Documentation for Sign Language Translator Project

Table of Contents

1. Introduction

The Sign Language Translator project is a groundbreaking initiative designed to facilitate communication between individuals who use American Sign Language (ASL) and those who do not. By translating ASL into text and speech, this project aims to make interactions more accessible and inclusive for everyone. The translator utilizes sophisticated machine learning models and APIs to provide accurate, real-time translations, thus fostering better understanding and interaction across different communities.

Features

- **User-Friendly Interface**: Designed for ease of use on both desktop and mobile platforms, making it accessible to a wide range of users.
- **API Integration**: Leverages state-of-the-art APIs to enhance the accuracy and efficiency of translations.
- **Real-time ASL Fingerspelling Detection**: Detects and translates ASL fingerspelling gestures into text in real-time.
- **Text-to-Sign Conversion**: Converts input text into fingerspelling animations, bridging communication gaps for non-ASL speakers.
- **Integrated Speech System**: Converts text to speech, making the communication output audible.
- **Lightweight and User-Friendly**: Optimized for accessibility and ease of use.

2. User Guide

Installation Instructions

Prerequisites:

To set up the Sign Language Translator, ensure you have the following:

Backend

- **Python**: Version 3.9 or higher. 3.11 considered as best.
- **Libraries**: MediaPipe, scikit-learn, OpenAI API, and ElevenLabs API.

Frontend

- **HTML**: Used for the basic structure of the web app.
- **CSS**: Used to make the web app's UI look professional and Good.
- **Javascript**: Used for Function calling and make the buttons responsive and do something integrating it to the flask app.

Step-by-Step Installation:

1. **Clone the Repository**: Use GitHub to clone the project repository to your local machine.

   ```
   git clone https://github.com/JashanMaan28/Sign-Language-Translator-Fingerspelling-Detector.git
   ```

   Or You could just download the zip file from our GitHub and unzip it. Then go to the folder containing all the components. Skip to Step - 3 after this.

2. **Navigate to Directory**: Open the terminal and navigate to the project directory.

3. **Install Dependencies**: Execute the command using the Terminal:

   ```
   MAKE SURE YOU HAVE PYTHON INSTALLED.
   ```

   ```
   pip install -r requirements.txt
   ```

   This will install all the necessary libraries.

4. **Configure API Keys**: Set up your API keys for OpenAI and ElevenLabs in the " .env " file (You need to create one). Look at .env.example for refrences.

5. **Run the Application**: Launch the application by going to the UI folder and running:

   ```
   python app.py
   ```

Usage

Running the Translator:

- **Launch the Application**: Launch the app.py file and it will provide you with a address mostly " http://127.0.0.1:5000/ " and paste it in your desired browser (Google Chrome Preffered for best Performance).
- **Use Webcam for Signing**: The application captures ASL gestures through your webcam.
- **View Translations**: The translated text will be displayed on the screen, and speech output will be generated on Respective Button Clicks as shown in the demo video at our GitHub Repository.

Example Use Cases:

- **Facilitating Communication**: Helps non-sign language users understand ASL.
- **Educational Tool**: Assists learners in practicing and improving their ASL skills.
- **Real-Time Translation**: Works as a effective Translation app between you and a deaf or mute person.

Troubleshooting

Common Issues:

- **Camera Detection**: Ensure your webcam is connected and functioning. Check system permissions if the camera is not detected.
- **API Errors**: Confirm that your API keys are correctly entered and that the APIs are accessible from the .env file.

FAQs:

- **Resetting the Application**: If needed, you can restart the application by closing and reopening it from the terminal or your code editor.
- **Multiple Sign Languages**: Currently, the translator supports ASL only. Future updates may add support for other sign languages if we get enough support.

3. Technical Details

Architecture

The system architecture of the Sign Language Translator is composed of several interconnected components:

- **Input Capture**: Uses Cumputer Vision to capture video input and process hand gestures via mediapipe.
- **Machine Learning Model**: A RandomForestClassifier from the scikit-learn library trained on a dataset of ASL gesture mediapipe landmarks to predict corresponding text.
- **API Integration**: Integrates OpenAI API for text processing and ElevenLabs API for converting text to speech.

Technologies Used

- **MediaPipe**: Provides real-time hand tracking and landmark detection to accurately capture sign language gestures.
- **scikit-learn**: Utilized for developing and training the machine learning model.
- **OpenAI API**: Enhances the text processing capabilities to improve translation accuracy.
- **ElevenLabs API**: Converts the text output into speech, enabling audible translations.

Model Training

Data Collection and Preprocessing:

- **Data Collection**: ASL gesture data is collected using the Open-cv and MediaPipe framework, capturing key landmarks of hand positions.
- **Preprocessing**: The data is normalized, augmented and cleaned to ensure consistency and reduce noise, improving the model's performance.

Model Selection:

- **RandomForestClassifier**: Selected for its balance between performance and simplicity, offering robust accuracy in classifying ASL gestures.

Evaluation Metrics:

- **Accuracy**: Measures how often the model correctly predicts the sign language gestures.
- **F1 Score**: Provides a balanced measure of precision and recall, offering insights into the model's performance across different classes.
- **Scores**: Our model achieved a score of 99.8% using our dataset.

4. Integration with APIs

OpenAI API

Purpose: The OpenAI API is used to process the text output from the machine learning model, ensuring that the translations are contextually accurate and coherent.

Usage:

- **Text Processing**: The API is called during the text translation phase to refine and enhance the output.
- **Example**:

```
response = openai.Completion.create(
    engine="text-davinci-003",
    prompt="Translate the following ASL signs to text:",
    input=data
)
```

ElevenLabs API

Role: The ElevenLabs API is responsible for converting the text output into speech, providing an auditory translation of the ASL gestures.

Usage:

- **Text-to-Speech Conversion**: The API generates speech output from the translated text, making the communication audible.
- **Example**:

```
speech = elevenlabs.Speech.create(
    text="Hello, how can I help you?",
    voice="Joanna"
)
```

Note: Those were just examples and do not comply with actual code. To view the actual code give visit to their files in the source code at GitHub.

5. UI/UX Design

User Interface

The user interface of the Sign Language Translator is designed to be straightforward and accessible, catering to users of all technical abilities. Key elements include:

- **Webcam View**: Displays the real-time video feed of the user signing in ASL, allowing immediate feedback.
- **Text Output Section**: Prominently shows the translated text, making it easy for users to read the output.
- **Audio Output Button**: Provides a simple button to play the speech output, ensuring that the translation can be heard as well as seen.

User Experience

- **Intuitive Navigation**: The application features a streamlined interface, reducing the learning curve for new users.
- **Responsive Design**: The interface is optimized for both desktop and mobile devices, ensuring a consistent experience across different platforms.

6. Future Improvements

Looking ahead, the Sign Language Translator project plans to introduce several enhancements:

- **Support for Multiple Sign Languages**: The system will be expanded to include translations for other sign languages such as British Sign Language (BSL), Indian Sign Language (ISL), Internation Sign Language (ISL) and many others.
- **Enhanced Model Accuracy**: By incorporating larger datasets and utilizing advanced algorithms, the accuracy of the translations will be improved.
- **Offline Mode**: An offline version of the application will be developed, allowing users to access translations without an internet connection, increasing its accessibility.
- **Support of Multiple spoken Languages**: The system will be expanded to include multiple languages other than just english to make sure including and helping a global audience.

7. Conclusion

The Sign Language Translator is a pioneering project that significantly contributes to breaking down communication barriers between sign language users and non-users. By transforming ASL gestures into text and speech, it promotes inclusivity and facilitates better interaction in diverse settings. This documentation provides a thorough guide for both users and developers, detailing every aspect of the project's installation, usage, and technical framework, along with plans for future developments.

8. Appendices

Glossary

- **ASL**: American Sign Language, a visual language used primarily in the United States and Canada also similar to International Sign Language which is used worldwide.
- **MediaPipe**: A framework developed by Google for building multimodal perception pipelines.
- **scikit-learn**: A machine learning library in Python, providing simple and efficient tools for data mining and data analysis.

References

- **MediaPipe Documentation**: https://mediapipe.dev/
- **scikit-learn Documentation**: https://scikit-learn.org/stable/
- **OpenAI API**: https://openai.com/api/
- **ElevenLabs API**: https://elevenlabs.io/api/

Code Snippets

- **MediaPipe Integration**:

```
import mediapipe as mp
hands = mp.solutions.hands.Hands()
```

- **scikit-learn Model Training**:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)
```