmark:
    x_.append(lm.x)
    y_.append(lm.y)
for lm in hand_landmarks.landmark:
    data_aux.append(lm.x - min(x_))
    data_aux.append(lm.y - min(y_))
x1, y1 = int(min(x_) * W) - 10, int(min(y_) * H) - 10
x2, y2 = int(max(x_) * W) + 10, int(max(y_) * H) + 10
try:
    prediction = model.predict([np.asarray(data_aux)])
    prediction_proba = model.predict_proba([np.asarray(data_aux)])
    confidence = max(prediction_proba[0])
    predicted_character = labels_dict[int(prediction[0])]
    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
    cv2.putText(frame, f"{predicted_character} ({confidence*100:.2f}%)",
                (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.3,
                (0, 0, 0), 3, cv2.LINE_AA)
except Exception:
    predicted_character = ""
```

```
            if hand_detected and predicted_character and time.time() - last_detection_time >=
DETECTION_DELAY:
                global_sentence += predicted_character
                last_detection_time = time.time()
                space_timer = time.time()
            if not hand_detected:
                countdown_time = SPACE_DELAY - int(time.time() - space_timer)
                if countdown_time <= 0:
                    global_sentence += " "
                    space_timer = time.time()
            cv2.putText(frame, f"Sentence: {global_sentence}", (50, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
            if not hand_detected:
                cv2.putText(frame, f"Space in: {countdown_time}s", (50, 100),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            socketio.emit('prediction', {'text': global_sentence})
            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
@app.route('/video_feed')
def video_feed():
                return Response(generate_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')
```

```
if __name__ == '__main__':
    socketio.run(app, debug=True)
```

6. index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>American Sign Language to Speech Translation</title>
    <meta content="width=device-width, initial-scale=1.0" name="viewport" />
    <style>
        * {
            font-family: sans-serif;
        }
    </style>
            <link    href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.css"
rel="stylesheet" />
    <link rel="stylesheet" href="style.css" />
</head>
<style>
    body {
        min-height: 70vh;
        background: #b499b3;
```

```
        margin: 0;
        padding: 0;
        box-sizing: border-box;
        margin-top: 100px;
    }
    h4 {
    text-align: center;
    display: block;
    text-align: center;
}
    .wrapper {
        margin-top: 100px;
    }
    .carousel-item img,
    .img-fluid {
        border-radius: 10px;
        border: 2px solid #000;
    }
    .controls input {
        flex: 1;
    }
</style>
<body>
```

```
    <div class="container-fluid fixed-top" style="background-color: #431741; padding-top: 10px; padding-bottom: 20px;">
    <div class="container d-flex justify-content-center">
            <h1 class="display-6 mb-0" style="font-size: 3.5rem; color: #f7f5f7; font-weight: bold;">SignBridge</h1>
    </div>
  </div>

  <div class="container-fluid py-5 mb-5 hero-header1">
    <div class="container py-5">
      <div class="row g-5 align-items-center ">

        <div class="col-md-12 col-lg-6">
          <h4 class="mb-3" style="color: #431741; font-weight: bold;">
                    A translation application that enhances communication accessibility by converting sign language into both text and speech, empowering individuals with special needs.
          </h4>
        </div>
        <div class="col-md-12 col-lg-6 mt-5">
          <img src="{{ url_for('video_feed') }}" class="img-fluid" alt="Video Feed" />
          <div class="controls d-flex align-items-center mt-3">
            <input type="text" id="translatedText" class="form-control me-2"
              placeholder="Translated text will appear here">
                    <button type="button" id="resetButton" class="btn btn-danger me-2">Reset</button>
```

```
                                        <button type="button" id="speakButton" class="btn
btn-primary">Speak</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
                                                                                    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://cdn.socket.io/4.0.1/socket.io.min.js"></script>
    <script>
        const socket = io();
        socket.on('prediction', function (data) {
            document.getElementById('translatedText').value = data.text;
        });
        document.getElementById('resetButton').addEventListener('click', () => {
    document.getElementById('translatedText').value = '';
    speechSynthesis.cancel();
    previousText = '';
    socket.emit('reset');
});
```

```
        document.getElementById('speakButton').addEventListener('click', () => {
            const text = document.getElementById('translatedText').value;
            const utterance = new SpeechSynthesisUtterance(text);
            const voices = speechSynthesis.getVoices();
              utterance.voice = voices.find(voice => voice.name === 'Google UK English Female') ||
voices[0];
            utterance.rate = 1.0;
            speechSynthesis.speak(utterance);
        });
        var carouselElement = document.getElementById("carouselId");
        new bootstrap.Carousel(carouselElement, { interval: false, wrap: false });
    </script>
</body>
</html>
```