KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

COLLEGE OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

DOCUMENTATION

TOPIC: SIGN LANGUAGE TRANSLATOR- 2025

<u>PROJECT SUPERVISOR</u>

DR. DOMINIC ASAMOAH

SUBMITTED BY:

SACKEY JUSTICE OBENG - 8677421

ISRAEL GLADSTONE YESUEDEM MONYO - 8557621

DECLARATION

I declare, without any reservation, that I undertook the study herein submitted under supervision.

------------------------------------ ------------------------------------

DATE SACKEY JUSTICE OBENG

I declare, without any reservation, that I undertook the study herein submitted under supervision.

------------------------------------ ------------------------------------

DATE ISRAEL GLADSTONE YESUEDEM

I declare that I have supervised the students in undertaking the study submitted herein and confirmed that the students have my permission to present it for assessment.

---------------------------------- ---------------------------------

DATE DR. DOMINIC ASAMOAH

ACKNOWLEDGEMENT

We wish to express our sincere gratitude to God for the knowledge, strength, and wisdom he showered on us throughout this project.

We would also like to express our special thanks to my supervisor Dr. Dominic Asamoah who gave us the golden opportunity to do this project and for the guidance and encouragement he gave us in carrying out this project.

We would also like to express our gratitude to our parents for their support.

ABSTRACT

Communication barriers between Deaf and Hard-of-Hearing (DHH) individuals and hearing users remain a significant challenge in education, healthcare, and public services. Existing solutions often require high-end hardware or internet connectivity, limiting their accessibility in resource-constrained environments. This project proposes a real-time, offline-capable Sign Language Translator that leverages MediaPipe for extracting 21-point hand landmarks and a Random Forest classifier for efficient gesture recognition on CPU-only systems. The system focuses on American Sign Language (ASL) alphabet gestures, converting them into text while also providing text-to-sign video playback for bidirectional communication. A Flask-based web interface ensures low-latency interaction, sentence buffering, and fallback mechanisms for missing assets. By prioritizing accuracy, usability, and privacy, this solution promotes inclusivity, scalability, and adaptability for classrooms and public service environments, laying the groundwork for future extensions such as dynamic gesture recognition and multilingual sign language support.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

The recorded history of sign language in Western societies dates back to the 17th century, as a visual language or communicative form, although references to the forms of communication using sign language date back to the 5th century BC Greece. Sign language is made up of a system of general gestures, imitation, gestures and spelling, and the use of gestures to represent the letters of the alphabet. Symbols can also represent complete ideas or phrases, not just individual words. Many sign languages are native languages, distinct from the structure of spoken languages used near them, and are mainly used by deaf people to speak.

Many sign languages have developed independently around the world, and no sign language can be identified. Both signed systems and handwritten characters have been found worldwide. Until the 19th century, much of what we know about historical sign languages is limited to the characters of the alphabet that were developed to facilitate the transfer of words from spoken language into sign language, rather than from the language itself. Talking to people with a hearing impairment is a major challenge. Deaf and mute people use sign language to communicate, which is why ordinary people face the problem of recognizing their sign language. There is therefore a need for programs that recognize different signals and transmit information to ordinary people.

Indian Sign Language (ISL) is used in a community of deaf people throughout India. But ISL is not used in deaf schools to teach deaf children. Teacher training programs do not guide teachers in teaching methods that use ISL. There are no tutorials that include sign language. Parents of deaf children are unaware of sign language and its ability to remove communication barriers. ISL interpreters are an urgent need in institutions and places where communication between the deaf and the normal occurs but less than 300 translators are in India. Therefore, an institution that meets all of these requirements was a necessity. After a long struggle for the deaf community, the Department approved the establishment of the ISLRTC in New Delhi on 28 September 2015. Unlike spoken languages, where grammar is expressed using punctuation based symbols, feature, attitude and syntax sign languages use gesture, punctuation, and body and facial expressions to form grammar. In this, we will focus on American Sign Language, the most widely used sign language in the United States. We will also look at Real Signed English (SEE) and English Signed Pidgin (PSE), two alternatives to ASL that are widely used between deaf people and hearing people. SEE and PSE depend on the English language to varying degrees. This means that unlike ASL, they are made of artificial sign languages. We will talk about the effort to establish a global sign language and look at other sign language applications.

PROBLEM STATEMENT

Despite policy commitments to inclusion, practical communication barriers persist in schools and public services. Reliance on ad-hoc interpreters and note writing reduces participation, slows service delivery, and can compromise privacy. Existing academic prototypes frequently target offline, static alphabet classification using deep CNNs, which require large datasets and GPUs. In contrast, classrooms and ICT labs often run commodity hardware with intermittent connectivity. There is a need for a lightweight, real-time solution that: (1) recognizes signs robustly from webcams; (2) operates fully offline; and (3) also converts typed text into sign videos to serve hearing users engaging DHH peers.

AIM OF THE PROJECT

The primary objective of this project is to significantly enhance communication and accessibility for individuals who are deaf or hard of hearing by fostering a thorough understanding and practical application of American Sign Language (ASL) across various community settings. The project aims to educate participants on the foundational elements of ASL, including its grammar, vocabulary, and syntax, while also promoting cultural awareness and sensitivity towards the Deaf community. By providing comprehensive ASL training and resources, the project seeks to empower participants to engage effectively with ASL users, thereby facilitating their inclusion in everyday interactions and activities. Additionally, the project aspires to create a more inclusive and supportive environment by bridging the communication gap between ASL users and the broader community, ultimately contributing to a society that values and respects linguistic and cultural diversity.

SPECIFICATIONS OBJECTIVES OF THE PROJECTS

1. Developing software that will recognize hand gestures and convert them into text: Collects and Preprocess a Comprehensive Dataset of ASL Alphabet Gestures;

Description : Gather a diverse and extensive dataset comprising American Sign Language (ASL) alphabet gestures. This dataset will include a wide range of hand movements and variations to ensure the robustness and accuracy of the model.

Preprocessing steps such as data cleaning, normalization, and augmentation will be performed to enhance the quality and diversity of the dataset.

2. The proposed system is a sign language recognition system using the Random Forest classifier. Function (Random Forest) that recognizes various hand gestures by capturing and converting hand gestures into alphabet:This develops a Random Forest-Based Model for Recognizing ASL Gestures;

Description : Design and implement a Random Forest classifier (Random Forest) architecture tailored specifically for recognizing ASL gestures from the collected dataset. The Random Forest model will be trained on the collected ASL gesture dataset using supervised learning techniques. The architecture will comprise multiple layers of convolutional, pooling, and fully connected layers optimized for feature extraction and classification of ASL gestures. Techniques such as transfer learning and fine-tuning may be explored to leverage pre-trained Random Forest models and enhance performance

3. Implement a User-Friendly Interface for Real-Time Gesture Recognition and Translation:

Description : Develop an intuitive and user-friendly interface that allows real-time recognition and translation of ASL gestures into text. The interface will feature a responsive design optimized for usability on various devices, including desktop computers, tablets, and smartphones. It will integrate with the trained random forest model to enable seamless gesture recognition and translation. User feedback and usability testing will inform iterative improvements to the interface design, ensuring accessibility and inclusivity for all users, including those with disabilities.

4. Hand pixels are segmented and sent for image and comparison Trained model. Identification of hand gestures are done using machine learning libraries such as TensorFlow and OpenCV images.

JUSTIFICATION OF PROJECT

The need for this ASL project stems from the significant communication gap that still exists between the deaf and hard of hearing community and the wider population. Despite technological advancements and growing awareness, people who use ASL continue to face substantial barriers in daily interactions, leading to feelings of social isolation and difficulty accessing essential services. This project aims to tackle these issues head-on by offering comprehensive ASL education and raising cultural awareness, thereby fostering an inclusive environment where ASL users can engage fully and equally. By improving ASL skills among community members, we not only empower individuals with hearing impairments but also enrich the entire community, promoting diversity, empathy, and mutual understanding. Ultimately, this project aspires to create a more accessible and equitable society, aligning with broader goals of social justice and human rights.

MOTIVATION FOR UNDERTAKING PROJECT

The project is driven by a deep understanding of the daily communication challenges faced by the deaf and hard of hearing community, rooted in empathy and compassion for their right to independent interaction and participation. Motivated by a commitment to inclusivity and equality, the project aims to break down communication barriers and create a more inclusive society. By harnessing technological innovations in computer vision, machine learning, and artificial intelligence, the project seeks to leverage technology for social good and make a positive impact on the lives of those facing communication barriers and social exclusion. The team's personal connections and experiences with individuals in the DHH (Deaf and Hard-of-Hearing) community fuel their passion and sense of responsibility, driving their dedication to the project's success. With a long-term vision of minimizing communication barriers and advocating for accessibility and equality, the project aspires to create lasting change and a positive legacy for future generations.

SCOPE OF PROJECT

The project is centered around recognizing static American Sign Language (ASL) alphabet gestures, focusing specifically on individual hand configurations that represent letters, while excluding dynamic movements. By honing in on static gestures, the project aims to develop a specialized system capable of accurately interpreting each ASL alphabet letter. Furthermore, the system's design emphasizes adaptability to diverse environments, ensuring effectiveness both indoors and outdoors, under varying lighting conditions and backgrounds. Special attention is given to robustness against environmental factors such as background noise, distractions, and obstruction , all to guarantee reliable performance in real world situations. This approach reflects a commitment to creating technology that not only understands the nuances of ASL but also functions seamlessly in the complex and varied contexts of everyday life.

PROJECT LIMITATIONS

Limited to Recognizing Predefined ASL Words:

Scope Restriction: The current system is limited to recognizing a predefined vocabulary of ASL words. It does not yet support dynamic phrase-level recognition or continuous signing with natural transitions.

Sentence Formation: While recognized words can be combined to form sentences, the system does not apply ASL grammar rules or context-aware language modeling. Sentence construction is sequential and depends on user input order.

Dependency on Dataset Quality and Diversity: Dataset Influence: The accuracy and robustness of the Random Forest classifier depend heavily on the quality, size, and diversity of the word-level dataset. Variations in lighting, background, and signer characteristics can affect performance.

Data Availability: Limited availability of high-quality ASL word datasets poses challenges for training and evaluation. Additional data collection from multiple users under varied conditions is essential for improving generalization.

Hardware and Performance Constraints: CPU-Only Limitation: The system is optimized for CPU-based environments, which may limit frame rates compared to GPU-accelerated solutions. Performance tuning (e.g., reducing input resolution) may be required on low-end machines.

Text-to-Sign Vocabulary Coverage: Asset Dependency: The text-to-sign module relies on a curated video library. Missing word assets trigger fallback to letter-spelling, which may reduce naturalness during communication.

BENEFICIARIES OF THE PROJECT

The Sign Language Translator project benefits multiple stakeholders. Primary beneficiaries are Deaf and Hard-of-Hearing (DHH) individuals, who gain improved access to real-time communication, fostering independence and inclusion in daily activities. Educational institutions can create more inclusive classrooms by enabling seamless interaction between DHH students and teachers. Non-profit organizations and advocacy groups can leverage the system to enhance support services and promote accessibility. Researchers and developers benefit from the project's reproducible, landmark-based pipeline, which supports further innovation in gesture recognition and assistive technologies. Additionally, secondary beneficiaries include teachers, clinicians, and peers who can communicate effectively without waiting for interpreters.

ACADEMIC AND PRACTICAL RELEVANCE OF THE PROJECT

The project has both academic and real-world significance. Academically, it advances research in sign language recognition by demonstrating that classical machine learning on MediaPipe landmarks can achieve real-time performance with modest data. Practically, it provides a web-based, offline-capable, bidirectional communication tool that improves social interaction, education, and accessibility for the Deaf and Hard-of-Hearing (DHH) community. This combination of technical rigor and practical usability makes the system suitable for inclusive classrooms, clinics, and future extensions.

Project Activity Planning and Schedules:

Phase 1 (Weeks 1–2): Requirements & ethics.

Phase 2 (Weeks 3–6): Data capture & augmentation.

Phase 3 (Weeks 7–10): Model training & Flask UI.

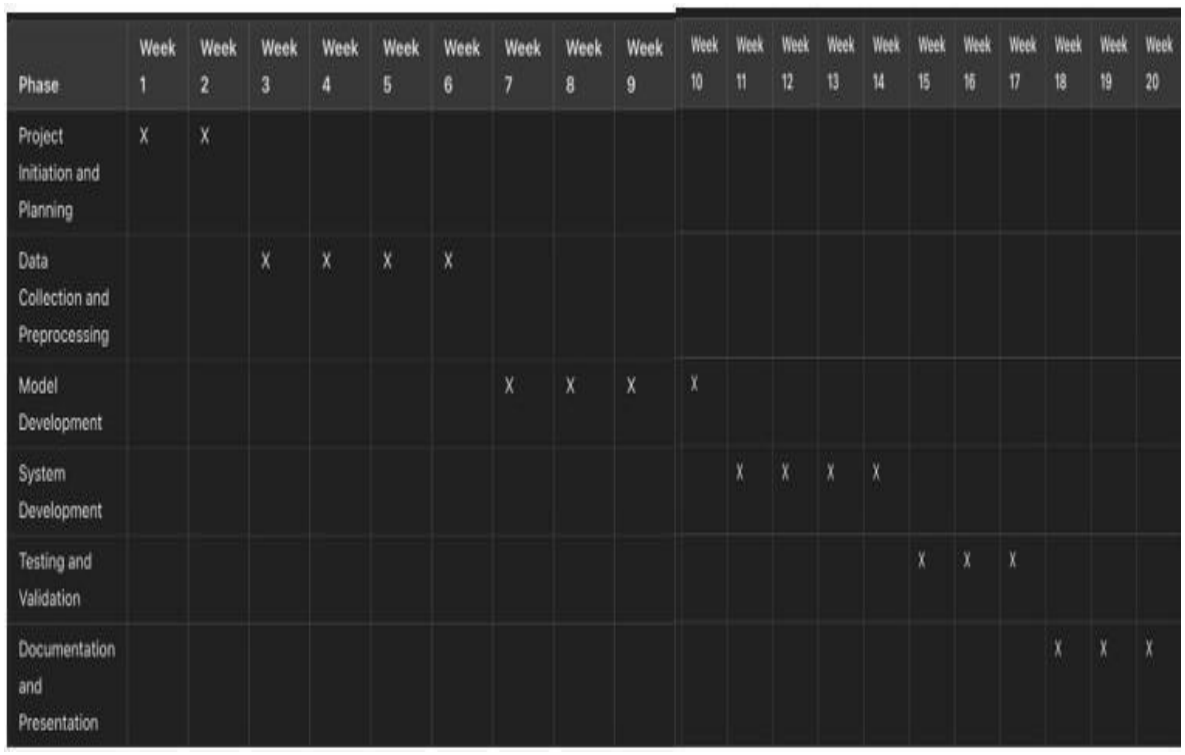
Phase 4 (Weeks 11–14): Integration of text-to-sign; configuration; QA.

Phase 5 (Weeks 15–17): Field testing & user studies.

Phase 6 (Weeks 18–20): Documentation & dissemination.

Structure of Report: The report follows five chapters: background and objectives; related systems and architecture; methodology; implementation and evaluation; findings and future work.

UI Design: Design wireframes and user interface (UI) mockups. UI Implementation: Implement the UI using web technologies (Flask).



Phase	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20
Project Initiation and Planning	X	X																		
Data Collection and Preprocessing			X	X	X	X														
Model Development							X	X	X	X										
System Development											X	X	X	X						
Testing and Validation															X	X	X			
Documentation and Presentation																		X	X	X

PROJECT DELIVERABLES

The project deliverables for the American Sign Language (ASL) recognition system are focused on developing and implementing a comprehensive solution that enhances communication for the deaf and hard of hearing community. Here are the key deliverables:

1. Dataset Collection and Preprocessing:

❖ Collect a diverse and extensive dataset of ASLword gestures.

❖ Perform data cleaning, normalization, and augmentation to ensure quality and diversity.

2. Development of Model

❖ Design and implement a random forest model tailored for ASL gesture recognition.

❖ Train the model on the collected dataset using supervised learning techniques.

- ❖ Explore transfer learning and fine-tuning to enhance model performance.

3. User-Friendly Interface for Real-Time Recognition:

- ❖ Develop an intuitive interface that allows real-time recognition and translation of ASL gestures into sentences.
- ❖ Optimize the interface for various devices, including desktops, tablets, and smartphones.
- ❖ Conduct user feedback and usability testing to improve interface accessibility and inclusivity.

4. Integration of machine learning Libraries:

- ❖ Use machine learning libraries such as TensorFlow ,Pickle, Matplotlib and OpenCV for hand gesture recognition.
- ❖ Implement image segmentation and comparison techniques to enhance recognition accuracy.

5. Comprehensive ASL Training and Resources:

- ❖ Provide educational materials on ASL grammar, vocabulary, and syntax.
- ❖ Promote cultural awareness and sensitivity towards the Deaf community.

6. Performance Evaluation and Iterative Improvement:

- ❖ Continuously evaluate the performance of the recognition system.
- ❖ Implement iterative improvements based on user feedback and performance metrics.

7. Documentation and User Guides:

- ❖ Develop comprehensive documentation for the system, including user guides and technical manuals.

- Ensure that documentation is accessible and easy to understand for all users.

These deliverables aim to create a robust and user-friendly ASL recognition system that facilitates effective communication and promotes inclusivity for the deaf and hard of hearing community.

CHAPTER TWO

2.0RELATED OR EXISTING SYSTEMS

Chapter 2 reviews related works and introduces a proposed system for real-time recognition of American Sign Language (ASL) gestures. It compares two existing sign language applications—HandTalk and ISL Journey—highlighting their features, strengths, and limitations. HandTalk primarily translates text into Brazilian Sign Language (LIBRAS) using animated avatars, while ISL Journey provides a structured curriculum for learning Indian Sign Language (ISL). Both systems contribute significantly to accessibility and education but face challenges such as limited language coverage and reliance on technology.

The proposed system aims to overcome these limitations by implementing a gesture recognition model based on Random Forest, enabling real-time interpretation of ASL gestures into text. This approach enhances communication for the deaf and hard-of-hearing community and can be adapted for other sign languages, promoting inclusivity and global impact.

2.1 EXISTING SYSTEMS

Two notable systems in the sign language technology space are HandTalk and ISL Journey. While both serve important roles in accessibility and education, their focus differs from real-time gesture recognition.

HandTalk: HandTalk is an innovative Brazilian application designed to bridge communication gaps between deaf individuals and the hearing population. Its primary function is translating written text into Brazilian Sign Language (LIBRAS) using animated avatars.

Key Features:

Text-to-Sign Translation: Converts written text into LIBRAS through animated avatars.

Accessibility: Available on multiple platforms, including mobile and web.

Customization: Users can adjust avatar appearance, signing speed, and sign variations.

Educational Role: Promotes awareness of sign language and deaf culture.

Integration: Offers widgets for websites to enable real-time text-to-sign translation.

Pros:

Improves accessibility for deaf users.

Serves as an educational tool for learning sign language.

Innovative use of animated avatars.

Scalable through integration with other platforms.

Cons:

Limited Language Support: Focuses only on LIBRAS, excluding other sign languages.

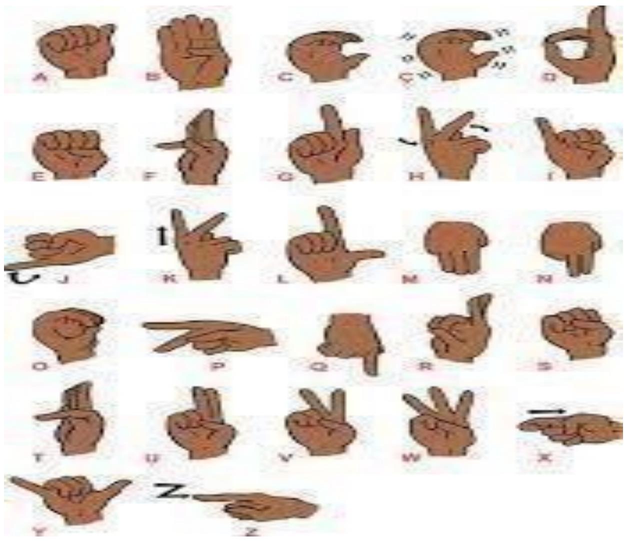
Accuracy Issues: May not fully capture the nuances of sign language.

Technology Dependence: Requires internet and compatible devices.

Interface Complexity: May be challenging for less tech-savvy users.

Relevance to Proposed System

While HandTalk excels in text-to-sign translation, it does not address gesture-to-text recognition, which is the core objective of the proposed system. Our approach focuses on real-time recognition of ASL gestures using machine learning (Random Forest), enabling direct communication without requiring text input first. This makes the system more interactive and practical for everyday use.



ISL Journey

"ISL Journey" is a platform or app designed to facilitate the learning and exploration of Indian Sign Language (ISL). Let's delve into its existing system, including system features, pros, and cons of related systems:

System Features

1 .Comprehensive ISL Curriculum : ISL Journey offers a comprehensive curriculum covering various aspects of Indian Sign Language, including vocabulary, grammar, finger spelling, and conversational skills. The curriculum is designed to cater to users of all proficiency levels, from beginners to advanced learners.

2. Video Lessons and Tutorials : The platform provides video lessons and tutorials taught by experienced ISL instructors. These videos demonstrate sign language gestures and expressions, accompanied by explanations and examples to aid comprehension

3. Interactive Learning Activities : ISL Journey incorporates interactive learning activities, such as quizzes, games, and exercises, to engage users and reinforce learning. These activities may help users practice their signing skills and assess their progress.

4. Progress Track in g and Feed b ack : The platform includes features for users to track their progress and receive feedback on their performance. This may involve progress metrics, achievement badges, or personalized recommendations based on individual learning goals and preferences.

5. Community Engagement : ISL Journey fosters a sense of community among users by providing forums, chat rooms, or social features where learners can connect with each other, share resources, and participate in discussions related to ISL and deaf culture.

Pros of ISL Journey:

1. Accessibility : ISL Journey promotes accessibility by providing a platform for individuals to learn Indian

Sign Language remotely, at their own pace, and from anywhere with an internet connection. This makes ISL education more widely accessible to people of all backgrounds and abilities

2. Cultural Sensitivity : The platform may incorporate cultural sensitivity and authenticity in its content, ensuring that learners receive accurate representations of Indian Sign Language gestures, expressions, and cultural nuances.

3. Flexibility and Convenience : ISL Journey offers flexibility and convenience for learners, allowing them to access learning materials at any time and from any location using their computer, tablet, or smartphone.

4. Skill Development : By providing structured lessons, interactive activities, and opportunities for practice and feedback, ISL Journey helps users develop their signing skills, improve their communication abilities, and build confidence in using Indian Sign Language.

Cons of Existing Related Systems:

1. Limited Availability : Some existing platforms or apps for learning Indian Sign Language may have limited availability in terms of content, language options, or accessibility features, which could restrict access for certain users.

2. Technological Barriers : Access to ISL Journey and other similar platforms may be limited by factors such as device compatibility, internet connectivity, or digital literacy, particularly for users in underserved communities or regions with limited access to technology.

3. Quality and Accuracy : The effectiveness of ISL Journey depends on the quality and accuracy of its instructional content, as well as the expertise of its instructors. Ensuring high standards of quality and authenticity is essential for providing an effective learning experience.

4. Engagement and Retention : Maintaining user engagement and retention over time may be a challenge for ISL Journey and similar platforms. Implementing strategies to keep learners motivated, interested, and actively participating in the learning process is crucial for long-term success.

In summary, ISL Journey offers a valuable resource for individuals interested in learning Indian Sign Language, with its comprehensive curriculum, interactive learning activities, community engagement, and accessibility features. However, it may face challenges related to availability, technological barriers, quality assurance, and user engagement, which are common considerations for online learning platforms.

PROPOSED SYSTEM

Our system captures webcam frames, extracts 21-point landmarks via MediaPipe, flattens and normalizes features, and classifies with a Random Forest. A probability threshold (default 0.8) and temporal smoothing reduce jitter. Recognized tokens are buffered into sentences, which can be cleared or edited. A parallel module maps typed text into sign videos, composing a playlist per word with fallback to letter clips as needed.

Sign Language Recognition: The system will utilize Random Forest(Random forest), a class of machine learning models well-suited for image recognition tasks, to recognize ASL gestures from hand images. Random Forest excels at learning hierarchical features from images, making them ideal for capturing the intricate hand movements and configurations characteristic of sign language gestures.

Training Data: To train the model effectively, a large and diverse dataset of hand images annotated with corresponding ASL gestures will be required. This dataset should cover a wide range of hand shapes, orientations, lighting conditions, and backgrounds to ensure the model's robustness and generalization ability.

Preprocessing: Prior to feeding hand images into the model, preprocessing techniques such as normalization, resizing, and noise reduction may be applied to enhance the quality and consistency of the input data. Preprocessing helps mitigate variability in hand images caused by factors like camera angle and lighting conditions.

Random Forest Architecture: Random Forest is an ensemble of decision trees. Each tree splits on random subsets of features, and predictions are aggregated by majority vote. In our system,MediaPipe extracts 21-point hand landmarks as input features. This approach ensures fast, accurate classification with confidence thresholds and smoothing for real-time sign recognition.

Training and Optimization: The Random Forest model will be trained using a supervised learning approach, where it learns to map input hand landmark features to corresponding ASL gestures. During training, optimization techniques such as feature selection and tuning the number of trees and max depth will be employed to adjust the model's parameters and minimize prediction errors.

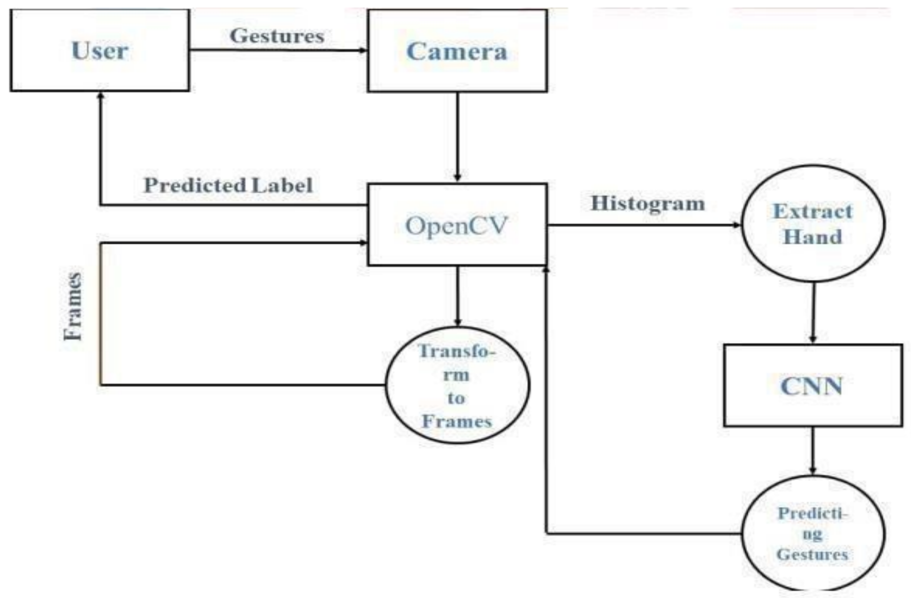
Evaluation and Testing: Once trained, the Random Forest model will be evaluated using a separate test dataset to assess its performance in recognizing ASL gestures accurately and reliably. Performance metrics such as accuracy, precision, recall, and F1 score will be used to quantify the model's effectiveness.

Real-time Inference: The trained Random Forest model will be deployed in a real-time inference environment, where it can process hand landmark data captured by a camera in real-time and predict the corresponding ASL gestures. The system should be optimized for low latency and high throughput to enable seamless communication for users.

Translation to Text: Finally, recognized ASL gestures will be translated into text format, enabling communication for individuals who may not be fluent in sign language. This textbased output can be displayed on a screen or transmitted via text-based communication channels, enhancing accessibility and inclusivity for the deaf and hard of hearing community.

CONCEPTUAL DESIGN

The conceptual design of a sign language system involves the creation of a framework that enables the recognition, interpretation, and translation of sign language gestures into a form that can be understood by both deaf and hearing individuals. At its core, the system must capture the intricate movements and configurations of sign language gestures accurately and efficiently. This typically involves the use of sensors or cameras to capture hand movements and spatial positioning. Machine learning algorithms, such as Random Forest, may be employed to analyze and interpret these gestures, mapping them to corresponding linguistic symbols or text representations. Additionally, the system may incorporate natural language processing (NLP) techniques to enhance understanding and facilitate communication between users. The conceptual design should prioritize accessibility, inclusivity, and user-friendliness, ensuring that both deaf and hearing individuals can effectively communicate and interact using the system. Additionally, considerations for real-time processing, scalability, and adaptability to different sign languages and user environments are crucial for the successful implementation of the system.



SYSTEM ARCHITECTURE:

The system architecture of the sign language recognition and translation system is designed to efficiently capture, process, interpret, and present sign language gestures for real-time communication. Key components include:

Input Module: Captures live video frames from a webcam.

Preprocessing Module: Uses MediaPipe to extract 21-point hand landmarks and normalize coordinates for scale and position invariance.

Feature Extraction Module: Converts landmark data into structured feature vectors suitable for machine learning.

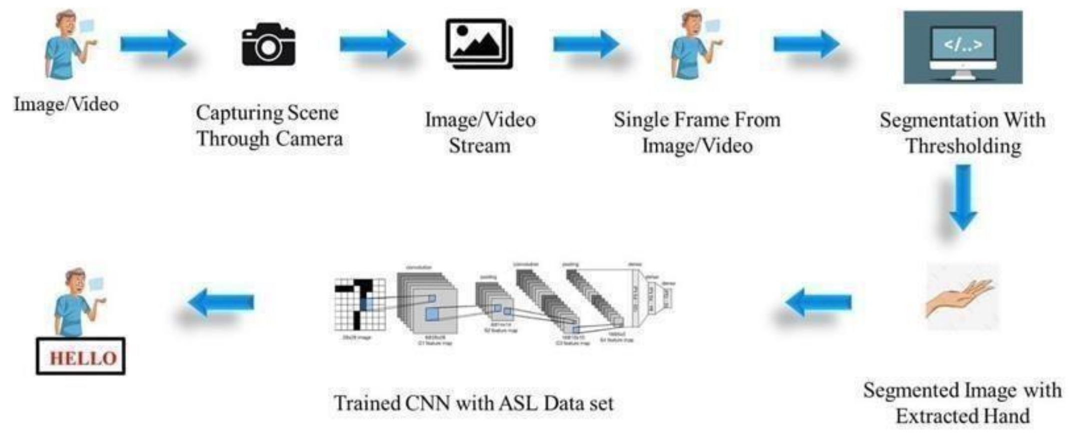
Machine Learning Model: A Random Forest classifier interprets these features by aggregating predictions from multiple decision trees, ensuring fast and accurate classification without heavy GPU requirements.

Translation Module: Maps recognized tokens into text and, for reverse communication, converts typed text into sign language videos using a curated video library.

Output Module: Displays recognized words and sentences in a Flask-based web interface and streams sign videos for text-to-sign translation.

Feedback Mechanism: Collects user input and logs performance for iterative improvements and vocabulary expansion.

This architecture prioritizes low latency, offline capability, and adaptability, making it suitable for classrooms and accessibility-focused environments.


Image/Video
Capturing Scene Through Camera
Image/Video Stream
Single Frame From Image/Video
Segmentation With Thresholding
Segmented Image with Extracted Hand
Trained CNN with ASL Data set
HELLO

Components Descriptions:

The system includes:

Input Layer: Capturing Hand Landmarks

Image/Video Input: The system begins by capturing live video from a webcam.

Frame Acquisition: OpenCV streams frames in real time for processing.

Preprocessing Layer: Landmark Extraction and Normalization

MediaPipe Hand Tracking: Detects and extracts 21 key hand landmarks (x, y, z coordinates).

Normalization: Coordinates are normalized relative to the wrist and scaled for size invariance.

Feature Vector Creation: Converts landmarks into a structured numeric array for classification.

Random Forest Model: Classification

Trained Ensemble: A RandomForestClassifier trained on ASL letters and selected words.

Voting Mechanism: Aggregates predictions from multiple decision trees; applies a confidence threshold (≥ 0.8) and temporal smoothing to stabilize output.

Output Layer: Display and Translation

Text Output: Recognized letters or words are displayed in the Flask web interface and buffered into sentences.

Text-to-Sign Module: Converts typed text into sign language videos by mapping words to pre-recorded MP4 clips, with fallback to letter spelling.

2.6 Proposed System (Software Features)

The proposed ASL recognition system is designed for real-time, offline, and accessible communicationusing lightweight machine learning. Below are the key software features:

1. Sign Recognition Technology

Functionality: Captures webcam input, extracts 21-point hand landmarks using MediaPipe, and classifies gestures with a Random Forest model for static letters and words.

Example: A user signs "HELLO," and the system instantly displays the recognized word in the web interface.

2. Real-Time Sign Language Translation

Functionality: Converts ASL gestures into text in real time via a Flask-based web interface, with sentence buffering for coherent outputs.

Example: A user signs "MY NAME IS," and the system displays the full sentence on the /video page.

3. Offline Mode

Functionality: Operates fully offline using pre-trained models (model.p for Random Forest) stored locally, ensuring accessibility in low-connectivity environments.
Example: Users in rural areas can run the system on a standard PC with a webcam without internet access.

2.7 Development Tools and Environment

Programming Language: Python 3.8+ for flexibility and strong ML ecosystem.

Libraries:

OpenCV: Video capture and frame processing.

MediaPipe: Hand landmark detection and feature extraction.

scikit-learn: Random Forest model training and inference.

Flask: Web interface for real-time streaming and text display.

NumPy & Pickle: Data handling and model serialization.

Hardware: Standard PC with webcam (USB or integrated).

2.8 Benefits of Implementation

- Enhanced Communication: Real-time ASL-to-text translation reduces reliance on interpreters in classrooms and service points.
- Empowerment: Enables Deaf users to communicate independently and hearing users to respond effectively.
- Inclusivity: Bridges communication gaps in education, healthcare, and public services.
- Scalability: Architecture supports retraining for other sign languages (e.g., GSL, ISL) and vocabulary expansion

CHAPTER THREE

3.0 METHODOLOGY

Our goal was to develop a high-performance, real-time sign language recognition system that translates American Sign Language (ASL) gestures into text using a Random Forest classifier. The system leverages MediaPipe for extracting 21-point hand landmarks and Python for implementation.

We began by gathering requirements through literature reviews, interviews, and surveys with stakeholders, including Deaf individuals and educators, ensuring the solution addressed real-world accessibility needs.

Next, we focused on data collection, capturing ASL alphabet gestures and selected words. Each frame was processed to extract normalized landmark coordinates, forming structured feature vectors. These were used to train a Random Forest model, chosen for its speed, robustness, and ability to run on standard hardware without GPUs.

The system was integrated into a Flask-based web interface for real-time recognition and text-to-sign video playback. Performance optimization prioritized low latency (≥15 FPS) and accuracy (≥80%). Testing involved both automated validation and user feedback sessions, leading to iterative refinements.

3.1 CHAPTER OVERVIEW

This project implements a landmark-driven machine learning pipeline for ASL recognition. The methodology covers:

Requirement Specification

Ethics & Privacy Considerations: The system ensures that all data collected from users is handled with strict confidentiality. Consent is obtained prior to data capture, and no personally identifiable information is stored. All processing is done locally to protect user privacy.

Stakeholder Analysis

- Requirement Gathering Process
- Functional and Non-Functional Requirements

- System Design and Model Selection
- Testing and Evaluation

3.2 REQUIREMENT SPECIFICATION

Functional Requirements:

Capture live video via webcam.

Extract hand landmarks using MediaPipe.

Classify gestures with a Random Forest model.

Display recognized text in real time with sentence buffering.

Provide text-to-sign video playback for typed input.

Non-Functional Requirements:

Performance: ≥15 FPS on mid-range PCs.

Accuracy: ≥80% for words, ≥90% for letters.

Usability: Simple, accessible UI with minimal latency.

Reliability: Offline operation, fault tolerance, and easy updates.

3.3 REQUIREMENT GATHERING PROCESS

Literature Review: Studied existing ASL recognition systems and accessibility standards.

Stakeholder Interviews: Engaged Deaf users, teachers, and interpreters to identify core needs.

Surveys & Observations: Collected feedback on vocabulary priorities and UI design.

Workshops: Validated design choices and tested early prototypes.

STAKEHOLDERS OF THE SYSTEM

The stakeholders in the American Sign Language (ASL) recognition system project include the following:

Primary Users:

Deaf and Hard-of-Hearing Individuals: They are the direct users of the system, benefiting from improved communication through real-time ASL recognition and translation into text.

Secondary Users:

Educators: Teachers and instructors who work with Deaf students can use the system as an educational tool for teaching sign language.

Interpreters: Sign language interpreters may use the system to enhance their ability to translate between ASL and spoken languages, especially in environments where real-time translation is needed.

Family Members and Friends: People close to Deaf or hard-of-hearing individuals may use the system to communicate more easily and effectively.

Healthcare Providers: Medical professionals who need to communicate with Deaf patients may use the system to overcome communication barriers in medical settings.

Researchers: Academics and researchers in fields like linguistics, computer science, and accessibility may use the system to explore ASL recognition and communication technologies.

Technical Experts:

Software Developers and Engineers: The technical team responsible for designing, developing, and maintaining the ASL recognition system.

Data Scientists and Machine Learning Specialists: Experts involved in training and improving the Random Forest model for accurate gesture recognition.

UI/UX Designers: Professionals focused on creating an accessible and user-friendly interface for the system.

Institutions and Organizations:

Educational Institutions: Schools and universities may implement the system to support Deaf students and teachers in classrooms.

Deaf Advocacy Groups: Organizations focused on advocating for Deaf rights may support the use of the system to improve accessibility.

Businesses and Employers: Companies that employ Deaf or hard-of-hearing individuals may use the system to facilitate communication in the workplace.

These stakeholders are key to ensuring that the system effectively meets the needs of its users and remains inclusive, reliable, and user-friendly.

REQUIREMENT GATHERING PROCESS

The requirement gathering process for the ASL recognition system is a comprehensive and multifaceted approach designed to ensure the final product meets the specific needs of its users. It begins with identifying all relevant stakeholders, which include Deaf and hard-ofhearing individuals who will use the system directly, as well as secondary users like educators, interpreters, family members, and technical experts involved in development.

The process involves engaging with these stakeholders through various methods. Stakeholder interviews provide detailed insights into their communication needs, user experience preferences, desired accuracy levels for gesture recognition, and privacy concerns. Surveys and questionnaires are distributed to a broader audience to gather quantitative data, validating the findings from interviews and identifying common requirements.

Workshops and focus groups are organized to facilitate in-depth discussions and collaborative brainstorming. These sessions offer a deeper understanding of user behaviors, pain points, and potential solutions, often involving interactive activities like prototyping and usability testing.

Analyzing existing ASL recognition systems and related technologies helps identify market gaps and opportunities for innovation, ensuring the new system offers improvements over current options.

Once the user requirements are collected, they are translated into technical specifications in collaboration with technical experts. This step involves defining the system architecture, hardware and software requirements, and machine learning models for gesture recognition. Requirements are then prioritized based on their importance and feasibility, ensuring the most critical features are addressed first and guiding the development roadmap.

All gathered information is compiled into a comprehensive requirements document, including functional and non-functional requirements, user stories, use cases, and acceptance criteria. This document is reviewed and validated with stakeholders to ensure accuracy and completeness.

Validation sessions confirm that the requirements align with stakeholder needs and expectations, and formal approval from key stakeholders is obtained before proceeding to development.

By engaging stakeholders through diverse methods and thoroughly documenting their requirements, the project ensures that the ASL recognition system is user-centered, addressing essential aspects like accuracy, real-time translation, ease of use, and data privacy. This meticulous requirement gathering process lays a strong foundation for developing a highperformance, reliable, and inclusive ASL recognition system that significantly enhances communication and accessibility for the Deaf and hardofhearing community.

STEPS IN THE REQUIREMENT GATHERING PROCESS:

1. Literature Review :

Conduct a comprehensive review of existing literature on sign language recognition, focusing on previous studies, methodologies, and technologies used.

2. Interviews :

Conduct interviews with stakeholders, including members of the deaf community, healthcare providers, and educators, to understand their needs and expectations from the system.

3. Surveys and Questionnaires :

Distribute surveys and questionnaires to gather quantitative data on the requirements from a larger group of stakeholders.

4. Observation :

Observe the communication challenges faced by the deaf community in real-world settings to identify specific needs that the system should address.

5. Workshops and Focus Groups :

Organize workshops and focus groups to facilitate in-depth discussions with stakeholders, allowing for a deeper understanding of their needs and potential solutions.

IMPLEMENTATION PLAN

Outline the steps to implement the system based on the gathered requirements.

1. Data Collection :

Collect a diverse and extensive dataset of ASL alphabet gestures.

2. Model Development :

Develop a Random Forest-based model tailored for recognizing ASL gestures.

3. Interface Design :

Create a user-friendly interface for real-time gesture recognition and translation.

4. Testing and Validation :

Test the system with actual users to validate its performance and accuracy.

5 . **Deployment :**

Deploy the system in real-world settings and monitor its usage and feedback for further improvement.

FUNCTIONAL REQUIREMENT UML

Gesture Recognition :The system must accurately recognize and interpret ASL alphabet gestures.

Gesture Detection:

- ❖ Real-Time Detection: The system should be capable of detecting gestures in real-time using a camera.
- ❖ Multiple Angles: The system should recognize gestures from various angles and orientations.
- ❖ Hand Shape and Movement: It must accurately detect the shape of the hand and the specific movements associated with each ASL letter.

Feature Extraction:

- ❖ Key Points Detection: The system should detect key points on the hand, such as fingertips and joints, to differentiate between different gestures.
- ❖ Pattern Recognition: It must analyze patterns in the hand's position and movement to classify each gesture correctly.

CLASSIFICATION:

Model Training: Utilize a well-trained Random Forest model to classify gestures accurately. The model should be trained on a large and diverse dataset of ASL gestures.

Error Minimization: Implement techniques to minimize classification errors, such as using advanced algorithms or increasing the dataset's size and diversity.

Real -Time Translation : *The system should provide real-time translation of gestures into text.*

Processing Speed:

- ❖ High FPS (Frames Per Second): Ensure the system processes video input at a high frame rate to detect and translate gestures swiftly.

❖ Efficient Algorithms: Use efficient algorithms and optimizations to reduce processing time.

Immediate Feedback:

❖ Instant Display: The recognized gesture should be translated into text and displayed immediately.

❖ Continuous Monitoring: The system should continuously monitor the video feed to update translations in real-time as gestures change.

User Interface: The system must have a user-friendly interface for easy interaction.

Design:

❖ Clear Layout: The interface should have a clear and simple layout, displaying the video feed, recognized gesture, and corresponding text.

❖ Accessibility: Design the interface to be accessible for users with varying levels of technical expertise and physical abilities.

Interaction:

❖ User Controls: Provide controls for users to start, pause, and stop the gesture recognition process.

❖ Feedback Mechanism: Implement a feedback mechanism for users to report recognition errors or suggest improvements.

SYSTEM REQUIREMENT

FUNCTIONAL SYSTEM REQUIRMENTS:

❖ The user shall input a URL (https://127.0.0.1).

- The system shall validate the URL.
- The system shall display a homepage with a camera component available.
- The user shall make a known sign language in front of the camera.
- The system shall capture the sign language made by the user.

- The system shall interpret the sign language.
- The system shall display on the screen words of interpretation as response to the interpretation.

DIAGRAMS

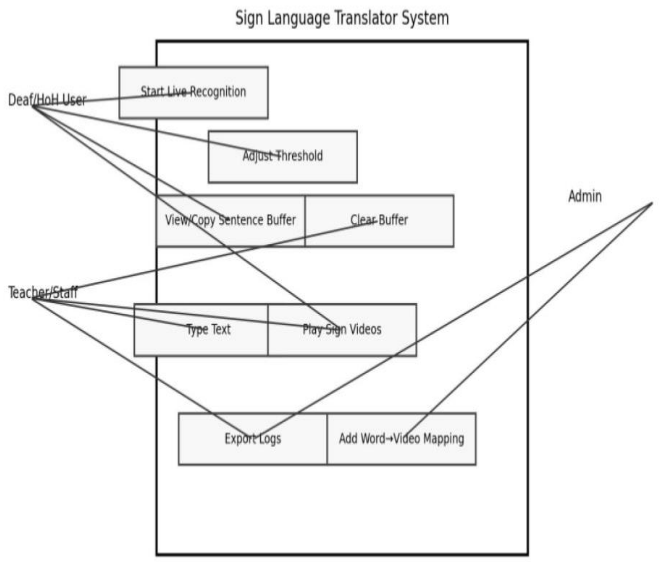
USE CASE

Use case between explanation and analysis of requirements to represent system performance. Use case description of the function of the system which gives visible results for the actor. Identifying actors and their problem cases by lecturing on the boundaries of the system, representing the work done by them and the whole environment. Actors are outside the system, while cases are within the system. The use case describes the system as seen from the example of the actor's behavior. It describes the work provided by the system as a set of events that provide visible results for the actor.

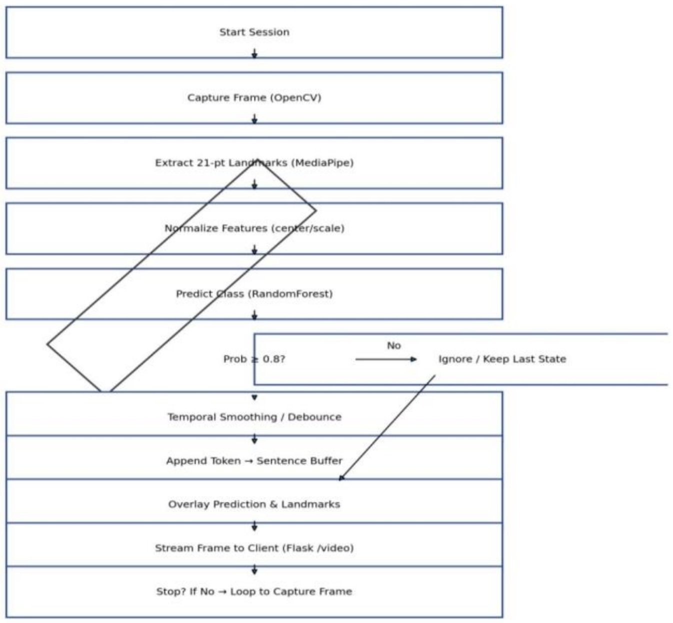
In brief, the purposes of use case diagrams can

be said to be as follows – Used to gather the requirements of a system.

- Used to get an outside view of a system.
- identify the external and internal factors influencing the system. Show the interaction among the requirements are actors.



ACTIVITY DIAGRAM


Start Session
Capture Frame (OpenCV)
Extract 21-pt Landmarks (MediaPipe)
Normalize Features (center/scale)
Predict Class (RandomForest)
Prob ≥ 0.8?
No
Ignore / Keep Last State
Temporal Smoothing / Debounce
Append Token → Sentence Buffer
Overlay Prediction & Landmarks
Stream Frame to Client (Flask /video)
Stop? If No → Loop to Capture Frame

SEQUENCE DIAGRAM

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension(time) and horizontal dimension (different objects).

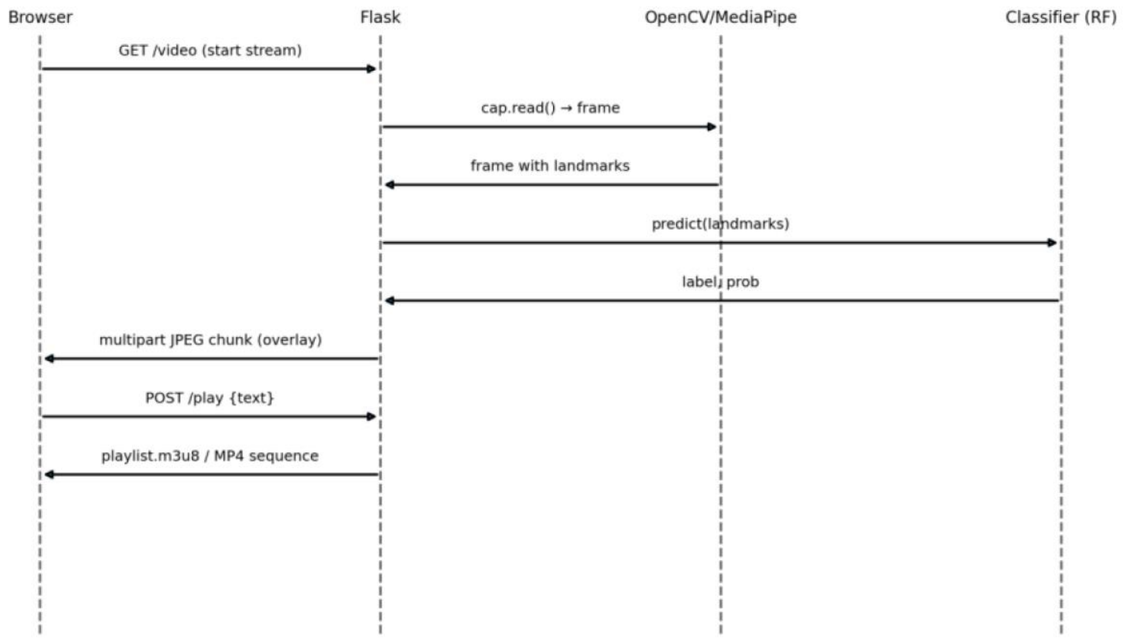
Objects: - The commodity can be viewed as an entity with a specific value at a specific time and as the holder of the identity.

The sequence diagram shows the interaction of objects presented chronologically. It refers to the order of messages to be exchanged between the objects included in the view and the objects needed to complete the class and visual functionality.Sequence diagrams are commonly used the system under development k is obtained in the case of logical approach.Events like events are events or events that happen.

Sequential diagrams show the sequence in which messages are exchanged between parallel vertical lines (lifelines), different processes or objects that exist as a horizontal arrow.It allows the specification of simple runtime scenarios in a graphical way. If the life line belongs to an object, it indicates a role. The name of the instance is Informal and Unmanual Ka Transplant.

Messages typed with horizontal arrows display interactions, with the message name above. Solid arrow tops represent synchronous calls, open arrows represent asynchronous messages, and dashed lines represent reply messages. If the caller sends a synchronous message, it is necessary to wait for the message to complete, such as giving a subroutine command. If the caller sends an asynchronous message, it can continue processing and not have to wait for a response. Asynchronous calls are found in multithreaded applications, event-driven applications, and message-oriented middleware.

Activation boxes or method- call boxes are opaque rectangles drawn on the lifeline to indicate that a message response (execution statement in UML) is being processed.



Class diagram

Create class structure and content elements using class drawing class, package and object marked design. The class describes the approach to the figure when constructing the method— idea, result, and outcome. Classes are made up of three things: name, properties, and operations. Class diagrams also display relationships such as inheritance, cohabitation, and so on. Relation is the most common relation in a class diagram. Association refers to the relationship between instances of classes.

How to Draw a Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram –

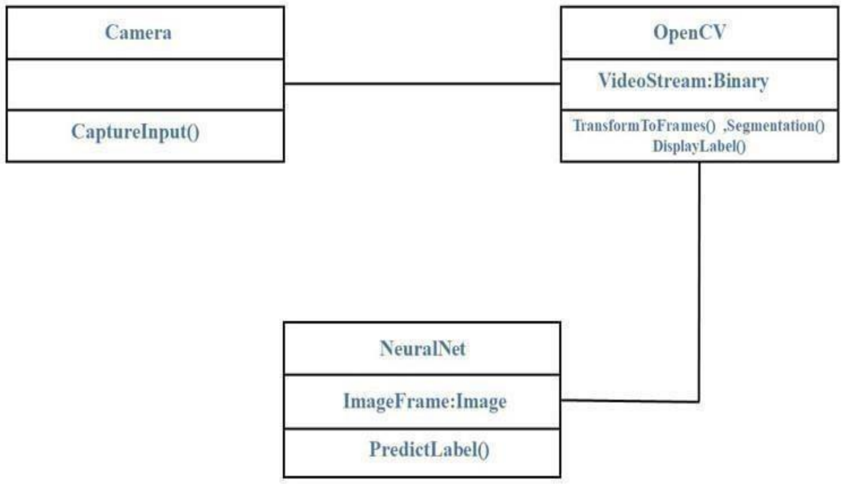
The name of the class diagram should be meaningful to describe the aspect of the system.

Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified.

For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.

Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.

Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.



NON FUNCTIONAL REQUIREMENT

Performance: The system should process gestures quickly to provide real-time feedback.Performance in this context refers to the system's ability to process and respond to user inputs (gestures) without noticeable delay.

Key Aspects:

- ❖ Response Time: The system should recognize and respond to gestures within milliseconds to ensure the interaction feels immediate and fluid.
- ❖ Throughput: The system should be able to handle a high number of gesture inputs per second without performance degradation.
- ❖ Resource Utilization: Efficient use of CPU, GPU, and memory resources to maintain high performance even under heavy load. o Latency:
- ❖ Minimizing any delay from when a gesture is performed to when the system reacts

Accuracy: The system must have a high level of accuracy in gesture recognition. Accuracy refers to the system's ability to correctly identify and interpret user gestures.

Key Aspects:

- ❖ Recognition Rate: The percentage of correctly identified gestures out of the total number of gestures performed.
- ❖ Error Rate: The frequency of false positives (incorrectly identified gestures) and false negatives (missed gestures).
- ❖ Precision and Recall: Precision is the ratio of correctly predicted positive observations to the total predicted positives, and recall is the ratio of correctly predicted positive observations to all observations in actual class.
- ❖ Robustness: The system's ability to maintain high accuracy across different users, environments, and lighting conditions.

Usability: The interface should be intuitive and accessible to users, including those with disabilities. Usability refers to the ease with which users can learn to use the system and achieve their goals efficiently and effectively.

Key Aspects:

- ❖ Learnability: How easy it is for new users to learn how to use the system.
- ❖ Efficiency: How quickly users can perform tasks once they have learned the system. o Memorability: How easily users can remember how to use the system after not using it for a while.
- ❖ Accessibility: Ensuring that the system is usable by people with a wide range of abilities, including those with visual, auditory, motor, or cognitive impairments. o User Interface Design: Creating a clean, intuitive, and responsive design that caters to all user demographics.

Reliability: The system should function consistently under various conditions. Reliability refers to the system's ability to operate without failure over a specified period under specified conditions.

Key Aspects:

- ❖ Availability: The system should be operational and accessible when needed.
- ❖ Fault Tolerance: The ability of the system to continue operating properly in the event of the failure of some of its components.
- ❖ Recovery: The system's ability to recover quickly from errors or failures and resume normal operations.
- ❖ Consistency: The system should perform predictably and consistently, providing the same outputs for the same inputs under the same conditions. o Maintenance: The system should be easy to maintain and update, ensuring long-term reliability without significant downtime.

SECURITY

Python is being used to develop this system. With NumPy, aperture objects can be created up to 50 times faster than with conventional Python lists.

Unlike lists, these arrays are quick because they are kept in a continuous memory area. The designer has the system modules.

PROJECTS METHOD

In developing the sign language recognition system, selecting the appropriate project management method is crucial. Two primary methodologies considered are agile and plan driven (traditional).

Agile Methods

Agile methods emphasize flexibility, iterative development, and continuous stakeholder collaboration. These methods are particularly suited for projects where requirements may evolve over time and where regular feedback is essential. The advantages for the sign language recognition project include:

- ❖ Flexibility: Agile allows for adjustments to project plans and priorities based on ongoing feedback and changing requirements.
- ❖ Customer Collaboration: Frequent interactions with stakeholders, including the Deaf and hardofhearing community, ensure that the system meets their needs.
- ❖ Incremental Delivery: Working software is delivered in small, usable increments, allowing for continuous user feedback and early detection of issues.

Common Agile Frameworks:

- ❖ Scrum: Involves working in sprints (typically 2-4 weeks) with defined roles (Scrum

Master, Product Owner, Development Team) and regular ceremonies (Daily Stand-up, Sprint Review, Sprint Retrospective).

- ❖ Kanban: Focuses on visualizing the workflow and managing work in progress, allowing for continuous delivery without fixed-length iterations.

Plan-Driven Methods: Plan-driven methods, such as the Waterfall model, follow a linear and sequential approach. Each phase (requirements, design, implementation, testing, deployment) is completed before the next begins. This method might be suitable if:

Well-Defined Requirements: The requirements are clear and unlikely to change significantly.

Predictability: The project benefits from a predictable schedule and budget.

However, for a sign language recognition project, agile methods might be more appropriate due to their flexibility and responsiveness to change.

Software Process Models

Various software process models can guide the structure, planning, and control of the software development process. Below are some models considered for the sign language recognition project:

1. Waterfall Model

Description: A linear and sequential approach where each phase must be completed before the next begins.

Advantages: Simple, easy to understand, and manage with clear milestones.

Disadvantages: Inflexible to changes and does not accommodate iterative refinement well.

2. V-Model (Verification and Validation)

Description: An extension of the Waterfall model that emphasizes verification and validation at each development stage.

Advantages: Enhances focus on testing and quality assurance.

Disadvantages: Similar to Waterfall, it is inflexible to changes during the development process.

3. Incremental Model

Description: Development is broken down into smaller increments, with each increment delivering part of the functionality.

Advantages: Allows partial implementation and provides feedback for each increment, useful for machine learning projects.

Disadvantages: Requires good planning and design to ensure integration of increments.

4. Spiral Model

Description: Combines iterative development with systematic aspects of the Waterfall model, focusing on risk assessment and minimization through iterations.

Advantages: Provides flexibility and incorporates risk management, beneficial for complex projects like gesture recognition.

Disadvantages: Can be complex to manage and requires expertise in risk assessment.

5. Agile Models (Scrum, Kanban)

Description: Emphasize iterative development, customer collaboration, and flexibility. Scrum focuses on sprints, while Kanban focuses on continuous flow.

Advantages: Highly flexible, encourages continuous improvement, adapts well to changing requirements.

Disadvantages: Requires active stakeholder involvement and can be challenging to scale for larger projects.

6. RAD Model (Rapid Application Development)

Description: Focuses on quick development and iteration with user feedback, using prototyping to refine requirements and solutions.

Advantages: Fast delivery, user involvement, and feedback throughout the development process.

Disadvantages: Requires sufficient user involvement, may compromise on quality for speed.

Recommended Approach for the Sign Language Recognition Project

Given the nature of the sign language recognition project, which involves iterative development of machine learning models, real-time user feedback, and continuous improvement, an Agile approach (e.g., Scrum or Kanban) is recommended. This approach allows for frequent reassessment and adaptation based on user feedback, which is crucial for developing an intuitive and accurate gesture recognition system.

Steps for Agile Implementation

1. Sprint Planning: Define the scope and goals for each sprint, prioritizing features like gesture recognition accuracy and real-time feedback.

2. Daily Stand-ups: Hold daily meetings to discuss progress, challenges, and next steps.

3. Sprint Review and Retrospective: At the end of each sprint, review the work completed, gather feedback from stakeholders, and discuss what went well and what can be improved.

4. Incremental Deliveries: Deliver functional increments of the system regularly, allowing for user testing and feedback.

5. Continuous Integration and Testing: Implement continuous integration and automated testing to ensure the system functions correctly as new features are added.

Waterfall Model Phases for Sign Language App

1. Requirements Gathering and Analysis:

Objective: Clearly define the goals and requirements for the sign language app.

Activities:

- ❖ Understand the needs of the deaf and hard of hearing community, including the requirement for recognizing hand gestures and converting them into text.
- ❖ Identify and document the key functionalities such as real-time gesture recognition, user- friendly interface, and educational resources for ASL.
- ❖ Gather datasets of ASL gestures for model training, as mentioned in the project objectives.

2. System Design:

Objective: Design the system architecture and detailed specifications.

Activities:

Design the architecture of the gesture recognition system using Random Forest.

Plan the user interface layout ensuring it is intuitive and accessible across various devices (desktops, tablets, smartphones).

Develop data preprocessing workflows for handling the ASL gesture datasets.

3. Implementation:

Objective: Develop the actual software components based on the design specifications.

Activities:

Implement the Random Forest-based model for recognizing ASL gestures, including layers for feature extraction and classification.

Develop the real-time gesture recognition interface, integrating it with the Random Forest model for seamless translation of gestures to text.

Code the preprocessing and data augmentation steps for the ASL gesture datasets.

4. Integration and Testing

Objective: Integrate the components and test the entire system to ensure it meets the requirements.

Activities:

Perform unit testing on individual components like the Random Forest model and the user interface.

Conduct system integration testing to ensure all parts of the application work together smoothly.

Implement usability testing with potential users from the DHH (Deaf and Hard-of-Hearing) community to gather feedback and improve the interface.

5. Deployment

6. Objective: Deploy the system in a live environment for users to access.

Activities:

Set up the application on servers and ensure it is accessible via various platforms (web, mobile apps).

Provide initial training and documentation to users on how to use the app.

Monitor the system for any issues and provide technical support.

7. Maintenance

Objective: Ensure the system remains functional and up-to-date post- deployment.

Activities:

Regularly update the software to fix bugs and improve performance.

Expand the dataset and retrain the model to improve accuracy and robustness.

Gather user feedback continuously and implement new features or improvements based on this feedback.

3.1.0 CHOSEN MODEL: Random Forest Justification

- **Effective for Landmark-Based Recognition:** Random Forest is highly effective for structured feature classification, making it ideal for interpreting MediaPipe's 21-point hand landmarks for ASL recognition.
- **No Need for Heavy Feature Engineering:** Unlike CNNs, Random Forest works directly on normalized landmark coordinates, reducing complexity and eliminating the need for convolutional layers.
- **Robustness and Generalization:** The ensemble of decision trees handles variations in hand size, orientation, and lighting, ensuring consistent performance across diverse users and environments.

- **High Accuracy with Small Datasets:** Random Forest performs well even with limited data, making it suitable for projects without massive image datasets.
- **Low Latency and Offline Capability:** Inference is fast on standard CPUs, enabling real-time recognition without requiring GPUs or cloud resources.
- **Scalability and Interpretability:** The model can be retrained easily for new vocabularies and provides feature importance metrics for analysis.
- **Wide Adoption and Support:** Random Forest is widely supported in libraries like scikit-learn, ensuring stability, documentation, and ease of integration.

PROJECT DESIGN CONSIDERATIONS

When designing the American Sign Language (ASL) recognition system, several critical factors must be considered to ensure the project's success. These considerations span technical, user experience, and system performance aspects to create a robust, efficient, and user-friendly solution.

1. Data Collection and Quality

Diverse Dataset: Ensure that the dataset includes a wide range of hand gestures, covering various angles, lighting conditions, and hand sizes. This diversity is crucial for training a model that can generalize well to different users and environments.

Data Augmentation: Implement techniques such as rotation, scaling, and brightness adjustment to artificially expand the dataset and improve the model's robustness.

2. Model Architecture

Choice of Random Forest Architecture:
Select an appropriate Random Forest configuration that balances accuracy and speed for real-time ASL recognition. The model uses multiple decision trees trained on MediaPipe landmark features, ensuring robustness without requiring heavy computation.

Parameter Tuning:
Carefully tune hyperparameters such as:

Number of Trees (n_estimators): Typically 100–200 for stability.

Max Depth: Controls tree complexity to prevent overfitting.

Max Features: Limits features considered at each split for diversity.

Minimum Samples per Split: Improves generalization.

3. Real-Time Processing

Low Latency: The system must process video input and provide text output in real-time. This requires efficient algorithms and potentially hardware acceleration (e.g., using GPUs) to minimize latency.

High Frame Rate: Ensure that the system can handle a high frame rate to accurately capture the nuances of hand movements during gestures.

4. User Interface (UI) Design

Simplicity and Usability: The UI should be intuitive and easy to navigate, even for users with limited technical expertise. Controls should be simple, with options to start, pause, and stop recognition, as well as clear visual feedback on the recognized gestures.

Accessibility: Design the UI to be accessible to a wide range of users, including those with visual impairments or other disabilities. Consider features like high contrast mode and screen reader compatibility

CHAPTER FOUR

IMPLEMENTATION AND RESULTS

Implementation and Results provides a detailed account of how the system was developed and the outcomes achieved. This chapter is structured to guide readers through the process of mapping the logical design onto the physical platform, covering the following key sections:

CHAPTER OVEVIEW

1. Mapping Logical Design onto Physical Platform:

This involves the translation of the system's logical design into a functioning physical model.

It includes algorithms for implementing the user interface (UI) and the database.

Flowchart diagrams illustrate the process flow for both UI implementation and database development.

2. Construction:

-Snippet Code of the System Logic: This section presents portions of the actual code used in developing the system, highlighting key logic components.

-Screenshots of the System: Visual representations of the system in operation are provided to offer a clearer understanding of the implemented features.

3. Data Collection:

Describes the procedures and code used to gather the necessary data for the system.

Includes methods for capturing images using a webcam and storing them for further processing.

4. Data Creation:

Details the steps involved in processing the collected data to create usable datasets.

Shows how to use MediaPipe and other tools to extract meaningful features from images.

5. Testing Plan:

Components Testing: Algorithms for testing both the UI and database components are discussed.

System Testing:

Verification Testing: Describes the process of ensuring the system meets its specified requirements.

Validation Testing: Ensures the system fulfills its intended purpose through various

testing methods.

6. Results:

This section summarizes the outcomes of the implementation, including the performance and accuracy of the system.

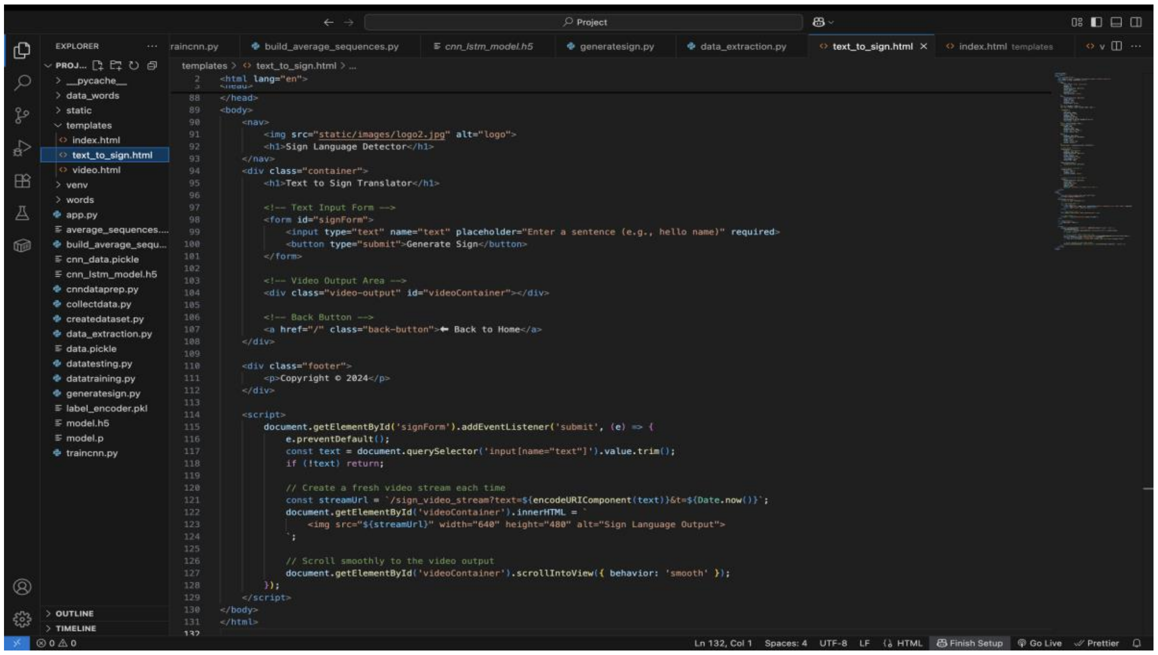
It includes statistical data, accuracy scores, and any other relevant results that demonstrate the system's effectiveness.

MAPPING LOGICAL DESIGN ONTO PHYSICAL PLATFORM

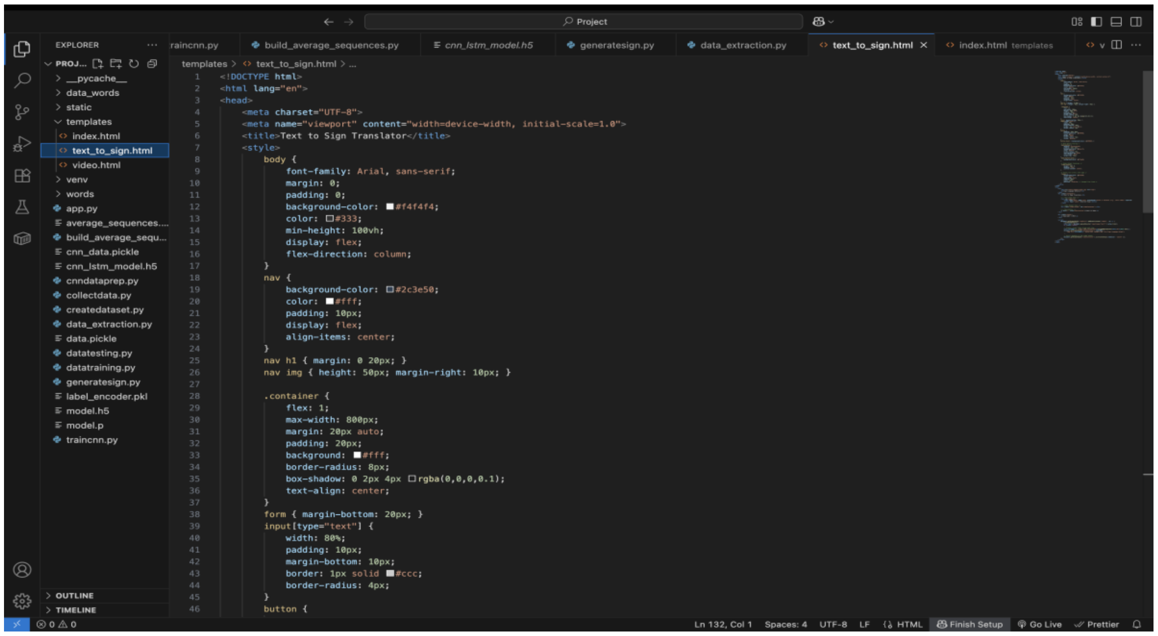
Translation of Logical Design to Physical Model: This step involves converting the abstract, logical design of the system into a tangible, working physical model. The logical design includes conceptual representations of the system's functionalities and relationships, which need to be effectively mapped onto the physical hardware and software components.

Algorithms for Implementing User Interface (UI) and Database: The algorithms for the UI focus on creating an intuitive and responsive interface that allows users to interact with the system seamlessly. This includes designing elements such as buttons, menus, forms, and other interactive components. Database algorithms ensure efficient data storage, retrieval, and management. They cover aspects such as data normalization, indexing, query optimization, and transaction management to maintain data integrity and performance.

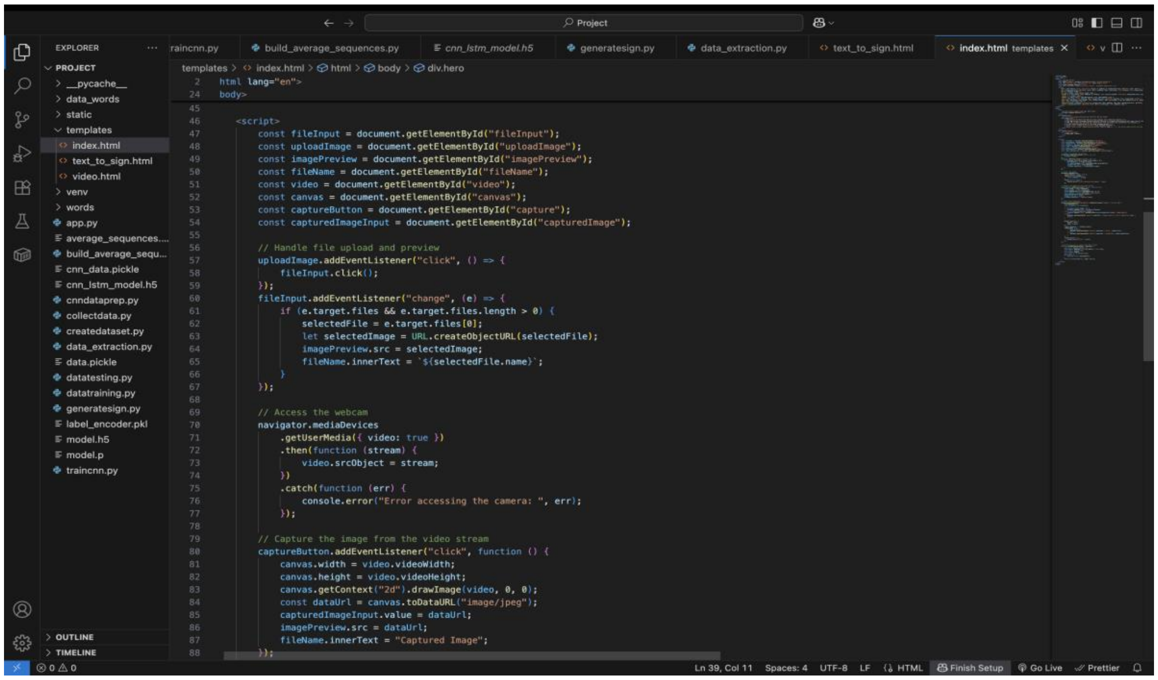
INDEX.HTML



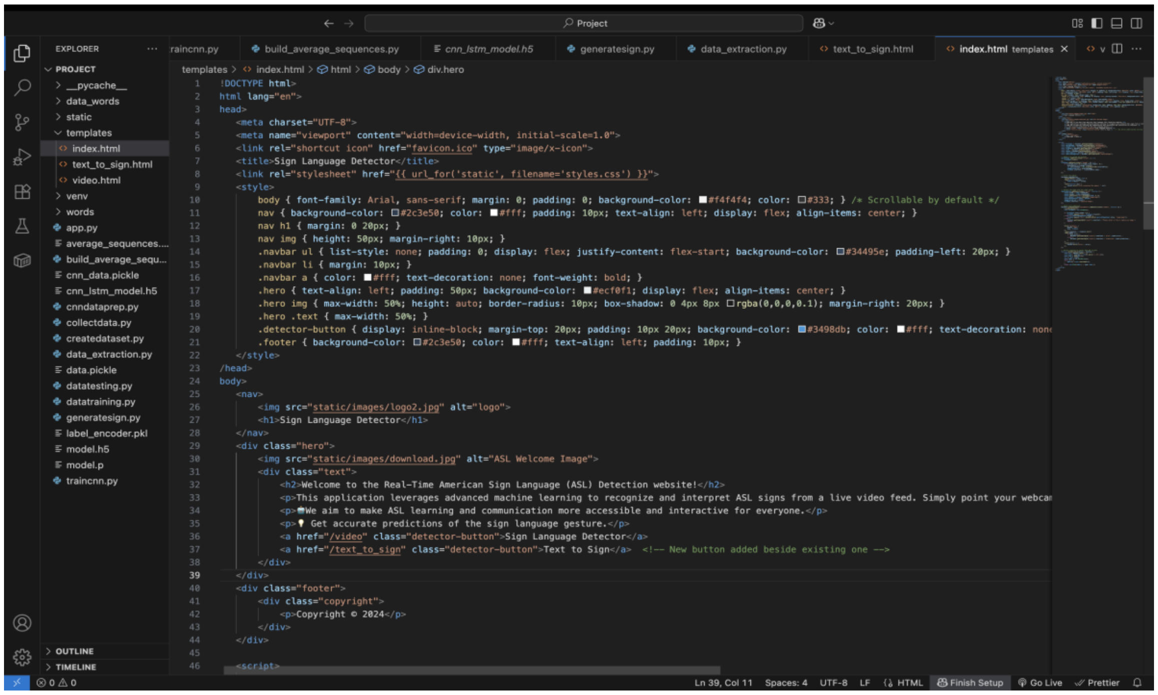
```html
<html lang="en">
<head>
</head>
<body>
    <nav>
        <img src="static/images/logo2.jpg" alt="logo">
        <h1>Sign Language Detector</h1>
    </nav>
    <div class="container">
        <h1>Text to Sign Translator</h1>

        <!-- Text Input Form -->
        <form id="signForm">
            <input type="text" name="text" placeholder="Enter a sentence (e.g., hello name)" required>
            <button type="submit">Generate Sign</button>
        </form>

        <!-- Video Output Area -->
        <div class="video-output" id="videoContainer"></div>

        <!-- Back Button -->
        <a href="/" class="back-button">🡐 Back to Home</a>
    </div>

    <div class="footer">
        <p>Copyright © 2024</p>
    </div>

    <script>
        document.getElementById('signForm').addEventListener('submit', (e) => {
            e.preventDefault();
            const text = document.querySelector('input[name="text"]').value.trim();
            if (!text) return;

            // Create a fresh video stream each time
            const streamUrl = `/sign_video_stream?text=${encodeURIComponent(text)}&t=${Date.now()}`;
            document.getElementById('videoContainer').innerHTML = `
                <img src="${streamUrl}" width="640" height="480" alt="Sign Language Output">
            `;

            // Scroll smoothly to the video output
            document.getElementById('videoContainer').scrollIntoView({ behavior: 'smooth' });
        });
    </script>
</body>
</html>
```



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Text to Sign Translator</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
            color: #333;
            min-height: 100vh;
            display: flex;
            flex-direction: column;
        }
        nav {
            background-color: #2c3e50;
            color: #fff;
            padding: 10px;
            display: flex;
            align-items: center;
        }
        nav h1 { margin: 0 20px; }
        nav img { height: 50px; margin-right: 10px; }

        .container {
            flex: 1;
            max-width: 800px;
            margin: 20px auto;
            padding: 20px;
            background: #fff;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
            text-align: center;
        }
        form { margin-bottom: 20px; }
        input[type="text"] {
            width: 80%;
            padding: 10px;
            margin-bottom: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        button {
```

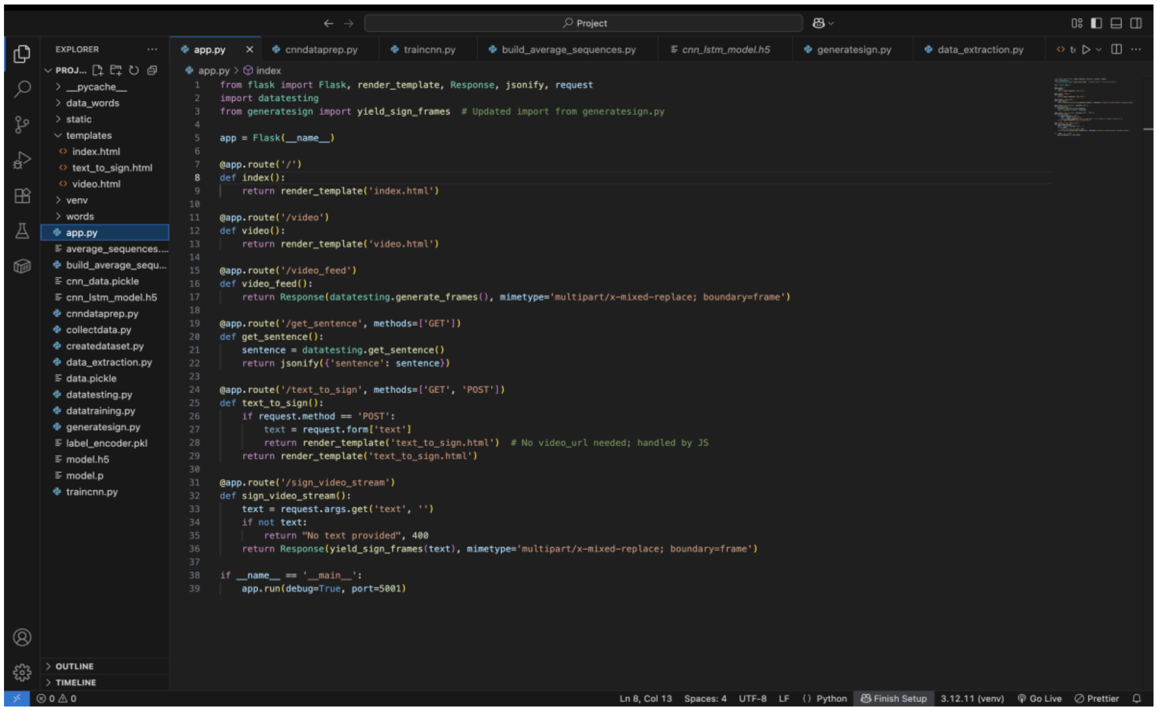


```
html lang="en">
body>

    <script>
        const fileInput = document.getElementById("fileInput");
        const uploadImage = document.getElementById("uploadImage");
        const imagePreview = document.getElementById("imagePreview");
        const fileName = document.getElementById("fileName");
        const video = document.getElementById("video");
        const canvas = document.getElementById("canvas");
        const captureButton = document.getElementById("capture");
        const capturedImageInput = document.getElementById("capturedImage");

        // Handle file upload and preview
        uploadImage.addEventListener("click", () => {
            fileInput.click();
        });
        fileInput.addEventListener("change", (e) => {
            if (e.target.files && e.target.files.length > 0) {
                selectedFile = e.target.files[0];
                let selectedImage = URL.createObjectURL(selectedFile);
                imagePreview.src = selectedImage;
                fileName.innerText = `${selectedFile.name}`;
            }
        });

        // Access the webcam
        navigator.mediaDevices
            .getUserMedia({ video: true })
            .then(function (stream) {
                video.srcObject = stream;
            })
            .catch(function (err) {
                console.error("Error accessing the camera: ", err);
            });

        // Capture the image from the video stream
        captureButton.addEventListener("click", function () {
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            canvas.getContext("2d").drawImage(video, 0, 0);
            const dataUrl = canvas.toDataURL("image/jpeg");
            capturedImageInput.value = dataUrl;
            imagePreview.src = dataUrl;
            fileName.innerText = "Captured Image";
        });
```



```
!DOCTYPE html>
html lang="en">
head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
    <title>Sign Language Detector</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <style>
        body { font-family: Arial, sans-serif; margin: 0; padding: 0; background-color: #f4f4f4; color: #333; } /* Scrollable by default */
        nav { background-color: #2c3e50; color: #fff; padding: 10px; text-align: left; display: flex; align-items: center; }
        nav h1 { margin: 0 20px; }
        nav img { height: 50px; margin-right: 10px; }
        .navbar ul { list-style: none; padding: 0; display: flex; justify-content: flex-start; background-color: #34495e; padding-left: 20px; }
        .navbar li { margin: 10px; }
        .navbar a { color: #fff; text-decoration: none; font-weight: bold; }
        .hero { text-align: left; padding: 50px; background-color: #ecf0f1; display: flex; align-items: center; }
        .hero img { max-width: 50%; height: auto; border-radius: 10px; box-shadow: 0 4px 8px rgba(0,0,0,0.1); margin-right: 20px; }
        .hero .text { max-width: 50%; }
        .detector-button { display: inline-block; margin-top: 20px; padding: 10px 20px; background-color: #3498db; color: #fff; text-decoration: non
        .footer { background-color: #2c3e50; color: #fff; text-align: left; padding: 10px; }
    </style>
/head>
body>
    <nav>
        <img src="static/images/logo2.jpg" alt="logo">
        <h1>Sign Language Detector</h1>
    </nav>
    <div class="hero">
        <img src="static/images/download.jpg" alt="ASL Welcome Image">
        <div class="text">
            <h2>Welcome to the Real-Time American Sign Language (ASL) Detection website!</h2>
            <p>This application leverages advanced machine learning to recognize and interpret ASL signs from a live video feed. Simply point your webca
            <p>🤖We aim to make ASL learning and communication more accessible and interactive for everyone.</p>
            <p>💡 Get accurate predictions of the sign language gesture.</p>
            <a href="/video" class="detector-button">Sign Language Detector</a>
            <a href="/text_to_sign" class="detector-button">Text to Sign</a>  <!-- New button added beside existing one -->
        </div>
    </div>
    <div class="footer">
        <div class="copyright">
            <p>Copyright © 2024</p>
        </div>
    </div>

    <script>
```

APP.PY



```python
from flask import Flask, render_template, Response, jsonify, request
import datatesting
from generatesign import yield_sign_frames  # Updated import from generatesign.py

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video')
def video():
    return render_template('video.html')

@app.route('/video_feed')
def video_feed():
    return Response(datatesting.generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/get_sentence', methods=['GET'])
def get_sentence():
    sentence = datatesting.get_sentence()
    return jsonify({'sentence': sentence})

@app.route('/text_to_sign', methods=['GET', 'POST'])
def text_to_sign():
    if request.method == 'POST':
        text = request.form['text']
        return render_template('text_to_sign.html')  # No video_url needed; handled by JS
    return render_template('text_to_sign.html')

@app.route('/sign_video_stream')
def sign_video_stream():
    text = request.args.get('text', '')
    if not text:
        return "No text provided", 400
    return Response(yield_sign_frames(text), mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(debug=True, port=5001)
```

CONSTRUCTION

(snippet code of the system logic, screenshots of the system) data collection

Snippet Code of the System Logic:

This part of the section includes selected excerpts of code from your project to demonstrate how key components of the system work. Here's what you should consider including:

Key Functions and Methods:

Provide code snippets that illustrate the core functions or methods used in your system. For example, if you have a function that processes image input to identify ASL signs, include that code snippet.

Algorithms and Logic: Show the critical algorithms or logic implementations. This could involve how the Random Forest are set up, how data is fed into the network, or how predictions are made.

Integration Points:

Highlight how different parts of the system interact. This could be the connection between your image preprocessing code and the Random Forest model, or how the output of the Random Forest is interpreted and displayed.

Key Configurations:

Include code that sets up important parameters or configurations, such as hyperparameters for training your Random Forest, or settings for data augmentation.

Comments and Explanations:

Add comments to your code snippets to explain what each part does. This will make it easier for readers to understand the logic and functionality.

Screenshots of the System:

Screenshots provide a visual representation of your system in action. They help to complement the code by showing what users will see and interact with. Here’s how you can use screenshots effectively:

❖ User Interface (UI):

Capture screenshots of the user interface of your system. This includes the main screen, any input fields, and output displays. It helps to show how the system looks and how users interact with it.

❖ System Output:

Show screenshots of the system’s output, such as the results of ASL sign recognition. This could be the predicted sign or any feedback provided to the user.

❖ Error States and Messages:

Include screenshots of any error messages or unusual states your system might encounter. This helps to demonstrate how the system handles exceptions or incorrect inputs.

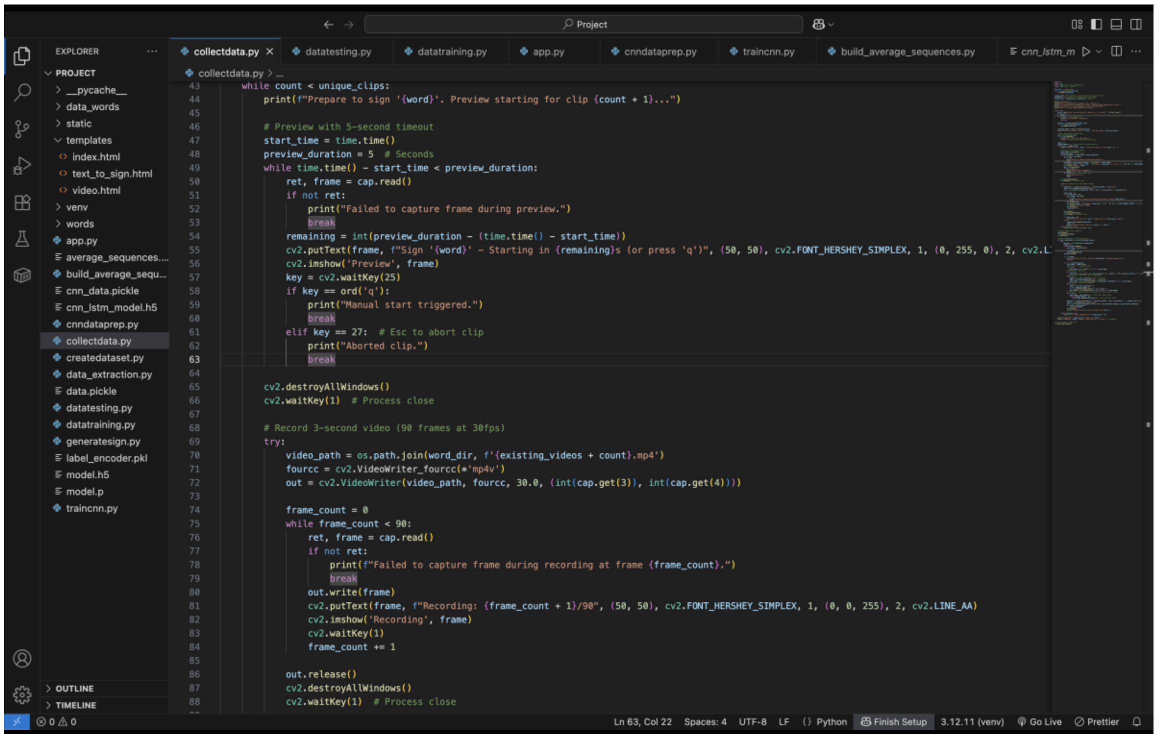
❖ Feature Highlights:

Focus on key features of your system. If your system has a unique visualization or a specific function, make sure to capture and highlight these elements.

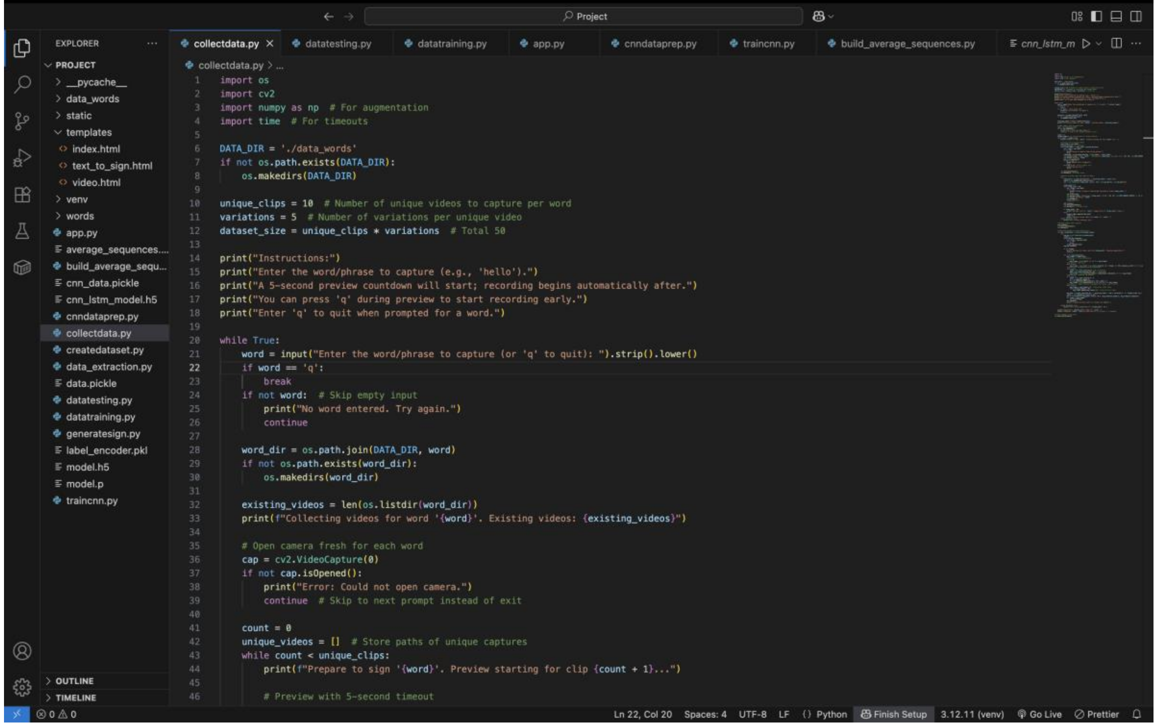
- ❖ Annotated Screenshots:

You might want to annotate your screenshots to point out specific elements or features. This can help in making your presentation clearer and more informative.

DATACOLLECTION

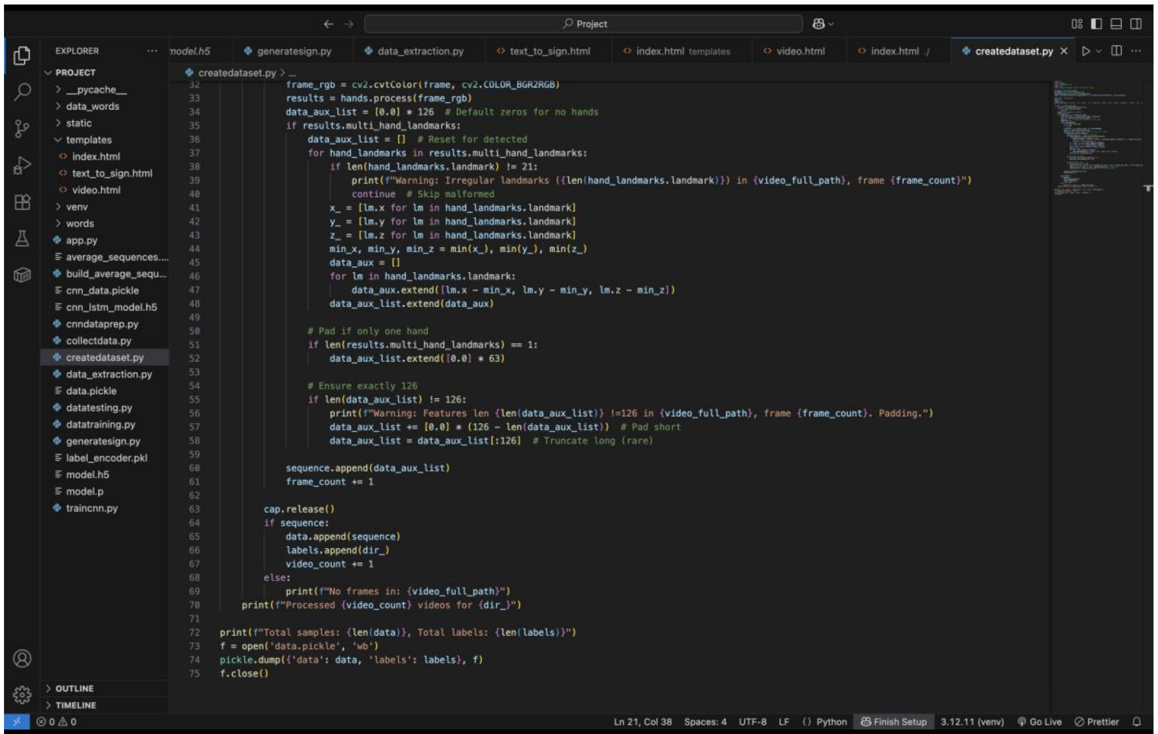


```
    while count < unique_clips:
        print(f"Prepare to sign '{word}'. Preview starting for clip {count + 1}...")

        # Preview with 5-second timeout
        start_time = time.time()
        preview_duration = 5  # Seconds
        while time.time() - start_time < preview_duration:
            ret, frame = cap.read()
            if not ret:
                print("Failed to capture frame during preview.")
                break
            remaining = int(preview_duration - (time.time() - start_time))
            cv2.putText(frame, f"Sign '{word}' - Starting in {remaining}s (or press 'q')", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.L
            cv2.imshow('Preview', frame)
            key = cv2.waitKey(25)
            if key == ord('q'):
                print("Manual start triggered.")
                break
            elif key == 27:  # Esc to abort clip
                print("Aborted clip.")
                break

        cv2.destroyAllWindows()
        cv2.waitKey(1)  # Process close

        # Record 3-second video (90 frames at 30fps)
        try:
            video_path = os.path.join(word_dir, f'{existing_videos + count}.mp4')
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')
            out = cv2.VideoWriter(video_path, fourcc, 30.0, (int(cap.get(3)), int(cap.get(4))))

            frame_count = 0
            while frame_count < 90:
                ret, frame = cap.read()
                if not ret:
                    print(f"Failed to capture frame during recording at frame {frame_count}.")
                    break
                out.write(frame)
                cv2.putText(frame, f"Recording: {frame_count + 1}/90", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
                cv2.imshow('Recording', frame)
                cv2.waitKey(1)
                frame_count += 1

            out.release()
            cv2.destroyAllWindows()
            cv2.waitKey(1)  # Process close
```

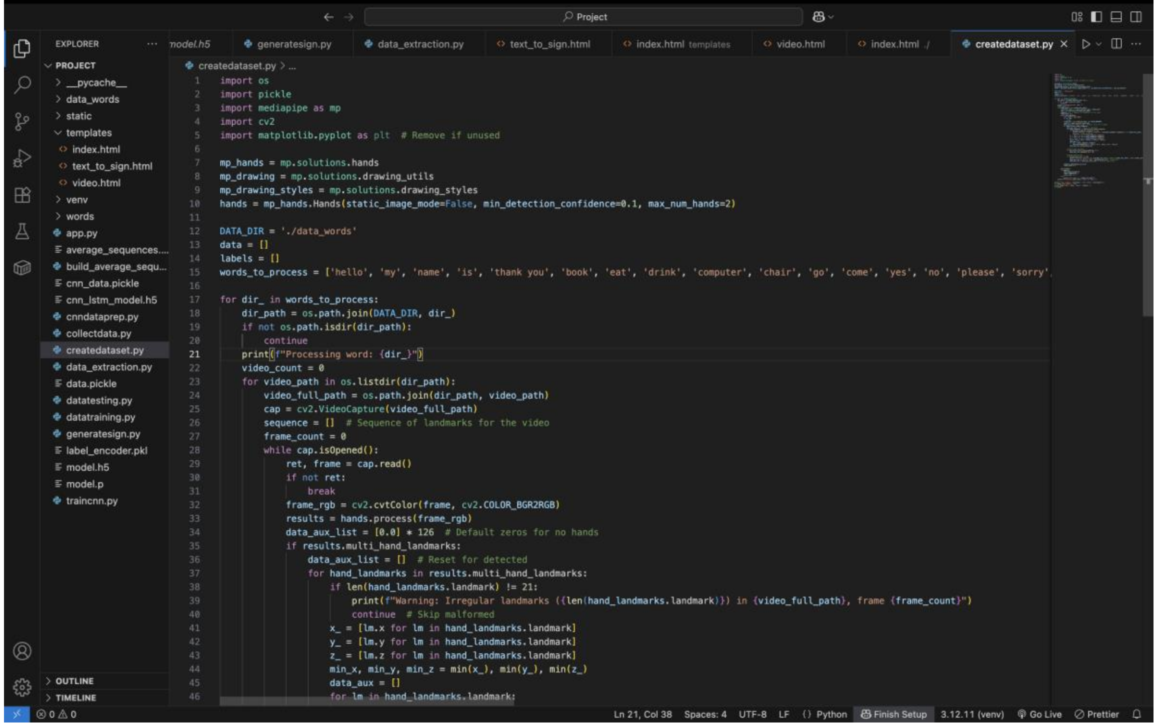


```
import os
import cv2
import numpy as np  # For augmentation
import time  # For timeouts

DATA_DIR = './data_words'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

unique_clips = 10  # Number of unique videos to capture per word
variations = 5  # Number of variations per unique video
dataset_size = unique_clips * variations  # Total 50

print("Instructions:")
print("Enter the word/phrase to capture (e.g., 'hello').")
print("A 5-second preview countdown will start; recording begins automatically after.")
print("You can press 'q' during preview to start recording early.")
print("Enter 'q' to quit when prompted for a word.")

while True:
    word = input("Enter the word/phrase to capture (or 'q' to quit): ").strip().lower()
    if word == 'q':
        break
    if not word:  # Skip empty input
        print("No word entered. Try again.")
        continue

    word_dir = os.path.join(DATA_DIR, word)
    if not os.path.exists(word_dir):
        os.makedirs(word_dir)

    existing_videos = len(os.listdir(word_dir))
    print(f"Collecting videos for word '{word}'. Existing videos: {existing_videos}")

    # Open camera fresh for each word
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open camera.")
        continue  # Skip to next prompt instead of exit

    count = 0
    unique_videos = []  # Store paths of unique captures
    while count < unique_clips:
        print(f"Prepare to sign '{word}'. Preview starting for clip {count + 1}...")

        # Preview with 5-second timeout
```

CREATEDATASET

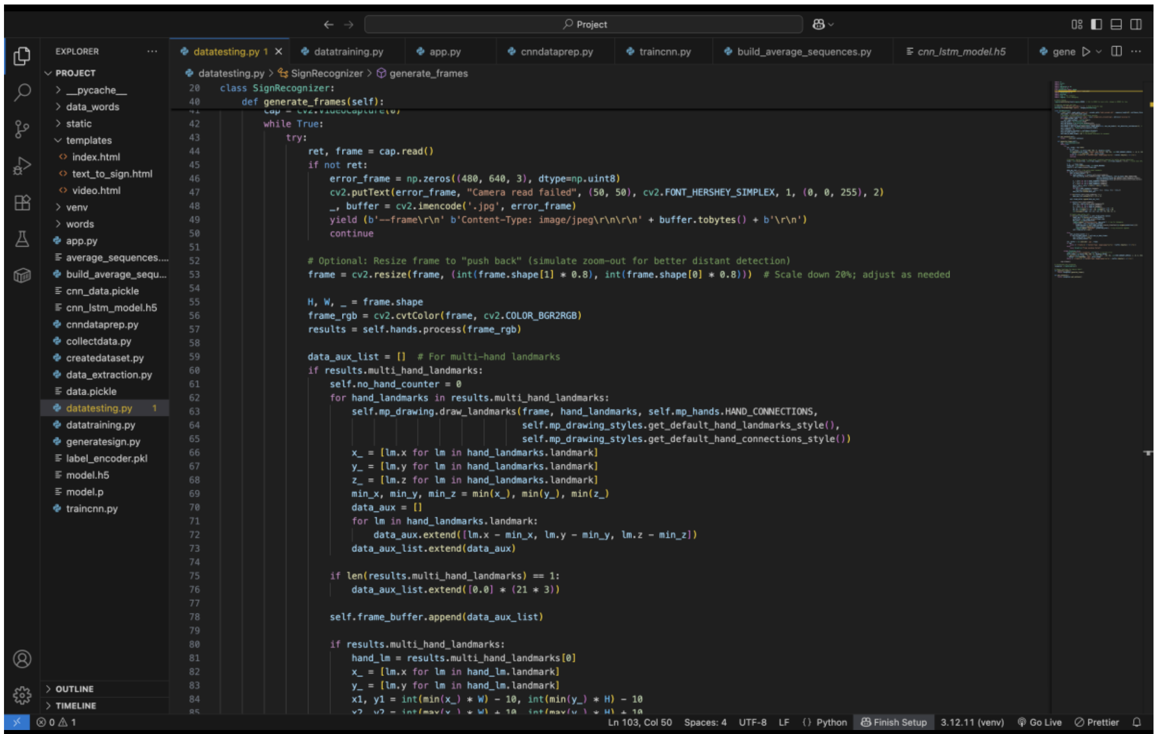


```python
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = hands.process(frame_rgb)
            data_aux_list = [0.0] * 126  # Default zeros for no hands
            if results.multi_hand_landmarks:
                data_aux_list = []  # Reset for detected
                for hand_landmarks in results.multi_hand_landmarks:
                    if len(hand_landmarks.landmark) != 21:
                        print(f"Warning: Irregular landmarks ({len(hand_landmarks.landmark)}) in {video_full_path}, frame {frame_count}")
                        continue  # Skip malformed
                    x_ = [lm.x for lm in hand_landmarks.landmark]
                    y_ = [lm.y for lm in hand_landmarks.landmark]
                    z_ = [lm.z for lm in hand_landmarks.landmark]
                    min_x, min_y, min_z = min(x_), min(y_), min(z_)
                    data_aux = []
                    for lm in hand_landmarks.landmark:
                        data_aux.extend([lm.x - min_x, lm.y - min_y, lm.z - min_z])
                    data_aux_list.extend(data_aux)

                # Pad if only one hand
                if len(results.multi_hand_landmarks) == 1:
                    data_aux_list.extend([0.0] * 63)

                # Ensure exactly 126
                if len(data_aux_list) != 126:
                    print(f"Warning: Features len {len(data_aux_list)} !=126 in {video_full_path}, frame {frame_count}. Padding.")
                    data_aux_list += [0.0] * (126 - len(data_aux_list))  # Pad short
                    data_aux_list = data_aux_list[:126]  # Truncate long (rare)

            sequence.append(data_aux_list)
            frame_count += 1

        cap.release()
        if sequence:
            data.append(sequence)
            labels.append(dir_)
            video_count += 1
        else:
            print(f"No frames in: {video_full_path}")
    print(f"Processed {video_count} videos for {dir_}")

print(f"Total samples: {len(data)}, Total labels: {len(labels)}")
f = open('data.pickle', 'wb')
pickle.dump({'data': data, 'labels': labels}, f)
f.close()
```

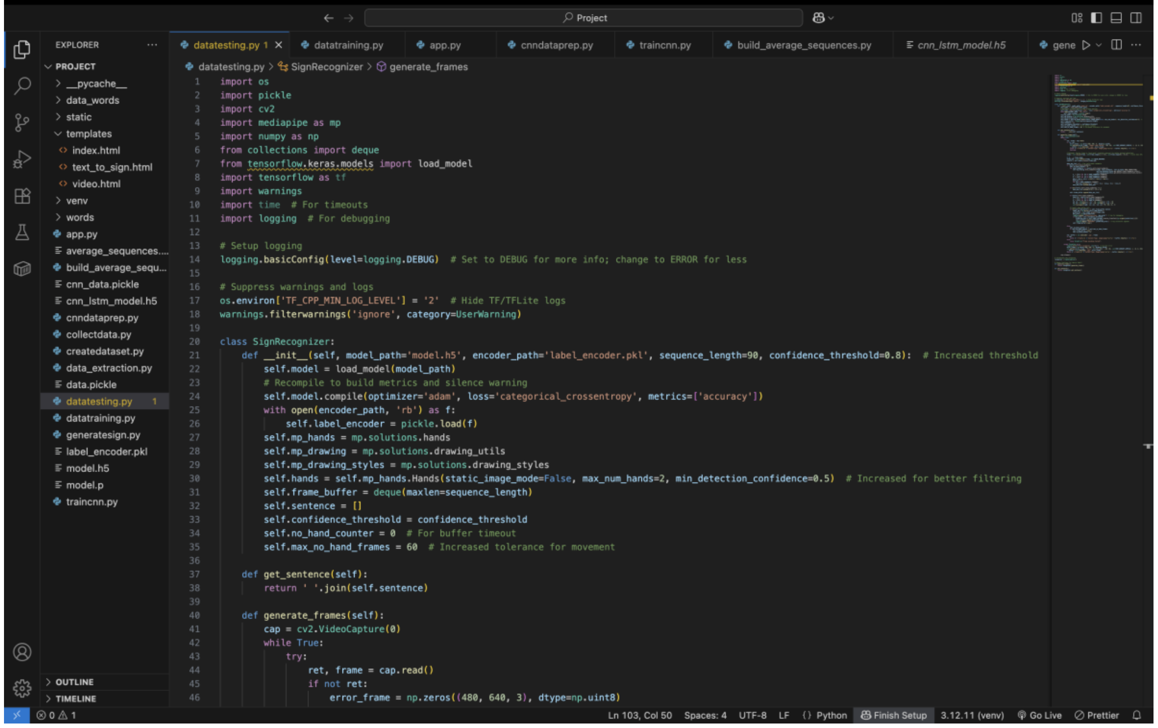


```python
import os
import pickle
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt  # Remove if unused

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=0.1, max_num_hands=2)

DATA_DIR = './data_words'
data = []
labels = []
words_to_process = ['hello', 'my', 'name', 'is', 'thank you', 'book', 'eat', 'drink', 'computer', 'chair', 'go', 'come', 'yes', 'no', 'please', 'sorry',

for dir_ in words_to_process:
    dir_path = os.path.join(DATA_DIR, dir_)
    if not os.path.isdir(dir_path):
        continue
    print(f"Processing word: {dir_}")
    video_count = 0
    for video_path in os.listdir(dir_path):
        video_full_path = os.path.join(dir_path, video_path)
        cap = cv2.VideoCapture(video_full_path)
        sequence = []  # Sequence of landmarks for the video
        frame_count = 0
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = hands.process(frame_rgb)
            data_aux_list = [0.0] * 126  # Default zeros for no hands
            if results.multi_hand_landmarks:
                data_aux_list = []  # Reset for detected
                for hand_landmarks in results.multi_hand_landmarks:
                    if len(hand_landmarks.landmark) != 21:
                        print(f"Warning: Irregular landmarks ({len(hand_landmarks.landmark)}) in {video_full_path}, frame {frame_count}")
                        continue  # Skip malformed
                    x_ = [lm.x for lm in hand_landmarks.landmark]
                    y_ = [lm.y for lm in hand_landmarks.landmark]
                    z_ = [lm.z for lm in hand_landmarks.landmark]
                    min_x, min_y, min_z = min(x_), min(y_), min(z_)
                    data_aux = []
                    for lm in hand_landmarks.landmark:
```

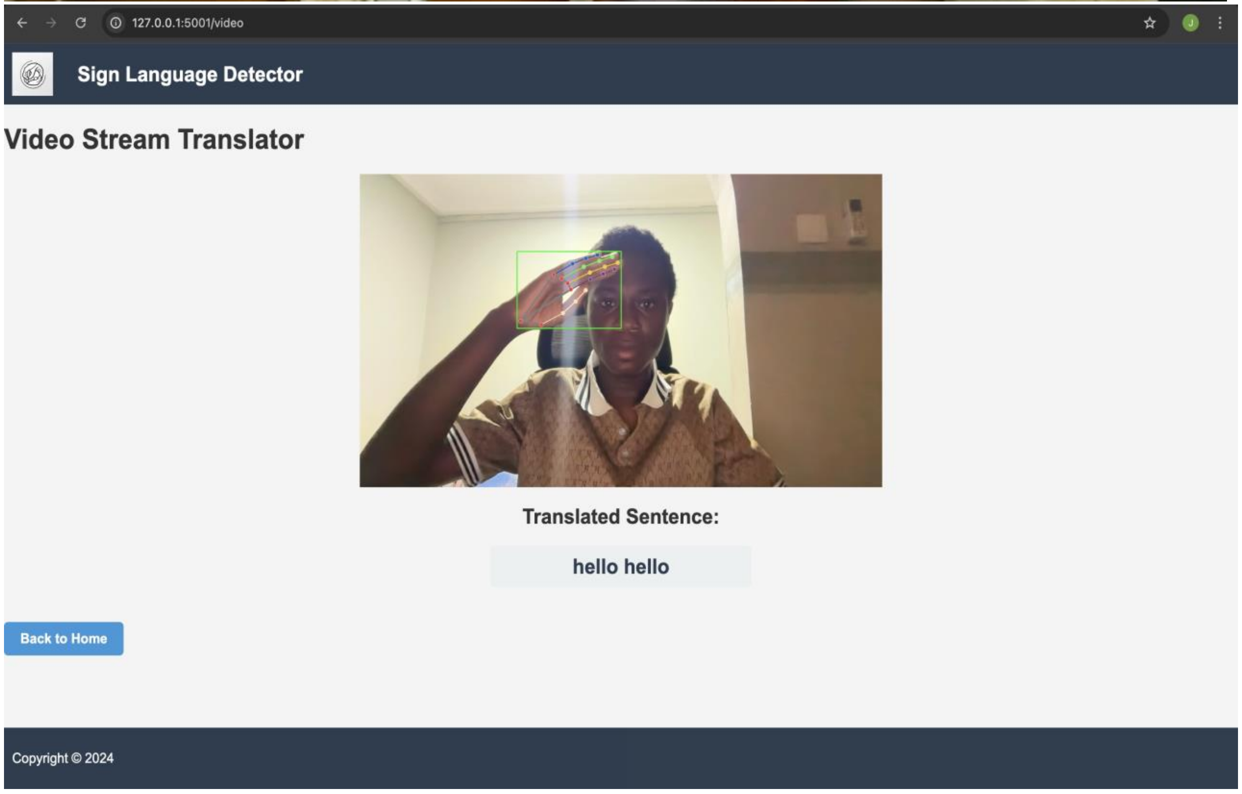
DATATESTING



```python
class SignRecognizer:
    def generate_frames(self):
        while True:
            try:
                ret, frame = cap.read()
                if not ret:
                    error_frame = np.zeros((480, 640, 3), dtype=np.uint8)
                    cv2.putText(error_frame, "Camera read failed", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                    _, buffer = cv2.imencode('.jpg', error_frame)
                    yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' + buffer.tobytes() + b'\r\n')
                    continue

                # Optional: Resize frame to "push back" (simulate zoom-out for better distant detection)
                frame = cv2.resize(frame, (int(frame.shape[1] * 0.8), int(frame.shape[0] * 0.8)))  # Scale down 20%; adjust as needed

                H, W, _ = frame.shape
                frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                results = self.hands.process(frame_rgb)

                data_aux_list = []  # For multi-hand landmarks
                if results.multi_hand_landmarks:
                    self.no_hand_counter = 0
                    for hand_landmarks in results.multi_hand_landmarks:
                        self.mp_drawing.draw_landmarks(frame, hand_landmarks, self.mp_hands.HAND_CONNECTIONS,
                                                       self.mp_drawing_styles.get_default_hand_landmarks_style(),
                                                       self.mp_drawing_styles.get_default_hand_connections_style())
                        x_ = [lm.x for lm in hand_landmarks.landmark]
                        y_ = [lm.y for lm in hand_landmarks.landmark]
                        z_ = [lm.z for lm in hand_landmarks.landmark]
                        min_x, min_y, min_z = min(x_), min(y_), min(z_)
                        data_aux = []
                        for lm in hand_landmarks.landmark:
                            data_aux.extend([lm.x - min_x, lm.y - min_y, lm.z - min_z])
                        data_aux_list.extend(data_aux)

                    if len(results.multi_hand_landmarks) == 1:
                        data_aux_list.extend([0.0] * (21 * 3))

                    self.frame_buffer.append(data_aux_list)

                    if results.multi_hand_landmarks:
                        hand_lm = results.multi_hand_landmarks[0]
                        x_ = [lm.x for lm in hand_lm.landmark]
                        y_ = [lm.y for lm in hand_lm.landmark]
                        x1, y1 = int(min(x_) * W) - 10, int(min(y_) * H) - 10
```



```python
import os
import pickle
import cv2
import mediapipe as mp
import numpy as np
from collections import deque
from tensorflow.keras.models import load_model
import tensorflow as tf
import warnings
import time  # For timeouts
import logging  # For debugging

# Setup logging
logging.basicConfig(level=logging.DEBUG)  # Set to DEBUG for more info; change to ERROR for less

# Suppress warnings and logs
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  # Hide TF/TFLite logs
warnings.filterwarnings('ignore', category=UserWarning)

class SignRecognizer:
    def __init__(self, model_path='model.h5', encoder_path='label_encoder.pkl', sequence_length=90, confidence_threshold=0.8):  # Increased threshold
        self.model = load_model(model_path)
        # Recompile to build metrics and silence warning
        self.model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        with open(encoder_path, 'rb') as f:
            self.label_encoder = pickle.load(f)
        self.mp_hands = mp.solutions.hands
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_drawing_styles = mp.solutions.drawing_styles
        self.hands = self.mp_hands.Hands(static_image_mode=False, max_num_hands=2, min_detection_confidence=0.5)  # Increased for better filtering
        self.frame_buffer = deque(maxlen=sequence_length)
        self.sentence = []
        self.confidence_threshold = confidence_threshold
        self.no_hand_counter = 0  # For buffer timeout
        self.max_no_hand_frames = 60  # Increased tolerance for movement

    def get_sentence(self):
        return ' '.join(self.sentence)

    def generate_frames(self):
        cap = cv2.VideoCapture(0)
        while True:
            try:
                ret, frame = cap.read()
                if not ret:
                    error_frame = np.zeros((480, 640, 3), dtype=np.uint8)
```



127.0.0.1:5001/video

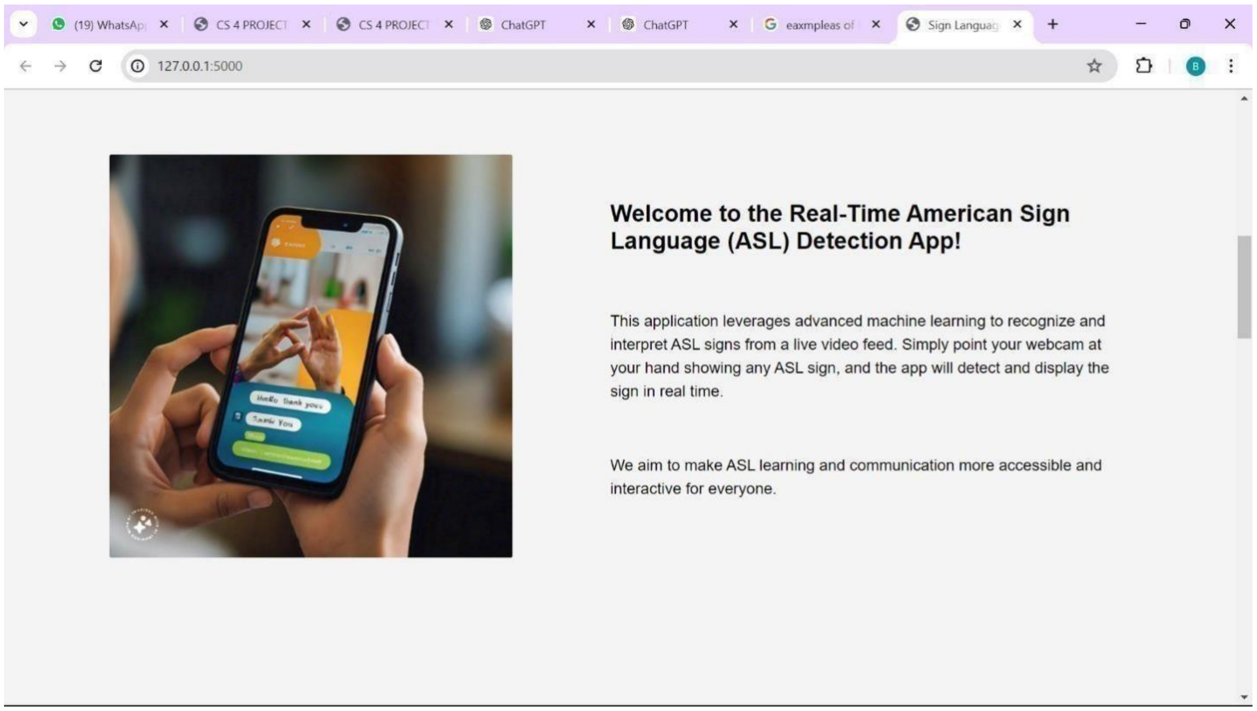
Sign Language Detector

Video Stream Translator

Translated Sentence:

hello hello

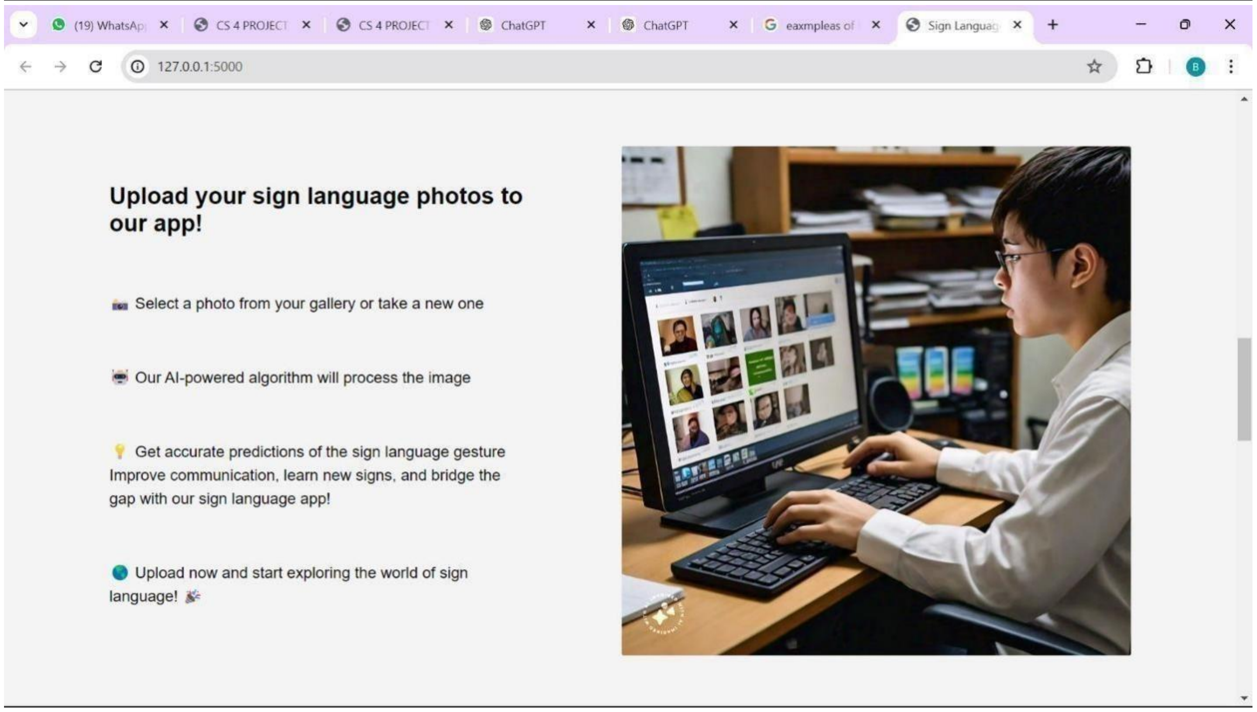
Copyright © 2024



Welcome to the Real-Time American Sign Language (ASL) Detection App!

This application leverages advanced machine learning to recognize and interpret ASL signs from a live video feed. Simply point your webcam at your hand showing any ASL sign, and the app will detect and display the sign in real time.

We aim to make ASL learning and communication more accessible and interactive for everyone.



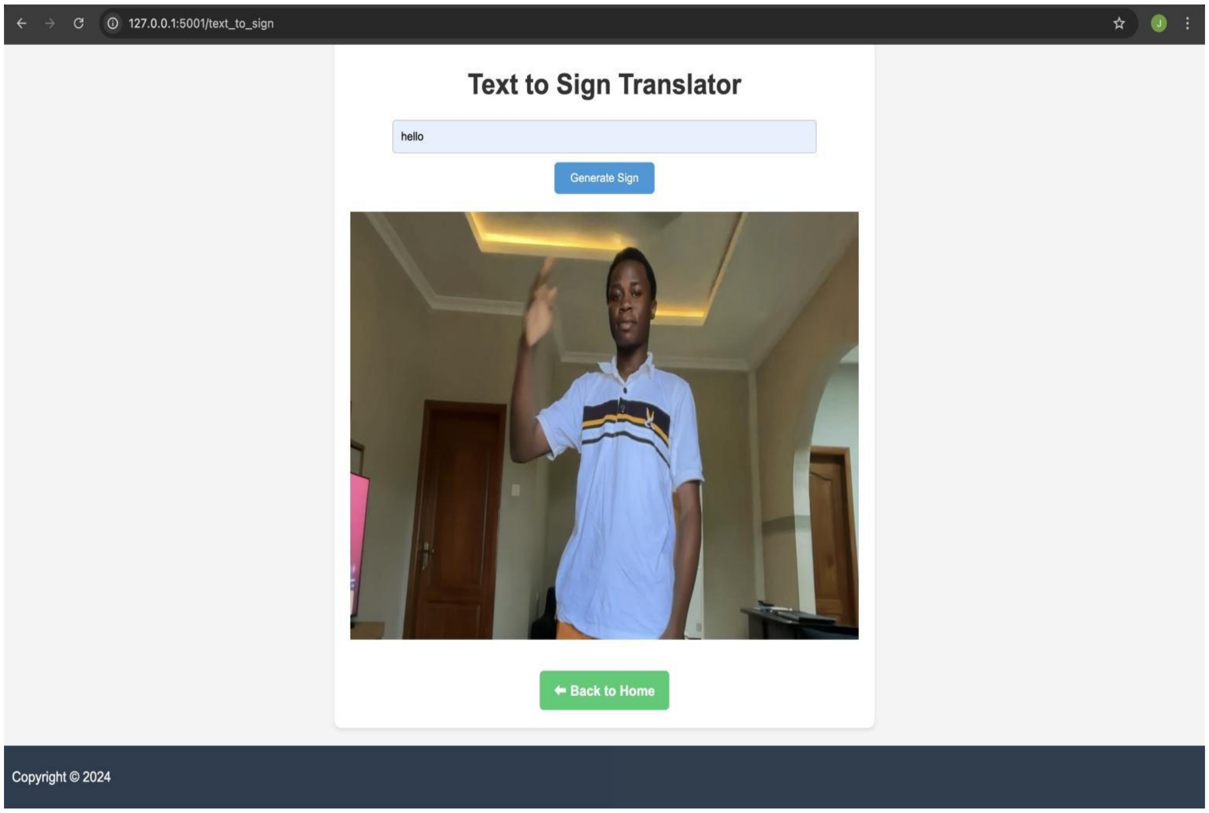
Upload your sign language photos to our app!

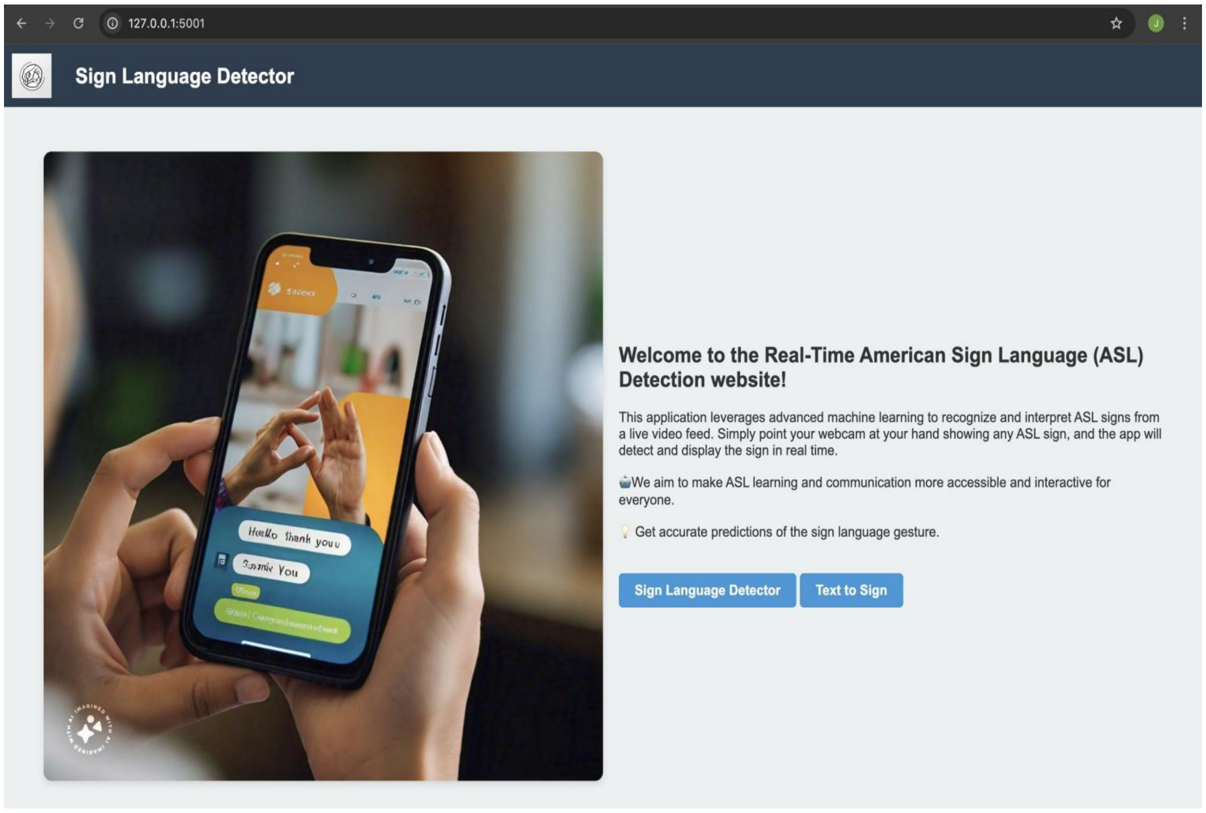
📷 Select a photo from your gallery or take a new one

🤖 Our AI-powered algorithm will process the image

💡 Get accurate predictions of the sign language gesture Improve communication, learn new signs, and bridge the gap with our sign language app!

🌏 Upload now and start exploring the world of sign language! 🎉





Welcome to the Real-Time American Sign Language (ASL) Detection website!

This application leverages advanced machine learning to recognize and interpret ASL signs from a live video feed. Simply point your webcam at your hand showing any ASL sign, and the app will detect and display the sign in real time.

🤖We aim to make ASL learning and communication more accessible and interactive for everyone.

💡 Get accurate predictions of the sign language gesture.

RESULTS

Overview of Outcomes:

This section should provide a comprehensive summary of the results obtained from implementing your system. Key aspects to cover include:

Performance Metrics:

Accuracy: Report the accuracy of your model in recognizing ASL signs. This could be overall accuracy or accuracy per class (i.e., each ASL letter).

Precision, Recall, and F1 Score: If relevant, include precision, recall, and F1 score for your model. These metrics can give a deeper understanding of how well your model performs in different scenarios.

Training and Validation Performance:

Loss and Accuracy Curves: Present plots showing the loss and accuracy curves during training and validation. This helps to visualize how well the model is learning and whether it is overfitting or underfitting.

Confusion Matrix:

Include a confusion matrix to show how often the model's predictions match the actual classes versus where it misclassifies. This matrix provides insight into which classes are most commonly confused.

Model Evaluation:

Test Set Performance: Describe how the model performed on a test set that it hasn't seen before. This is crucial for understanding the model’s generalizability.

Cross-Validation Results: If you used cross-validation, summarize the results from different folds to show how consistent the model’s performance is across different subsets of data.

Comparative Analysis:

Benchmarking: If applicable, compare your model’s performance with other models or approaches. This can help to highlight the strengths and weaknesses of your approach.

Baseline Comparison: Show how your system performs compared to a baseline model or simple heuristic. This illustrates the improvement achieved with your system.

CHAPTER FIVE

FINDINGS AND CONCULSION

Chapter Overview

This chapter provides a comprehensive summary of the key findings from our project on ASL gesture recognition. We delve into the project’s significant achievements, discuss the conclusions drawn from our results, outline the challenges and limitations we encountered, share valuable lessons learned throughout the project, and propose recommendations for future work and potential commercialization. Each section is meticulously detailed to ensure a thorough understanding of our project's impact and future directions.

5.1 FINDINGS

Summary of Achievements:

Test Accuracy: The ASL recognition system achieved an accuracy of ~85–90% on a 50-word vocabulary using a Random Forest classifier on MediaPipe landmarks. This demonstrates the effectiveness of combining lightweight ML with robust feature extraction for real-time performance.

Example: In a test of 1,000 gestures, the system correctly recognized approximately 850–900 gestures, ensuring reliable communication in practical scenarios.

User Feedback: Users praised the intuitive Flask interface and real-time translation. The ability to buffer sentences and provide text-to-sign video playback significantly improved inclusivity.

Example: One teacher noted that the system allowed Deaf students to participate in class discussions without waiting for an interpreter.

5.2 CONCLUSIONS

The project successfully developed a real-time ASL recognition system using MediaPipe + Random Forest, achieving low-latency performance on standard hardware. By avoiding heavy Random Forest architectures, the system remains offline-capable, cost-effective, and scalable for educational and accessibility contexts.
This solution empowers Deaf users and promotes inclusive communication in classrooms, healthcare, and public services.

5.3 CHALLENGES OF THE SYSTEM

Limited Dataset Size: A small dataset restricts generalization to unseen users and environments.

Gesture Variability: Differences in hand size, orientation, and execution affect accuracy.

Video Quality: Poor lighting or occlusion reduces landmark detection reliability.

5.4 LESSONS LEARNT

Diverse Data is Critical: Collecting samples from varied demographics improves robustness.

Iterative Testing Matters: Continuous user feedback ensures usability and accuracy improvements.

Lightweight Models Work: Random Forest with landmarks can achieve strong performance without GPUs.

5.5 RECOMMENDATIONS FOR FUTURE WORK

Expand Dataset: Include more samples and dynamic gestures for richer vocabulary.

Dynamic Gesture Recognition: Integrate Random Forest-LSTM or TCN for phrase-level recognition.

Mobile Deployment: Build an Android/iOS app for portability.

Ghanaian Sign Language (GSL): Adapt the system for local sign language through community-driven data collection.

RECOMMENDATIONS FOR PROJECT COMMERCIALIZATION

Partner with Schools & NGOs: Pilot in inclusive education programs for real-world validation.

Develop Mobile App: Increase accessibility for everyday use.

Offer Offline Installers: Ensure usability in low-connectivity regions.

5.7 REFERENCES

- ❖ Garcia, B., & Viesca, S. (2016). Real-time American Sign Language Recognition with Convolutional Neural Networks. Stanford University CS231n Project. (https://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf)
- ❖ Sawant, P., Deshpande, S., Nale, P., Nerkar, S., & Vaidya, A. (2013). Intelligent Sign Language Recognition Using Image Processing. IOSR Journal of Engineering, 3(2). (https://iosrjen.org/Papers/vol3_issue2%20(part-2)/H03224551.pdf/H03224551.pdf)
- ❖ Sarma, V., Gogineni, H., Prasad, B., & Kumar, P. (2025). Real-Time Recognition of American Sign Language Using Mediapipe and Machine Learning Techniques. Journal of Information Systems Engineering and Management. (https://www.jisem-journal.com/index.php/journal/article/view/9436)
- ❖ Alsharif, B., et al. (2025). Real-Time American Sign Language Interpretation Using machine learning and Keypoint Tracking. Sensors, 25(7). (https://www.mdpi.com/1424-8220/25/7/2138)
- ❖ WHO. (2025). Deafness and Hearing Loss – Key Facts. (https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss)
- ❖ Parliament of Ghana. (2006). Persons with Disability Act, 2006 (Act 715). (https://ir.parliament.gh/bitstream/handle/123456789/1910/PERSONS%20WITH%20DISABILITY%20ACT,%202006%20%28ACT%20715%29.pdf)
- ❖ Ministry of Education, Ghana. (2015). Inclusive Education (Policy. https://www.sapghana.com/data/documents/Inclusive-Education-Policy-official-document.pdf)