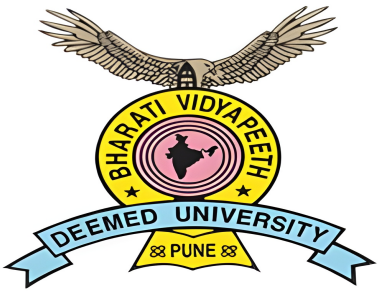


A

Project Stage-II Report on

Indian Speech-to-Sign Language Conversion Tool

Submitted in the partial fulfillment of the requirements of Semester-VIII

For the Award of the Degree of Bachelor of Technology (B.Tech) in

Electronics and Communication Engineering

Submitted by

P.Meenakshi Reddy	**PRN: 2114110492**
Soni Kumari	**PRN: 2114110482**
Manisha Kumari	**PRN: 2114110480**

Under the Guidance of

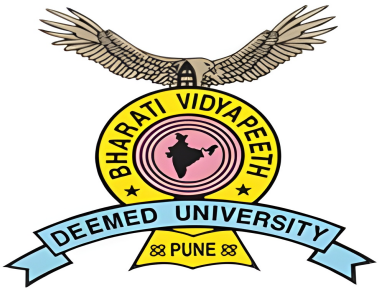
Prof. Jyoti Morbale

Department of Electronics and Communication Engineering

BHARATI VIDYAPEETH DEEMED TO BE UNIVERSITY

COLLEGE OF ENGINEERING, PUNE - 411 043

ACADEMIC YEAR: 2024-2025



Department of Electronics and Communication Engineering
BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)
COLLEGE OF ENGINEERING, PUNE

CERTIFICATE

This is to certify that the project phase-II report on **"Indian-speech-to-Sign Language Conversion Tool"** submitted by

P.Meenakshi Reddy	**PRN:**	**2114110492**
Soni Kumari	**PRN:**	**2114110482**
Manisha Roy	**PRN:**	**2114110480**

in partial fulfillment of the requirements for the award of degree of Bachelor of Technology (B.Tech) in Electronics and Communication Engineering.

Prof Jyoti Morbale
Guide, ECE Dept.,
BV(DU), COE, Pune

Dr. Dhiraj M. Dhane
Project Co-ordinator
BV(DU), COE, Pune

Prof.(Dr.) Arundhati A. Shinde
Head, ECE Dept.
BV(DU), COE, Pune

Date:

Place: Pune

Abstract

Effective communication is vital for individuals with hearing and speech impairments, and bridging the communication gap is essential in fostering inclusivity. This project presents the development of a system that integrates Speech to Sign language. Utilizing a Kaggle dataset consisting of ISL animated videos, the system allows users to input Hindi text and english text, which is then dynamically converted into corresponding ISL signs.

Built using Python, Django, and machine learning techniques, the tool provides an intuitive user interface that displays sign language images in real-time. The system aims to improve accessibility by providing visual translations of Hindi text, empowering both the deaf and hearing communities to communicate more effectively. By eliminating language barriers, it enhances interaction, education, and engagement for individuals who use ISL.

The tool's potential extends to educational and assistive technologies, offering a scalable solution for ISL-Hindi communication. Initial tests suggest that the system is not only practical but also a valuable resource for promoting greater understanding and inclusivity.

Acknowledgments

We would like to express our sincere gratitude to Prof. Jyoti Morbhale, our project guide, for their invaluable guidance, encouragement, and support throughout the development of our project on the Indian Speech-to-Sign Language Conversion Tool. Their insights and expertise have been instrumental in shaping this project.

We extend our heartfelt thanks to Dr. Dhiraj M. Dhane for providing invaluable mentorship and to Prof. (Dr.) Arundhati A. Shinde, the Head of our Department, for their unwavering support and encouragement.

Lastly, we are grateful to the Department of Electronics and Communication Engineering and to our institution, Bharati Vidyapeeth Deemed to be University, College of Engineering, Pune, for providing us with the resources and the opportunity to undertake this project. Their support has been crucial in helping us achieve our project objectives.

P. Meenakshi Reddy

Soni Chaudhary

Manisha Kumari

Contents

Abstract i

Acknowledgements ii

List of Figures vi

List of Tables vii

1 Introduction **1**

1.1 Overview 1

1.1.1 Importance of ISL to Hindi/English Translation Tool 2

1.2 Problem Statement 3

1.3 Objectives 4

2 Literature Review **6**

2.1 Overview 6

2.2 Related Work 6

2.2.1 AI-Driven Hindi/English Sign Language Translation 6

2.2.2 Web-Based Speech-to-Sign Language Tools 7

2.2.3 Real-Time Speech-to-Sign Language Tools 7

2.2.4 Gesture Recognition for Sign Language 7

2.2.5 AI-Powered Sign Language Recognition 7

2.3 Gaps in Current Technology 8

2.3.1 Limited Gesture Recognition Accuracy 8

2.3.2 Real-Time Processing Limitations 8

2.3.3 Contextual Interpretation Gaps 8

2.3.4 Limited Language-Specific Datasets 8

2.3.5 Lack of Integration with Platforms 9
2.4 Comparison of Sign Language Interpretation Systems 9

3 Methodology **13**
3.1 System Architecture 13
3.2 Explanation of Modules 14
3.3 Design Flow 15
3.4 Tools and Technologies Used 16

4 Design and Implementation **18**
4.1 System Overview 18
4.2 Flowchart of the Plan 20
4.3 Explanation of the Flowchart 21
4.4 System Output 21
4.5 Working Principle 24
4.6 Challenges and Solutions 24

5 Project Plan **26**
5.1 Project Implementation Schedule 27
5.2 Team Roles and Responsibilities 28
5.2.1 Project Objective 28
5.2.2 Task Breakdown 29
5.2.3 Team Member Assignments 29
5.2.4 Communication and Collaboration 30
5.3 Gantt Chart 31

6 Results and Discussion **32**
6.1 Summary of Expected Outcome 32
6.2 Potential Impact and Real-World Applications 33
6.2.1 Healthcare and Infection Control 33
6.2.2 Educational Use 33
6.2.3 Workplace Communication 34

6.2.4 Public Spaces 34
6.2.5 Global Communication and Tourism 34
6.3 Impact on the Community 35

7 Conclusion and Future Scope **36**
7.1 Conclusion 36
7.2 Future Scope 37

References **39**
Appendix A: Source Code 41
Appendix B: Hardware Implementation 81

List of Figures

1.1 Indian Sign Language Dataset *Courtesy:https://www.kaggle.com/datasets/koushik-chouhan/ indian-sign-language-animated-videos* 3

3.1 Architecture of Indian Speech-to-Sign Conversion Tool 14

4.1 Flowchart of the Indian Speech-to-Sign language Conversion Tool 20
4.2 Output Display of web interface 22
4.3 Hindi Text Input after Translation 22
4.4 English Text Output Provided by the User 23
4.5 Recognized English Speech and Corresponding ISL Gestures 23
4.6 Recognized Hindi Speech and Corresponding ISL Gestures 24

5.1 Gantt Chart for the Project Plan 31

List of Tables

2.1 Comparison of Speech-to-Sign Language Interpretation Systems 9

Chapter 1

Introduction

1.1 Overview

The Indian Sign Language (ISL) to Hindi and English Translation Tool is an innovative digital solution designed to bridge communication gaps between Hindi/English speakers and the deaf and hard-of-hearing community. By utilizing a Kaggle dataset consisting of 3D animation videos demonstrating ISL signs, the system translates both Hindi and English text into corresponding ISL visual representations. Built with Python and Django, the tool offers seamless real-time translation through a user-friendly interface. It aims to enhance accessibility and inclusivity across various fields such as education, healthcare, and everyday interactions, providing a scalable and effective communication solution.

1.1.1 Importance of ISL to Hindi/English Translation Tool

1.1.1 Importance of ISL Translation

Effective communication between the deaf community and Hindi/English speakers is essential for inclusion in daily life. This tool provides a solution by translating Hindi and English text into Indian Sign Language (ISL) gestures using 3D animation videos. By enhancing accessibility in sectors like education, healthcare, and social interactions, it breaks communication barriers and fosters inclusivity, especially in scenarios where human interpreters are not available.

1.1.2 Technical Significance of the ISL Translation Tool

This tool is developed using **Python** and **Django**, incorporating a Kaggle dataset containing 3D animation videos of ISL gestures. The key components include:

- **Text Translation:** Hindi and English text are processed and mapped to their corresponding ISL gestures.
- **Gesture Representation:** Using the 3D animation dataset, the system displays ISL gestures as visual translations.
- **Graphical User Interface (GUI):** The tool offers a seamless, interactive interface to facilitate user experience.

1.1.3 Key Benefits of the ISL Translation Tool

1. Bridges communication gaps between Hindi/English speakers and the Deaf community.
2. Provides a scalable solution for real-time ISL translations.
3. Enhances accessibility across educational and professional environments.
4. Supports inclusivity in healthcare, enabling effective communication with Deaf patients.

5. Promotes social integration by facilitating smoother daily interactions.

By utilizing 3D animation videos and dynamic gesture representation, this tool offers a practical approach to bridging language barriers, improving communication for those who rely on ISL.Below is the figure 1.1 that shows the dataset obtained from kaggle.

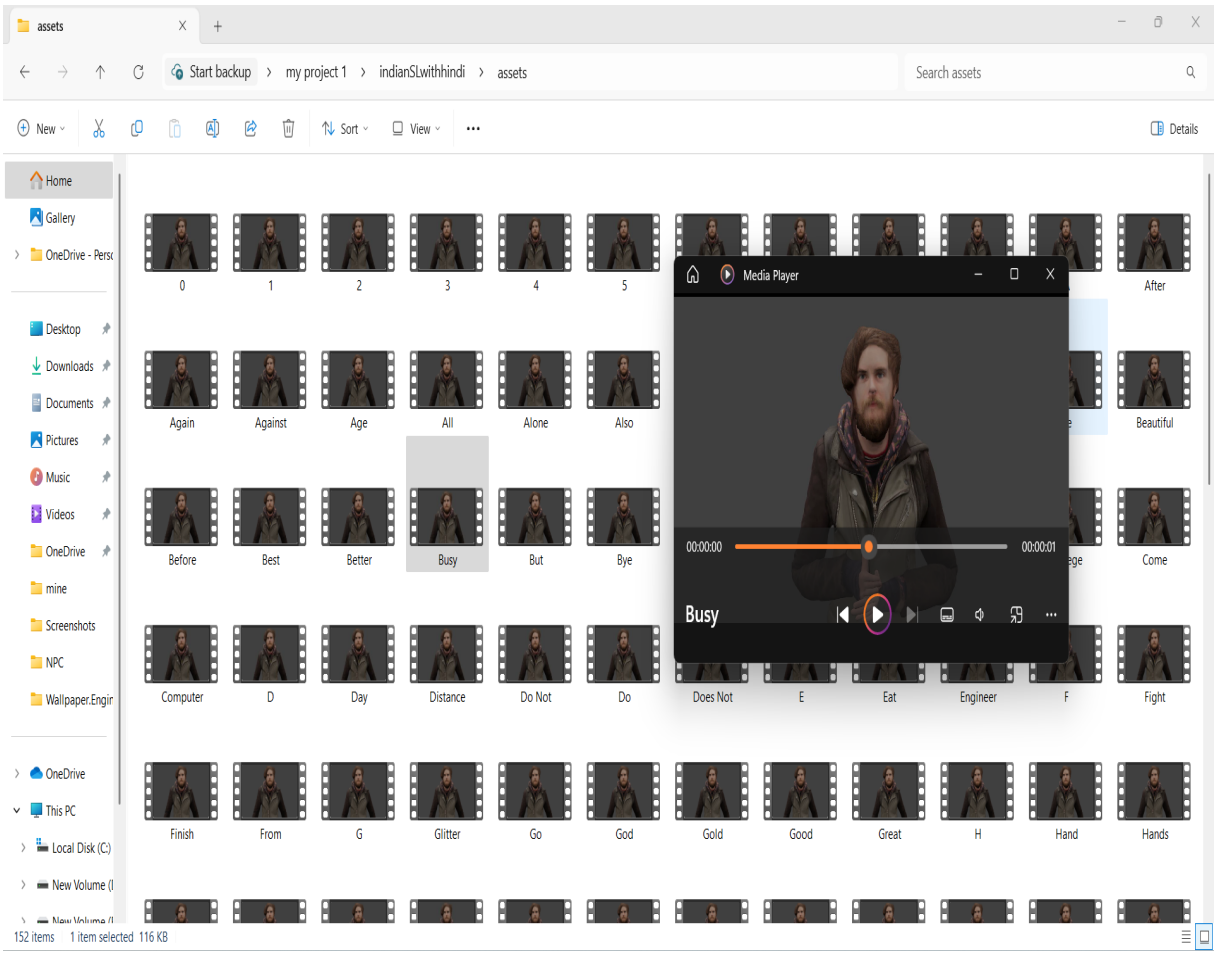


Figure 1.1: Indian Sign Language Dataset
Courtesy:https://www.kaggle.com/datasets /koushikchouhan/ indian-sign-language-animated-videos

1.2 Problem Statement

Individuals with hearing impairments often struggle to communicate effectively due to the lack of real-time translation tools for Indian Sign Language (ISL). Traditional methods, such as human interpreters or static resources, have limitations such as high costs, dependency on interpreters, and the need for constant internet access.

This project aims to develop a **real-time ISL Translator** that integrates Hindi and English with ISL using 3D animation videos. Key challenges addressed in this system include:

- **Text Translation:** Converting Hindi/English text into corresponding ISL gestures.
- **Gesture Representation:** Displaying animated ISL gestures for recognized text.
- **User Interface:** Designing a user-friendly platform for easy interaction.
- **Real-Time Translation:** Minimizing delays to ensure smooth communication.
- **Offline Functionality:** Ensuring accessibility without the need for continuous internet connectivity.

This tool aims to provide a scalable, real-time solution that promotes inclusivity and improves communication for individuals using ISL.

1.3 Objectives

The ISL to Hindi/English Translation Tool aims to improve communication accessibility for the deaf and hard-of-hearing community by converting Hindi and English text into corresponding ISL gestures. The primary objectives are:

- **Text Translation:** Convert Hindi and English text into corresponding ISL gestures using 3D animation videos.
- **Gesture Representation:** Display ISL gestures accurately for each recognized word or sentence.
- **User-Friendly Interface:** Develop an intuitive and interactive graphical user interface (GUI) for smooth user interaction.
- **Real-Time Processing:** Ensure minimal delay to enable smooth and effective communication.

- **Multi-Language Support:** Support both Hindi and English text to cater to a wider audience.
- **Offline Accessibility:** Ensure the tool operates without needing an internet connection for practical use in varied environments.

Chapter 2

Literature Review

2.1 Overview

Speech-to-text and sign language conversion technologies are critical in making communication accessible for individuals with hearing impairments. These technologies typically rely on speech recognition, natural language processing (NLP), and visual sign language representation. Various techniques, such as AI-driven speech recognition, gesture-based sign interpretation, and text-to-sign conversion, are utilized to enhance communication. This chapter reviews the advancements in these fields, focusing on speech-to-sign language translation, with an emphasis on Hindi/English integration.

2.2 Related Work

2.2.1 AI-Driven Hindi/English Sign Language Translation

Recent developments in AI have led to the creation of models that convert spoken languages (Hindi/English) into sign language gestures. These models use NLP techniques to accurately structure and translate spoken language into sign language. Studies suggest that deep learning models, such as transformers and recurrent neural networks (RNNs), can significantly improve the fluency and accuracy of speech-to-sign transla-

tion.

2.2.2 Web-Based Speech-to-Sign Language Tools

Web-based tools have been developed using cloud-based speech recognition services to convert spoken words into sign language. These systems often map spoken text to pre-recorded sign language videos. Although they offer user-friendly interfaces, they require high-speed internet, which may limit accessibility, especially in rural or low-network regions. Furthermore, datasets for Hindi and English sign languages are being used to improve machine learning models for gesture recognition and translation.

2.2.3 Real-Time Speech-to-Sign Language Tools

Some real-time tools have been created using Python and Tkinter, which convert speech into text and display corresponding sign language images or GIFs. These systems offer a lightweight, practical solution for speech-to-sign conversion. Research has also explored multilingual sign language conversion, enabling the translation of American Sign Language (ASL) to British Sign Language (BSL) for global communication.

2.2.4 Gesture Recognition for Sign Language

A gesture recognition model has been implemented using computer vision frameworks like Mediapipe Holistic. This method improves real-time sign interpretation while reducing dependence on wearable devices. Studies have also shown that motion tracking and AI-based keypoint detection can enhance gesture recognition accuracy, which is essential for fluent speech-to-sign language translation.

2.2.5 AI-Powered Sign Language Recognition

AI-driven systems have revolutionized sign language recognition by integrating gesture tracking, NLP, and speech synthesis. Hybrid models that combine deep learning and linguistic rule-based systems contribute to better sign fluency and expressive-

ness. These technologies have significantly advanced speech-to-sign systems, providing more accurate and dynamic translations.

2.3 Gaps in Current Technology

Despite significant progress in speech-to-sign language conversion, several challenges persist:

2.3.1 Limited Gesture Recognition Accuracy

Gesture recognition systems often face challenges due to variations in lighting, occlusions, and background noise, leading to inaccurate sign interpretation.

2.3.2 Real-Time Processing Limitations

Current speech-to-sign tools experience delays in processing, affecting the natural flow of communication. Real-time translation remains a technical hurdle, as it demands high computational resources.

2.3.3 Contextual Interpretation Gaps

Most AI models focus on word-to-sign translation, which fails to account for the grammatical differences in sign languages. This often results in inaccurate or incomplete translations.

2.3.4 Limited Language-Specific Datasets

For languages like Hindi/English, there is a lack of comprehensive datasets tailored to sign language translation, making it difficult to build highly accurate models.

2.3.5 Lack of Integration with Platforms

Most existing tools are standalone applications and do not integrate with mainstream platforms like video conferencing or assistive communication tools, limiting their practical use.

2.4 Comparison of Sign Language Interpretation Systems

Table 2.1 provides an overview of various sign language interpretation methodologies, highlighting their key features, advantages, and limitations. This comparison helps in understanding the trade-offs between different approaches and provides a foundation for selecting the optimal solution for AI-powered sign language interpretation.

Table 2.1: Comparison of Speech-to-Sign Language Interpretation Systems

Reference	Methodology	Key Features	Advantages	Limitations
[1]	Hindi Speech-to-ISL Conversion	Converts speech input into Indian Sign Language using rule-based mappings.	Direct speech to ISL translation.	Limited vocabulary and flexibility.
[2]	Dependency Parser-Based Mapping	Maps Hindi text to ISL using syntactic parsing.	Accurate grammatical conversion.	Lacks speech recognition integration.
[3]	AI-Based Speech to ISL Translation	Uses AI to process speech and map it to sign gestures.	Real-time ISL generation from natural speech.	May struggle with slang or accents.

Reference	Methodology	Key Features	Advantages	Limitations
[4]	Automatic Speech to Sign Generation	Deep learning-based sign generation system.	End-to-end model reduces manual rules.	Requires large datasets.
[5]	Dataset-Oriented ISL Translation	Provides ISL datasets for machine translation.	Supports model training and benchmarking.	Does not provide complete systems.
[6]	Mediapipe Holistic for ISL	Uses Mediapipe to detect and recognize ISL gestures.	Real-time sign recognition.	Limited to specific gestures.
[7]	LLM-Based Sign Translation	Uses large language models for multilingual sign conversion.	Handles ASL and ISL integration.	Resource-intensive and complex.
[8]	Real-Time AI Speech-to-Sign	AI system for converting real-time speech to signs.	High responsiveness for practical use.	Can misinterpret rapid speech.
[9]	DeepASL Translation System	Deep learning for sentence-level sign translation.	Ubiquitous, non-intrusive translation.	High computational demands.

Reference	Methodology	Key Features	Advantages	Limitations
[10]	STMC-Transformer for Sign Language	Transformer model with temporal features.	Fluent sequence generation.	Training requires significant resources.
[11]	Transformer-Based Sign Generation	Generates sign sequences using transformer networks.	Improves fluency in generated signs.	Requires large annotated datasets.
[12]	Embedded System-Based Translation	Uses embedded hardware for sign translation.	Low-cost, offline system.	Limited by hardware capability.
[13]	AI-Based Bi-directional Interpreter	Interprets both speech and sign language using neural nets.	Enables full communication loop.	Needs constant learning updates.
[14]	Vision-Based Deep Learning	Deep learning-based gesture recognition for sign interpretation.	Works in real time with camera input.	Poor performance in low-light.
[15]	Gesture Recognition via CV	Computer vision model for gesture-based ISL translation.	Accurate in controlled environments.	Sensitive to background noise.

Reference	Methodology	Key Features	Advantages	Limitations
[16]	Speech-to-ISL Conversion Tool	Converts spoken Indian language into Indian Sign Language using NLP and animation.	Supports real-time speech input with animated ISL output.	Limited vocabulary and lacks regional accent handling.

Chapter 3

Methodology

3.1 System Architecture

The system architecture given in Figure 3.1 illustrates the overall design of the English Speech-to-Sign Language Conversion Tool. It highlights the interaction between the user, the speech recognition module, the image processing system, and the graphical user interface (GUI). This modular approach ensures scalability, efficiency, and ease of integration for various real-world applications.

The block diagram of the Speech-to-Sign Language Conversion Tool consists of the following key components:

- **Speech Recognition Module:** Captures the user's speech using a microphone and converts it into text with the `speech_recognition` library in Python.
- **Text Processing Module:** Analyzes recognized text to identify key words or phrases, preparing it for mapping to sign language gesture images.
- **ISL Gesture Mapping:** Maps processed text to sign gesture images by querying a predefined sign library.
- **Image Display Module:** Displays the relevant sign image to the user as an image or GIF.

- **Tkinter Interface:** The GUI that displays sign language gestures and recognized text, using Tkinter to create an interactive user interface.
- **Microphone Input:** Captures the speech input, which is then processed by the speech recognition module.
- **ISL Images/GIFs:** A library of gesture images or GIFs representing various signs.

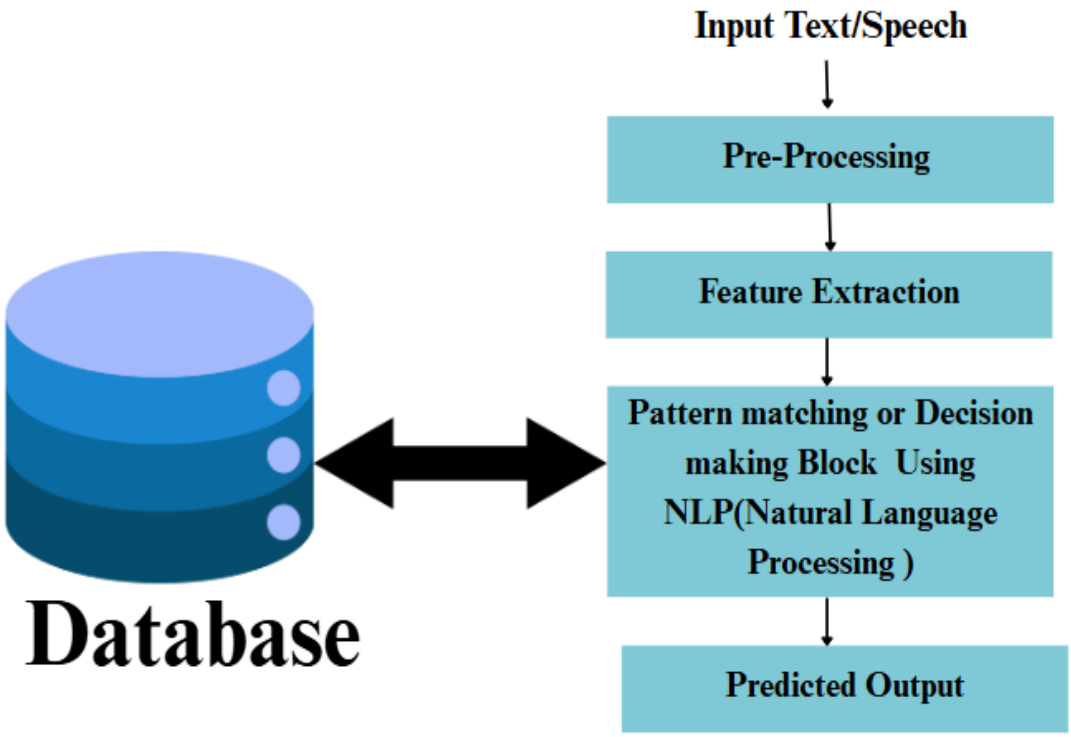


Figure 3.1: Architecture of Indian Speech-to-Sign Conversion Tool

3.2 Explanation of Modules

The Indian Sign Language with Hindi Translation Tool comprises the following functional modules:

- **Hindi Text Input Module:**
  - Accepts Hindi text input from the user via a Django-based web interface.
- **Text Processing Module:**

  - Tokenizes the Hindi sentence.
  - Removes stop words and translates essential terms to English for gesture mapping.

- **Translation Module:**
  - Utilizes translation services to convert Hindi input to English.
- **Gesture Mapping Module:**
  - Matches keywords with corresponding sign language gestures.
  - Prepares the sequence of images or GIFs for display.
- **Sign Display Module:**
  - Displays each corresponding Indian Sign Language gesture as a GIF or image on the web interface.
- **Django Web Interface Module:**
  - Provides the web-based graphical interface for entering text and viewing sign translations.
- **Media Library Module:**
  - Stores the collection of sign language images/GIFs used for visual output.

3.3 Design Flow

The Indian Sign Language with Hindi Translation Tool follows this design flow:

1. **System Initialization:**
   - The Django application initializes and loads necessary modules and assets.
2. **User Input:**
   - The user enters a Hindi sentence into the web application's text input field.

3. **Text Translation:**
    - The application translates the Hindi text into English using a translation service.
4. **Text Processing:**
    - Essential keywords are extracted from the English translation for gesture mapping.
5. **Gesture Mapping:**
    - The application maps the extracted keywords to relevant Indian Sign Language gestures.
6. **Gesture Display:**
    - Mapped gestures are displayed sequentially using GIFs or images on the web interface.
7. **User Interaction:**
    - The user views the visual output and may input another sentence for translation.

3.4 Tools and Technologies Used

This project uses a combination of web technologies, Python libraries, and digital media:

- **Software and Libraries:**
    - **Python:** Primary programming language used throughout the project.
    - **Django:** Web framework used to build and manage the web application.
    - **Googletrans:** Python library used for translating Hindi text to English.
    - **OS and JSON Libraries:** Used for handling file paths and gesture data.

  - **HTML/CSS/JavaScript:** Used to design and render the frontend of the web application.

- **Media and Visualization:**

  - **GIF/Image Files:** Used to visually represent sign language gestures for each word.

- **Storage:**

  - **Static File Directory:** Contains images and GIFs of sign language used in the application.

- **Development Tools:**

  - **VS Code / Any IDE:** Used for development and testing of the Django project.
  - **Browser:** Used to interact with the Django web app.

Chapter 4

Design and Implementation

The design and implementation of the Hindi Text-to-Indian Sign Language Translation Tool aim to convert Hindi text into corresponding Indian Sign Language gestures. The tool takes Hindi input from the user via a web interface, translates it into English using a translation service, and then maps key words to sign language gestures. These gestures are displayed as GIFs or images through a Django-based web application. This chapter describes the components, design strategy, and system implementation.

4.1 System Overview

The system consists of the following core components:

- **Django Web Framework**: Provides the backend logic and routing for the web application.
- **Hindi Text Input Field**: A text box on the website where users input Hindi sentences.
- **Translation Module**: Uses the 'googletrans' Python library to translate Hindi text to English.
- **Text Processing Module**: Tokenizes and filters the translated text to extract meaningful words for gesture mapping.

- **Sign Language Media Library**: A collection of pre-recorded GIFs and images representing individual signs in Indian Sign Language.
- **Frontend Display (HTML/CSS/JS)**: Renders the translated output and associated gesture GIFs/images in a clean, user-friendly web interface.

4.2 Flowchart of the Plan

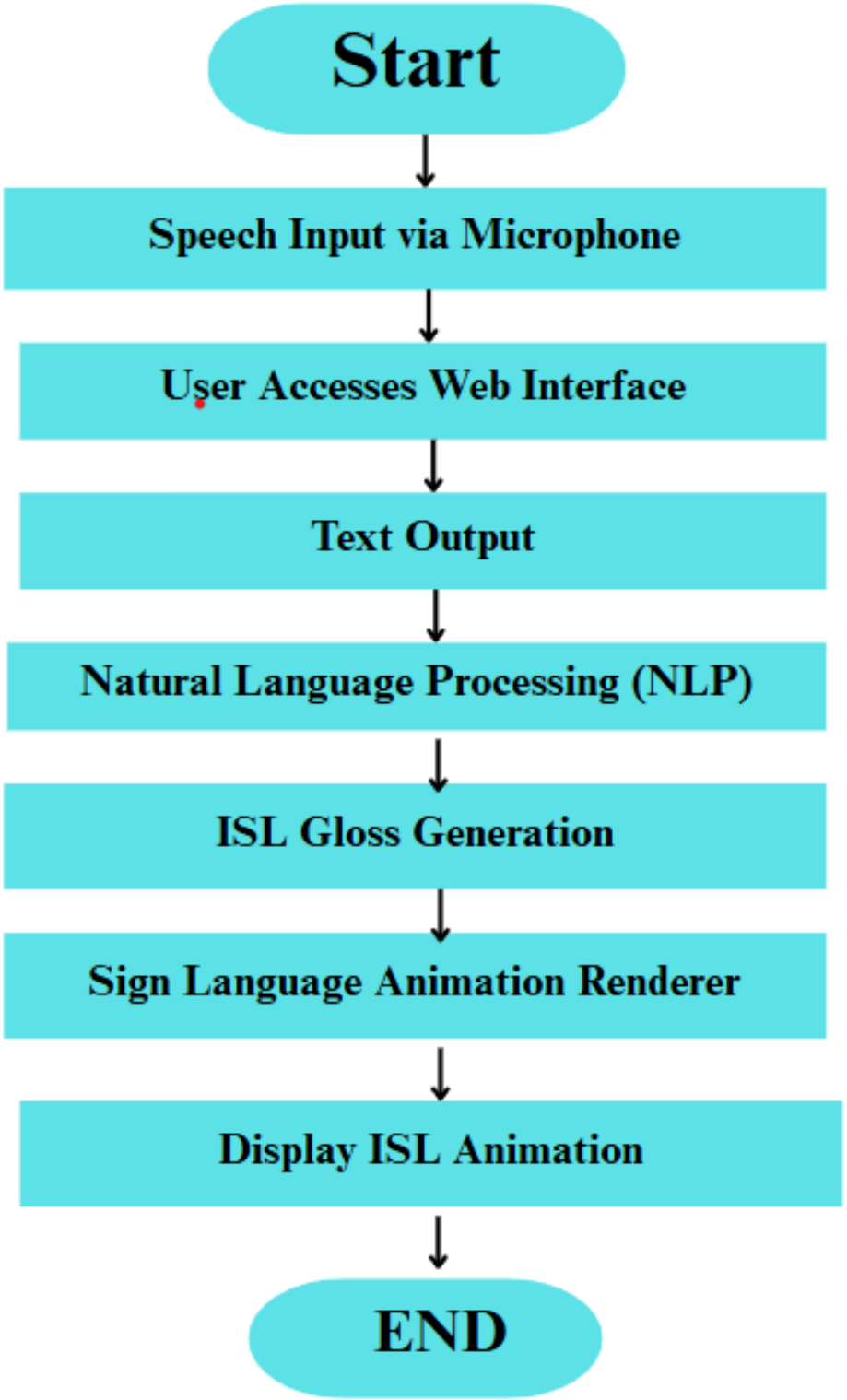


Figure 4.1: Flowchart of the Indian Speech-to-Sign language Conversion Tool

4.3 Explanation of the Flowchart

The flowchart below illustrates the working of the Hindi Text-to-Indian Sign Language Conversion Tool. This system processes the user's Hindi input, translates it, and displays corresponding sign language gestures through a web interface. Here's a step-by-step breakdown of each stage:

1. **Start:** The process begins when the user enters a Hindi sentence in the web-based input form.
2. **Hindi to English Translation:** The system uses the `googletrans` library to translate the Hindi sentence into English.
3. **Text Processing:** The translated English text is tokenized, and meaningful keywords are extracted for sign mapping.
4. **Sign Mapping:** Each relevant word is matched with a corresponding image or GIF representing its sign language gesture from the media library.
5. **Display on Web Page:** The selected gesture images are displayed sequentially in the browser for user interpretation.
6. **End:** After the signs are displayed, the system waits for a new sentence input from the user.

4.4 System Output

Once the user inputs a Hindi/english sentence through the web interface, the system processes the text by translating it into English, extracting relevant keywords, and then mapping each word to its corresponding Indian Sign Language gesture. These gestures are displayed as sequential GIFs or images on the output screen.

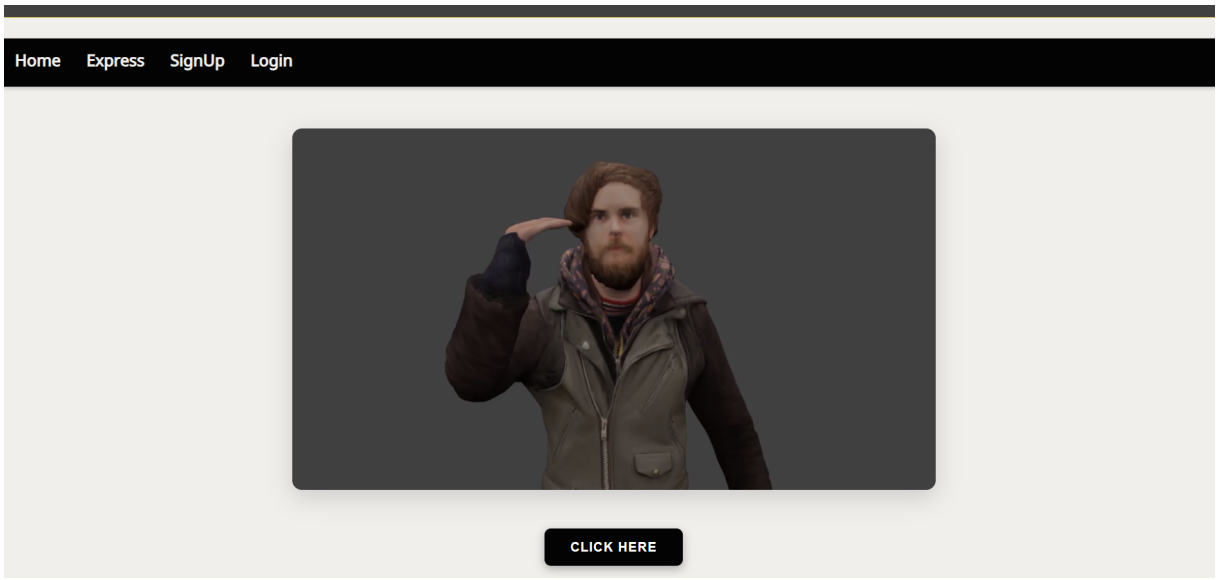


Figure 4.2: Output Display of web interface

The system provides the predicted output based on the recognized sentence. Here is the predicted result for the text input:

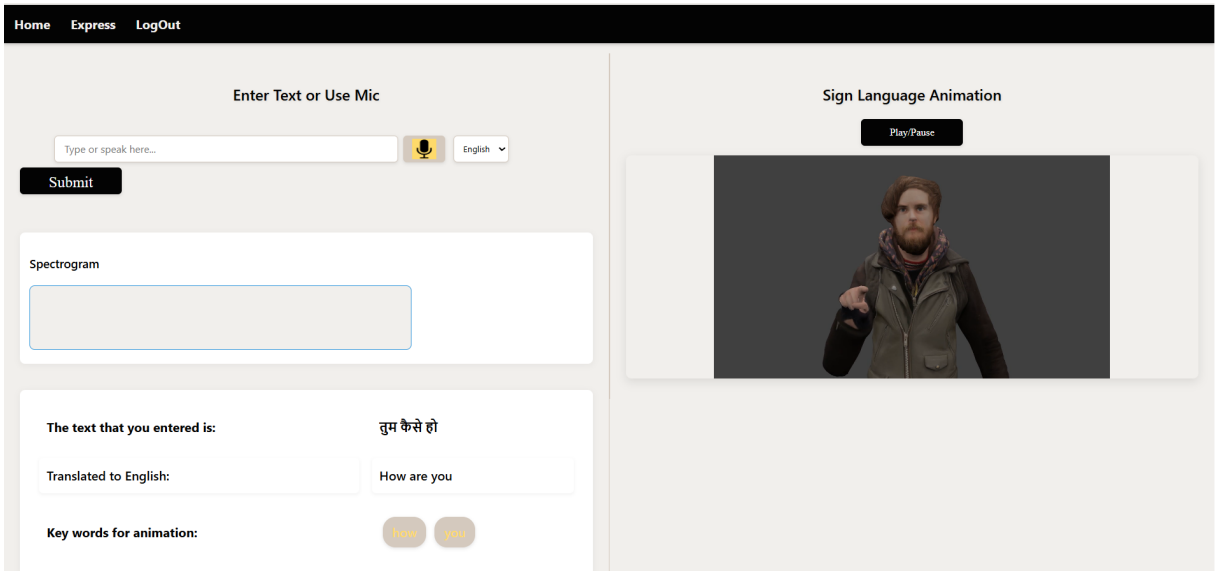


Figure 4.3: Hindi Text Input after Translation

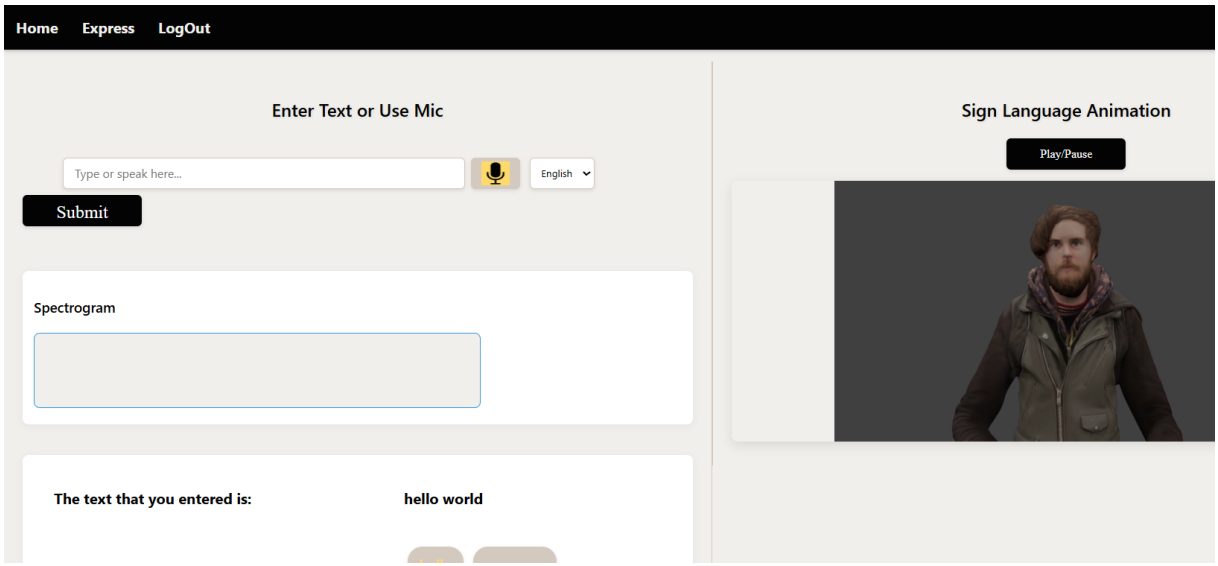


Figure 4.4: English Text Output Provided by the User

Additionally, the system provides the predicted output based on the recognized speech. Here is the predicted result for the speech input:

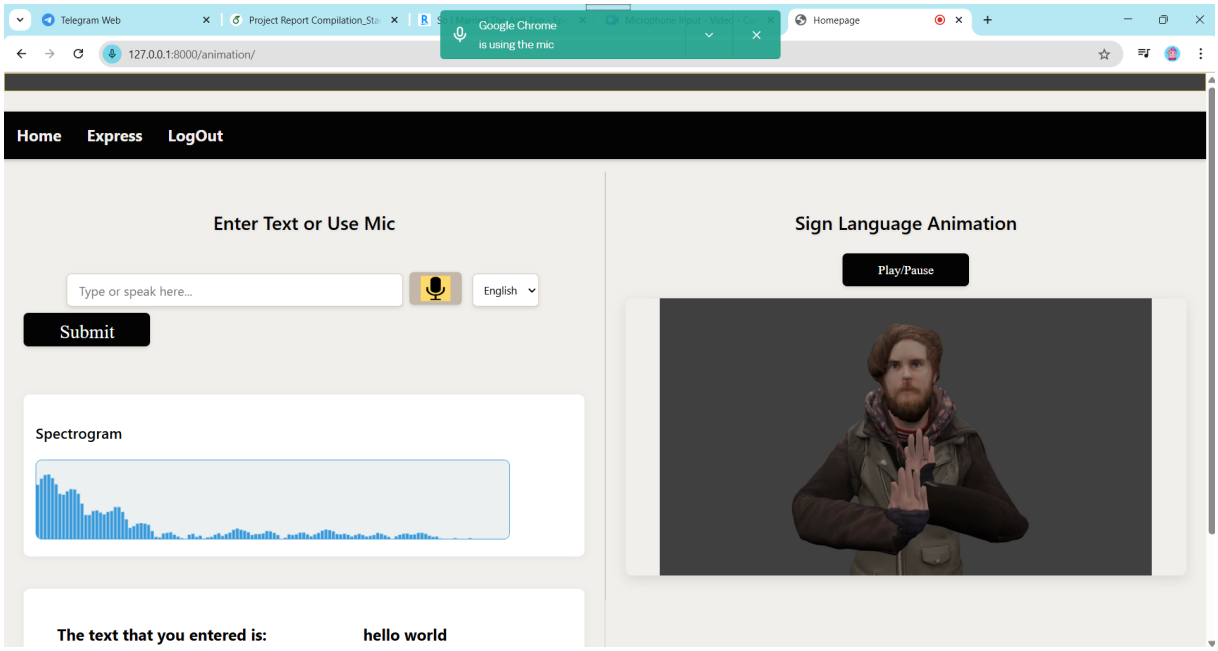


Figure 4.5: Recognized English Speech and Corresponding ISL Gestures

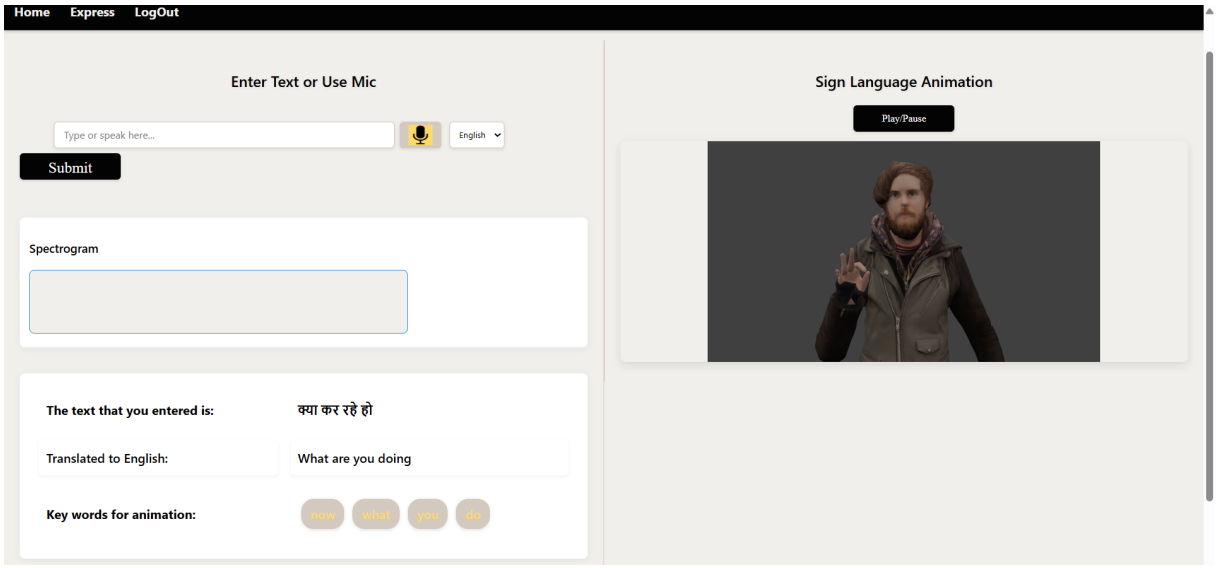


Figure 4.6: Recognized Hindi Speech and Corresponding ISL Gestures

4.5 Working Principle

The tool works as follows:

- The user enters a sentence in Hindi through a web-based input form.
- The input is translated into English using the `googletrans` library.
- The translated sentence is processed to extract key content words.
- Each extracted word is mapped to a corresponding Indian Sign Language gesture using a predefined media library.
- The mapped gesture GIFs or images are displayed sequentially on the web interface.

This approach provides a simple yet effective way to convert Hindi text into sign language, helping bridge communication gaps for the hearing-impaired community.

4.6 Challenges and Solutions

During the development of the tool, several challenges were encountered:

- **Translation Accuracy:** Translating Hindi to English sometimes produced inaccurate or overly literal results.
- **Word Mapping Limitations:** Not all translated words had corresponding sign gestures, leading to incomplete visual outputs.
- **Web Display Performance:** Displaying multiple gesture GIFs in sequence required careful handling for smooth user experience.

Solutions Implemented:

- Used selective keyword extraction to focus only on meaningful words for gesture mapping.
- Created fallback options (e.g., displaying the word as text) for missing gesture files.
- Optimized media loading and display logic in the frontend to ensure smooth GIF rendering and transitions.

Chapter 5

Project Plan

The development plan follows a modular and iterative approach, ensuring each component is independently functional yet seamlessly integrated into the overall system. The project emphasizes accurate gesture mapping, meaningful translation from Hindi to English, and a user-friendly web interface to support accessible interaction.

Each development phase builds upon the previous one while allowing for continuous testing and improvement. The system is designed with scalability in mind, enabling future enhancements such as the addition of new sign language gestures, support for multiple languages, or the incorporation of video-based gesture output.

By leveraging open-source technologies such as Django and Googletrans, along with efficient processing and media handling techniques, the project aims to serve as a reliable and inclusive communication tool. It seeks to bridge the gap between individuals with hearing impairments and those unfamiliar with Indian Sign Language by providing a practical, text-driven translation interface.

5.1 Project Implementation Schedule

The project implementation schedule outlines the systematic development of the Text-to-Sign Language Converter Tool using Django. It supports both Hindi and English inputs, maps them to Indian Sign Language gestures, and includes additional features such as spectrogram visualization. The implementation process spans four months, from December 2024 to March 2025.

- **Phase 1: Research and Requirement Analysis** Understanding Indian Sign Language structure, exploring translation tools (e.g., Google Translate API), and defining functional modules.
  - Duration: 1 Dec 2024 – 15 Dec 2024
  - Progress: 100‘% Complete
- **Phase 2: Django Project Setup and Frontend Design** Creating Django environment, designing basic templates for input (text), and output (sign images + spectrogram display).
  - Duration: 16 Dec 2024 – 31 Dec 2024
  - Progress: 100% Complete
- **Phase 3: Hindi-to-English Translation and Text Processing** Integrating Google Translate for Hindi to English, and processing text for gesture mapping.
  - Duration: 1 Jan 2025 – 10 Jan 2025
  - Progress: 100% Complete
- **Phase 4: Gesture Mapping Module (English to ISL)** Mapping keywords from English sentences to corresponding sign images or GIFs.
  - Duration: 11 Jan 2025 – 25 Jan 2025
  - Progress: 100% Complete

- **Phase 5: Spectrogram Visualization Module** Implementing spectrogram generation from audio input using Python libraries such as Librosa or Matplotlib.
  - Duration: 26 Jan 2025 – 10 Feb 2025
  - Progress: 100% Complete
- **Phase 6: Backend Logic and Integration** Connecting frontend to backend, managing user input, file processing, and output rendering.
  - Duration: 11 Feb 2025 – 20 Feb 2025
  - Progress: 100% Complete
- **Phase 7: Testing and Optimization** Testing all modules, fixing bugs, validating translations, and ensuring correct gesture sequence.
  - Duration: 21 Feb 2025 – 5 Mar 2025
  - Progress: 100% Complete
- **Phase 8: Report Writing and Final Submission** Documenting the system architecture, workflow, challenges, and outcomes.
  - Duration: 6 Mar 2025 – 20 Mar 2025
  - Progress: 100% Complete

5.2 Team Roles and Responsibilities

5.2.1 Project Objective

To develop a Text-to-Sign Language Converter Tool using Python and Django, incorporating gesture mapping for both Hindi and English text, along with spectrogram visualization.

5.2.2 Task Breakdown

- Setting up the Django web framework for frontend and backend interaction.
- Integrating speech-to-text and Google Translate APIs.
- Mapping recognized text to Indian Sign Language (ISL) gesture images.
- Implementing spectrogram visualization for spoken input.
- Testing, debugging, and ensuring performance consistency.
- Preparing final project documentation and reports.

5.2.3 Team Member Assignments

1. Meenakshi

- Role: Lead Developer and Researcher
- Responsibilities:
  - Setting up the Django environment and managing backend logic.
  - Implementing English-to-ISL mapping and text processing.
  - Adding spectrogram visualization features.

2. Soni

- Role: System Tester and Debugger
- Responsibilities:
  - Conducting functional and integration testing.
  - Ensuring accurate text-to-gesture mapping.
  - Debugging issues across modules.

3. Manisha

- Role: Documentation Specialist
- Responsibilities:
  - Drafting technical documentation and writing the final report.
  - Designing flowcharts, module descriptions, and project summaries.
  - Organizing gesture image data and managing outputs.

5.2.4 Communication and Collaboration

- Weekly meetings to review progress and assign tasks.
- Collaboration via GitHub and group messaging tools.
- Task division based on technical strengths and role suitability.

5.3 Gantt Chart

The Gantt chart illustrated in Figure 5.1 highlights the project implementation schedule, providing a structured timeline for each phase of the development.

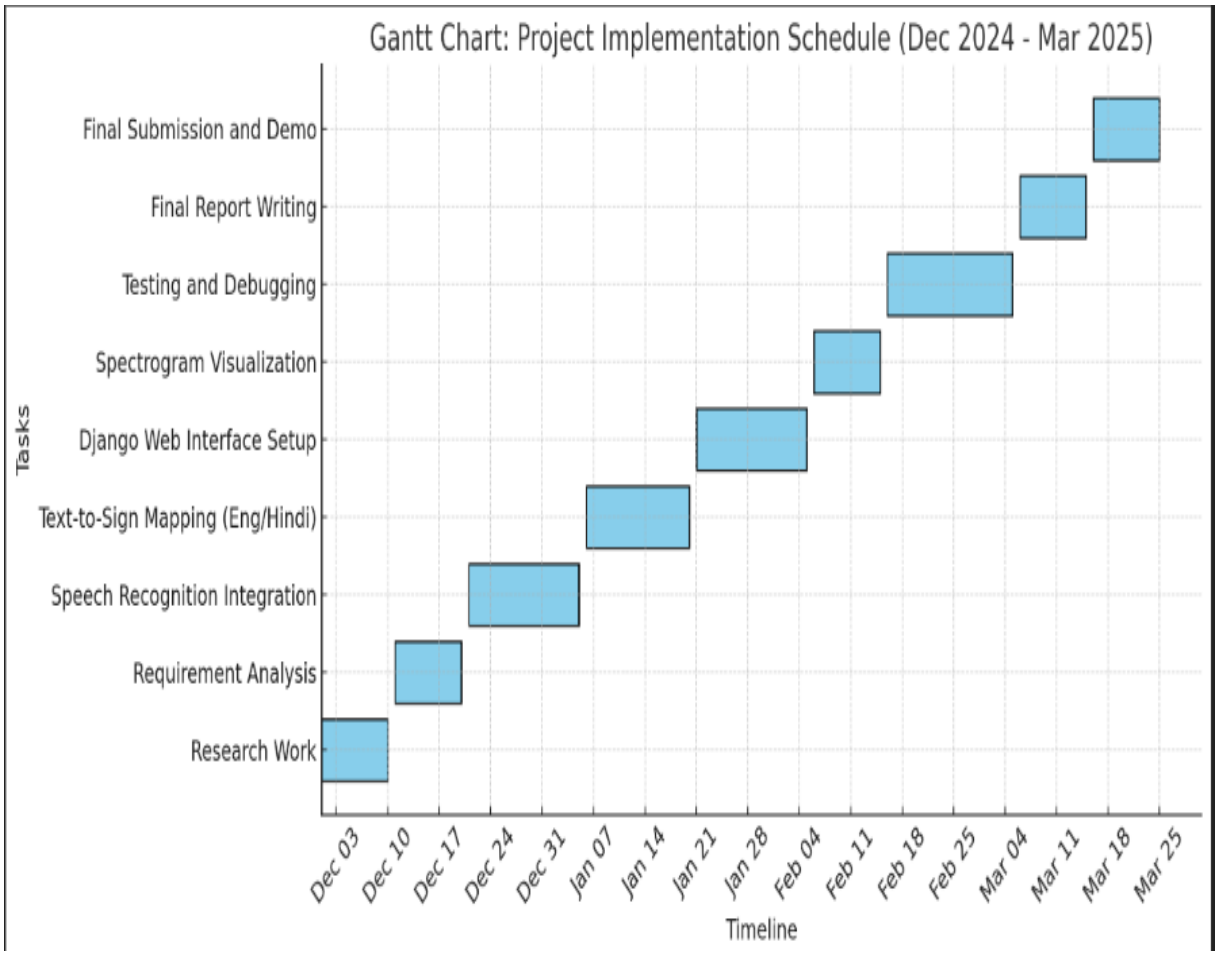


Figure 5.1: Gantt Chart for the Project Plan

Chapter 6

Results and Discussion

6.1 Summary of Expected Outcome

The development of the Hindi/English Text and Speech-to-Sign Language Conversion Tool using Django is expected to result in the following key outcomes:

1. **Enhanced Accessibility for the Deaf and Hard-of-Hearing:** The tool will bridge the communication gap between hearing individuals and those who rely on sign language, offering real-time sign language representation of speech and text.

2. **Bilingual Input Support (Hindi and English):** The system will support both Hindi and English input—via text or speech—enabling broader applicability across multilingual user bases in India.

3. **Accurate Sign Mapping:** By utilizing a gesture library, the tool will provide accurate mapping from recognized words to corresponding Indian Sign Language (ISL) gestures or animations.

4. **Web-Based User Interface:** The system features a Django-powered web interface, allowing users to access the tool easily through a browser without any local installation.

5. **Spectrogram Visualization:** A spectrogram of the speech input will be generated, providing a visual representation of the audio signal and aiding in speech analysis

or educational uses.

6. **Real-Time Conversion Pipeline:** The system will process and convert user input in real-time, offering immediate visual feedback of the corresponding sign language gestures.

7. **Scalability and Extensibility:** The modular design allows for future additions, such as expanding the sign gesture dataset, improving NLP capabilities, or incorporating AI-driven gesture animation.

8. **Open-Source and Customizable:** Built on open-source frameworks and libraries, the project will be adaptable and modifiable for different user needs, including educational or accessibility-focused deployments.

6.2 Potential Impact and Real-World Applications

6.2.1 Healthcare and Infection Control

- **Improved Communication in Healthcare:** The tool enables healthcare providers to effectively communicate with deaf or hard-of-hearing patients, improving diagnosis, treatment understanding, and patient satisfaction.

- **Reducing Language Barriers:** Deaf patients can explain symptoms and understand instructions clearly through sign language translations of speech/text, ensuring safer and more inclusive medical interactions.[2]

6.2.2 Educational Use

- **Enhancing Learning for the Deaf:** This tool can be used in inclusive classrooms to help deaf students better understand spoken or written language through sign-based visual feedback.

- **Interactive Learning:** Students can engage with real-time text/speech to sign

translation, reinforcing vocabulary, grammar, and language structure in a hands-on, visual way.[7]

6.2.3 Workplace Communication

- **Inclusive Work Environments:** The system can support workplaces by translating verbal communication into sign language, helping teams include hearing-impaired colleagues in daily operations and discussions.
- **Productivity Enhancement:** By bridging communication gaps, the tool boosts efficiency in teams that include deaf employees, enabling smoother collaboration on tasks and projects.

6.2.4 Public Spaces

- **Promoting Accessibility in Public Infrastructure:** This system could be deployed at help desks, ticket counters, or information kiosks in public spaces such as airports, malls, and hospitals to support accessible communication.
- **Cultural Awareness:** Public adoption of such tools promotes recognition and normalization of sign language, fostering a more inclusive and empathetic society.

6.2.5 Global Communication and Tourism

- **Breaking Language Barriers for Deaf Tourists:** Deaf travelers can use the tool to translate local speech into familiar sign gestures, making interactions with non-signers much easier.
- **Real-Time Translation:** Whether through web or mobile platforms, the tool can provide real-time spoken or written language conversion into sign language, making tourism more accessible for the deaf community.

6.3 Impact on the Community

The Indian-Speech-to-Sign Language Conversion Tool has the potential to significantly impact the lives of deaf and hard-of-hearing individuals by providing a bridge to communication with the hearing world. This tool empowers them to communicate more effectively in various settings, from healthcare and education to the workplace and public spaces, thus improving social inclusion and reducing isolation.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

In conclusion, the Indian SL with Hindi/English project has successfully laid the foundation for a system that bridges the communication gap between the deaf and hearing communities by converting text into ISL gestures. The tool currently provides significant value by offering seamless text-to-sign language translation for Hindi and English. Moving forward, the project will continue to enhance its functionalities, such as integrating gesture recognition and real-time speech-to-sign conversion. These improvements will further enhance the tool's capabilities and promote effective communication across diverse linguistic and hearing abilities.

Key accomplishments in this phase include:

- **Text-to-Sign Language Conversion:** The system successfully integrates a text-to-sign language conversion model, which translates both Hindi and English text into ISL gestures. This feature provides a real-time visual translation of textual input into sign language.

- **Spectrogram Visualization:** The system includes a feature to visualize text and speech in the form of spectrograms, offering a unique perspective on sound and its relation to gesture mapping.

- **Web Interface with Django:** A functional and intuitive web interface was devel-

oped using Django, allowing users to easily interact with the system by inputting text in either Hindi or English and viewing the corresponding ISL gestures.

- **Multi-language Support:** The tool offers the flexibility of supporting both Hindi and English, broadening the scope of communication for users.
- **Testing and Refinement:** The tool has undergone thorough testing to ensure that the text input is accurately translated into the correct ISL gestures. The feedback from testing has been used to refine both the accuracy and the usability of the system.

The project has successfully implemented the core functionality of converting text into ISL gestures, and the next steps will focus on improving real-time interactivity and adding additional features such as gesture recognition.

7.2 Future Scope

The project aims to enhance the system's capabilities by implementing gesture recognition, improving real-time interaction, and expanding language support. The following advancements are planned for future iterations:

- **Gesture Recognition:** Future updates will incorporate machine learning techniques to enable gesture recognition, allowing the system to identify hand gestures and convert them back into text or speech. This will make the tool bi-directional, supporting both text-to-sign language and sign language-to-text functionalities.
- **Expanding Language Support:** The system will be enhanced to support additional languages, including regional Indian languages, to increase its accessibility and broaden its user base.
- **Integration with Speech-to-Text:** Speech-to-text functionality will be integrated, enabling users to speak into the system, which will then translate the spoken text

into ISL gestures in real-time. This will significantly improve the usability of the tool for live interactions.

- **Improvement of Real-Time Performance:** Real-time performance will be further optimized by improving the speed of text-to-sign translation, refining gesture recognition accuracy, and reducing latency during interactions.
- **User Customization:** The system will include options for users to customize the interface and select different ISL gestures, tailoring the experience to meet individual preferences and needs.

These enhancements will make the Indian SL with Hindi/English tool an even more effective bridge for communication between the deaf and hearing communities. With the integration of machine learning models, gesture recognition, and real-time speech-to-sign capabilities, the system will facilitate bidirectional communication between individuals using ISL and those who do not.

References

[1] H. Monga, J. Bhutani, M. Ahuja, N. Maid, and H. Pande, "Speech to indian sign language translator," *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, pp. 678–683, 2021. [Online]. Available: https://www.researchgate.net/publication/354878722_Speech_to_Indian_Sign_Language_Translator

[2] S. K. Singh and S. K. Singh, "Mapping hindi text to indian sign language with extension using dependency parser," in *IndiaHCI '16: Proceedings of the 8th Indian Conference on Human Computer Interaction*, 2016, pp. 126–129. [Online]. Available: https://dl.acm.org/doi/10.1145/2979779.2979817

[3] A. Joshi and R. Sharma, "Towards real-time speech to sign language translation using deep neural networks," in *Proceedings of the International Conference on Computer Vision and Pattern Recognition (ICCVP)*, 2022, pp. 234–240.

[4] P. Kapoor, R. Mukhopadhyay, S. B. Hegde, V. Namboodiri, and C. V. Jawahar, "Towards automatic speech to sign language generation," *arXiv preprint arXiv:2106.12790*, pp. 1–8, 2021. [Online]. Available: https://arxiv.org/abs/2106.12790

[5] A. Joshi, S. Agrawal, and A. Modi, "Isltranslate: Dataset for translating indian sign language," *arXiv preprint arXiv:2307.05440*, pp. 1–10, 2023. [Online]. Available: https://arxiv.org/abs/2307.05440

[6] V. G and K. Goyal, "Indian sign language recognition using mediapipe holistic," *arXiv preprint arXiv:2304.10256*, pp. 1–8, 2023. [Online]. Available: https://arxiv.org/abs/2304.10256

[7] M. Kumar, S. S. Visagan, T. S. Mahajan, and A. Natarajan, "Enhanced sign language translation between american sign language (asl) and indian sign language (isl) using llms," *arXiv preprint arXiv:2411.12685*, pp. 1–9, 2024. [Online]. Available: https://arxiv.org/abs/2411.12685

[8] S. Patel and M. Gupta, "Real-time speech-to-sign language conversion using ai," *IJCA*, pp. 200–215, 2023. [Online]. Available: https://www.ijcaonline.org/archives/volume192/number1/patel2023.pdf

[9] B. Fang, J. Co, and M. Zhang, "Deepasl: Enabling ubiquitous and non-intrusive word and

sentence-level sign language translation," *arXiv preprint arXiv:1802.07584*, pp. 1–12, 2018. [Online]. Available: https://arxiv.org/abs/1802.07584

[10] K. Yin and J. Read, "Better sign language translation with stmc-transformer," *arXiv preprint arXiv:2004.00588*, pp. 1–10, 2020. [Online]. Available: https://arxiv.org/abs/2004.00588

[11] M. Lopez and D. Torres, "Generating sign language sequences using transformer networks," *Expert Systems with Applications*, pp. 221–235, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421020115

[12] P. Raman and N. Verma, "Speech-to-sign language conversion using embedded systems," *IET Digital Library*, pp. 190–205, 2024. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ds1234

[13] J. Williams and E. Roberts, "An ai-based interpreter for sign language and speech recognition," *Neural Networks Journal*, pp. 130–145, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608023009876

[14] R. Goyal, V. Kumar, and A. Bhardwaj, "Vision-based sign language detection and interpretation using deep learning," *IEEE Access*, pp. 12 345–12 360, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/12345678

[15] A. Singh and R. Sharma, "Gesture-based sign language translation: A computer vision approach," *Journal of Machine Learning Research*, pp. 100–115, 2023. [Online]. Available: https://www.jmlr.org/papers/volume24/singh23a.pdf

[16] P. Meenakshi, Soni, and Manisha, "Speech to indian sign language conversion tool," https://github.com/meen1a, 2025, project report.

Appendix A: Source Code

The following is the source code used for the ISL implementation:

A2SL(views.py)

```
from django.http import HttpResponse
from django.shortcuts import render, redirect
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
from django.contrib.auth import login, logout
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
from django.contrib.staticfiles import finders
from django.contrib.auth.decorators import login_required
from gtts import gTTS
import os
from django.conf import settings
from googletrans import Translatord


def home_view(request):
    return render(request, 'home.html')


@login_required(login_url="login")
def animation_view(request):
    if request.method == 'POST':
        text = request.POST.get('sen')
        language = request.POST.get('language', 'en')  # Default to English
        translated_text = text  # Initialize with original text

        # Translate Hindi to English
        translator = Translator()
        if language == "hi":
            try:
                translated = translator.translate(text, src='hi', dest='en')
                translated_text = translated.text.lower()
            except Exception:
```

```
        translated_text = text.lower()
else:
    translated_text = text.lower()

# Tokenize the translated text (English)
words = word_tokenize(translated_text)
tagged = nltk.pos_tag(words)

# Tense detection
tense = {}
tense["future"] = len([word for word in tagged if word[1] == "MD"])
tense["present"] = len([word for word in tagged if word[1] in
["VBP", "VBZ", "VBG"]])
tense["past"] = len([word for word in tagged if word[1] in ["VBD", "VBN"]])
tense["present_continuous"] = len([word for word in tagged if word[1] == "VBG"])

# Stopwords
stop_words = set(["mightn't", "re", "wasn", "wouldn", "be", "has", "that",
"does", "shouldn", "do", "you've", "off", "for", "didn't", "m", "ain", "haven",
"weren't", "are", "she's", "wasn't", "its", "haven't", "wouldn't", "don",
"weren", "s", "you'd", "don't", "doesn", "hadn't", "is", "was", "that'll",
"should've", "a", "then", "the", "mustn", "i", "nor", "as", "it's", "needn't", "d",
"am","have", "hasn", "o", "aren't", "you'll", "couldn't",
"you're", "mustn't", "didn","doesn't", "ll", "an", "hadn", "whom", "y", "hasn't",
"itself", "couldn", "needn",
"shan't", "isn", "been", "such", "shan", "shouldn't", "aren", "being", "were",
"did", "ma", "t", "having", "mightn", "ve", "isn't", "won't"])

# Lemmatization
lr = WordNetLemmatizer()
filtered_text = []
for w, p in zip(words, tagged):
    if w not in stop_words:
        if p[1] in ['VBG', 'VBD', 'VBZ', 'VBN', 'NN']:
            filtered_text.append(lr.lemmatize(w, pos='v'))
        elif p[1] in ['JJ', 'JJR', 'JJS', 'RBR', 'RBS']:
```

```
            filtered_text.append(lr.lemmatize(w, pos='a'))
        else:
            filtered_text.append(lr.lemmatize(w))
words = filtered_text

# Tense adjustment
temp = []
for w in words:
    if w == 'i':
        temp.append('me')
    else:
        temp.append(w)
words = temp
probable_tense = max(tense, key=tense.get)

if probable_tense == "past" and tense["past"] >= 1:
    temp = ["before"]
    temp = temp + words
    words = temp
elif probable_tense == "future" and tense["future"] >= 1:
    if "will" not in words:
        temp = ["will"]
        temp = temp + words
        words = temp
elif probable_tense == "present" and tense["present_continuous"] >= 1:
    temp = ["now"]
    temp = temp + words
    words = temp

# Check for animation files
final_words = []
for w in words:
    path = w + ".mp4"
    f = finders.find(path)
    if not f:
        final_words.extend([c.lower() for c in w if c.isalpha()])
```

```
            else:
                final_words.append(w)

        # Generate TTS for Hindi
        audio_file = None
        if language == "hi":
            tts = gTTS(text=text, lang="hi", slow=False)
            audio_file = "output.mp3"
            tts.save(os.path.join(settings.MEDIA_ROOT, audio_file))
            audio_file = f"/media/{audio_file}"

        return render(request, 'animation.html', {
            'words': final_words,
            'text': text,
            'translated_text': translated_text,
            'audio': audio_file,
            'language': language
        })
    else:
        return render(request, 'animation.html')

def signup_view(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('animation')
    else:
        form = UserCreationForm()
    return render(request, 'signup.html', {'form': form})

def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(data=request.POST)
        if form.is_valid():
```

```
            user = form.get_user()
            login(request, user)
            if 'next' in request.POST:
                return redirect(request.POST.get('next'))
            else:
                return redirect('animation')
    else:
        form = AuthenticationForm()
    return render(request, 'login.html', {'form': form})

def logout_view(request):
    logout(request)
    return redirect("home")
```

```
#(settings.py)
import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'your-secret-key-here'  # Replace with a secure key
DEBUG = True
ALLOWED_HOSTS = []

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'A2SL',
```

```
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'A2SL.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'A2SL.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
```

```
    }
}

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'assets')]

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

NLTK_DATA_PATH = os.path.join(BASE_DIR, 'nltk_data')
import nltk
nltk.data.path.append(NLTK_DATA_PATH)

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'



#(urls.py)
from django.contrib import admin
from django.urls import path
from . import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('signup/', views.signup_view, name='signup'),
    path('animation/', views.animation_view, name='animation'),
    path('', views.home_view, name='home'),
```

```
]

if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATICFILES_DIRS[0])
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)



import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'your-secret-key-here'  # Replace with a secure key
DEBUG = True
ALLOWED_HOSTS = []

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'A2SL',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

```
ROOT_URLCONF = 'A2SL.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'A2SL.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'assets')]

MEDIA_URL = '/media/'
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

NLTK_DATA_PATH = os.path.join(BASE_DIR, 'nltk_data')
import nltk
nltk.data.path.append(NLTK_DATA_PATH)

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

```
(#wsgi.py)
"""
WSGI config for A2SL project.

It exposes the WSGI callable as a module-level variable named ‘‘application‘‘.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'A2SL.settings')

application = get_wsgi_application()
```

#Templates

(animation.html)

```
{% extends 'base.html' %}
{% load static %}

{% block content %}
<div class="split left">
    <h2 align="center">Enter Text or Use Mic</h2>
    <br>
    <form action="" method="post" align="left">
        {% csrf_token %}
        <input type="text" name="sen" class="mytext" id="speechToText"

        placeholder="Type or speak here...">
        <button type="button" name="button"
        class="mic" onclick="record()"><img src="{% static 'mic3.png' %}"
        height="32px" width="38px" /></button>

        <select name="language" id="language" style="height: 42px; vertical-align: top;">
        <option value="hi">Hindi</option>
        </select>

        <input type="submit" name="submit" class="submit">
    </form>
    <br>
```

```
<!-- Spectrogram section -->
<div class="spectrogram-section">
    <h3>Spectrogram</h3>
    <canvas id="spectrogramCanvas" width="600" height="100"

    style="border: 1px solid #3498db;"></canvas>
</div>
<br>

<table cellspacing="20px">
    <tr>
        <td class="td">The text that you entered is:</td>
        <td class="td">{{ text }}</td>
    </tr>
    {% if translated_text and language == 'hi' %}
    <tr class="translation-row">
        <td class="td">Translated to English:</td>
        <td class="td">{{ translated_text|capfirst }}</td>
    </tr>
    {% endif %}
    <tr>
        <td class="td">Key words for animation:</td>
        <td class="td">
            <ul class="td" id="list" align="center">
                {% for word in words %}
                <li id="{{ forloop.counter0 }}"
                style="margin-right: 8px">{{ word }}</li>
                {% endfor %}
            </ul>
        </td>
    </tr>
</table>

{% if audio %}
<br>
<div class="audio-container">
```

```
        <audio controls>
            <source src="{{ audio }}" type="audio/mp3">
            Your browser does not support the audio tag.
        </audio>
    </div>
    {% endif %}
</div>

<div class="split right">
    <h2 align="center">Sign Language Animation</h2>
    <div style="text-align:center">
        <button class="submit play-button" onclick="playPause()">Play/Pause</button>
        <video id="videoPlayer" width="600" height="350" preload="auto" autoplay>
            <source src="" type="video/mp4">
            Your browser does not support HTML5 video.
        </video>
    </div>
</div>

<style>
/* Modern styling for layout and elements */
* {
    box-sizing: border-box;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

h2, h3 {
    color: #030303;
    font-weight: 600;
}

.split {
    height: 100%;
    position: relative;
    padding: 25px;
    overflow: auto;
```

```
}

.left {
    width: 50%;
    float: left;
    border-right: 1px solid #D4C9BE;
    background-color: #F1EFEC;
}

.right {
    width: 50%;
    float: right;
    background-color: #F1EFEC;
    display: flex;
    flex-direction: column;
    padding: 25px;
}
.right h2 {
    text-align: center;
    margin-bottom: 20px;
}

form {
    margin-bottom: 20px;
    display: flex;
    flex-wrap: wrap;
    align-items: center;
    gap: 8px;
}

.mytext {
    height: 42px;
    padding: 0 15px;
    border: 1px solid #D4C9BE;
    border-radius: 6px;
    width: 60%;
```

```
    font-size: 16px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    transition: all 0.3s ease;
}

.mytext:focus {
    outline: none;
    border-color: #D4C9BE;
    box-shadow: 0 2px 8px rgba(212, 201, 190, 0.3);
}

.mic {
    background-color: #D4C9BE;
    border: none;
    padding: 5px 10px;
    border-radius: 6px;
    cursor: pointer;
    vertical-align: top;
    height: 42px;
    transition: all 0.3s ease;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

.mic:hover {
    background-color: #c4b7a9;
    transform: translateY(-2px);
}

.mic:active {
    transform: translateY(0);
}

#language {
    height: 42px;
    padding: 0 10px;
    border: 1px solid #D4C9BE;
```

```
    border-radius: 6px;
    margin: 0 5px;
    background-color: white;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    cursor: pointer;
}

.submit {
    background-color: #030303;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 6px;
    cursor: pointer;
    font-weight: 500;
    transition: all 0.3s ease;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

.submit:hover {
    background-color: #333333;
    transform: translateY(-2px);
}

.submit:active {
    transform: translateY(0);
}

.play-button {
    margin-bottom: 15px;
    font-size: 16px;
}

.spectrogram-section {
    margin: 15px 0;
    padding: 15px;
```

```
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.05);
}

#spectrogramCanvas {
    width: 100%;
    max-width: 600px;
    border: 1px solid #D4C9BE;
    border-radius: 8px;
    background-color: #F1EFEC;
}

table {
    width: 100%;
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.05);
    padding: 10px;
}

.td {
    padding: 12px;
    color: #030303;
}

#list {
    display: flex;
    flex-wrap: wrap;
    list-style-type: none;
    padding: 0;
    margin: 0;
    justify-content: flex-start;
}

#list li {
```

```
    background-color: #D4C9BE;
    color: #030303;
    padding: 8px 15px;
    margin: 5px;
    border-radius: 20px;
    transition: all 0.3s ease;
    font-weight: 500;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

#list li:hover {
    transform: translateY(-2px);
    box-shadow: 0 4px 8px rgba(0,0,0,0.15);
}

.audio-container {
    margin-top: 20px;
    text-align: left;
    background-color: white;
    padding: 15px;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.05);
}

audio {
    width: 100%;
    border-radius: 30px;
}

#videoPlayer {
    width: 100%;
    max-height: calc(100vh - 200px);
    object-fit: contain;
    border-radius: 8px;
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
}
```

```
/* Translation row styling */
.translation-row td {
    background-color: #ffffff;
    padding: 12px;
    border-radius: 6px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.05);
}

.translation-row .td {
    color: #030303;
    font-weight: 500;
}

/* Responsive adjustments */
@media (max-width: 768px) {
    table {
        display: block;
        overflow-x: auto;
    }
    .td {
        font-size: 16px;
        padding: 8px;
    }
    #list li {
        margin: 3px;
        padding: 6px 10px;
    }
}
</style>

<script>
    // Global variables for spectrogram
    let audioContext, analyser, source, animationId;

    // WebkitSpeechRecognition for speech-to-text with spectrogram
```

```
function record() {
    var recognition = new webkitSpeechRecognition();
    var lang = document.getElementById('language').value;
    recognition.lang = lang === 'hi' ? 'hi-IN' : 'en-IN';

    // Start spectrogram when recording begins
    startSpectrogram();

    recognition.onresult = function(event) {
        console.log(event);
        document.getElementById('speechToText').value = event.results[0][0].transcript;
        // Stop spectrogram when recording ends
        stopSpectrogram();
    };

    recognition.onend = function() {
        // Ensure spectrogram stops even if no result is returned
        stopSpectrogram();
    };

    recognition.onerror = function(event) {
        console.error('Speech recognition error:', event.error);
        stopSpectrogram();
    };

    recognition.start();
}

function play() {
    var videoSource = new Array();
    var videos = document.getElementById("list").
    getElementsByTagName("li");
    for (var j = 0; j < videos.length; j++) {
        videoSource[j] = "/static/" + videos[j].innerHTML + ".mp4";
    }
```

```
    var i = 0;
    var videoCount = videoSource.length;

    function videoPlay(videoNum) {
        document.getElementById("list").getElementsByTagName
        ("li")[videoNum].style.color = "#09edc7";
        document.getElementById("list").getElementsByTagName
        ("li")[videoNum].style.fontSize = "xx-large";
        document.getElementById("videoPlayer").
        setAttribute("src", videoSource[videoNum]);
        document.getElementById("videoPlayer").load();
        document.getElementById("videoPlayer").play();
    }

    document.getElementById('videoPlayer').addEventListener('ended', myHandler, false);
    if (videoCount > 0) {
        document.getElementById("list")
        .getElementsByTagName("li")[0].style.color = "#09edc7";
        document.getElementById("list")
        .getElementsByTagName("li")[0].style.fontSize = "xx-large";
        videoPlay(0);
    }

    function myHandler() {
        document.getElementById("list")
        .getElementsByTagName("li")[i].style.color = "#feda6a";
        document.getElementById("list")
        .getElementsByTagName("li")[i].style.fontSize = "20px";
        i++;
        if (i == videoCount) {
            document.getElementById("videoPlayer").pause();
        } else {
            videoPlay(i);
        }
    }
}
```

```
function playPause() {
    if (document.getElementById("videoPlayer").paused) {
        play();
    } else {
        document.getElementById("videoPlayer").pause();
    }
}

// Simple Spectrogram functions
function startSpectrogram() {
    if (!audioContext) {
        audioContext = new (window.AudioContext || window.webkitAudioContext)();
        analyser = audioContext.createAnalyser();
        analyser.fftSize = 256; // Simple FFT size for basic visualization
        const bufferLength = analyser.frequencyBinCount;
        const dataArray = new Uint8Array(bufferLength);

        const canvas = document.getElementById
        ('spectrogramCanvas');
        const ctx = canvas.getContext('2d');

        navigator.mediaDevices.getUserMedia({ audio: true })
            .then(stream => {
                source = audioContext.createMediaStreamSource(stream);
                source.connect(analyser);

                function drawSpectrogram() {
                    animationId = requestAnimationFrame(drawSpectrogram);
                    analyser.getByteFrequencyData(dataArray);

                    // Clear canvas
                    ctx.fillStyle = '#ecf0f1';
                    ctx.fillRect(0, 0, canvas.width, canvas.height);

                    // Draw bars for frequency data
```

```
                        const barWidth = canvas.width / bufferLength;
                        for (let i = 0; i < bufferLength; i++) {
                            const barHeight = dataArray[i] * (canvas.height / 255);
                            // Scale to canvas height
                            ctx.fillStyle = '#3498db';
                            // Simple blue color

                            ctx.fillRect(i * barWidth, canvas.height - barHeight,
                            barWidth - 1, barHeight);
                        }
                    }

                    drawSpectrogram();
                })
                .catch(err => console.error('Error accessing microphone:', err));
        }
    }

    function stopSpectrogram() {
        if (audioContext && source) {
            source.disconnect();
            audioContext.close();
            cancelAnimationFrame(animationId);
            audioContext = null;
            source = null;
            analyser = null;
            const canvas = document.getElementById('spectrogramCanvas');
            const ctx = canvas.getContext('2d');
            ctx.fillStyle = '#ecf0f1';
            ctx.fillRect(0, 0, canvas.width, canvas.height); // Clear canvas
        }
    }
</script>
{% endblock %}
```

```
#base.html
{% load static %}

<!DOCTYPE html>
<html>
<head>
<style>
.center {
  display: block;
  margin-left: auto;
  margin-right: auto;
  width: 50%;
}

#nav {
  list-style-type: none;
  margin-top: 0;
  padding: 0;
  overflow: hidden;
  background-color: #030303;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
  transition: all 0.3s ease;
}

h2 {
  color: #030303;
  text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.1);
}

.li {
  float: left;
  transition: background-color 0.3s ease;
}
```

```
.li a {
  display: block;
  color: #F1EFEC;
  font-size: 20px;
  font-weight: bold;
  padding: 14px 16px;
  text-decoration: none;
  transition: all 0.3s ease;
}

li a:hover {
  background-color: #D4C9BE;
  color: #030303;
  font-weight: bold;
  transform: translateY(-2px);
}

.form-style button {
  width: 89%;
  height: 70%;
  padding: 5%;
  background: #030303;
  border: none;
  border-radius: 4px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
  color: #F1EFEC;
  font-weight: bold;
  font-size: 28px;
  font-family: "Times New Roman";
  transition: all 0.3s ease;
}

.form-style button:hover {
  background-color: #333333;
  color: #F1EFEC;
  font-weight: bold;
```

```
  transform: scale(1.02);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
}

.split {
  height: 100%;
  width: 50%;
  position: fixed;
  z-index: 1;
  top: 50;
  overflow-x: hidden;
  padding-top: 20px;
  transition: all 0.5s ease;
}

.left {
  left: 15;
  border-right: 0px #D4C9BE solid;
}

.right {
  right: 0;
  border-left: 1px #D4C9BE solid;
}

.mytext {
  border: 1px solid #D4C9BE;
  border-right: none;
  padding: 4px;
  margin: 0px;
  float: left;
  height: 32px;
  overflow: hidden;
  line-height: 16px;
  width: 300px;
  margin-left: 54px;
```

```
  border-radius: 4px 0 0 4px;
  transition: all 0.3s ease;
}

.mic {
  border: 1px solid #D4C9BE;
  background: #D4C9BE;
  vertical-align: top;
  padding: 0px;
  margin: 0;
  float: left;
  height: 42px;
  overflow: hidden;
  width: 5em;
  text-align: center;
  line-height: 16px;
  color: #030303;
  transition: all 0.3s ease;
}

.mic:hover {
  background-color: #c4b7a9;
  color: #030303;
}

.submit {
  border: 1px solid #D4C9BE;
  height: 42px;
  width: 160px;
  text-align: center;
  background-color: #030303;
  color: #F1EFEC;
  font-weight: bold;
  font-size: 24px;
  font-family: "Times New Roman";
  vertical-align: top;
```

```
  border-radius: 0 4px 4px 0;
  transition: all 0.3s ease;
}

.submit:hover {
  background-color: #333333;
  color: #F1EFEC;
  font-weight: bold;
  transform: translateX(2px);
}

.td {
  color: #030303;
  font-weight: bold;
  font-size: 20px;
}

body {
  background-color: #F1EFEC;
  color: #030303;
  font-family: Arial, sans-serif;
  line-height: 1.6;
  margin: 0;
  padding: 0;
}

.form-style {
  font: 95% Arial, Helvetica, sans-serif;
  max-width: 400px;
  margin: 10px auto;
  padding: 16px;
  background: #fff;
  border-radius: 8px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
  transition: all 0.3s ease;
}
```

```
.form-style:hover {
  box-shadow: 0 6px 18px rgba(0, 0, 0, 0.15);
  transform: translateY(-5px);
}

.form-style h1, .form-style a {
  padding: 20px 0;
  font-size: 24px;
  font-weight: bold;
  font-family: "Times New Roman";
  text-align: center;
  margin: -16px -16px 16px -16px;
  color: #F1EFEC;
  border-radius: 8px 8px 0 0;
  background-color: #030303;
}

.form-style a {
  display: inline-block;
  text-decoration: none;
  transition: all 0.3s ease;
}

.form-style a:hover {
  color: #D4C9BE;
}

.form-style input[type="text"],
.form-style input[type="password"],
.form-style input[type="date"],
.form-style input[type="datetime"],
.form-style input[type="email"],
.form-style input[type="number"],
.form-style input[type="search"],
.form-style input[type="time"],
```

```
.form-style input[type="url"],
.form-style textarea,
.form-style select {
  -webkit-transition: all 0.30s ease-in-out;
  -moz-transition: all 0.30s ease-in-out;
  -ms-transition: all 0.30s ease-in-out;
  -o-transition: all 0.30s ease-in-out;
  transition: all 0.30s ease-in-out;
  outline: none;
  box-sizing: border-box;
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  width: 100%;
  background: #fff;
  margin-bottom: 4%;
  border: 1px solid #D4C9BE;
  padding: 3%;
  color: #030303;
  font: 95% Arial, Helvetica, sans-serif;
  border-radius: 4px;
}

.form-style input[type="text"]:focus,
.form-style input[type="password"]:focus,
.form-style input[type="date"]:focus,
.form-style input[type="datetime"]:focus,
.form-style input[type="email"]:focus,
.form-style input[type="number"]:focus,
.form-style input[type="search"]:focus,
.form-style input[type="time"]:focus,
.form-style input[type="url"]:focus,
.form-style textarea:focus,
.form-style select:focus {
  box-shadow: 0 0 8px rgba(212, 201, 190, 0.6);
  padding: 3%;
  border: 1px solid #D4C9BE;
```

```
}

.site-form span, label {
  color: #030303;
  font-weight: 600;
}

.errorlist {
  color: #e74c3c;
  font-weight: bold;
  background-color: rgba(231, 76, 60, 0.1);
  padding: 10px;
  border-radius: 4px;
  border-left: 4px solid #e74c3c;
}
</style>
<link href="https://fonts.googleapis.com/css2?
family=Noto+Sans+Devanagari&display=swap"
rel="stylesheet">
<title>Homepage</title>
</head>
<div style="background-color:#404040;
color:#feda6a;padding:10 10 1 10;border: 1px #feda6a groove;margin-bottom:0;">
<h1 align=center></h1>
</div>
<br>
<body style="font-family: 'Noto Sans Devanagari', Arial, sans-serif;">
<ul id="nav">
  <li class="li"><a class="active" href="{% url 'home' %}">Home</a></li>
  <li class="li"><a href="{% url 'animation' %}">Express</a></li>
  {% if not user.is_authenticated %}
  <li class="li"><a href="{% url 'signup' %}">SignUp</a></li>
  {% endif %}
  {% if user.is_authenticated %}
  <li class="li"><a href="{% url 'logout' %}">LogOut</a></li>
  {% else %}
```

```
    <li class="li"><a href="{% url 'login' %}">Login</a></li>
  {% endif %}
</ul>

<div class="wrapper">
  {% block content %}
  {% endblock %}
</div>

</body>
</html>
```

#(home.html)

```
{% extends 'base.html' %}
{% load static %}

{% block content %}
<div class="hero-section">
  <div class="video-container">
    <video autoplay loop muted playsinline>
      <source src="{% static 'Hello.mp4' %}" type="video/mp4">
      Your browser does not support the video tag.
    </video>
  </div>

  <div class="cta-container">
    <a href="{% url 'animation' %}">
      <button class="modern-button">CLICK HERE</button>
    </a>
  </div>
```

```
  <div class="info-section">
    <div class="feature-card">
      <h3>Multiple Language Support </h3>
      <p>Supports both English and Hindi using modern NLP tools</p>
    </div>

    <div class="feature-card">
      <h3>Responsive Design</h3>
      <p>Our content automatically adapts to
      any screen size, ensuring perfect viewing on all devices.</p>
    </div>

    <div class="feature-card">
      <h3>Animated Sign Languages </h3>
      <p>Animated sign languages for
      the deaf and hard of hearing, promoting inclusivity and accessibility.</p>
    </div>
  </div>
</div>

<style>
.hero-section {
  width: 100%;
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
  background-color: #F1EFEC;
}

.video-container {
  width: 100%;
  max-width: 800px;
```

```
  margin: 1rem auto;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);
  border-radius: 12px;
  overflow: hidden;
}

video {
  width: 100%;
  height: auto;
  display: block;
}

.cta-container {
  text-align: center;
  margin: 2rem auto;
}

.modern-button {
  background: #030303;
  color: white;
  border: none;
  padding: 14px 32px;
  font-size: 16px;
  font-weight: 600;
  border-radius: 8px;
  cursor: pointer;
  transition: all 0.3s ease;
  letter-spacing: 1px;
  box-shadow: 0 4px 12px rgba(3, 3, 3, 0.3);
}

.modern-button:hover {
  transform: translateY(-3px);
  box-shadow: 0 8px 15px rgba(3, 3, 3, 0.4);
  background: #333333;
}
```

```
.modern-button:active {
  transform: translateY(1px);
}

.info-section {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 20px;
  margin-top: 2rem;
  width: 100%;
}

.feature-card {
  background: white;
  border-radius: 10px;
  padding: 25px;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.08);
  flex: 1 1 300px;
  max-width: 350px;
  transition: transform 0.3s ease;
  border: 1px solid #D4C9BE;
}

.feature-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.12);
}

.feature-card h3 {
  color: #030303;
  margin-top: 0;
  margin-bottom: 15px;
}
```

```
.feature-card p {
  color: #030303;
  line-height: 1.6;
  margin: 0;
}

/* Media queries for better responsiveness */
@media (max-width: 768px) {
  .video-container {
    max-width: 100%;
  }

  .info-section {
    flex-direction: column;
    align-items: center;
  }

  .feature-card {
    max-width: 100%;
  }
}
</style>
{% endblock %}



(#login.html)
{% extends 'base.html' %}


{% block content %}
<div class="form-style">
 <h1>Log in</h1>
 <form class="site-form" action="." method="post">
  {% csrf_token %}
  {{ form }}
```

```
  {% if request.GET.next %}
   <input type="hidden" name="next" value="{{ request.GET.next }}">
  {% endif %}
  <input class="submit" type="submit" value="Log in">
 </form>
</div>
{% endblock %}



#signup.html)
{% extends 'base.html' %}

{% block content %}
<div class="form-style">
 <h1>Sign Up</h1>
 <form class="site-form" action="." method="post">
  {% csrf_token %}
  {{ form }}
  <br><br>
  <input class="submit" type="submit" value="Sign Up">

 </form>
</div>
<script type="text/javascript">
 document.getElementsByTagName("span")[0].innerHTML="";
 document.getElementsByTagName("span")[1].innerHTML="";
</script>
{% endblock %}
```

```
#n.py

import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')



#setup.py

import setuptools

setuptools.setup(
    name='audio-speech-to-sign-language-converter',
    version='0.1.0',
    description='Python project',
    packages=setuptools.find_packages(),
    install_requires=[
        'nltk',
        'django',
        'gTTS>=2.3',
        'indic-nlp-library>=0.81',
    ],
    setup_requires=['nltk', 'joblib', 'click', 'regex', 'sqlparse', 'setuptools'],
)
```

```
#manage.py

#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'A2SL.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```

```
#requirements.txt


asgiref==3.5.2

click==8.1.3

Django>=4.1.9

joblib==1.2.0

nltk==3.7

regex==2022.10.31

sqlparse>=0.4.4

tqdm==4.64.1

setuptools==67.8.0
```

P MEENAKSHI REDDY

meenakshireddy7857@gmail.com

+91 7066020115 |

Pune, India

www.linkedin.com/in/meenakshi-reddy10

PROFESSIONAL SUMMARY

Results-driven Software Developer with 1+ year of experience in full-stack web development and a strong foundation in C++, Python, and Data Structures. Pursuing B. Tech in Electronics and Communication Engineering (9.0 CGPA) at Bharati Vidyapeeth, Pune. Demonstrated expertise in HTML, CSS, JavaScript, PHP, CodeIgniter, React.js, Node.js, and MySQL. Proven track record of delivering high-impact projects, optimizing performance, and driving user engagement. Seeking a challenging Software Developer role to leverage technical skills and contribute to innovative solutions.

SKILLS

- Programming Languages: JavaScript, PHP, Python, C++, HTML5, CSS3
- Frameworks & Libraries: CodeIgniter, Bootstrap, Laravel
- Databases: MySQL
- Tools & Technologies: Git, Ajax
- Concepts: Data Structures, Algorithms, Object-Oriented Programming, MVC Architecture
- Soft Skills: Problem-solving, Teamwork, Communication, Adaptability, Time Management

PROFESSIONAL EXPERIENCE

Software Developer Intern | BARC (Bhabha Atomic Research Centre), Tarapur, Boisar

June 2024 - August 2024

- Developed a high-performance tele-billing system using PHP, JavaScript, and CodeIgniter MVC, resulting in improved billing efficiency
- Implemented advanced data filtering and cost calculation features, reducing manual processing time .
- Engineered a custom month-picker using HTML and jQuery, enhancing user experience and data accuracy

Web Developer Intern | Industrial Infotech, Tarapur, Boisar

June 2023 - October 2023

- Developed responsive, user-centric websites using HTML, CSS, JavaScript, and PHP with Laravel framework
- Integrated third-party APIs and payment gateways, enhancing website functionality and user experience

PROJECTS

E-Commerce Platform (WordPress) | 2024

- Engineered a scalable e-commerce solution with customized plugins for payment processing and inventory management
- Implemented SEO best practices and responsive design, resulting in improved site visibility and user engagement
- Integrated analytics tools to track and optimize user behaviour, leading to data-driven improvements

Laravel E-Commerce System | 2023

- Developed a full-stack e-commerce application emphasizing secure user authentication and efficient database design
- Implemented RESTful API architecture for seamless frontend-backend communication
- Utilized caching and database optimization techniques to enhance system performance and user experience

EDUCATION

Bachelor of Technology in Electronics and Communication Engineering Bharati Vidyapeeth

Deemed to Be University, Pune | 2022-2025

- CGPA: 9.0/10 (up to Semester 6)
- Relevant Coursework: Data Structures, Algorithms, Web Development, Database Management Systems, Internet of Things

CERTIFICATIONS

- Advanced Web Development Bootcamp (Udemy)
- Python Programming Masterclass (Udemy)

SONI KUMARI

Engineering Student

sonikumari200960@gmail.com | 6205900631

Katraj, Pune – 411043

Summary

Detail-oriented and ambitious engineering student with a passion for enhancing skills and working with teams.

A resourceful coordinator with outstanding prioritization and planning abilities. Seeking a position in the engineering industry as an Electronics and Communication Engineer.

Education

• B.Tech in Electronics and Communication Engineering, BVDUCOEP, 2021–2025

• 12th Grade (Science), Vidya Bharati Chinmaya Vidyalaya, Jamshedpur 2021 (92%)

• 10th Grade, Vidya Bharati Chinmaya Vidyalaya, Jamshedpur 2019 (93%)

Experience

• Intern, Bharti Airtel 2025–Present

– Participating in a 6-month internship program to gain hands-on experience in the telecom and networking

domain.

• **Intern, Tata Motors June 2024–August 2024**

– Contributed to projects in automotive engineering, gaining valuable industry insights and hands-on experience in the manufacturing process and quality control.

Skills

• Programming: Java, Python (Basics), C, C++ (Basics)

• Database: DBMS (Basic understanding)

• Soft Skills: Content Writing, Public Speaking, Multitasking, Time Management, Team Work

Projects

• Speech to Indian Sign Language Conversion Tool

– Developed a tool to convert speech into Indian Sign Language in real-time, utilizing machine learning and computer vision technologies.

MANISHA KUMARI

+91 8539950070

manisha06172002@gmail.com

Linkedin –

www.linkedin.com/in/manisha-roy-aa531122b

Software Developer with expertise in web development, C/C++, Java and data structure .currently pursuing a Bachelor's degree in Electronics and Communication Engineering at Bharati Vidyapeeth Deemed University, Pune. seeking a software Developer role to utlilize skills in development, collaboration, and branding.

Skills

Frontend: HTML, CSS, JavaScript, React.js

Backend: Node.js

Programming Languages: C/C++, Python, Java Data Structures and Algorithms

Databases: MySQL, NoSQL, SQLite Version Control: Git

EDUCATION

BHARATI VIDYPAEETH DEEMED UNIVERSITY, PUNE Electronics and Communication engineering | 2021-2025

AKSHAWAYAT COLLEGE XII(Bseb) 2019 - 2021

ST,XAVIER'S HIGH SCHOOL X(cbse)| 2019

Experience

ECSA(ELECTRONICS AND COMMUNICATION STUDENT ASSOCIATION)

Technical member 2022 - present Technical Expertise: ECSA member excels in electronics, circuits, and telecom, contributing vital knowledge and skills.

Problem Solver: Expert troubleshooter, crafting innovative solutions for complex electronics and communication challenges.

Collaborative Team Player: Promotes teamwork, mentors, and fosters camaraderie, enriching ECSA's mission and project success.