****

**Software Engineering Department  
 Braude College**

**Capstone Project Phase B – 61999**

**SoundSigns: Translating English Audio to Sign Language**

**25-1-D-19**

**Students: Ahmad Ataba & Waseem Saleem**

**Supervisor: Dr. Reuven Cohen**

**GitHub:-** [**https://github.com/Ataba29/Final-Project**](https://github.com/Ataba29/Final-Project)

**Table of Contents**

[**Abstract 3**](#_5tp4tbm9l512)

[**1. Introduction 3**](#_9hyaz8d8aw6)

[**2. The Product 5**](#_x9rz19pikxbd)

[2.1 The Product Architecture 5](#_b326kwk8e7s9)

[2.2 Interaction with the Application 7](#_b9b444gmpxvg)

[1. Voice Input and Transcription 7](#_tt5blk1aqn3l)

[2. Text-to-Gloss Conversion 7](#_n5tsd6n9wujk)

[3. Gloss Tokenization and Video Matching 8](#_ppywhz44qrny)

[4. Video Assembly and Playback 8](#_djjhuvn30b4y)

[5. Output Display 8](#_fy0qe98r9juj)

[2.3 Activity Diagram 9](#_yd05g8buoskx)

[2.4 Technologies Used 10](#_ixcuys177pd2)

[2.5 Development Environment 10](#_o5rlvyfn1rs5)

[2.6 Dataset 11](#_vubdsy7myeqn)

[2.7 Testing 12](#_vitt88mkl71x)

[2.9 Development Process 13](#_ghyev2v1rcph)

[**3. Challenges 14**](#_ijekab57w6oc)

[3.1 Key Challenges 14](#_18nlvnja505c)

[3.2 Limitations and Comparative Evaluation 16](#_n0f5e1suflof)

[3.2.1 Comparison with Native Signers 16](#_iu96v0nuk0wr)

[3.2.2 Comparison with Existing Tools 17](#_htdpjykjosmt)

[3.2.3 Summary of Findings 18](#_eb6xv42sekxk)

[3.3 Summary and Conclusions 18](#_vxix1jr73emo)

[**4. User Guide 20**](#_gerxclxy9hta)

[**5. Maintenance Guide 23**](#_lb4kg816w51h)

[5.1 System Requirements 23](#_wwmthiihdzcd)

[5.2 Backend Setup and Maintenance 24](#_mf6kj65oinzl)

[5.3 Frontend Setup and Maintenance 24](#_3546qmafkf83)

[5.4 Folder Structure 25](#_lweh3a9sqq5q)

[**6. References 26**](#)

# 

# **Abstract**

Despite ongoing efforts to improve accessibility for the deaf and hard-of-hearing communities, significant challenges persist in providing consistent, scalable sign language translations for spoken and written media content. Addressing the shortage of professional interpreters and the technical complexity of real-time motion capture systems, we propose a streamlined web-based tool that delivers sign language translations using pre-rendered 3D animations in International Sign Language (ISL). Unlike systems that rely on live avatars or dynamic motion synthesis, our solution employs a curated dataset of MP4 video clips, each featuring a 3D model performing standardized ISL signs. The system takes English input and translates it into ISL gloss using ChatGPT. This gloss sequence is then used to match and play the corresponding pre-rendered videos in order, creating a coherent sign language output. By shifting the animation workload to the frontend and removing dependencies on real-time speech recognition systems, our architecture improves efficiency, reliability, and ease of deployment. This modular structure simplifies the development pipeline and ensures that users receive consistent, high-quality sign language interpretation across devices. Through ongoing refinement and expansion of the sign video dataset, our system aims to bridge communication gaps and promote greater inclusivity for deaf individuals engaging with digital content.

# **Introduction**

In today’s world, communication between deaf and non-deaf individuals has greatly improved. Innovations like sign languages, hearing aids, and increased accessibility awareness have all contributed to this progress. Still, significant gaps remain, especially when it comes to consuming audio-heavy or spoken media, for many deaf individuals who rely on sign language, traditional solutions like subtitles are not always effective.

Subtitles can be helpful, but they are typically designed for users fluent in written English. For native sign language users, subtitles may not provide the natural flow and clarity of communication that signing does. On the other hand, sign language interpreters offer more intuitive understanding, but they are rarely used in everyday media. Interpreters are generally reserved for formal events, news programs, or institutional content, not casual or fast-paced online videos.

This leaves a large portion of digital content, especially user-generated media like vlogs, tutorials, and social media clips, inaccessible to the deaf community. Creating subtitles for such content is often time-consuming and technically challenging. Many creators lack the tools, time, or resources to provide accessible alternatives.

To address this issue, we propose a web-based tool that automatically converts English text into International Sign Language (ISL)—a pidgin language used for cross-cultural communication among deaf individuals from diverse linguistic backgrounds. Unlike natural sign languages such as American Sign Language (ASL), ISL is not the native language of any community; rather, it draws on widely recognized, iconic signs and gestures to support mutual understanding.

Our system uses a modular design built for efficiency and ease of use. Instead of relying on real-time animations or motion capture, we use a curated library of pre-rendered MP4 videos. These clips feature a high-quality 3D avatar signing standardized ISL terms.

The backend system focuses on translating English text into ISL gloss. Gloss is a simplified form of sign language that reflects its unique grammar and word order. After the gloss is generated, the frontend system matches each term to its corresponding video. The system then assembles the clips to create a coherent signed message.

This method avoids the complexity of live rendering or speech recognition. By separating translation and video playback, our tool maintains smooth performance and works across different devices. The user interface is simple. Users can speak a short English text, and the system generates a signed message using the avatar. It’s designed to be intuitive, fast, and accurate.

By using ISL, the tool can serve an international audience, rather than being limited to one country’s sign language. This broader reach helps increase inclusivity and allows for cross-cultural accessibility.

Beyond media accessibility, the tool also has potential in education. Users learning ISL can see how gloss terms map to specific signs and observe sign language grammar in action. Compared to other tools that may require downloading software or dealing with low-quality visuals, our platform is fully browser-based and visually engaging.

This document is structured to guide the reader through the development and capabilities of our web-based International Sign Language tool. Chapter 2 presents a detailed overview of the product, including its architecture. Technologies used and the development environment are also discussed.

Chapter 3 addresses the key challenges encountered during development and outlines potential future improvements to enhance usability and translation quality. Chapter 4 serves as a user guide, providing instructions on how to operate the application effectively. Chapter 5 offers a maintenance guide covering system requirements and steps for backend and frontend upkeep to ensure continued functionality.

# 2. The Product

## 2.1 The Product Architecture

Figure 1: System Architecture

The SoundSigns system is a web-based application designed to translate spoken English into International Sign Language (ISL) in real time. The architecture is composed of three primary components: the frontend client application, the backend server, and a curated video dataset, as illustrated in the system architecture diagram.

**The Frontend (Client Application)**

The frontend is developed using React.js, ensuring a responsive, interactive, and modern user experience, and styled with Tailwind CSS to maintain a clean and consistent visual design. The user interface features an intuitive design that includes a microphone button for capturing audio input, two text displays that show real-time transcribed speech and ISL gloss text, and a 3D animated avatar that performs sign language output generated from pre-rendered video clips.

The frontend processing pipeline consists of four key components:

**Transform Speech-to-Text**: This component captures spoken input and converts it into written text using the Web Speech API, a native browser interface for speech recognition. It provides real-time transcription capabilities directly within the browser environment, eliminating the need for external libraries or services. This stage forms the initial step in the frontend processing pipeline, enabling voice-based user interaction.

**Tokenize Gloss Text**: This component breaks down the ISL gloss text received from the backend into individual tokens or components. Each token represents a specific sign, letter, or concept that needs to be visually represented, making it easier for the system to match each element with its corresponding video clip.

**Get Videos from Dataset**: This component retrieves the appropriate video files from the stored collection of sign language clips. Based on the tokenized gloss text, it identifies and fetches the corresponding videos for each sign or letter needed.

**Concatenate Videos into One**: This component takes multiple individual sign language video clips and seamlessly combines them into a single, continuous video sequence. It is responsible for the video stitching process that creates fluid sign language animations displayed to the user.

**Backend (Server)**

The backend server is responsible for converting spoken English (transcribed as text) into ISL gloss—a structured, simplified form of sign language representing its grammar and word order. The backend processing involves two main components:

**GPT API**: This component processes the transcribed English text and converts it into ISL gloss format. It utilizes ChatGPT's natural language processing capabilities to understand the meaning and context of the spoken words, then generates the appropriate sign language grammar and word order structure.

**Send to Frontend**: This component handles the communication back to the client application, transmitting the generated ISL gloss text to the frontend.

**Dataset (Video Library)**

The system utilizes a curated dataset comprising approximately 150 pre-rendered MP4 videos, segmented into three categories:

* **Alphabet signs**: Individual videos for letters A–Z
* **Numeric signs**: Individual videos for digits 0–9
* **Common vocabulary**: Videos for frequently used words and expressions

Each video is linked to specific gloss terms, allowing the frontend to assemble coherent sign sequences by retrieving and stitching together relevant clips. This modular approach enables the system to construct complex sign language translations from individual video components.

## 2.2 Interaction with the Application

The SoundSigns application delivers real-time sign language translation by coordinating browser-based speech recognition, backend language processing, and frontend-driven animation using pre-rendered video assets. The process follows five primary steps:

### 1. Voice Input and Transcription

* The user initiates voice capture by clicking a microphone button in the interface.
* Spoken English is transcribed directly in the browser using the Web Speech API.

### 2. Text-to-Gloss Conversion

The transcribed English text is converted into an ISL gloss using the ChatGPT API (*gpt-3.5-turbo*). The process was guided by a structured system prompt to ensure consistency with ISL grammar and compatibility with the available video dataset. Key aspects of the prompt design include:

* ISL Grammar Rules: The model was instructed to follow standard ISL gloss conventions, including time-topic-comment structure and the use of classifiers.
* Controlled Vocabulary: While a fixed dictionary was not provided, the prompt encouraged the use of simplified, gloss-compatible vocabulary aligned with our dataset.
* Formatting Standards: Signs were required to be written in UPPERCASE, and proper nouns were fingerspelled using hyphens (e.g., J-O-H-N).
* Exclusion of Non-Gloss Words: Common English function words (e.g., "is", "the", "a") were explicitly excluded to maintain gloss accuracy.
* Direct Output: The model was instructed to return only the ISL gloss, without any explanatory or surrounding text.

### 3. Gloss Tokenization and Video Matching

* Once the frontend receives the gloss, it is tokenized into individual sign components.
* Each token is then matched to a corresponding pre-rendered MP4 video stored locally in the frontend. These videos feature a 3D avatar performing standardized ISL signs.

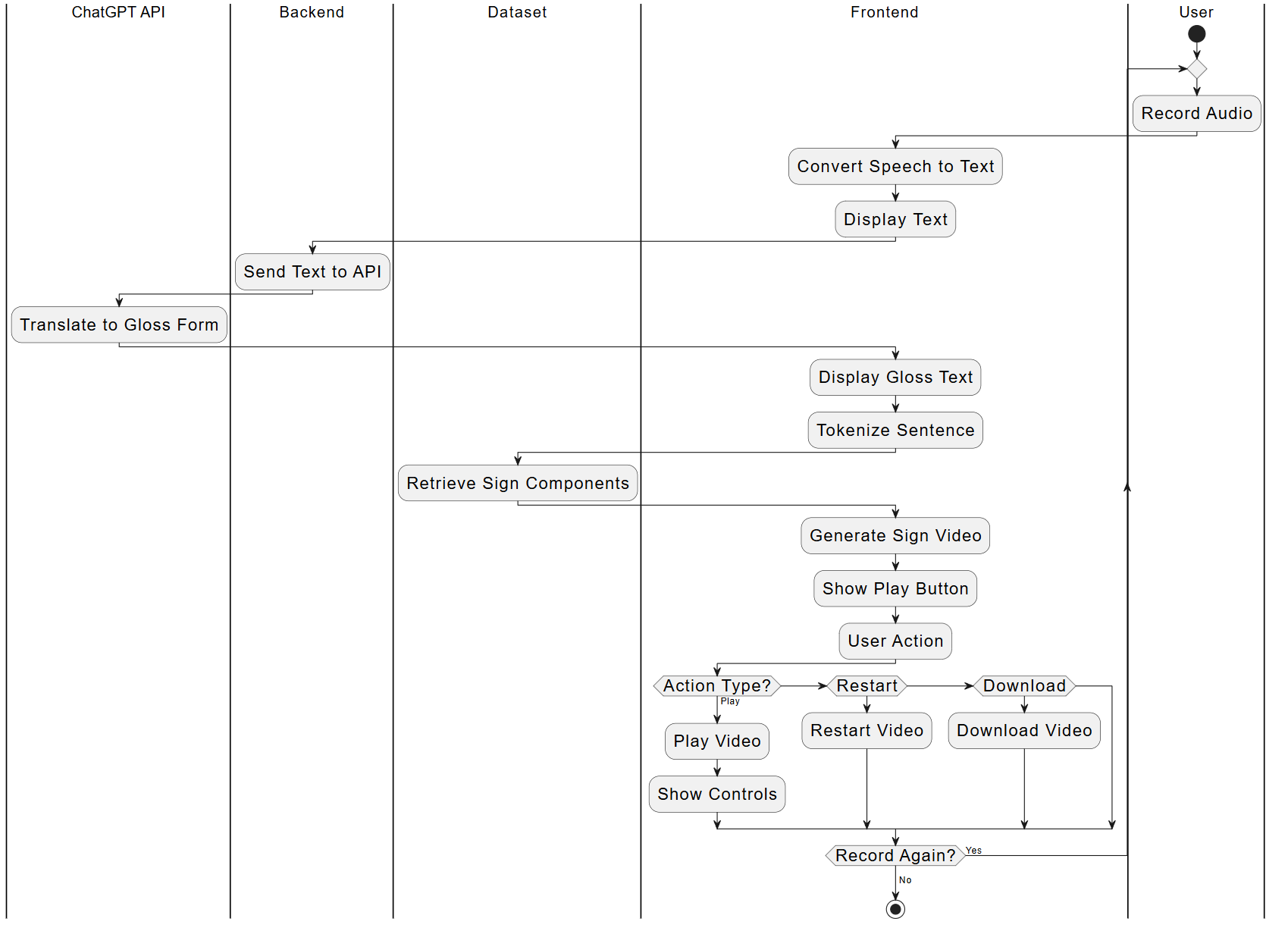
### 4. Video Assembly and Playback

* The matched video clips are assembled in sequence on the client side to form a grammatically accurate ISL sentence.
* The resulting animation is played directly in the browser, with no reliance on real-time rendering, motion capture, or backend video generation.

### 5. Output Display

* The user interface presents both the transcribed English text and the assembled sign language animation, allowing users to follow the spoken content in both textual and visual formats.

## 2.3 Activity Diagram

Figure 2: System Activity Diagram

This diagram illustrates the process of converting spoken language into a sign language video using multiple system components. The process begins with the User recording audio, which is converted in the front end into text. This text is sent to the Backend, which forwards it to the ChatGPT API for conversion into a gloss form—a simplified, grammatical version of the sentence suitable for sign language translation. The Frontend then displays this gloss text and tokenizes it into text components. These components are used to query the Dataset, retrieving the corresponding sign language videos. The Frontend assembles these videos into a single sign language video, displayed by a play button. The user can choose to play, restart, or download the video. The process can be repeated if the user wants to record another sentence.

## 2.4 Technologies Used

**React.js**

The user interface is built using React, a JavaScript library for creating dynamic, component-based web applications. React allows the system to manage state effectively and provide a smooth, responsive user experience, especially during real-time voice capture, gloss translation, and video playback.

**Tailwind CSS**

Tailwind CSS is a utility-first CSS framework used to style the application. It helps maintain a clean, modern, and responsive design across different screen sizes and devices, contributing to both accessibility and ease of use.

**Flask**

The system’s backend is built using Flask, a lightweight Python web framework. Flask handles API routing and acts as a bridge between the frontend and the OpenAI API, ensuring secure and efficient processing of the text-to-gloss conversion.

**OpenAI API (ChatGPT)**

The OpenAI API is used to perform the core translation task within the system. Specifically, it converts transcribed English text into International Sign Language (ISL) gloss, which reflects the grammatical structure of sign language. This API acts as a lightweight, cloud-based language processing service, allowing the backend to focus solely on accurate and efficient natural language translation.

**Web Speech Recognition API**

The application utilizes the Web Speech API to transcribe spoken English directly within the user's browser. This built-in browser feature enables real-time voice input without requiring external dependencies or server-side audio processing, making the system faster and more accessible.

## 2.5 Development Environment

* Code Editor:  
   The project was developed using Visual Studio Code, a lightweight and flexible code editor ideal for web development. It provided essential features like syntax highlighting, integrated terminal access, and helpful extensions for JavaScript, Python, and code formatting.
* Local Development Setup:  
   We used a local environment for running and testing the application:
  + The frontend (built with React) was run using a development server.
  + The backend (using Flask) ran in a separate terminal window.  
     This two-terminal workflow enabled real-time testing of the full system, from speech input to video playback, without the need for external deployment or complex configuration.
* Browser Testing:  
   Google Chrome was used as the primary browser for testing, thanks to its support for the Web Speech API, which handled voice transcription.
* Version Control:  
   Git and GitHub were used throughout the project for version control and collaborative development, ensuring consistency and traceability across the frontend and backend codebases.

## 2.6 Dataset

The dataset utilized in this project consists of MP4 video clips featuring a 3D avatar performing standardized signs in International Sign Language (ISL). It includes three primary categories of signs: alphabet letters (A–Z), numerical digits (0–9), and a selection of common words. Each video represents a single sign and maintains a consistent visual style, which supports smooth playback and seamless composition when assembling signed sentences in the frontend.

This dataset was obtained from a publicly available open-source project by the developer credited under the alias JS-Coderr (2024) [1]. The dataset is part of the "Text-Speech to Sign Language Generator" project, which is hosted on GitHub. Proper credit is given to the original author for creating and sharing these video assets under open license terms, which made it possible to integrate them into our application.

The use of pre-rendered video assets allowed us to focus development efforts on translation and video sequencing, while still delivering high-quality visual output. Since the signs are standardized and performed by a 3D model, they provide a clear and accessible representation of ISL gestures suitable for both accessibility and educational use.

## 2.7 Testing

| **NUM** | **Test** | **Expected Result** | **Achieved Result** |
| --- | --- | --- | --- |
| 1 | Test the accuracy of speech-to-text conversion | The system transcribes speech with high accuracy | The system transcribes speech with good accuracy, but occasional errors occur |
| 2 | Test UI responsiveness across different browsers | UI loads and functions properly on Chrome, Firefox, and Safari | UI loads and functions properly on Chrome, Firefox, and Safari |
| 3 | Test the accuracy of gloss transformation | System converts English to ISL gloss, preserving meaning | System converts English to ISL gloss; results are variable and may not always preserve full meaning |
| 4 | Test video retrieval from the dataset | The system successfully retrieves the correct videos for gloss tokens | The system successfully retrieves the correct videos for gloss tokens |
| 5 | Test video stitching and playback | The system seamlessly combines multiple videos into smooth animations | The system seamlessly combines multiple videos into smooth animations |
| 6 | Test audio capture quality | Audio is captured clearly with minimal distortion | Audio captures clearly in quiet environments; some degradation with background noise |
| 7 | Test system latency | The complete speech-to-sign process completes within 1 second | The complete process usually completes within 3-5 seconds |
| 8 | Test multi-device compatibility | System functions on desktop, tablet, and mobile devices | System functions on desktop, tablet, and mobile devices |

Test 3 showed that, while the system generally succeeds in converting English to ISL gloss, the output quality varies and does not always fully preserve the original meaning. This inconsistency primarily arises from the reliance on ChatGPT, a generative language model that operates without a fixed glossary or strict domain constraints. As a result, the model occasionally produces gloss terms or grammatical structures that do not align perfectly with our video dataset or established ISL conventions. Furthermore, complex or ambiguous English input can challenge the model’s ability to generate accurate gloss translations.

Aside from gloss transformation, the system performed well across most tests. The speech-to-text component demonstrated good accuracy, although occasional transcription errors were noted, particularly in noisy environments—an expected limitation of browser-based speech recognition technologies. The user interface exhibited consistent responsiveness across multiple browsers, supporting broad accessibility for users. Video retrieval and concatenation operated reliably, delivering smooth and continuous sign language animations.

## 2.9 Development Process

The development of the SoundSigns system followed a structured, iterative approach that focused on integrating existing resources with custom-built components to achieve a functional and user-friendly application.

The process began with sourcing the dataset of sign language videos. We identified and adopted a comprehensive collection of MP4 clips depicting a 3D avatar performing International Sign Language (ISL) signs, including letters, numbers, and common words. This dataset formed the foundation for the visual representation of signed content.

Next, we developed the frontend interface using React.js and Tailwind CSS. The initial interface prioritized a clean and accessible design. We implemented core functionality such as a microphone input button that allowed users to record spoken English. Speech-to-text transcription was integrated using the browser’s native Web Speech API, and the recognized English text was displayed in real time on the interface.

Parallel to the frontend work, the backend was developed with Flask to handle the translation process. The backend interacted with the OpenAI ChatGPT API to convert English text into ISL gloss, which is a simplified representation of sign language grammar. The backend solely managed this translation step, ensuring a clear separation of concerns within the system.

Once both components were functional, the frontend and backend were integrated. The integration enabled sending the transcribed English text captured on the frontend to the backend, translating into ISL gloss, and returning the output to the frontend for display. At this stage, the application displayed both the original English transcription and its corresponding ISL gloss output.

To effectively manage the video assets, the dataset was organized into three categories—letters, words, and digits—and placed within the frontend’s assets directory. This organization facilitated efficient lookup and retrieval of the appropriate videos for each gloss token.

The next development milestone was enabling the sequential playback of multiple video clips. By tokenizing the gloss sentence on the frontend, the system could assemble and display a series of videos in order, thereby producing a coherent signed message.

Finally, an option was added for users to download the assembled sign language video and play it back locally. This feature enhanced usability by allowing users to save and share the signed content outside of the web application.

Through these steps, the development process emphasized modularity, usability, and leveraging existing resources, resulting in a scalable and accessible sign language translation tool.

# 3. Challenges

## 3.1 Key Challenges

During the development of SoundSigns, our team faced a range of technical and practical challenges that shaped both the architecture and the development process. These difficulties required careful consideration, iterative testing, and creative problem-solving to ensure the system’s reliability and accessibility. Examining these challenges offers valuable insight into the complexities involved in building scalable and user-friendly sign language translation tools, especially when working within resource and technology constraints.

1. **Converting videos to animation:** One of the most significant hurdles was the inability to find a dependable solution for converting sign language videos into dynamic animation data. Accurate animation is crucial for natural sign language interpretation, as subtle hand shapes and movements carry essential meaning. We explored Kalidokit yeemachine (2021) [3], a tool designed to refine motion capture data, but its precision was insufficient for the intricate gestures of sign language. We then turned to SignAvatars Zhengdi Yu (2024) [4], an open-source project aimed at animating avatars for sign language, but were unable to run it due to incomplete documentation and unspecified package versions. This lack of viable options led us to rely on pre-rendered video clips, which, while limiting animation flexibility, provided a reliable and consistent visual output.
2. **Limited datasets of pre-rendered sign language videos:** Another major obstacle was the scarcity of publicly available datasets containing 3D rendered videos of sign language. High-quality, standardized datasets are essential for building accurate and inclusive translation systems. Despite extensive searching across academic and open-source repositories, we identified only a single dataset that met the project’s criteria for International Sign Language (ISL). This dataset’s limited scope constrained the variety of signs available and highlighted the broader issue of resource availability in this field.
3. **Learning and implementing Flask for backend development:** Involved designing an efficient pipeline to interact with the OpenAI API and manage requests, which required experimentation with API integration, server routing, and security best practices. This process deepened our understanding of backend architectures and emphasized the importance of modular design and maintainability.
4. **Securing the OpenAI API key:** Protecting sensitive credentials such as API keys is critical in any web-based application. We encountered challenges in safeguarding the OpenAI key from exposure on public platforms like GitHub, which could lead to misuse and potential financial liability. Implementing environment variables and excluding keys from version control required careful configuration, underscoring the importance of security awareness even in small-scale projects.
5. **Integrating the Web Speech API for speech-to-text functionality:** Implementing real-time speech recognition within the browser was a new challenge for us. Initially, we considered using Whisper, a powerful speech-to-text model, but integrating it directly into the React frontend proved complex and impractical given our project timeline and technical constraints. Instead, we opted for the browser’s native Web Speech API, which provided a simpler and more accessible interface for capturing audio and transcribing speech in real time. However, achieving reliable transcription required multiple rounds of testing and fine-tuning to address issues such as variations in audio input quality, browser compatibility differences, and user experience considerations. This iterative development was crucial in delivering responsive and accurate voice input, which is a core feature of the application’s usability and overall accessibility.

Collectively, these challenges highlight both the technical limitations and opportunities present in sign language translation technology. Overcoming them has informed our approach to system design and positioned the project for future expansion and refinement.

## 3.2 Limitations and Comparative Evaluation

This section focuses on assessing the accuracy of ChatGPT’s English-to-sign language gloss translation. While our system produces 3D sign animations, this evaluation specifically emphasizes the linguistic accuracy of the gloss output. Since our dataset primarily uses International Sign (IS) glosses and a hybrid vocabulary, we evaluate the generated gloss sequences rather than the visual animations. This linguistic evaluation verifies whether the glosses reflect natural sign language structures before considering visual quality.

### **3.2.1 Comparison with Native Signers**

We compare our tool’s gloss output with gloss sequences used by native signers of International Sign and other established sign languages. Testing involved simple, intermediate, and complex English sentences to evaluate how well the gloss translations capture natural sign language syntax and meaning.

For simple sentences such as “What is your name?” and “How are you?”, the tool generates glosses such as “YOU NAME WHAT” and “HOW YOU,” which align closely with natural sign ordering. For intermediate sentences, such as “I do not understand” and “Can you help me?”, the original English phrases were translated into glosses “UNDERSTAND ME NOT” and “HELP CAN YOU ME,” respectively. These glosses demonstrate reasonable syntactic structure, although occasionally the word order or inclusion of function words differs slightly from native usage.

More complex sentences, such as “If I had known about the meeting, I would have attended” and “She decided to walk even though it was raining,” were glossed by the tool as “ME KNOW MEETING, ME ATTEND” and “RAIN FALLING, WALK DECIDE SHE,” respectively. While these outputs effectively capture the essential meaning, they simplify the structure by omitting key grammatical markers. For example, native signer glosses would typically include a conditional marker and more natural verb usage, as in “IF ME KNOW MEETING, ME WILL GO,” and a contrastive element such as “STILL” to indicate opposing ideas, as in “RAIN STILL SHE DECIDE WALK.” These examples highlight that while the model reliably conveys the core semantic content in a form accessible to sign language users, it does not fully express the nuanced grammar, spatial structuring, or non-manual markers—such as facial expressions and body posture—that are typical of fluent, native signing.

The sign video dataset combines signs from multiple sign languages, resulting in a hybrid gloss vocabulary that may differ from any single regional variety. Although the 3D animations offer a visual representation of the signs, the system currently lacks the non-manual markers necessary for expressing grammatical and emotional nuance.

This evaluation highlights the system’s effectiveness in generating syntactically and semantically coherent gloss sequences, while also acknowledging current limitations in expressiveness and grammatical depth. Future work will focus on integrating non-manual features and refining the dataset to improve naturalness.

### **3.2.2 Comparison with Existing Tools**

To better understand the capabilities of our system, we conducted a comparative evaluation with Sign.mt (Moryossef, 2023) [2], a publicly available text-to-sign translation system that primarily focuses on American Sign Language (ASL) gloss conventions and includes detailed avatar facial animations.

For comparison, we input a series of English sentences into both systems and analyzed the gloss results. For simple queries such as “What is your name?” and “How are you?”, both tools produced similar glosses, including “YOU NAME WHAT” and “HOW YOU.” This suggests that for basic, everyday expressions, our tool captures the expected gloss structure accurately and in line with natural sign word order. While Sign.mt includes additional non-manual markers and subtle timing variations, our system focuses on the gloss and sign motion, which can suffice for many communication contexts.

When testing more intermediate-level sentences such as “Do you understand me?” and “Can you help me?”, our tool produced glosses like “UNDERSTAND ME NOT” and “HELP CAN YOU ME.” These outputs maintained the semantic intent of the original sentences, even if the gloss form occasionally deviated from strict regional standards. In comparison, Sign.mt tends to follow more consistent ASL glossing conventions, whereas our outputs reflect a hybridized, internationally-inspired approach. Nonetheless, the meaning remained clear and sign-accessible.

Finally, for more complex expressions like “If I had known about the meeting, I would have attended” and “She decided to walk even though it was raining,” our glosses—“ME KNOW MEETING, ME ATTEND” and “RAIN FALLING, WALK DECIDE SHE”—demonstrate the tool’s ability to reduce English input into simplified, conceptually accurate sign language structures. While these glosses omit finer grammatical constructs like conditional clauses or temporal markers, they preserve the essential message, consistent with simplified International Sign glossing conventions. In comparison, Sign.mt produces more language-specific glosses but similarly reduces sentence complexity.

Overall, this comparison is intended to contextualize the strengths and current limitations of our tool rather than provide a competitive benchmark. While currently lacking non-manual markers and some grammatical detail, our gloss outputs are consistently intelligible and aligned with expected sign language structure. These findings reinforce the potential of our system to support accessible sign language translation, particularly in multilingual or informal settings where simplified International Sign glosses are suitable

### **3.2.3 Summary of Findings**

.

| **Sentence Type** | **English Input** | **Gloss (Your Tool)** | **Native Signer / Ideal Gloss** | **Notes on Accuracy** |
| --- | --- | --- | --- | --- |
| Simple | What is your name? | YOU NAME WHAT | YOU NAME WHAT | Accurate and natural |
| Simple | How are you? | HOW YOU | HOW YOU | Accurate; typical ISL structure |
| Intermediate | I do not understand | UNDERSTAND ME NOT | ME NOT UNDERSTAND | Acceptable; minor variation in order |
| Intermediate | Can you help me? | HELP CAN YOU ME | YOU CAN HELP ME | Clear meaning; word order slightly off |
| Complex | If I had known about the meeting, I would have attended | ME KNOW MEETING, ME ATTEND | IF ME KNOW MEETING, ME WILL GO | Simplified; missing conditional marker |
| Complex | She decided to walk even though it was raining | RAIN FALLING, WALK DECIDE SHE | RAIN STILL SHE DECIDE WALK | Lacks contrastive element ("STILL"); meaning preserved |

## 3.3 Summary and Conclusions

A key area for future development is the creation or integration of a system capable of rendering real-time 3D models performing sign language. Currently, the reliance on pre-rendered video clips limits the fluidity and expressiveness of the output. Enhancing this capability would allow for more natural and dynamic signing, improving user engagement and comprehension.

Additionally, the existing dataset covers only a limited vocabulary, necessitating frequent use of finger spelling for words not included in the collection. Expanding the dataset to include a broader range of commonly used words and phrases is essential to reduce dependence on finger spelling, which can be slow and less efficient for communication.

Furthermore, although ChatGPT played a key role in translating English text into ISL gloss, it is not specifically designed or optimized for sign language translation. This limitation resulted in inconsistent and sometimes inaccurate gloss outputs that did not fully conform to ISL grammar or vocabulary. The model occasionally generated signs that were not present in our video dataset, which complicated video retrieval and affected the fluency of the final output. To address these challenges, developing a dedicated sign language translation model trained on extensive sign language datasets would likely improve both the accuracy and naturalness of gloss generation. Such a specialized model could better capture the unique grammatical structures and semantic nuances of ISL, thereby enhancing the overall reliability and effectiveness of the system.

Working on SoundSigns gave us practical experience in full-stack development, system integration, and accessibility-focused design. We learned how to critically evaluate third-party tools, balance technical feasibility with user needs, and address unexpected constraints such as limited datasets and documentation gaps. Most importantly, we deepened our understanding of the challenges involved in building inclusive technologies — a perspective we will carry into future projects.

# 4. User Guide

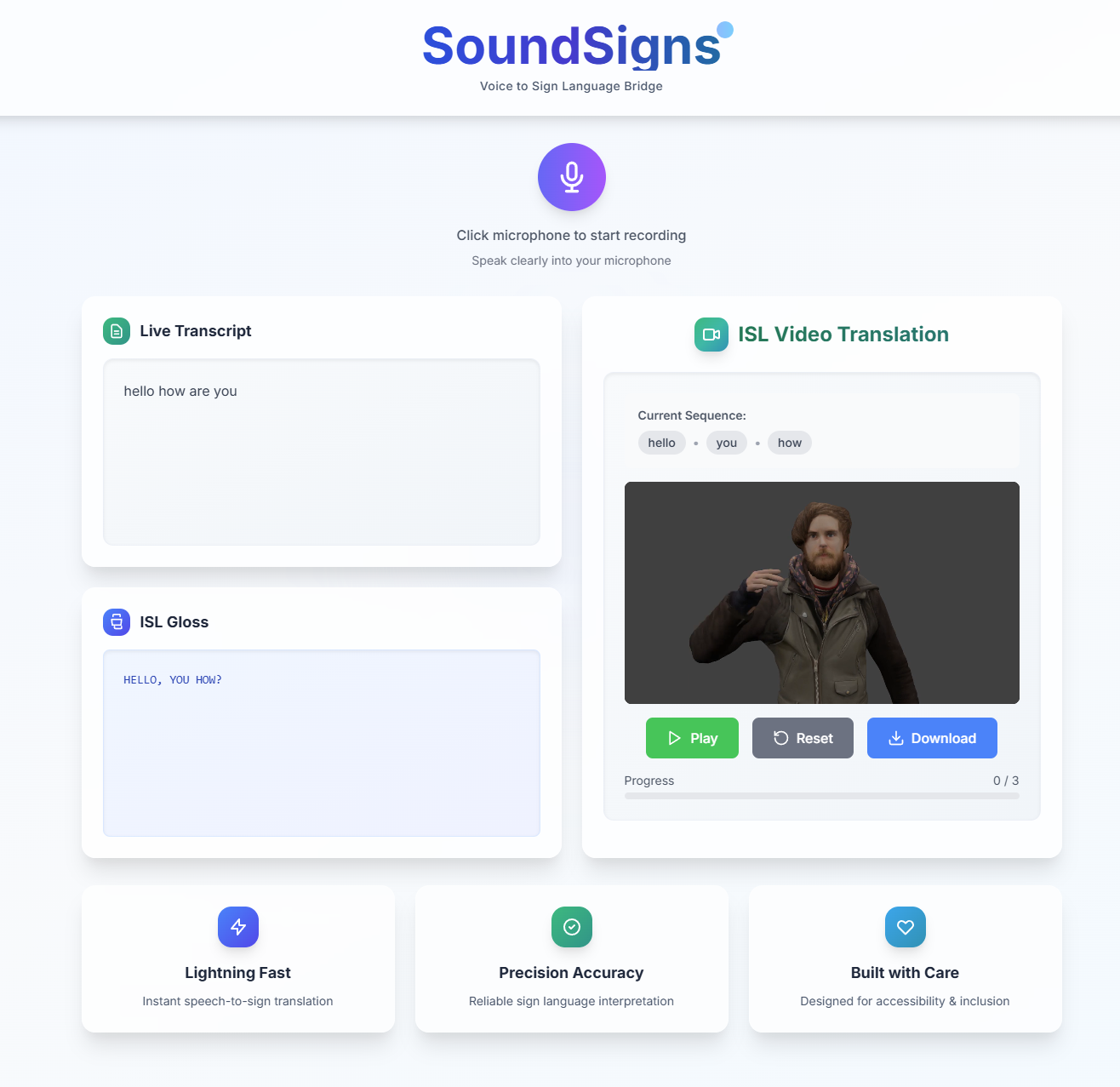


Figure 3: Home Page

**4.1 Interface Overview**

The entire application is accessible through a single screen, designed to be simple and intuitive. The main components are:

* Microphone Button
* English Transcription Text Area
* ISL Gloss Text Area
* Video Player with Sign Language Playback
* Playback and Download Controls

**4.2 Using the Microphone Button**

* Starting a Recording:  
   To begin, click the microphone button prominently displayed on the left side of the screen. This activates the browser’s built-in speech recognition, allowing you to speak in English naturally.
* Automatic Detection of Speech End:  
   The system automatically detects when you pause or stop speaking and ends the recording. There is no need to manually stop the recording.
* Real-Time Transcription Display:  
   As you speak, or immediately after, your words are converted to text and displayed in the English Transcription Text Area below the microphone button. This allows you to confirm what the system has understood.
* Error Handling:  
   If speech is unclear or not detected, the system may prompt you to try again, or the text area may remain empty. Ensure your microphone is enabled and your speech is clear for best results.

**4.3 Viewing the Translated ISL Gloss**

* Gloss Translation:  
   After transcription, the backend sends the English text to the translation model, which converts it into International Sign Language (ISL) gloss. This gloss is a simplified text representation of sign language syntax.
* Display:  
   The translated gloss appears in its dedicated text area next to the English transcription. This gives users a textual preview of how the original sentence is broken down into signs.  
   Displaying gloss helps users familiar with sign language understand the structure and flow of the translation before watching the avatar perform the signs.

**4.4 Sign Language Video Playback**

* Video Player Location:  
   To the right of the text areas is the video player that displays the sign language translation.
* Synchronized Gloss Highlighting: Above the video, each gloss word lights up in synchronization with the avatar performing its corresponding sign. This feature allows users to follow along visually and understand the timing of each sign.
* Pre-rendered 3D Videos: The video consists of a sequence of pre-rendered MP4 clips showing a high-quality 3D avatar performing each sign. These clips are played to create a coherent and accurate signed message.

**4.5 Playback Controls**

* Play Button:  
   Pressing this button starts playing the sign language video from the current position.
* Replay Button:  
   This restarts the entire video from the beginning, allowing users to watch the translation again.
* Download Button:  
   Users can download the full sign language video as a single file for offline viewing, sharing, or educational purposes.

**4.6 Restarting the Translation Process**

* To translate a new sentence or phrase, simply click the microphone button again. This clears the previous transcription, gloss, and video.
* The application then restarts the process, allowing you to record new speech, view the transcription, and watch the new sign language translation.

**4.7 Additional Features and Tips**

* Browser Compatibility:  
   The application works best on modern browsers that support the Web Speech API, such as Google Chrome and Microsoft Edge. Some features may not work properly on older or unsupported browsers.
* Microphone Permissions:  
   When first accessing the application, your browser will request permission to use the microphone. Please allow this to enable speech recognition.
* Audio Quality:  
   For the best transcription accuracy, use a good quality microphone and speak clearly in a quiet environment.
* Limitations:  
   Currently, the dataset covers only a limited vocabulary. Words not included in the dataset will be spelled out letter-by-letter using finger spelling videos, which may slow down the translation.
* Feedback:  
   If you encounter any issues or inaccuracies in the translation, restarting the application or refreshing the page can often resolve temporary glitches.

**4.8 Summary of Workflow**

1. Click the Microphone Button and speak your sentence in English.
2. Watch the English Transcription to confirm your speech was captured.
3. View the ISL Gloss text that represents the translation structure.
4. Use the Video Player to watch the avatar perform the signed translation, following the highlighted gloss words above.
5. Use the playback controls to play, Replay, or download the video.
6. Repeat the process by clicking the microphone again for new translations.

# 5. Maintenance Guide

This guide outlines the necessary steps and best practices for setting up, running, and updating both the backend and frontend components, as well as securing sensitive information and managing dependencies effectively.

## 5.1 System Requirements

To successfully run and maintain the application, the following software prerequisites must be met:

* **Python 3.8 or higher** — Required for the backend server and API interactions.
* **Node.js version 14 or higher** — Necessary for the frontend development environment.
* **OpenAI API Key** — Enables access to the GPT-based language translation services.

## 5.2 Backend Setup and Maintenance

The backend is implemented in Python using the Flask framework and serves as the interface for natural language processing and API communication.

**Installation of Dependencies:** Ensure all required Python packages are installed by running:

pip install sounddevice numpy openai flask flask-cors python-dotenv

These libraries provide audio processing, API handling, and web server functionality.

**API Key Configuration:** The OpenAI API key must be stored securely in an .env file located in the backend directory. This environment file should contain:

OPENAI\_API\_KEY=your\_openai\_key\_here

For security, never commit this file to public repositories.

**Running the Backend:** To start the backend server, execute the transcription script from the root directory:

py backend/transcription.py

Regular monitoring and updating of dependencies are recommended to maintain compatibility and security.

## 5.3 Frontend Setup and Maintenance

The frontend, built with React and styled using Tailwind CSS, manages user interactions, audio recording, and video playback.

**Dependency Installation:** Navigate to the frontend directory and install all necessary Node.js packages:

cd frontend

npm install

**Running the Frontend:** Launch the development server by executing:

npm run dev

This will start the React application locally, enabling live testing and interface updates.

## 5.4 Folder Structure

The project follows a modular and hierarchical directory structure to promote maintainability and clarity:

project-root/

├── backend/

│ ├── .env

│ └── transcription.py

├── frontend/

│ ├── src/

│ │ ├── components/

│ │ ├── App.jsx

│ └── assets/

│ └── videos/

│ ├── letters/

│ ├── numbers/

│ └── words/

The project is organized into a clear and modular directory hierarchy to facilitate development, maintenance, and scalability:

* backend/: Contains the server-side components. The .env file holds environment-specific variables, in our case, an API key, which is kept private and excluded from version control. The transcription.py script implements the core backend logic for converting English text into International Sign Language gloss.
* frontend/: Houses the client-side application built with React. The src/ folder contains the React source code, including reusable UI components located in components/ and the main application file App.jsx.
* assets/videos/: Located within the frontend, this folder stores pre-rendered MP4 videos categorized by content type — letters/ for individual alphabet signs, numbers/ for numerical signs, and words/ for commonly used vocabulary. This structure supports efficient video retrieval and playback by the avatar during ISL translation.

This organized structure ensures the separation of concerns between frontend and backend code, enables straightforward updates to video assets, and supports maintainable, scalable development practices.

# 6. References

[1] JS-Coderr. (2024). *Text-Speech to Sign Language Generator* [GitHub repository]. GitHub.<https://github.com/JS-Coderr>

[2] Moryossef, A. (2023). sign.mt: Effortless Real-Time Sign Language Translation [Web application]. <https://sign.mt/>

[3] Yeemachine. (2022). Kalidokit (Version 1.1.5) [Computer software]. GitHub. <https://github.com/yeemachine/kalidokit>

[4] Yu, Z., Huang, S., Cheng, Y., & Birdal, T. (2024)**.** *SignAvatars: A large‑scale 3D sign language holistic motion dataset and benchmark* (arXiv:2310.20436) [Preprint]. arXiv.<https://arxiv.org/abs/2310.20436>