# Sign Language Translator Glove

***Abstract -*** The Sign Language Translator Glove is designed to bridge the communication gap between sign language users and non-sign language users. We use hardware such as flex sensors and the ESP-32 joined with an Android mobile application, the glove translates sign language gestures into text and speech in real time.

## Introduction

In Sprint 2, significant progress was made towards the development of our project. This sprint focused on two major milestones: the construction of the glove and being able to modify the output of existing signals.

The glove was designed and built successfully using a perf board, 100 ohm resistors and flex sensors. We soldered 5 100 ohm resistors, each in a voltage divider circuit for successful connection. Afterwards, we used zip ties and hot glue to place the perf board and the sensors for each finger on the glove. Additionally, we implemented the gesture activity. This feature enables users to interact with the application to modify specific hand gestures which are captured and processed by the glove.

These advancements in the project bring us closer to delivering a fully functional system capable of translating sign language gestures.

## Design

### Android Activities

#### Main Activity

When the user launches the app, they are brought to the Main Activity, also referred to as the Translation Page. If no glove is currently connected, a "No device connected" message is displayed. To initiate a connection, the user taps the “More Options” button, which opens a dialog listing available Bluetooth devices. Upon selecting and connecting to the glove, the app begins processing the user’s hand gestures. Recognized gestures are then visually displayed alongside their corresponding image and translated text.

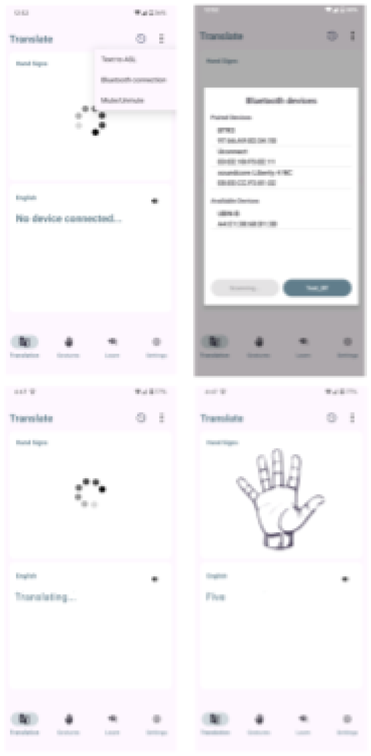
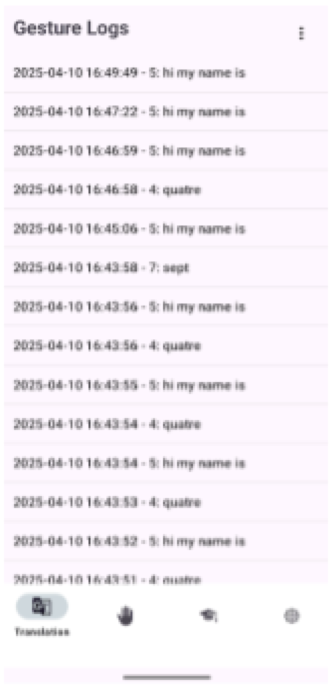


Figure 1. Translation activity wireframe

#### Gesture Logs Activity

The Gesture Logs Activity allows users to view a chronological history of their previously signed gestures. Each entry includes the gesture's label, translation, and timestamp, giving users insight into their signing habits and activity over time.



Gesture Logs

```
2025-04-10 16:49:49 - 5: hi my name is
2025-04-10 16:47:22 - 5: hi my name is
2025-04-10 16:46:59 - 5: hi my name is
2025-04-10 16:46:58 - 4: quatre
2025-04-10 16:45:06 - 5: hi my name is
2025-04-10 16:43:58 - 7: sept
2025-04-10 16:43:56 - 5: hi my name is
2025-04-10 16:43:56 - 4: quatre
2025-04-10 16:43:55 - 5: hi my name is
2025-04-10 16:43:54 - 4: quatre
2025-04-10 16:43:54 - 5: hi my name is
2025-04-10 16:43:53 - 4: quatre
2025-04-10 16:43:52 - 5: hi my name is
2025-04-10 16:43:51 - 4: quatre
```

Figure 2. Gesture Logs activity wireframe

## Text-To-ASL Activity

Accessible from the Main Activity, the Text-to-ASL Activity enables non-ASL speakers to communicate back with ASL users. Users can type or speak words and sentences, which are then visually translated into ASL gesture images. This feature supports two-way interaction and improves communication for both signers and non-signers.

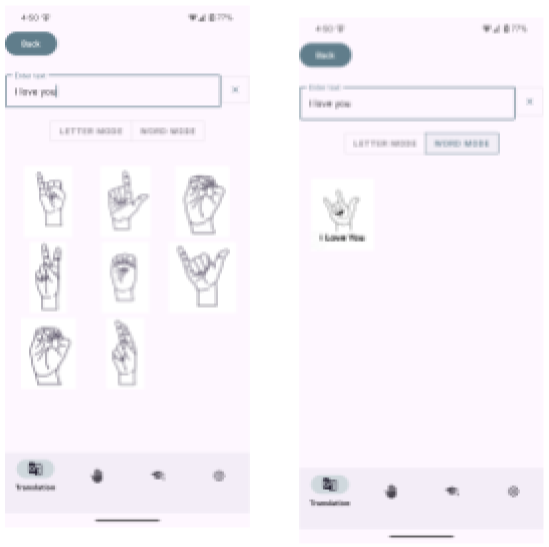


Figure 3. Text to ASL Activity

## Gesture Activity

In the Gestures Activity, users are presented with a grid of gesture cards. Each card displays an image of a gesture along with its associated translation. Tapping on a card opens a modal that allows users to customize the translation or reset it to its default meaning.

Additionally, users can create and switch between multiple custom gesture collections. This allows for different sets of gesture-to-translation mappings, useful for context-specific communication such as workplace, education, or casual settings.

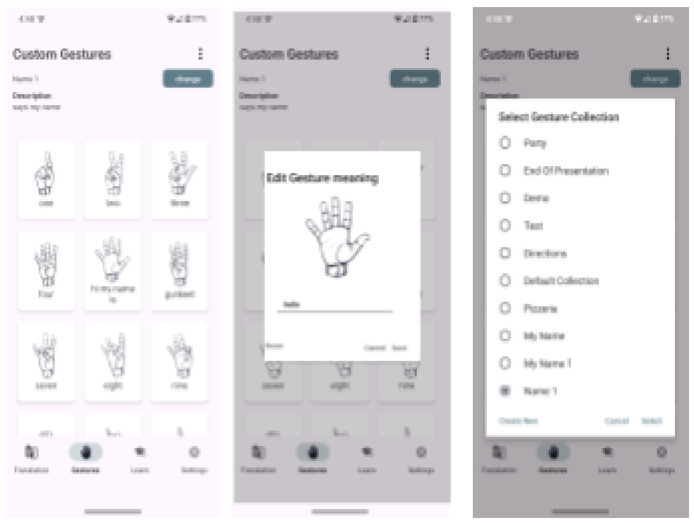


Figure 4. Gesture Activity wireframe

## Learning Activity

The Learning Activity provides an interactive environment for users to practice sign language. The app prompts the user to perform a specific gesture. If the gesture is signed incorrectly, the app provides immediate feedback and encourages the user to try again. Once the correct gesture is performed, the system acknowledges success and randomly selects the next gesture to practice, fostering an engaging and educational experience.

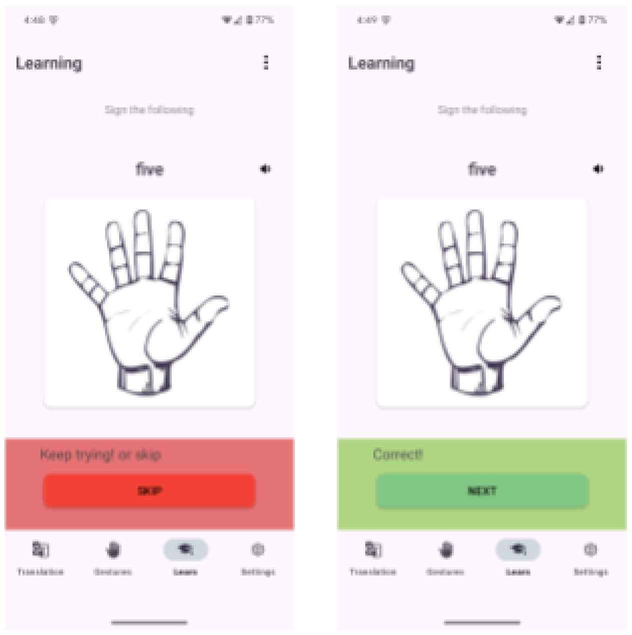


Figure 3. Learning Activity Wireframe

## Settings Activity

In the Settings Activity, users can toggle between light and dark mode and choose the default app language (English or French). Changing the language updates both the interface and the default translations for all gestures, providing a fully localized experience.

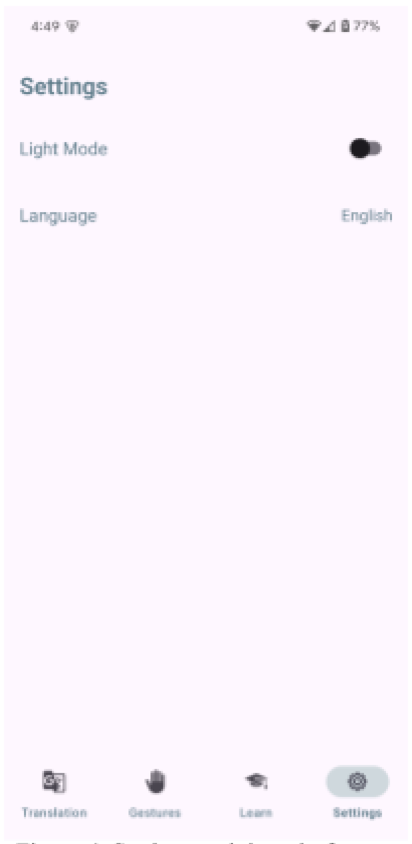


Figure 4. Settings activity wireframe

## Dark Themes

Below is the dark theme mode for some activities

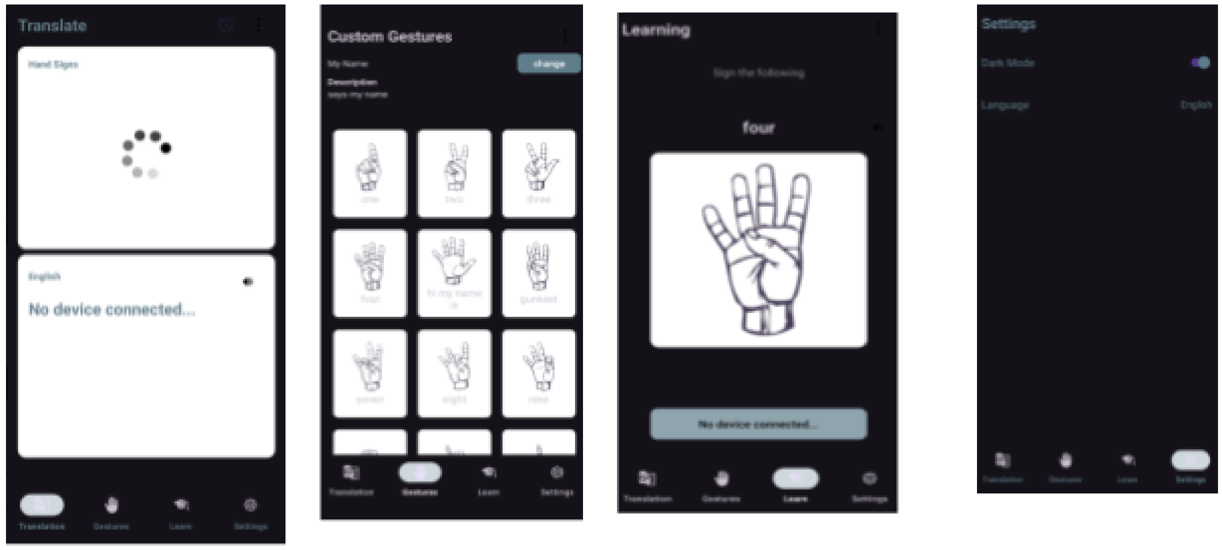


Figure 5. Activities in Dark Theme

## System Architecture

Our system architecture is composed of two interconnected components working seamlessly to deliver gesture translation functionality:

- **Hardware Component (Glove):**
  The glove integrates five flex sensors, strategically placed to measure precise finger movements. These sensors are connected to an ESP32 microcontroller, which continuously captures and digitizes the sensor data. The microcontroller then transmits this raw data in real-time to the mobile application via Bluetooth using a serial communication protocol.

- **Mobile Application (Android):**
  The mobile application acts as the central processing hub. It receives incoming sensor

data through a persistent Bluetooth connection established by the app's Bluetooth module. Once received, this data passes through a data-processing pipeline managed by the Gesture Controller. The controller ensures gesture stability using the Gesture Stability Checker and subsequently classifies stable gestures with a TensorFlow Lite machine learning model. Finally, classified gestures are integrated into various user-facing activities within the application, such as real-time translation, gesture learning sessions, and gesture history logging, providing an intuitive and interactive experience for users.

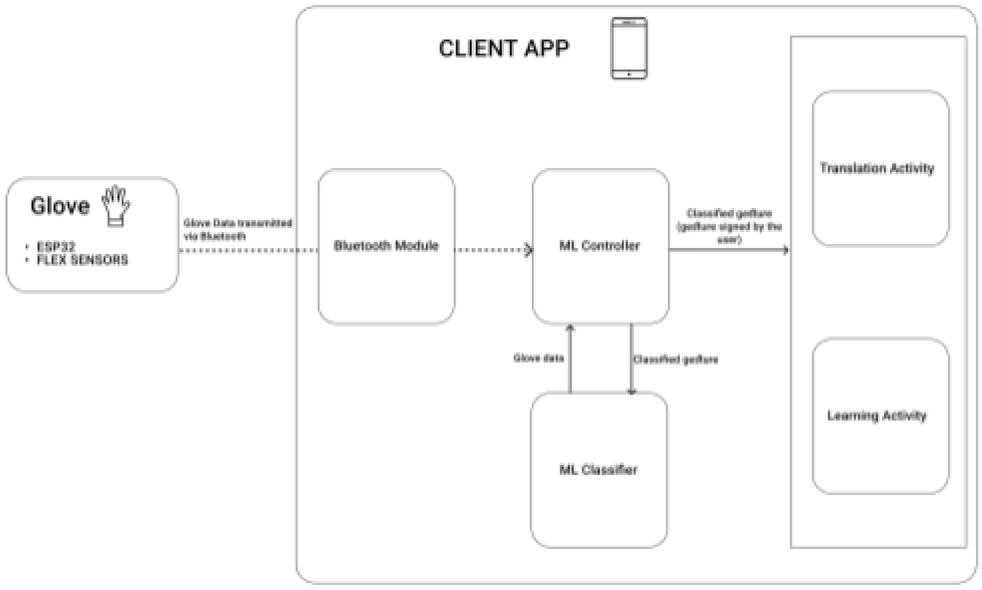


Figure 6. System Architecture for the product

## Hardware Architecture

Our hardware solution centers on a wearable glove that integrates an ESP32 microcontroller with five flex sensors, one for each finger. The flex sensors continuously monitor the bending of each finger, capturing precise movement data. The ESP32 reads the sensor outputs in real time and, with every new set of measurements, transmits this data through its integrated Bluetooth serial port. This design ensures that gesture data is delivered promptly and reliably to the mobile application for immediate processing and interpretation.

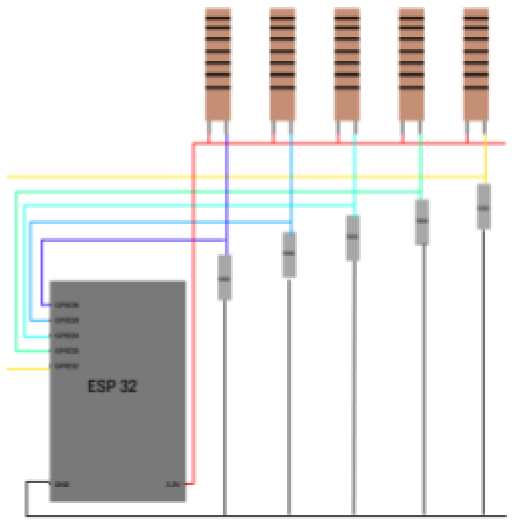


Figure 7. Glove's Circuit Diagram

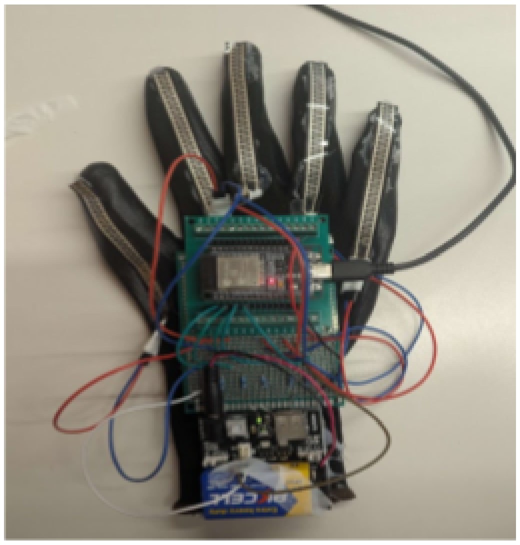


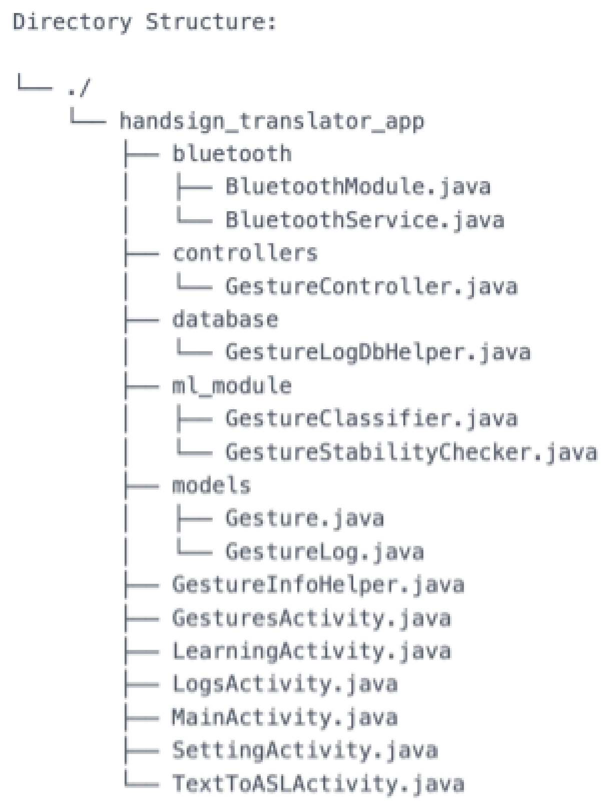
Figure 8. Image of the final Glove product

Each of the five fingers is equipped with a flex sensor, forming a voltage divider in conjunction with a 100Ω resistor. The top end of each flex sensor is tied to the 3.3 V rail, while the lower end connects to a 100Ω resistor leading to ground. The midpoint between the flex sensor and resistor is routed to an ESP32 analog input pin (GPIO36, GPIO39, GPIO34, GPIO35, and GPIO32).

This circuit was soldered onto a perfboard, ensuring stable connections and a compact form factor for the glove's embedded electronics.

## Software Architecture

Here is a breakdown of the directory structure:



```
Directory Structure:

└── ./
    └── handsign_translator_app
        ├── bluetooth
        │   ├── BluetoothModule.java
        │   └── BluetoothService.java
        ├── controllers
        │   └── GestureController.java
        ├── database
        │   └── GestureLogDbHelper.java
        ├── ml_module
        │   ├── GestureClassifier.java
        │   └── GestureStabilityChecker.java
        ├── models
        │   ├── Gesture.java
        │   └── GestureLog.java
        ├── GestureInfoHelper.java
        ├── GesturesActivity.java
        ├── LearningActivity.java
        ├── LogsActivity.java
        ├── MainActivity.java
        ├── SettingActivity.java
        └── TextToASLActivity.java
```

Figure 9. Directory Structure the Sign Language Translator Android App

Our application is organized in a modular fashion to ensure a clear separation of concerns and ease of maintenance. The codebase follows an MVC-like pattern where:

- Model: Represents the gesture data.
- View: Consists of the UI components defined in our activity classes.
- Controller: Manages the application logic, data processing, and interactions between the model and view.

Component Breakdown

**Bluetooth Module (bluetooth):**

This module is responsible for all Bluetooth-related operations.

BluetoothModule.java: Establishes a Bluetooth serial connection with the ESP32 glove, continuously reads incoming sensor data streams, and parses these streams into usable sensor values.

BluetoothService.java: A persistent Android service maintaining a single instance of the BluetoothModule throughout the app's lifecycle. Once initialized, the Bluetooth connection established in the Main Activity persists and can be reused seamlessly across multiple activities without needing to reconnect.

**Gesture Controller (controllers/GestureController.java):**

The Gesture Controller acts as the central coordinator of the system. It acquires real-time sensor data from the Bluetooth module, maintains a sliding window of recent sensor readings, and uses the GestureStabilityChecker to ensure the stability of gestures before processing. Upon detection of a stable gesture, it passes the data to the GestureClassifier, which translates it into a recognizable gesture. Results are then communicated to the user interface via callback methods.

**Machine Learning Module (ml_module/):**

GestureClassifier.java: Utilizes a TensorFlow Lite model, converted from a neural network trained using TensorFlow and Keras, to classify gestures based on the readings from five flex sensors.

GestureStabilityChecker.java: Analyzes sensor data within a sliding window to confirm gesture stability, preventing premature or incorrect gesture classifications caused by temporary sensor fluctuations.

**Model (models/Gesture.java):**

The Gesture model encapsulates classification results, holding both the textual translation and the corresponding image used for visual representation.

**Helper Classes:**

GestureInfoHelper.java: Manages gesture metadata, including translations and associated image resources, by loading them from CSV files. This information maps classification results to user-friendly representations.

database/GestureLogDbHelper.java: Provides an interface to a local SQLite database used for logging gestures previously signed by the user, maintaining a historical record for review.

**Activities:**

MainActivity.java: Also referred to as the "Translation Activity," this primary screen enables users to establish Bluetooth connections with the glove. Once connected, it leverages the Gesture Controller to translate and visually display recognized gestures in real-time.

GesturesActivity.java: Enables users to manage and switch between different custom gesture collections, allowing personalized gesture definitions for various use cases or contexts.

LearningActivity.java: Facilitates sign language practice by prompting users to perform specific gestures. It provides immediate feedback, indicating whether gestures are performed correctly or if additional attempts are needed, subsequently proceeding to new randomly selected gestures for continuous practice.

LogsActivity.java: Displays a chronological history of previously recognized gestures, allowing users to review past interactions.

TextToASLActivity.java: Allows users to input words or sentences via text, subsequently displaying corresponding ASL gesture images, aiding communication and learning through visual translation.

## Use Cases and Sequence Diagrams

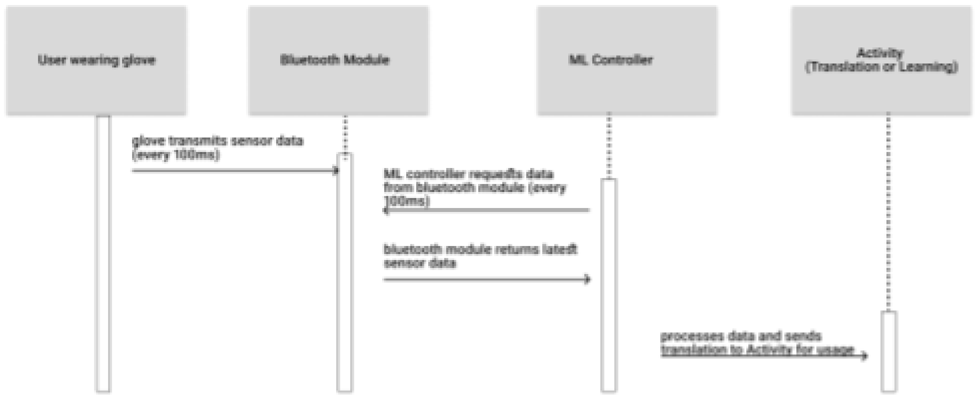

User wearing glove
Bluetooth Module
ML Controller
Activity (Translation or Learning)
glove transmits sensor data (every 100ms)
ML controller requedts data from bluetooth module (every 100ms)
bluetooth module returns latedt sensor data
processes data and sends translation to Activity for usage

Figure 10. Sequence diagram: App translates gesture done with the embedded glove

Figure 4 illustrates the real-time gesture data flow within our Sign Language Translator system. Initially, the smart glove, equipped with flex sensors, continuously measures finger movements, transmitting sensor readings approximately every 100 milliseconds. These real-time data streams are wirelessly sent to the mobile application via Bluetooth, ensuring rapid and accurate gesture detection.

On the mobile device, the Bluetooth Module continuously receives and caches these sensor readings, making the data instantly accessible. The Gesture Controller periodically polls the Bluetooth Module at short intervals (around 100 milliseconds), ensuring minimal latency and providing fresh sensor data for gesture processing.

Upon receiving the latest sensor values, the Gesture Controller assesses the stability of the gestures using the Gesture Stability Checker. Once a gesture is deemed stable, the controller forwards the sensor data to the Gesture Classifier, where a trained machine learning model translates these sensor patterns into recognized gestures.

The recognized gesture, along with its corresponding translation, is then sent to the active user-facing activity, such as the Main Activity (for real-time gesture translation) or the Learning

Activity (for interactive sign practice sessions). These activities immediately display the result to the user, potentially accompanied by additional feedback mechanisms like text-to-speech output or visual representations. Through this continuous and responsive loop of data acquisition, classification, and user feedback, the application achieves near-instantaneous interpretation and interactive engagement with sign language gestures.

## Testing

**Story ID: HW-01**

We want to build a basic glove prototype with all sensors and board connected to the glove.

| Test Case 1 | | |
|---|---|---|
| Pre-condition: Ensure the voltage divider circuit is connected to the perf board successfully and the sensors are soldered with the perf board. | | |
| Steps: | Expected Results | Actual Results |
| Flex the sensors | Sensors should give you some reading of the flex. | Sensors gave a successful reading |
| Apply a flex to your fingers | Data is sent over to the Android app via bluetooth | Data was sent successfully to the app. |
| Result: PASS | | |

| Test Case 1.1 | | |
|---|---|---|
| Pre-condition: Ensure the voltage divider circuit is connected to the perf board successfully and the sensors are soldered with the perf board. Have a glove ready to construct the actual glove. | | |
| Steps: | Expected Results | Actual Results |
| 1. Place the perf board on the hand using zip ties. | Perf board should successfully remain on the glove. | Perf board successfully remains on the perf board. |
| 2. Place the ESP-32 board on the hand | ESP-32 should remain on the glove | ESP-32 successfully remained on the glove |

| Result: PASS |
|---|

| Test Case 1.2 | | |
|---|---|---|
| Pre-condition: Ensure the voltage divider circuit is connected to the perf board successfully and the sensors are soldered with the perf board. Have a glove ready to construct the actual glove. Perf board is connected steadily to the glove. | | |
| Steps: | Expected Results | Actual Results |
| 1. Place the flex sensors on all 5 fingers using zip ties | Once placed, flex sensors should remain on the fingers and give a reading when fingers are flexed. | Flex sensors did not remain on the fingers and would slip out of the zip lock. |
| 2. Place the flex sensors on all 5 fingers using velcro | Once placed, flex sensors should remain on the fingers and give a reading when fingers are flexed. | ESP-32 Flex sensors did not remain on the fingers and would slip out of the velcro. |
| 3. Place the flex sensors on all 5 fingers using tape | Once placed, flex sensors should remain on the fingers and give a reading when fingers are flexed. | It would create gaps in between tapes which leader to bad readings and would eventually come off |
| 4. Hot glue to flex sensors on to the glove directly | Once placed, flex sensors should remain on the fingers and give a reading when fingers are flexed. | The flex sensors remained on the glove successfully and gave readings. |
| Result: PASS | | |

**Story ID: HW-02**

We want to be able to send data to the app when wearing gloves and the finger motions flex is sent to the mobile app in real time.

| Test Case 1 | | |
|---|---|---|
| Pre-condition: Arduino and flex sensors are connected and powered on, the software on the Arduino is set up to compile and upload code to the ESP-32. | | |
| Steps: | Expected Results | Actual Results |

| | | |
|---|---|---|
| Connect to the bluetooth on mobile phone | Phone successfully connects to the bluetooth module of the ESP-32 | Phone successfully connected to the bluetooth module. |
| Apply a flex to your fingers | Data is sent over to the Android app via bluetooth | Data was sent successfully to the app. |
| Result: PASS | | |

**Story ID: HW-04**

We want the glove to be comfortable and have different hand sizes able to wear the glove.

| Test Case 1 | | |
|---|---|---|
| Pre-condition: Arduino and flex sensors are connected and powered on, the software on the Arduino is set up to compile and upload code to the ESP-32. Everything is connected to the glove. | | |
| Steps: | Expected Results | Actual Results |
| 1. Have different people in the group wear the glove | All should be able to wear the glove and flex the sensors | Everyone was able to wear the glove. |
| Result: PASS | | |

| Test Case 1.1 | | |
|---|---|---|
| Pre-condition: Arduino and flex sensors are connected and powered on, the software on the Arduino is set up to compile and upload code to the ESP-32. Everything is connected to the glove. | | |
| Steps: | Expected Results | Actual Results |
| 1. Wear glove for long time and conduct signs | Glove should be comfortable and not hurt you in any capacity | Glove was not comfortable and hurt your hands. |
| Result: FAIL | | |

**Story ID: HW-09**

Users should be able to hear their English translation of their sign.

| Test Case 1 | | |
|---|---|---|
| Pre-condition: Arduino and flex sensors are connected and powered on, the software on the Arduino is set up to compile and upload code to the ESP-32. Everything is connected to the glove. | | |
| Steps: | Expected Results | Actual Results |
| 1. Perform an operation on the glove. | A text-to-speech voice should output the English translation | The English translation was outputted. |
| Result: PASS | | |

## Story ID: HW-06

We want the glove to detect dynamic gestures through the integration of an IMU sensor.

**Pre-condition:**

IMU sensor is soldered and wired to the ESP32. Arduino IDE is ready to upload and monitor sensor data.

| **Steps** | **Expected Results** | **Actual Results** |
|---|---|---|
| Connect IMU sensor to ESP32 via I2C | Sensor should power on and be recognized in I2C scanner | IMU failed to appear consistently |
| Upload test sketch to read motion/orientation | Serial monitor should show live motion data | Serial monitor showed no or sporadic data |
| Move glove (tilt, rotate) | IMU data should reflect real-time motion | No usable data during movement |
| Attempt to integrate IMU data with ML Model | Combined data should be used for dynamic gesture detection | Integration was unsuccessful |
| **Result:** | | FAIL |

**Story ID: UI-03**

Users should be able to modify the output of certain signs to their liking

| Test Case 1 | | |
|---|---|---|
| Pre-condition: Arduino and flex sensors are connected and powered on, the software on the Arduino is set up to compile and upload code to the ESP-32. Everything is connected to the glove. Gestures activity page should be implemented. | | |
| Steps: | Expected Results | Actual Results |
| 2. Change the gesture output for “one” | When one is signed, the changed sign should output | The changed sign was outputted. |
| Result: PASS | | |

**Story ID: UI-04**

Settings Page

| Test Case 1 | | |
|---|---|---|
| As a user, I want to configure settings such as language, theme, and gesture preferences. | | |
| Steps: | Expected Results | Actual Results |
| Change the language to French | The app updates to display content in French | All UI elements switched to French |
| Toggle dark mode | Theme switches from light to | Dark mode applied successfully |
| Result: PASS | | |

**Story ID: UI-05**

Gesture Training UI

| Test Case 1 | | |
|---|---|---|
| As a user, I want an interface to train custom gestures so I can personalize my experience. | | |
| Steps: | Expected Results | Actual Results |
| Record a new gesture for "peace." | UI allows recording and saving the gesture | Gesture was saved recognized in test |
| Test the new gesture | Glove recognizes and outputs the saved gesture | Output displayed as expected |
| Result: PASS | | |

**Story ID:UI-10**

Text-to-Gesture Interface

| Test Case 1 | | |
|---|---|---|
| As a user, I want to input text and see the corresponding sign translation. | | |
| Steps: | Expected Results | Actual Results |
| Enter the word “hello”” | App shows the correct sign animation | Animation for “hello” was displayed |
| Enter a full sentence | Gesture sequence plays in correct order | Sentence was signed with correct hand motions |
| Result: PASS | | |

**Story ID: UI-12**

Gesture Sets

| Test Case 1 | | |
|---|---|---|
| As a user, I would like to have different sets with distinct saved gestures. | | |
| Steps: | Expected Results | Actual Results |
| Create multiple new sets | Set page add new sets | The list of sets showed new sets |
| Set the same gesture, 2 different meanings in | Different text appears depending on which set is | The gesture outcome depends on what set is used |

| | | |
|---|---|---|
| two distinct sets | selected | |
| Result: PASS | | |

## Story ID: UI-13

**Goal:** As a user, I want to view previously used gestures for better tracking.

| **Steps** | **Expected Results** | **Actual Results** |
|---|---|---|
| Open history log | Previously used gestures should appear chronologically | List showed date and time stamps for all gestures |
| **Result** | | **PASS** |

## Story ID: UI-14

**Story Title:** Learning Activity

**Goal:** As a new user, I want to learn ASL through the app.

| **Steps** | **Expected Results** | **Actual Results** |
|---|---|---|
| Launch ASL lesson 1 | App shows images/videos of signs | Lesson showed basic hand signs clearly |
| Complete a quiz | App provides explanation, and to redo errors | Feedback results were shown and had to be redone to pass |
| **Result** | | **PASS** |

## Story ID: ML-09

**Story Title:** Gesture Accuracy Feedback

**Goal:** As a user, I want the app to give feedback when I make incorrect gestures.

| **Steps** | **Expected Results** | **Actual Results** |
|---|---|---|
| Perform an incorrect version of “yes” | System flags and highlights the error | Red X and “Try Again” prompt displayed |
| Repeat the correct version | Gesture is recognized and marked correct | Confirmation message shown, |
| **Result** | | **PASS** |