Sign Language Decoder

Team Members: Sidharth Uppuluri, Vivek Krishnagiri,

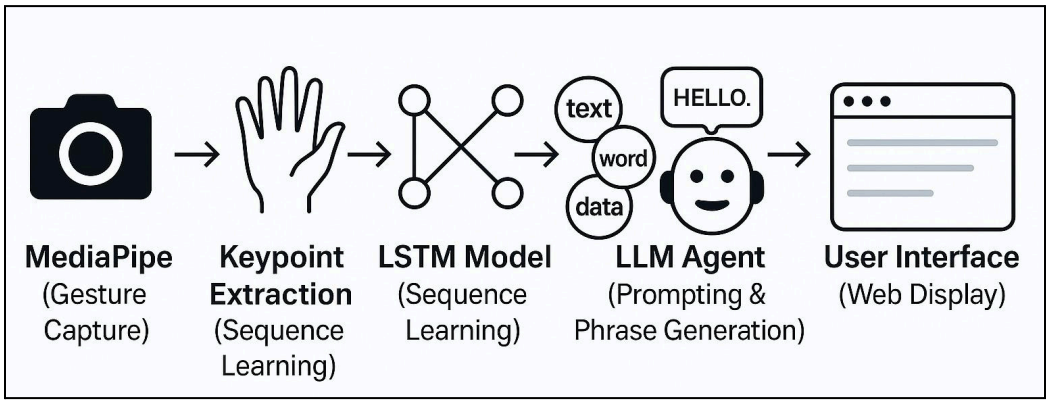
Abhiram Menon, Jonathan Tang

Sponsor: Chad Price (crprice@asu.edu) - ASU SAILS

04/29/25

Technical documentation

Architecture Diagram



File navigation

File	Purpose
`config.py`	Central hub for all parameters like which gestures are supported, where files are stored, and model settings.
`data_collection_refined.py`	Used to record new gesture data using the webcam. It stores the gesture data as numerical files for model training.
`model_refined.py`	Trains the gesture recognition model using previously collected data. This produces the `.keras` model file that makes predictions.
`realtime_detection_openai.py`	The main script that runs the live recognition system. It uses the webcam to detect gestures and convert them into full sentences.
`mediapipe_utils.py`	Contains helper functions to process video and extract landmark data using Google's MediaPipe library.
`requirements.txt`	Contains all necessary dependencies to run the project.

Installation Process

System Requirements:

- A computer with a webcam running Windows, macOS, or Linux.
- Python (version 3.7 to 3.10 recommended).
- A stable internet connection (for sentence generation using OpenAI).

Setup Process:

1. **Install Python**

   Download from https://www.python.org/downloads/
2. **Download the Project Files** - https://github.com/siduppuluri/ActionDetection

   Clone the repository or place all project files in a single directory.

Create a Virtual Environment (Optional but Recommended)

This keeps dependencies clean and separate. Run:

```
python -m venv venv
source venv/bin/activate  # On Mac/Linux
venv\Scripts\activate     # On Windows
```

Install Required Libraries
Run the following to install all necessary packages:

```
pip install requirements.txt
```

3. **Add OpenAI API Key**

   You will need a valid API key from OpenAI (found at https://platform.openai.com/account/api-keys).

   Store it in your environment:

   - On Mac/Linux:

     ```
     export OPENAI_API_KEY=your_key_here
     ```

   - On Windows (Command Prompt):

```
set OPENAI_API_KEY=your_key_here
```

Licenses

Software Component	License Type
MediaPipe (by Google)	Apache 2.0
TensorFlow	Apache 2.0
OpenCV	BSD 3-Clause
OpenAI API	Commercial (terms set by OpenAI)
This Project	Internal-use license

Libraries

All the software libraries used are publicly available and widely supported. The main libraries used are listed below. Full list can be found in `requirements.txt`:

Library	Version	Purpose
`mediapipe`	0.10.0	Detects face, hand, and body landmarks in live video.
`opencv-python`	4.8.0	Handles camera input and display of video.
`tensorflow`	2.12.0	Powers the machine learning model (LSTM).
`openai`	1.6.0	Connects to OpenAI's GPT-4 for sentence generation.
`numpy`	1.24.3	Handles numerical data and arrays.
`scikit-learn`	1.3.0	Used for model evaluation (accuracy, confusion matrix).

API Use

OpenAI GPT-4 API

- Used for translating a list of signed words into grammatically correct sentences.
- The code ensures that only the recognized words are used in the sentence (no extra or made-up words).
- A working internet connection is required.
- This API requires a valid key from OpenAI and may incur usage costs depending on the plan.

Best practices

Code updates and maintenance

- Modular code structure allows isolated updates (e.g., you can improve model training without touching the webcam code).
- Each script can be tested independently.
- Use comments and logging to make changes traceable.

APIs, Libraries updates and maintenance

- Revisit `pip list --outdated` every 3–6 months.
- Update TensorFlow and MediaPipe only after confirming compatibility with your hardware (especially GPU).
- Monitor OpenAI documentation for changes to API limits or billing.

Version Control and Repository Management

- Store the code on GitHub.
- Use branches for development and main for stable versions.
- Include `README.md`, `LICENSE`, and `.gitignore` files.

Licensing management (Future Consideration)

- Keep a record of each external library's license in a `third_party_licenses.txt` file.
- Include a license for your own code if sharing externally (e.g., MIT License).

Hosting and domain services (Future Consideration)

- Currently runs locally on the user’s machine.

- If expanded to a public-facing system, consider deploying to a web platform (e.g., using Streamlit, Flask, or WebRTC for camera access).
- Domain hosting may be needed for a web version.

Software distribution platforms

- Can be bundled into a standalone app using:
    - `PyInstaller` – creates an executable that runs without installing Python.
    - `cx_Freeze` – another good option for creating desktop apps.
- This is helpful for classrooms where installations need to be simple.

Data and Storage

- All gesture data is stored as `.npy` (NumPy array) files under the `MP_Data/` folder.
- Each gesture (e.g., "hello", "thanks") has its own folder with 30 short recordings.
- The trained model is saved as `action.keras`.

User guide

How the system works

The ASL Decoder application is designed to translate American Sign Language (ASL) gestures into real-time, readable text using AI-driven computer vision technology. It operates through the following workflow:

- **Input Capture:** The system captures live video input of ASL gestures via webcam.
- **Gesture Recognition:** Utilizing MediaPipe Holistic, the application detects and extracts keypoints (hand, face, body) from video frames.
- **Data Processing:** The extracted keypoints are processed by an LSTM-RNN model to identify and classify gestures.
- **Real-Time Translation:** Recognized gestures are then translated into textual form, enhanced through integration with OpenAI's ChatGPT API, generating grammatically correct and contextually accurate English sentences.
- **Adding and Training New ASL Words:** Users can input new gestures by performing them clearly in front of the webcam. The system records 30 frames per gesture to ensure temporal accuracy and consistency. These frames are saved and integrated into the training dataset. Users then retrain the model to improve recognition accuracy.
- **Output Display:** Translated sentences are displayed on a user-friendly interface.

Goals and functionality - EPICS/User Stories

Primary Goals:

- Enable seamless communication for individuals who use ASL.
- Provide a reliable tool to reduce dependence on interpreters.
- Integrate smoothly into educational and social environments.

EPICS/User Stories:

- **As a Deaf student,** I want real-time translation of my ASL gestures so that I can participate actively in classroom discussions.
- **As a non-ASL teacher,** I need accurate translations of ASL into text to communicate effectively with ASL-using students.

- **As a healthcare provider,** I want the app to help me understand signed descriptions of symptoms from patients accurately and quickly.

Major Functionalities:

- Real-time video processing and gesture recognition.
- Robust accuracy in gesture-to-text translations (>90% accuracy).
- Accessible user interface.
- Integration with OpenAI for enhanced linguistic accuracy.

Input & Outputs

Inputs:

- Live Video Stream: Webcam input capturing ASL gestures.
- Gesture Dataset: Pre-collected and preprocessed keypoint data (.npy format).
- User Gestures: Real-time gestures performed by the user.

Outputs:

- Real-Time Text Translation: ASL gestures translated into readable English sentences.
- Enhanced Text Outputs: Grammatically correct sentences via ChatGPT integration.
- User Interface Display: Continuous display of translated text alongside video input.

Services/Features

- Real-time Gesture Recognition: Immediate identification of common ASL phrases and finger-spelling.
- Natural Language Processing: Leveraging ChatGPT for contextually accurate translations.

Transferring ownership

Code Repository

Location of Code:
The project is currently hosted on the platform GitHub. The sponsor will have full and complete access to the repository of the project. This can then be cloned locally to successfully deploy the application locally.

This is the easiest way to clone the repository.

- Go to the GitHub link: (`https://github.com/siduppuluri/ActionDetection`).
- On the top right of the page, click the green "Code" button.
- In the dropdown, click "Download ZIP".
- A ZIP file of the entire project will be downloaded to your computer.
- Once downloaded, double-click the ZIP file to unzip it.
- You now have all the project files. You can open the folder and follow the installation process to get started.
- **Documentation Included**: This transfer includes:
    - All Python source files
        - `config.py`
        - `data_collection_refined.py`
        - `model_refined.py`
        - `realtime_detection_openai.py`
        - `mediapipe_utils.py`
        - `requirements.txt`
    - Trained model file (`action.keras`)
    - All collected gesture data in MP_DATA
    - README and Deployment Documentation

Next Steps for Sponsor:

- Maintain a secure copy of the repository.
- If using GitHub, set repository visibility (public/private) as desired.
- Assign a designated technical point-of-contact to manage any future updates.

Database Format

There is no traditional database (like MySQL or MongoDB) in use. Instead, data is stored locally as `.npy` files using NumPy. These are structured as:

```
MP_Data/
  ├── hello/
  │    ├── 1/
  │    │    ├── 0.npy
  │    │    └── ...
  └── thanks/
       ├── 1/
       │    └── 0.npy
```

Contents:

- Each `.npy` file is a short "snapshot" of hand/pose landmark data from a single video frame.
- These are used to train and test the gesture recognition model.

Recommendation:

Backup the `MP_Data` directory. This contains the entire dataset used for model training. It can be expanded or retrained later if additional signs are to be added.

Algorithms and Model Used

Machine Learning Model

- **Model Type:** LSTM (Long Short-Term Memory), a type of neural network optimized for time-series data like video frames.
- **Trained Model File:** `action.keras`
- **Input:** A sequence of 30 frames worth of body and hand landmark coordinates.
- **Output:** A prediction of the performed gesture (e.g., “hello”).

Supporting Algorithm Details

- **Preprocessing Pipeline:** Uses Google MediaPipe to extract coordinates for the face, pose, and hands.
- **Postprocessing:** Aggregates predictions, filters for confidence and consistency, and passes results to OpenAI for sentence generation.

How to Train Data

- Run `data_collection_refined.py` to add new signs. Words will be detected from the `actions` parameter in `config.py`.
- Then run `model_refined.py` to retrain the model.
- New model overwrites the existing `action.keras` file.

Recommendation:

Do not delete the training scripts, as they allow future extension of the vocabulary.

Server Credentials

Note:
This project does **not use a hosted server**. It runs locally on any modern personal computer with a webcam and Python installed.

If a server-based version is created in the future (e.g., for deployment to a school website or accessibility center):

- Web hosting credentials and IP addresses should be documented here.
- No server credentials are needed at this time.

Online Service Credentials

OpenAI API Key

- Required for converting recognized words into full English sentences
- This is **not hardcoded** in the script and must be set as an environment variable:
    - On Mac/Linux:

```
export OPENAI_API_KEY=your_key_here
```

    - On Windows:

```
set OPENAI_API_KEY=your_key_here
```

- Sponsor will need to:
    - Create an OpenAI account at platform.openai.com
    - Generate and manage the API key
    - Monitor API usage and billing through OpenAI's dashboard

Security Reminder:

Never share the API key publicly or store it in plain text within code files.

Contact for Ongoing Support

Should questions arise during deployment or future development, the following support options are recommended:

Contact Person	Role	Contact Info
Sidharth Uppuluri	Developer	siduppuluri12@gmail.com
Abhiram Menon	Developer	abhiram.am64@gmail.com
Vivek Krishnagiri	Developer	vivek.krishnagiri@gmail.com
Chad Price	Academic Supervisor	crprice@asu.edu