Give Me A Sign

Authors: Leia Park, Ran Fang, Sejal Madan
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A phone attachment and mobile application package that provides dual-screen, real-time, immediate translations of American Sign Language into full, grammatically correct English.**

***Index Terms*—American Sign Language, Arduino, Bluetooth Low Energy, Computer Vision, Machine Learning, Web Application, Large Language Model**

1 INTRODUCTION

In today's interconnected world, effective communication is essential for social interaction, professional advancement, and access to vital services. However, communication barriers persist, particularly between the deaf community and individuals who are not proficient in American Sign Language (ASL). These barriers hinder meaningful interaction, limit opportunities, and contribute to feelings of isolation and exclusion among the deaf population[1].

Our solution, a Real-time ASL Translator App + Phone Attachment, addresses this pressing issue by providing a seamless means of communication between deaf individuals and those unfamiliar with sign language. This innovative application, coupled with a convenient phone attachment, enables real-time translation of spoken language into ASL gestures, facilitating smooth and efficient communication regardless of language proficiency.

The primary use case for our product arises in everyday interactions, where deaf individuals encounter communication challenges in various settings, including educational institutions, workplaces, healthcare facilities, dining locations and social gatherings. Our device is designed as a portable device that users can easily set up. The compact and lightweight design allows users to carry it with them wherever they go, ensuring accessibility and convenience in various situations. For example, if a user wants to order a coffee at a coffee shop, they can simply attach the device to their phone, set up the camera angle, and start signing into the camera. The app automatically translates the sign language into written English and displays the translation as subtitles on both the phone screen for ASL users to view and the attached screen for non-ASL users to understand, enabling smooth and efficient communication for both parties. This empowers deaf individuals to confidently place orders, customize their preferences, and engage in meaningful interactions, thereby enhancing their ordering experience and promoting inclusivity.

Overall, our Real-time ASL Translator App + Phone Attachment offers a comprehensive solution to the communication barriers faced by the deaf community. Our goal is to empower individuals with hearing impairments to engage confidently in conversations, participate fully in social and professional settings, and access essential services without the hindrance of communication barriers. Through our commitment to inclusivity and innovation, we strive to enhance the quality of life for individuals with hearing impairments and promote equal opportunities for communication and interaction.

2 USE-CASE REQUIREMENTS

To ensure effective communication between individuals proficient in American Sign Language (ASL) and those who are not, several critical quantitative metrics have been established, with specifications regarding user distance from camera, sign language detection, recognition, accuracy, and user satisfaction.

Firstly, the person using the app must be within a certain distance from the camera for their gestures to be visible and accurately tracked. This distance is specified to be between 1 to 3.9 feet from the mobile device front camera. Our application is designed to operate on a mobile device, so these distances come from our research about the optimal distance of the phone front camera best resolution range, and the distance of a normal conversation [2]. The prescribed distance ensures that the user's gestures are well within the camera's field of view, allowing our computer vision (CV) algorithm to accurately detect and analyze the intricate movements of their hands along with their pose (upper body). Besides from distance, the ambient lighting condition could interfere with the CV detection quality. Since our ideal use-case is a cafe or restaurant setting, we expect our product remains intact CV function under a brightness range of 10-500 lux (lumen per square meter), approximately from dimmest as an evening restaurant to brightest as indoor workplace under sunlight[3].

Furthermore, the accuracy of gesture recognition is important to the functionality of our app. With a minimum requirement of 95% accuracy for gesture detection and recognition, users can trust that their signing gestures will be correctly interpreted and translated into written English. We will be leveraging MediaPipe for temporal gesture recognition, which reports average accuracy rates up to 99% [4]. Specifically, we will be using MediaPipe's holistic model to create landmarks for the user's hands and pose, which will then be used in the machine learning (ML) algorithm to predict these translations. We are giving a 5% error to account for potential challenges such as variations in environmental conditions, lighting, and other factors that may affect the accuracy of gesture recognition. This high level of accuracy is essential for facilitating clear and meaningful communication between ASL users and non-ASL users.

In addition to accurate gesture recognition, the accuracy of translation from sign language to written English is another crucial requirement. The app must achieve a minimum accuracy of 95% for translation to ensure that the intended message is conveyed accurately and effectively. To achieve this level of accuracy, we will employ a ML model trained with extensive data specifically curated for dynamic signing recognition. This model will be trained on a diverse range of ASL gestures and expressions to accurately interpret sign language in various contexts. This minimum accuracy is based on empirical accuracy findings:

Long Short-Term Memory (LSTM) networks achieve an empirical accuracy of 95.21%[5], and are a type of recurrent neural network (RNN) that are specifically designed to overcome the limitations of traditional RNNs in capturing long-range dependencies in sequential data, such as video frames. We allow a tolerance for inaccuracies up to 5% due to factors such as variations in hand movements, environmental conditions, or the complexity of the sign.

Furthermore, to provide a truly immersive and real-time communication experience, the translation must be relatively immediate, functioning as ”live subtitles.” While existing translation models, such as Microsoft Translation Services, boast impressively low latency (8.9 - 13.9 milliseconds) [6], our aim is to set a slightly slower latency requirement to account for processing times involved in capturing and analyzing dynamic sign language gestures and model complexity. With a latency requirement of between 1 and 3 seconds, users can enjoy smooth and natural conversations without hindering delays in translation.

Lastly, ensuring good accessibility and a positive user experience is crucial for the widespread adoption and satisfaction of our Real-time ASL Translator App + Phone Attachment. Not only must the app itself provide a seamless experience, but the phone attachment must also enhance usability and convenience. With a target user satisfaction rate of approximately 90%, our goal is to create an intuitive interface that is easy to navigate for both ASL and non-ASL users. Specifically, we asked at least 10 ASL users to try the device and ask questions about the usability and accessibility. Some questions included ”Were you able to perform tasks efficiently with the app and phone attachment?” and ”How satisfied are you with the accuracy of ASL translation provided by the app?” Clear instructions guide users through the setup process, ensuring that they can quickly and easily attach the device to their smartphone. Seamless functionality is essential for smooth communication between users. By meeting these quantitative metrics, our app aims to enhance the overall communication experience for both ASL and non-ASL users, promoting inclusivity and accessibility in diverse social and professional settings.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

There are two primary modules to our product: (1) the software module shown in Fig. 1 and (2) the hardware module shown in Fig. 2. The interactions between and within the two modules are indicated in the block diagram in Fig. 3. The software module acts as the data processing hub of the system, consisting of computer vision (CV), machine learning (ML) and large language model (LLM) algorithms, and the web application. The CV receives raw input data from the phone camera and the ML performs gesture recognition on that data. The collated information is processed through LLM to interpret the gestures into grammatical English, transmitting the finalized translations to our hardware side. The hardware module serves as the physical product the user attaches to their phone that receives data from the web app to display ASL translations on its monitor. It consists of components packaged into an adjustable 3D-printed case, providing dual-screen ASL translations for the user and their conversant.

3.1 Software Module

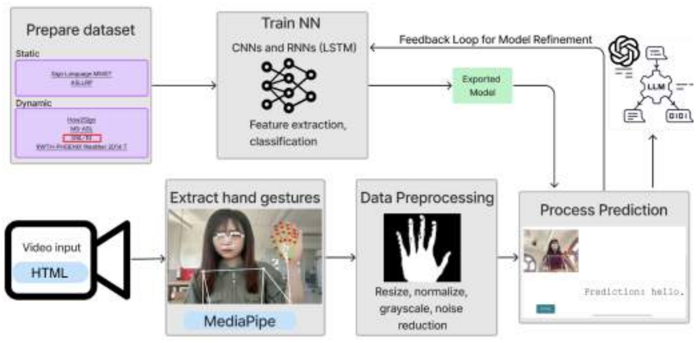


Figure 1: Subsystem diagram - Software module

MediaPipe’s web framework runs in HTML and the underlying JavaScript. The HTML camera session handles and propagates real-time video input to pre-trained MediaPipe models in JavaScript, which extracts hand and pose gestures and allows dynamic visualization. The combined programs identify and outline skeletons for hands and upper body.

The machine learning subsystem is implemented by Tensorflow and Keras. Dynamic datasets of ASL are prepared and run through a Long Short-Term Memory (LSTM) Model, training the neural network model for feature extraction and classification.

The exported model reads the skeleton defined by the CV to predict the gestures, and the CV data is propagated to the model in a feedback loop for model refinement.

The language processing utilizes a Large Language Model (LLM), specifically GPT-3.5, receiving the final prediction from the ML model to standardize the text with correct spelling and proper grammar according to the English language. The resulting text is sent to the frontend for retrieval by the web application.

The web application is created with Javascript as our frontend, Python as our backend, and Django as our framework to main the web application. The app also serves as a communication terminal, sending live camera feed to the CV subsystem and taking text data from the backend to transfer to the hardware module and the HTML of the web app.

3.2 Hardware Module

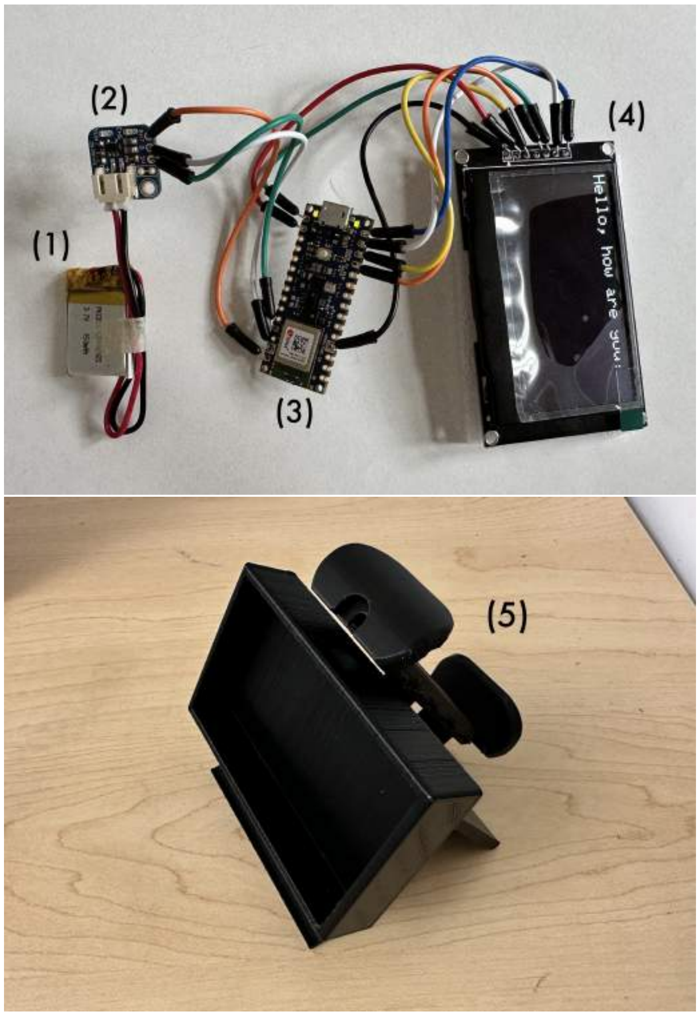


Figure 2: Hardware Components

The hardware module consists of five components: (1) a rechargeable Lithium-Polymer (Li-Po) battery 3.7V 150mAh, (2) Adafruit Li-Po Backpack, (3) an Arduino Nano 33 BLE unit, (4) an OLED 2.42" display screen, and (5) a 3D-printed phone attachment.

The Arduino uses Bluetooth Low Energy (BLE) to wirelessly connect and locally interact with the web application, acquiring the translation text. Connected with jumper wires, the Arduino unit shares the text to the OLED screen for display and is powered by the Li-Po battery, which is charged via inductive coil as phones are powered through this same technology. The battery is connected to the Arduino and OLED through the Adafruit Li-Po Backpack, which is a voltage regulator that manages the amount of power entering the rest of the components as well as preventing the battery from overcharging.

The components are packaged into a 3D-printed phone attachment. Designed with CAD software and printed with polyactic acid (PLA), the device is tilt-adjustable for clear image capturing by the phone camera according to the user's preference and grip-adjustable to accommodate a wide variety of mobile phones and different sizes. It can be easily removed or affixed to the phone, even if the phone has a case.

4 DESIGN REQUIREMENTS

Firstly, to maintain the effective functionality of our product under various environments, we enhance the CV module performance. According to our defined use-case requirements, CV module must detect signing occurring at 1-3.9 feet from the camera under a lighting condition of 10-500 lux. Driven by this specification, we preprocess the video frames with either OpenCV's embedded functions or customized algorithms. Specifically, we resize the captured image by a scale factor of 28, convert it to grayscale using `cvtColor` function, and normalize it by a factor of 256.

Subsequently, our gesture and pose detection models must output the correct output 95% of the time, preparing a valid input dataset to the following ML translation module. Hence, we adopt the combination of OpenCV and MediaPipe as the action recognition solution. Through these softwares' official documents and our initial trials, the OpenCV-MediaPipe pipeline possesses smooth and accurate performance in gesture and pose recognition on frames of dynamic, real-time camera feed. MediaPipe recognizes the hands and upper body and ignores the distracting background objects, generating 21 landmarks per hand[7] and 33 landmarks for the body pose [8] at maximum, depending on the proportion of the figure inside the camera frame. Further analysis on CV components will be elaborated in later sections.

Thirdly, to give the users accurate ASL-to-English translation results, we specify our ML module to reach a 95% accuracy rate in prediction. Accordingly, we trained our LSTM ML model for dynamic signing on multiple videos that approximate our use case in the real world. We have trained the algorithms with 6 datasets. The detailed ML training, testing and validation procedures will be discussed in the following sections.

Furthermore, with a goal of minimizing latency between ASL signing and feedback reception, our design emphasizes efficiency across multiple pipeline stages. The Computer Vision (CV) module targets a frame rate of approximately 30 fps for gesture and pose recognition. Through

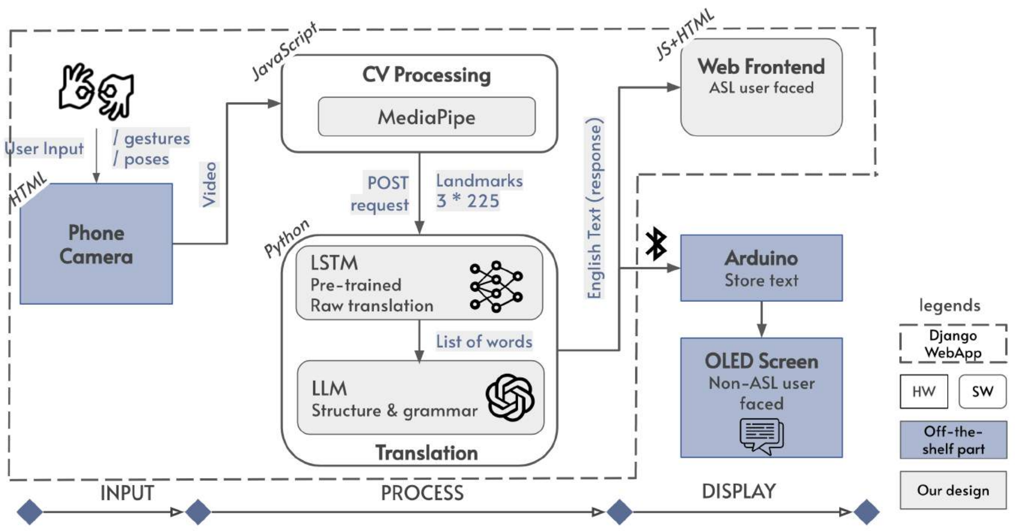


Figure 3: System Block Diagram - Pipeline

our testing and validation, we have proved that this benchmark can be consistently achieved through MediaPipe processing, as detailed in Section 7. Subsequently, our Machine Learning (ML) prediction aims for results within a 1000-1500 milliseconds window, allowing ample time for Large Language Model (LLM) API calls. The OpenAI API's response time fluctuates between 1000 to 2500 milliseconds, dependent on the length and complexity of the raw translation result produced by ML model. We optimized prompt tokens to be both clear and concise for swift GPT3.5 responses. Besides, initial experiments reveal minimal latency in frontend-backend data transmission, facilitated by transmitting constant-length float arrays. However, bluetooth-enabled data exchange between JavaScript and the Arduino board presents latency challenges stemming from the speed-accuracy trade-off.

Lastly, to reach the expected user satisfaction rate of over 90%, our mobile app UI/UX design must be navigable and of minimalist style. Our another essential user interactive component, the screen-integrated phone attachment, must be fabricated and connected for the ease of use and carry around. The electric power source should operate at a human-safe voltage of below 15 VAC. In addition, to avoid bias in the process of collecting user feedback, we should recruit at least 3 pairs of users who know neither each other nor our team members beforehand. Admittedly, since we do not plan to distribute the product commercially and have limited access to ASL user community, we are likely to include personal acquaintances in the final survey process.

5 DESIGN TRADE STUDIES

5.1 Product Design

We initially thought of three unique designs for our product. First, we considered a web application for its efficiency as no hardware is required and the user can pull out their phone or laptop to access. However, given that seamless conversation in a physical setting is our aim, this method is difficult for the user to work with casually. Moreover, they must flip their device screen between themselves and the person they are conversing with, making the modus operandi a hassle. Second, we considered glove tech, which would remove CV from our software implementation and provide higher gesture tracking accuracy because of its wearability. However, when imagining its use in daily interactions, the setup has potential to be inconvenient as the user must put gloves on both hands and then prepare their device to display translations. Third, we considered a triangular prism device that had two screens, a camera, and a microprocessor. However, its camera angle and portability were major concerns. Finally, with insight and a recommendation by a TA, we settled on our current implementation as we found it to be cost-effective, reasonable to achieve, and practical for users.

5.2 Software Systems

5.2.1 OpenCV vs. MediaPipe

For computer vision processing tools, after we examined OpenCV's JavaScript API and MediaPipe's web framework, we determined that the latter would assist in a more accurate and easier implementation. OpenCV utilizes Gaussian blur and Canny edge detection algorithms to subtract backgrounds and grayscale the images[9]. Through this process, it detects the precise configuration of the target object in a given frame. While we are able to filter noise pixels with this feature, OpenCV lacks an expertise in gesture and pose recognition. The existing gesture recognition package implemented solely in OpenCV contains 10 actions and 3 poses at maximum[10]. Admittedly, we would have control over the entire pipeline if we tailored the detection algorithm and dataset from scratch[11]. However, with MediaPipe library, we would simplify the process without losing any functionality, because MediaPipe is dedicated to gesture, pose and facial recognition. Its pre-trained dataset generates accurate coordinate representations on human hands and body, resulting in 21 landmarks per hand and 33 landmarks for the body pose. Most importantly, MediaPipe is optimized for real-time performance. Lastly, as we migrate from a local mobile app design to a web application, MediaPipe provides a scalable web framework that can be conveniently integrated and modified. Since our ultimate objective is to efficiently process real-time, dynamic signing, we eventually chose to use a combination of MediaPipe's web framework, where a camera session in HTML captures the frames and MediaPipe is responsible for gesture and pose recognition.

5.2.2 LSTMs vs. GRU vs. 3D CNNs

In the realm of recognizing dynamic signs, various neural network architectures offer distinct advantages and trade-offs. Long Short-Term Memory networks (LSTMs) are favored for their ability to capture long-range dependencies in sequential data, making them well-suited for tasks like sign language recognition that require understanding gestures over time. Despite being computationally intensive, LSTMs excel in retaining important information across extended sequences, crucial for accurate interpretation. Alternatively, Gated Recurrent Unit (GRU) networks offer a simpler architecture with comparable performance, albeit at the cost of potentially weaker long-term memory retention. Meanwhile, 3D Convolutional Neural Networks (3D-CNNs) provide an alternative approach by extracting spatial and temporal features from video data, offering potential benefits in capturing motion dynamics. However, their complex architecture and computational demands may pose challenges, especially with limited training data. Ultimately, the choice of LSTMs for recognizing dynamic signs seems like the best option from its balance of effective long-term memory retention and suitability for sequential data processing, aligning closely our the requirements of sign language recognition [5].

5.2.3 NLP vs. LLM

In considering the methods for converting directly translated sign language to proper English grammar, two main methods were considered, natural language processing (NLP) and large language models (LLMs) like GPT (Generative Pre-trained Transformer). NLP, relying on predefined rules and linguistic patterns, offers precise control over grammar and syntax, ensuring adherence to grammatical rules and linguistic conventions. However, it requires constant maintenance to accommodate changes in language usage and may struggle with understanding context or handling ambiguous language. Conversely, LLMs leverage vast amounts of text data to generate coherent and contextually relevant text responses without explicit rule definitions, providing adaptability and contextual understanding. However, they may lack control over specific grammatical structures and require extensive training data, potentially limiting their effectiveness in specialized domains like sign language recognition. We chose NLP over LLMs due to the need for precise grammatical control and the availability of specialized linguistic knowledge for sign language translation, making NLP a more suitable and customizable option for ensuring accurate and grammatically correct English outputs from sign language inputs [12].

5.2.4 iOS Mobile Application vs. Web Application

In our design review stage, we examined different platforms and programming languages to decide which is optimal for our mobile app development. We chose to develop an iOS app coded in Swift on the Xcode integrated development environment (IDE), because provided by Apple, Xcode supports a unified ecosystem that covers design, development, and testing across multiple Apple devices. Some MediaPipe CV packages are available to be integrated in an iOS app, and CoreML[13] provides interface to incorporate trained ML models. However, while the MediaPipe hand landmarks module was successfully integrated to our elementary iPhone app, we discovered later that pose recognition was not supported on the iOS platform. Besides, the keras-trained ML model presented compatibility problems with CoreML to compile the prediction input. Unfortunately, these two difficulties could not be resolved in a limited time, and the absence of pose landmarks would severely impacted the accuracy of dynamic signing prediction.

Consequently, in light of these constraints and the need for a viable solution, we made the strategic decision to transition our subsystems into a full-stack web application framework. We chose Django framework to integrate multiple codelayers needed in our web application. Django offers extensive support for integrating third-party libraries and tools, providing us with greater flexibility in incorporating CV and ML functionalities without the constraints imposed by the iOS ecosystem. Specifically, we did unit testing on both MediaPipe hand and pose frameworks and ensured its seamless performance with HTML and JavaScript implemented in Django frontend. Moreover, by migrating to a web application framework like Django, we can ensure broader compatibility across various platforms and devices, thus expanding our potential user base. Furthermore, the learning curve for a Django web app becomes negligible compared to an iOS app development, because our team members have previous course project experience with it. Nevertheless, the decision to abandon the previous mobile app development and move to a new solution imposes significant pressure on our schedule. The depending mechanism of software-to-hardware communication also needs to be re-designed and re-tested. Overall, we made this transition to prioritize the pivotal viability and functionality of our end product.

5.2.5 Local Implementation vs. Cloud Storage

When researching how to transmit live, real-time video to a cloud server, we found these exist numerous ways to implement this, but finding in addition to deploying a compatible method was not conducive to our timeframe, so we configured our web app to locally execute the machine learning and language processing rather than delegate those operations to an external source.

5.2.6 Minimal Number of Phrases vs. Accuracy

Our system admittedly possesses a minimal number of phrases: Hello, How are, Love, Mask, No, Please, Sorry, Thanks, Wear, You. Because there are extremely few datasets of dynamic sign language gestures for our machine learning model to train with, we even had to develop our own videos for our neural network. This resulted in limited sentencing, but we tried to value quality over quantity and did our best to improve the accuracy of the few words our system understood.

5.3 Hardware Components

5.3.1 Arduino vs. Raspberry Pi

A microcontroller is needed for the product to communicate with the mobile app and relay translations on an additional screen. Raspberry Pi (RPi) modules are powerful, acting as miniature computers with high processing capacity and their own operating system. Arduino units are straightforward, consisting of programmable circuits boards and containing their own integrated development environment. After a comparative analysis, we decided to utilize an Arduino fundamentally because our product does not require the immense complexity that RPi's provide [14]. Additionally, the RPi consumes more current than the Arduino and must be properly turned off to power it down, whereas the Arduino can simply be plugged and unplugged without damage to the operating software. Within the Arduino families, the Nano 33 BLE was selected because of its Bluetooth abilities and compact size. Other units in the Nano family had extra features our product would not use.

5.3.2 OLED vs. LCD vs. E-Ink Displays

For clear readability, we examined three different types of monitors. OLEDs consist of organic light emitting diodes, offering high contrast, bright images, and precise, independent pixel control. LCDs are liquid crystal displays that emit a backlight to illuminate crystals, displaying bright images and possessing low power consumption. E-Ink similarly uses liquid that colored pixel particles float within when charged. The screen incurs much less eye strain than the other monitors as it imitates paper reading. Currently, we chose an OLED screen for our product because of its conceptual finer clarity, but all the distinct kinds will be tested to identify which is most appropriate for our goals. All the screens were available in a 2.4" - 2.74" diagonal length and had similar pricing within a range of \$19.99 - \$23.99. Despite the extensive research on which monitor would suit our product, we recognized actual experimentation is needed for confidence in using one, specific display. According to our findings, we have settled with the OLED display because of its clear text and steady function.

5.3.3 Gear Adjuster vs. Adjuster Brackets

We considered using adjuster brackets to control tilt function of the device. Adjuster brackets are lines of spiked hooks that look like arrays of teeth in which the foot of the stand digs into to lean the viewing side. However, this adjustment mechanism was rejected because it was incompatible to the components that need to be attached to it, such as the phone grip and the hardware components casing. We went forward with a simple gear-based phone stand because of its simplicity and smooth sides that can easily accommodate the other parts to our product.

6 SYSTEM IMPLEMENTATION

6.1 Computer Vision

Firstly, when the web page for detection is rendered, it starts loading the required MediaPipe module, including its pre-trained landmark detecting ML model, to prepare for gesture and pose recognition. Corresponding prompt display on the web page to notify users the completion of module loading. Then, users enable a capture session embedded in HTML that takes in real-time video from the device's webcam. The underlying JavaScript starts to continuously grab image from webcam stream and detect it with a sampling rate of 30 fps (frames per second). The integrated MediaPipe then extracts landmark data of the hands and pose detected in the video via its `model.detectForVideo` function based on its pre-loaded `HandLandmarker` and `PoseLandmarker` models [15]. Meanwhile, MediaPipe's configured `DrawingUtils` display the landmarks in form of straight lines (for bones) and points (for joints) on the feedback screen. For each frame where target objects (hands and/or pose) are detected, 21 landmarks for each hand and 33 landmarks for the upper body will be processed in JavaScript into flattened `float` arrays. For example, we fill missing hand / pose data as zeros. At this stage, the CV module should have readily prepared the raw datasets for training and testing in the following ML module.

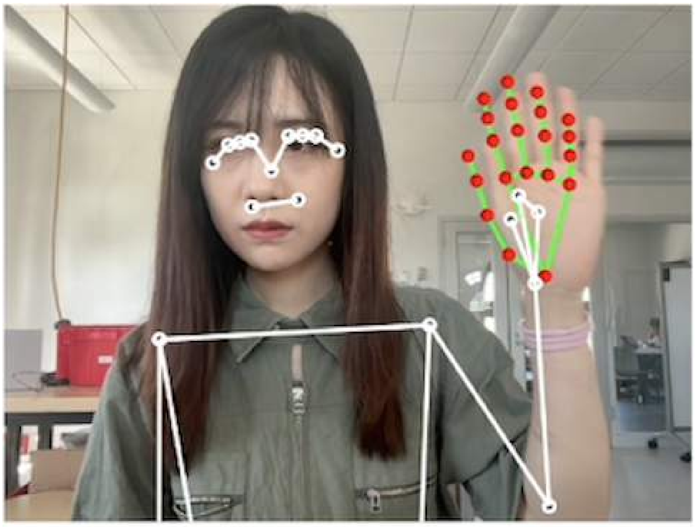


Figure 4: MediaPipe Recognition

6.2 Machine Learning

The training process for sign language recognition begins with dataset preparation, comprising of dynamic sign language gestures. The dataset includes existing datasets and data inputs created ourselves using OpenCV's video capturing function. Existing datasets include the DSL-10 dataset, which is recorded and split into 80% training and 20% testing subsets. MediaPipe's holistic model is employed with a minimum detection and tracking confidence set at 0.5 to extract landmarks for both hands and poses from the videos. These landmarks serve as features for training the model. In this training process, we utilize TensorFlow along with Keras, a high-level neural networks API, for building and training the LSTMs model. The model architecture, defined as a sequential neural network with LSTMs and dense layers, is compiled using the Adam optimizer and categorical cross-entropy loss function. During training, the model is trained for a specified number of epochs using the model.fit function, with the training data. After training, the model's performance is evaluated on the testing data using the model.evaluate function, providing insights into its loss and accuracy metrics. Finally, the trained model is utilized to predict the sign language gestures from the testing data using the model.predict function, where the `argmax` function selects the class with the highest probability for each sample, effectively choosing the final predicted class for classification. These predictions are then saved to be integrated into the CV algorithm to compute a prediction. Fine tuning the number of LSTMs and dense layers enhanced the model's capacity to capture complex patterns in sequential data, but also introduces the risk of overfitting and computational overhead. Similarly, adjusting the number of epochs can influence the model's convergence and generalization performance. Therefore, multiple variations were trained and the model that outputted the best accuracy results was used in the final system. The figure below illustrates the optimized architecture with three LSTM layer and three dense layers, with a softmax activation function applied within the last Dense layer.

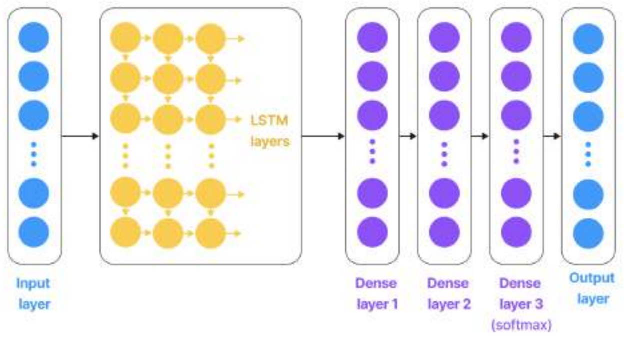


Figure 5: Neural Network Architecture

6.3 Large Language Model

After the machine learning model translates sign language gestures into written English, the output undergoes further processing through a large language model (LLM) to enhance its linguistic analysis and semantic interpretation. The LLM, specifically the GPT-3.5 Turbo model, is utilized to refine the translated text by generating more natural and contextually appropriate sentences. This process involves providing the LLM with a prompt containing the predicted text output from the ML model. The LLM then generates multiple potential translations based on the provided prompt, leveraging its vast language understanding capabilities to produce accurate and coherent linguistic interpretations of the sign language gestures. The structured sentence is then send back to the frontend of the web application to be displayed to the user.

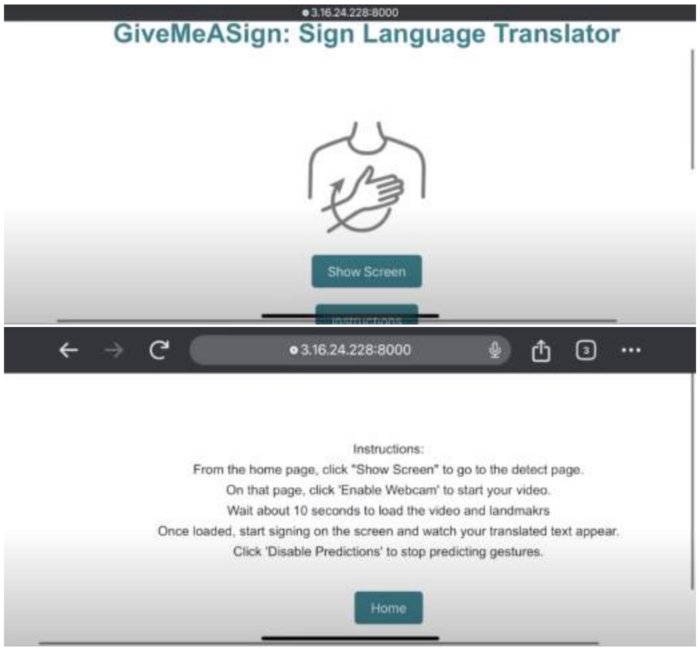
6.4 Screen Device

The LiPo battery has simple black ground and red power wires, which are connected to the corresponding ports of the Arduino Nano 33 BLE unit. The OLED screen has a resolution of 128 x 64 pixels with a SSD1309 driver chip, a a single-chip CMOS OLED driver with controller for organic/polymer light emitting diode dot-matrix graphic display systems and commonly used for OLED panels [16]. The screen supports a serial peripheral interface (SPI), used for facilitating short-distance communication between microcontroller and one or more peripheral integrated circuits, and has 6 pins according to the SPI interface:

1. GND (power ground)
2. VCC (power supply positive)
3. SCL (clock line)
4. SDA (data line)
5. RES (reset line)
6. DC (data / command)

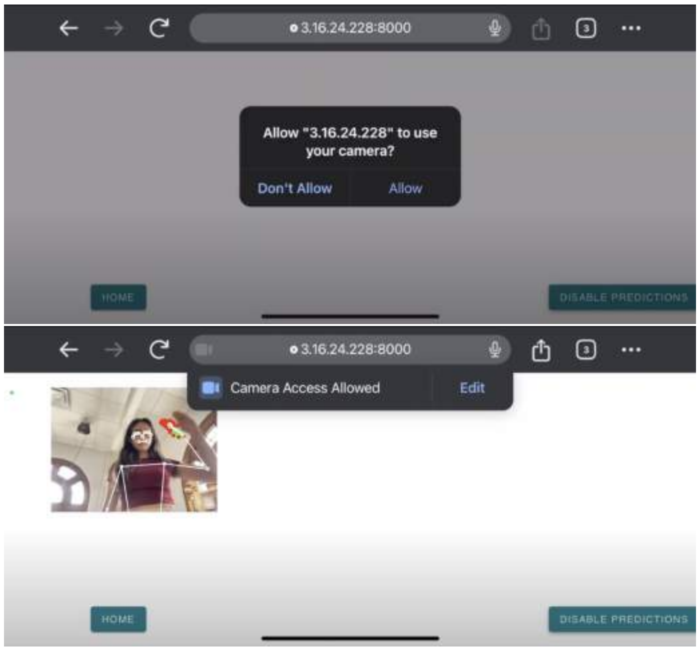
The appropriate headers of the Arduino are connected to the OLED pins, and an SPI interfacing library is already pre-bundled in the Arduino IDE. Everything is connected to each other with jumper wires via soldering.

6.5 Web Application



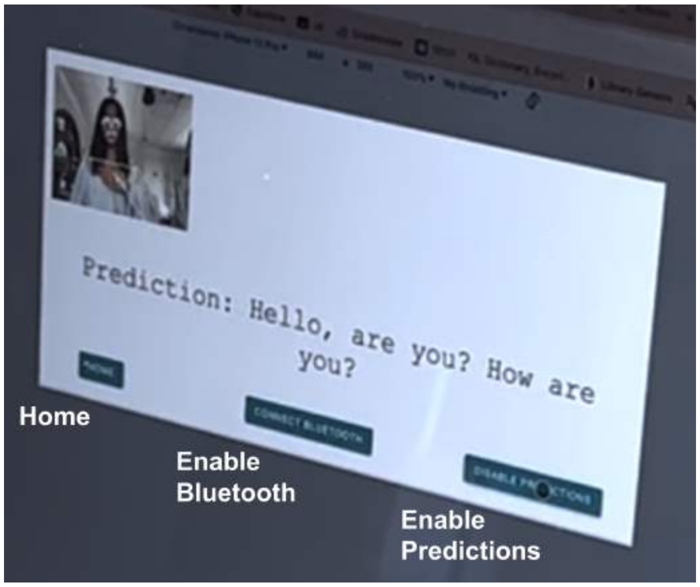
Instructions:
From the home page, click "Show Screen" to go to the detect page.
On that page, click 'Enable Webcam' to start your video.
Wait about 10 seconds to load the video and landmakrs
Once loaded, start signing on the screen and watch your translated text appear.
Click 'Disable Predictions' to stop predicting gestures.

Figure 6: Home & Instructions Pages



Allow "3.16.24.228" to use your camera?

Figure 7: Camera Permissions Demo



Prediction: Hello, are you? How are you?

Figure 8: Translation Page - Buttons are Home, Enable Bluetooth, Enable Predictions

The web application follows a structured architectural pattern, using frontend and backend responsibilities. At the frontend, JavaScript orchestrates user interaction and presentation, managing dynamic content rendering and user input. Leveraging MediaPipe functionalities, the frontend detects and processes hand and body gestures from webcam input, enhancing user engagement and interaction. Additionally, the script facilitates Bluetooth Low Energy (BLE) connectivity, enabling seamless communication with external devices. It manages data transmission, ensuring the smooth relay of collected data, such as hand and body landmark coordinates, to the backend for further analysis and translation.

The backend of the web application handles the prediction of sign language gestures and their translation into English text. Upon receiving a request, it processes the landmark data representing the hand and body gestures captured from the webcam. Using the machine learning model that was previously trained, it predicts the sign language gesture based on the processed landmark data. Subsequently, the predicted sign language gesture is translated into English text using a large language model provided by OpenAI. This translation process involves generating English sentences that convey the meaning of the sign language gesture. The translated text is then returned as a response to the client, enabling seamless communication between users using sign language and those understanding English. Additionally, the backend incorporates functionalities to manage the translation process, including handling intervals between consecutive translation requests and maintaining the continuity of translated sentences.

To deploy the Django web application on AWS, we utilized EC2 (Elastic Compute Cloud), AWS's service for deploying and managing web applications. We configured the necessary environment settings, including specifying the Python runtime and dependencies, and then deployed the Django application. To enable HTTPS, we obtained an SSL/TLS certificate from AWS Certificate Manager and configured it for the application's domain.

6.6 Attachment Creation

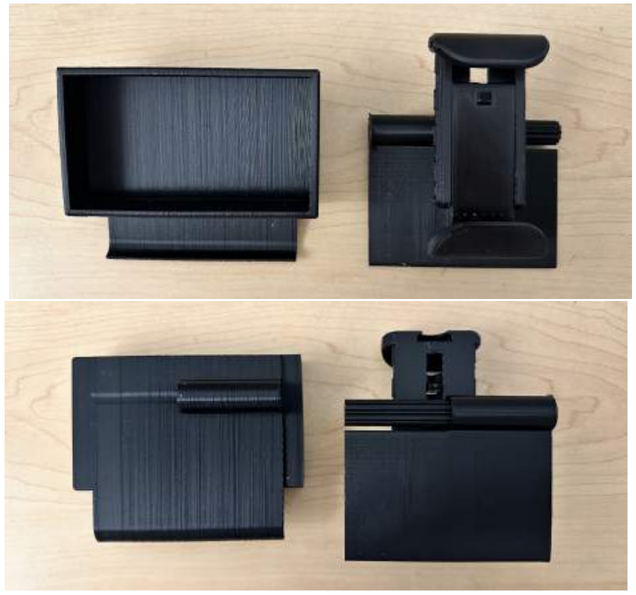


Figure 9: 3D Print Casing - Disassembled

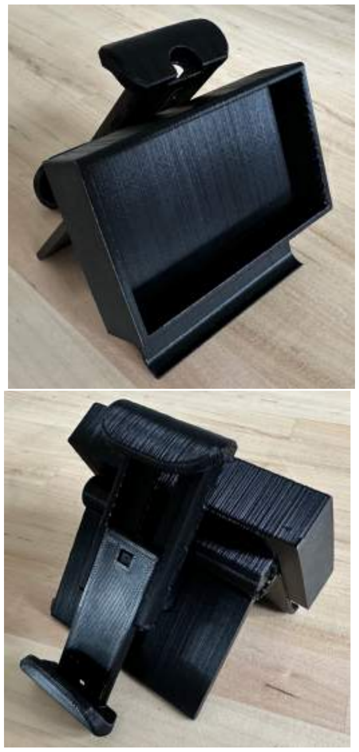


Figure 10: 3D Print Casing - Assembled

Designed with CAD and created out of polyactic acid filament, the most popular material for 3D printing, the attachment is grip and tilt adjustable to accommodate a variety of phone sizes and manipulate the phone camera angle to the user's preferences [17].

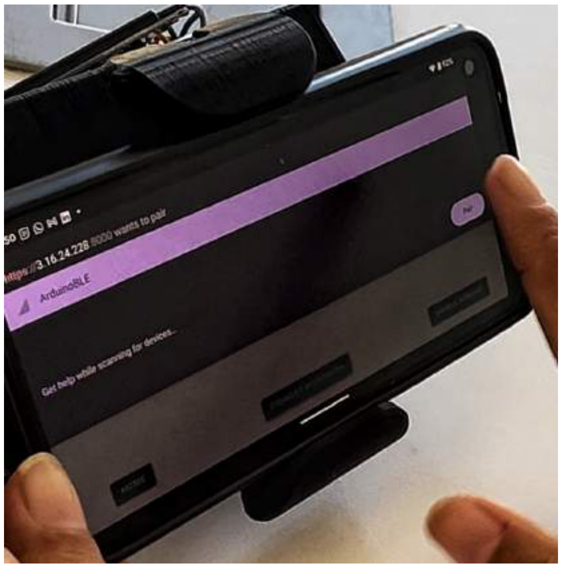
There are three components to the attachment: casing, grip, and stand. The casing is box-like to house all the hardware components into a compact, flat, and organized layout, and where the OLED screen displays translation for dual-screen functionality. The grip adjustment is controlled by a C-Clamp by hand and where the phone is inserted. As shown in Fig. 6 & 7, the bottom part of the grip can extend or retract just by pushing or pulling [18]. The stand's tilt adjustment is controlled with a gear mechanism, adjustable to five different angles between 20 and 73 degrees.

Dimensions of the components are: (W x H x L)

- Arduino Nano 33 BLE – 18 x 15 x 45 mm
- LiPo Battery – 34.5 x 10.6 x 56.0 mm
- Adafruit Li-Po Backpack – negligible due to small size
- OLED Screen – 43 x 6 x 74 mm
- Grip Clamp – 30 x 20 x 60 mm (not extended)
- Adjustable Stand - 60 x 15 x 85 mm (closed/flattened)

The current overall product dimension is about 60 x 55 x 85 mm.

6.7 Device Integration & Bluetooth Low Energy Functionality


https://3.16.24.228 8000 wants to pair
ArduinoBLE
Get help while scanning for devices...
Pair
Figure 11: Bluetooth Pairing Web App Pop-Up

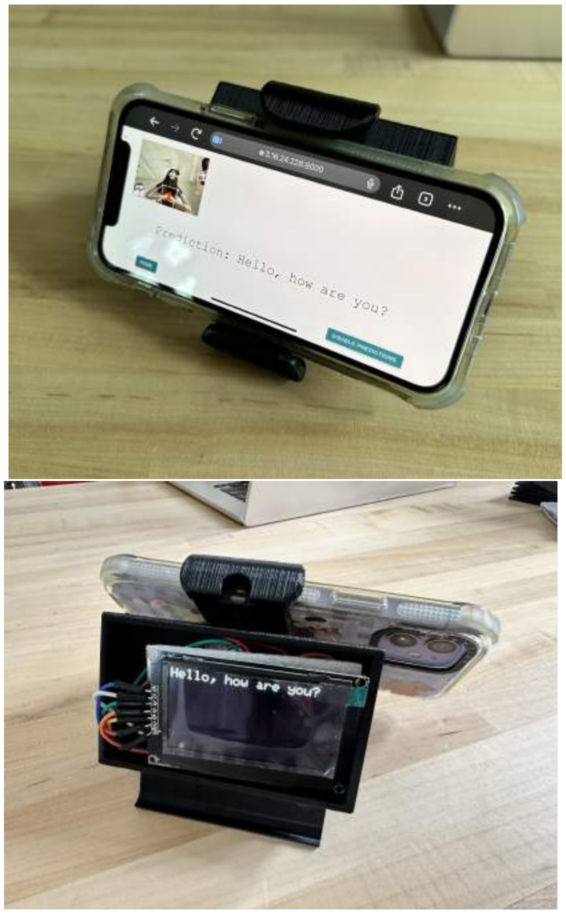


Figure 12: Final Integration

The Arduino Nano 33 BLE connects to the web application via Bluetooth Low Energy technology, a wireless personal area network utilizing client-server architecture called the General Attribute Profile (GATT), and running on reduced 0.01 - 0.05W power consumption and 2.4GHz radio frequencies like Classic Bluetooth [19].

The GATT is a communication interface that establishes how data is organized and transferred. Data is passed and stored in the form of characteristics or services, which are a collection of characteristics, in the memory of the BLE device. Between two devices: one is the server, or peripheral, that accepts an incoming connection request after advertising and contains the characteristic database that is being read or written by a client; the other is the client, or central, that initiates an outgoing connection request to an advertising peripheral device and is reading or writing data from or to the server.

The Arduino program, or sketch, is built on the Arduino IDE. Universally Unique Identifiers (UUIDs) are, as the name suggests, unique identifiers that identify services and characteristics of Arduino packages such as the BLE communication protocol. After they are user-defined for human-readability, the required services are initialized and characteristics are added to their corresponding services. The sketch is given a local name, and the Arduino, acting as the peripheral, advertises it for the web app, our central, to find. Entering a broadcast loop, the Arduino listens for our web app, and once connected, starts a loop for handling communication, executing periodic reading of data between Arduino and app. Checks and event-handler functions are implemented for the sketch to respond to the central device and update its data.

Specifically to our Arduino code, we have one service defined as the ArduinoBLE, which our web app identifies the UUID of to pair with the device. After a translated sentence is finalized by the web app, it has a function to write each character at a time as an ASCII value into the characteristic for the Arduino to receive and read, displaying the whole sentence letter by letter on the OLED screen. Moreover, at the beginning of a new sentence, the web app attaches the "@" symbol for the Arduino to recognized the ASCII value of so the screen is cleared before the next sentence is exhibited.

Since our demo web app to test bluetooth connectivity of the Arduino was a p5.js sketch, a Javascript library for creative coding, we incorporated the p5.ble.js library into our final web app to enable communication with our BLE device, which is the specific p5.js library that uses the Web Bluetooth API.

7 TEST, VERIFICATION & VALIDATION

We structured our testing strategy to primarily assess the performance of our entire system, reflecting its functionality in real-world scenarios. However, we also conducted targeted testing of subsystems, particularly focusing on latency, as they represent integral components of our final product. Our overarching design requirements, including latency, accuracy and user satisfaction, guided our testing efforts and were systematically addressed across both the entire system and its individual components. By adopting this approach, we aimed to ensure comprehensive validation of our product's capabilities while also identifying areas for refinement and optimization at both the system and subsystem levels.

7.1 Results for Distance Requirements of MediaPipe Landmarks

Our initial design specification required that MediaPipe landmarks should be accurately tracked within a distance range of 1-3.9 feet. To verify this requirement, we conducted a series of tests at various distance intervals.

We collected data by positioning the camera at different distances from the subject: 1 foot, 1.5 feet, 2.0 feet, 2.5 feet, 3 feet, 3.5 feet, and 4 feet. At each distance, we captured five samples to ensure consistency and reliability of our measurements.

During the testing process, we observed the accuracy of the detected landmarks on both hand gestures and body poses. We considered landmarks to be accurate if they were correctly identified and aligned with the corresponding anatomical features.

Our analysis revealed that the detected landmarks met the accuracy requirement across all tested distances, with proper landmarks appearing 100% of the time. This indicates that the system successfully tracked MediaPipe landmarks within the specified distance range.

To visualize the relationship between the tested distances and the accuracy of landmark detection, we plotted the percentage of accurate landmarks against the distance intervals. The plot illustrates a consistent trend of 100% accuracy within the 1-3.9 feet range, validating our design specification.

Our measurements align closely with the theoretical predictions, demonstrating the effectiveness of our system in accurately tracking landmarks at varying distances.

We did not modify the original system design to achieve a better value, as MediaPipe already inherently delivers high accuracy in landmark detection. Our results validate the robustness of the system's design and its ability to consistently meet the specified requirements. By leveraging the advanced capabilities of MediaPipe, we ensured that our system maintains reliable performance across various distance intervals without the need for extensive modifications

7.2 Results for Accuracy Requirements of Gesture Recognition

Our design specification mandated that MediaPipe, should accurately display landmarks of the hands and upper body at least 95% of the time. To evaluate this requirement, we designed a series of tests focusing on landmark detection performance under varying background settings.

We conducted trials with different background settings, varying the number of distractors present in the scene. Each trial consisted of two samples, allowing us to assess consistency in landmark detection across different backgrounds. The number of distractors ranged from 1 to 10, providing a diverse range of scenarios representative of real-world usage.

The distractors in our test scenarios encompassed a variety of objects and elements commonly encountered in real-world environments. These included items such as furniture, plants, electronic devices, miscellaneous objects, and other humans typically found in indoor or outdoor settings. Additionally, we incorporated background textures and patterns to simulate complex visual scenes commonly encountered in natural environments. By including diverse distractors representative of real-world scenarios, we aimed to evaluate the system's ability to maintain accurate landmark detection even with varying background complexities.

During testing, we examined the accuracy of landmark detection on the target subject, focusing on the hands and pose regions. Landmarks were considered properly displayed if they were correctly identified and positioned on the subject's anatomy.

Our analysis of the test results revealed that landmarks were accurately drawn on the target subject 95% of the time, meeting the specified requirement. However, the 5% failure rate in landmark detection occurred notably in scenarios where humans were present in the background. The presence of additional humans in the scene occasionally interfered with landmark detection, resulting in deviations from the expected accuracy level.

To visualize the impact of background settings on landmark detection accuracy, we plotted the percentage of properly displayed landmarks against the number of distractors in the background. The plot illustrates a consistent trend of high accuracy, with occasional deviations observed in scenarios with multiple human distractors.

Overall, our findings demonstrate the effectiveness of the CV/MediaPipe system in achieving the specified accuracy threshold for landmark detection. While the presence of human distractors poses challenges to accuracy in certain scenarios, the system maintains robust performance across a range of background settings, ensuring reliable landmark detection in most real-world situation

7.3 Results for Accuracy Requirements of Translation

Our design requirement was that English text translations should achieve a semantic accuracy level of at least 95%. To evaluate this criterion, we employed a testing method involving the translation of different sign language phrases and complex sentences into English.

The initial step involved plotting the training and validation accuracy curves during the model training process. The plotted results indicated that the training accuracy reached approximately 97%, while the validation accuracy reached around 94%. This observation suggests that the model was effectively learning from the training data, as evidenced by the high training accuracy. Furthermore, the relatively close alignment between the training and validation accuracies indicates that the model was not overfitting to the training data, as the validation accuracy remained high. Therefore, based on these results, it was anticipated that the model would perform well when tested on unseen data, with an expected accuracy falling within the range of the achieved training and validation accuracies. Furthermore, the confusion matrix yielded promising results, characterized by high true positive rates, low false positive and false negative rates, and overall high accuracy. This indicates that the model's predictions closely aligned with the ground truth across different classes, adding to the anticipation of good real time results on unseen data.

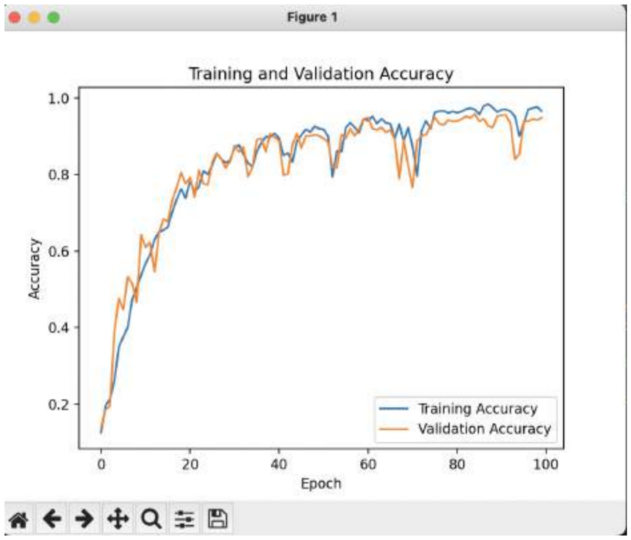


Figure 13: Training and Validation Accuracy Results

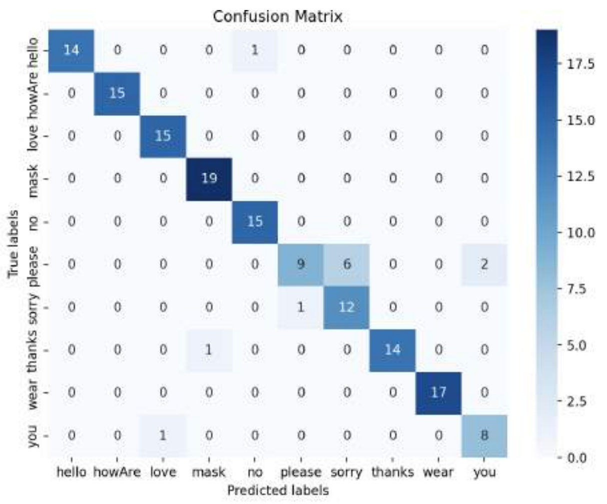


Figure 14: Confusion Matrix

Additionally, we conducted several rounds of model training with varying configurations, including different numbers of LSTM and dense layers, as well as adjusting the number of epochs. Through this iterative process, we aimed to optimize the model's performance. Ultimately, we found that training with epochs was ideal, as it consistently yielded the highest accuracy levels without the gap between training and validation accuracies becoming too high. Specifically, the configuration that produced 97% training accuracy and a 94% produced the most accurate results during real time testing. Therefore, we selected this configuration as the final model to be used for testing, as it demonstrated the most promising performance characteristics.

To perform real time testing of our system's prediction outputs, for each phrase, three members of our team independently signed the phrase three times, ensuring a diverse range of signing styles and variations. Similarly, for complex sentences, each member signed the sentence three times to capture the nuances of sentence structure and expression. Complex sentences were multi-phrase sentences that could be made from our set of phrases.

During the signing sessions, we translated the signed phrases and sentences into English text using our translation system. We then compared the translated text with the intended meaning conveyed by the sign language expressions.

Our analysis of the test results revealed the following accuracy rates:

- Phrases Translation Accuracy: 90%
- Sentences Translation Accuracy: 76%

These results indicate that while the translation system demonstrates proficiency in capturing the translation of signed phrases, although it did not reach our requirement of 95%, it exhibits a lower accuracy level when translating more complex sentences.

The discrepancy between the translation system's high accuracy in training/validation and its performance during real-world usage via the web application likely stems from disparities in data distribution, domain adaptation challenges, potential overfitting, and the presence of ambiguities/noise in real-world data. While the system may have been well-trained on representative examples, it may struggle to generalize effectively to unseen instances encountered in live scenarios. Variations in signing styles, environmental conditions, and user behaviors can introduce domain shifts that the system is not adequately adapted to handle, leading to decreased accuracy. Additionally, the system may have become overly specialized or overfitted to the training data, limiting its ability to generalize, while ambiguities and noise in real-world inputs further challenge its performance. Addressing these challenges may require strategies such as collecting more diverse training data, enhancing robustness to domain shifts, and incorporating techniques to mitigate overfitting and handle noise effectively.

While the achieved accuracy rates fall short of the 95% requirement, they provide valuable insights into the system's strengths and areas for improvement.

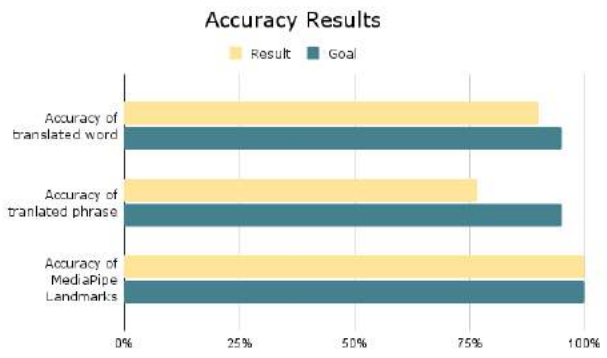


Figure 15: Accuracy Results

7.4 Results for Latency of Translation

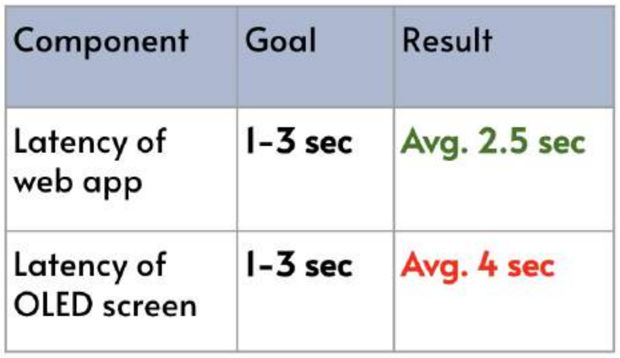
The requirement stipulated that translations should be displayed within a time frame of 1000 to 3000 milliseconds (ms) after a gesture is detected. To assess compliance with this requirement, we conducted tests involving each team member performing sign language gestures three times. We measured the time elapsed in milliseconds before the translation appeared on both the web application and the OLED screen, representing different parts of the system.

Our results indicate that the translation latency for the web application was approximately 2.5 seconds (2500 ms), falling within the specified range of 1000 to 3000 ms. However, for the OLED screen component, the translation latency was observed to be around 4 seconds, exceeding the upper limit of the specified range.

The longer 4-second latency in displaying translations on the OLED screen can be attributed to the serial communication protocol and the constraints it imposes on data transmission. When data is sent via serial communication, it is transmitted character by character. In this case, when translations were sent to the Arduino BLE 33 model to be displayed on the OLED screen, each character of the translation had to be transmitted individually. As a result, the process of sending the entire translation letter by letter introduced additional processing time. This led to the observed 4-second latency in displaying translations on the OLED screen.

This discrepancy suggests that there may be implementation differences or processing constraints between the web application and the OLED screen, impacting the speed at which translations are displayed. Further investigation is warranted to identify potential bottlenecks or optimizations that could reduce latency and ensure timely translation delivery across all components of the system.

We weighed our options and considered potential solutions to reduce the 4-second latency in displaying translations on the OLED screen. However, many of these options would require additional processing or exploring alternative communication protocols, which could introduce complexity and potentially disrupt other aspects of the system. Since the observed latency of 4 seconds is close to our upper bound of 3 seconds specified in the design requirement, we decided that the current latency is acceptable for practical use.



Component	Goal	Result
Latency of web app	1-3 sec	Avg. 2.5 sec
Latency of OLED screen	1-3 sec	Avg. 4 sec

Figure 16: Latency Results

7.5 Results for User Satisfaction Requirements

In assessing our requirement for achieving 90% user satisfaction, we conducted user testing sessions involving invited sign language users. During these sessions, we collected oral feedback and survey responses, asking them to rate each component from 1-5, to gauge user satisfaction with the system's usability and overall performance. Our results indicate that the ease of use of the web application received a perfect rating of 5 out of 5, highlighting the system's intuitive interface and user-friendly design. Additionally, the ease of use of the phone attachment received a high rating of 4.5, indicating a positive user experience with this component. Overall, the ease of use of the entire system received a rating of 4.7, reflecting a high level of user satisfaction with the system as a whole. These results suggest that the system meets our requirement for achieving 90% user satisfaction, as evidenced by the positive feedback and high ratings provided by sign language users during the testing sessions.

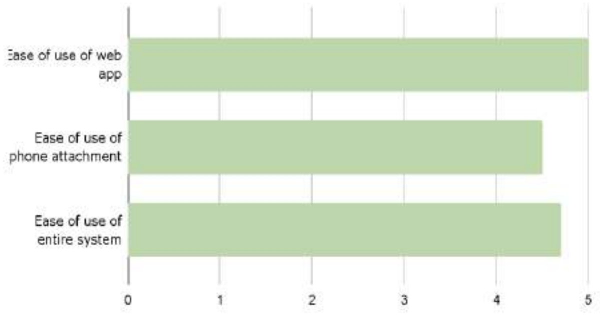


Figure 17: User Satisfaction Results

8 PROJECT MANAGEMENT

8.1 Schedule

Our schedule is represented by our Gantt chart, Fig. 20, and divided into 7 major sections: Hardware Device, Computer Vision, Machine Learning Model, Mobile App, Testing & Verification, Integration, and Misc. Across them there are 6 milestones:

1. Word Recognition
2. Real-time Recognition
3. Word Translation
4. Sentence Translation
5. Launch of Mobile App
6. Display Device

All tasks are color-coded in the Gantt chart according to the respective team member(s) meant to handle them. We accounted for Spring Break and buffers for integration of parts in case of challenges and difficulties. Our schedule changed from the design document by adding extra weeks for final integration and mobile to web app transitioning.

8.2 Team Member Responsibilities

All tasks have been equally and reasonably distributed to all team members according to their specialization and enthusiasm as shown in the figure below, which derives its calculations from Fig. 20.

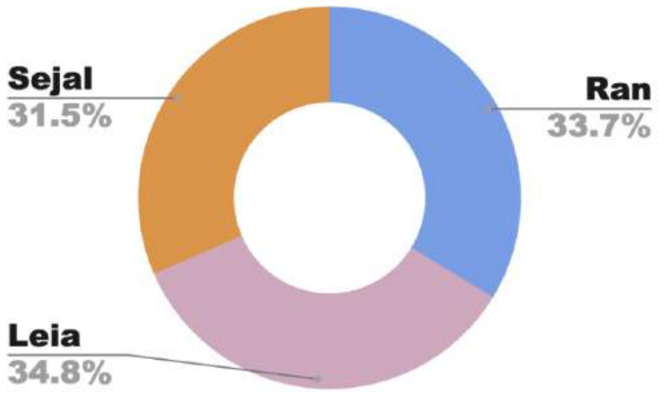


Figure 18: Distribution percentages of tasks across team members

Ran handles video processing with OpenCV and hand and pose detection with MediaPipe, and web application backend development.

Sejal handles the word translation ML model and sentence structuring and optimization, directing the neural network and NLP algorithm.

Leia handles the device fabrication, screen integration, and bluetooth development, creating the final product the user interacts with.

Integration of modules and their verification were handled in pairs depending on the specific mergers, and the complete intersectional amalgamation was tackled all together by the team. Each member was responsible for testing their individual assignments, and for final verification tests, we executed them as a team for fluid progress of the overall project.

8.3 Bill of Materials and Budget

See Table 1 for the breakdown of purchases that make the complete product. Our product is meant to be made and distributed at very low costs, given its purpose for equity and diversity. The product's total manufacturing cost (before the divider line, exempt of the purchases below it) came out to be $80.97 in total. The actual sum when including purchases made for testing and unused components was $165.47.

8.4 Risk Management

A primary risk element was possible faultiness in integrating all parts into a final product. Three of our design tradeoffs were actually risk mitigation decisions in this regard: the conversion of NLP to LLM, mobile app to web app, and cloud storage usage to local packaging. For all of the above as described in the earlier design trade studies section, we encountered significant difficulties in the near-endgame stage of integration that pushed us to our fall-back plans. They were executed with consideration of our schedule and feasibility, cutting out risk of an unfinished or defective project.

Ensuring the accuracy of gesture detection and sign language translation was one of the most critical aspect of our solution due to its fundamental role in facilitating effective communication between users of ASL and those unfamiliar with it. Any inaccuracies could undermine user trust, impede communication, and compromise the app's overarching goal of fostering accessibility and inclusivity.

To mitigate environment-based inaccuracies, we provide an instructions page to guide users for optimal signing conditions. We also set a rest gesture feature indicated by no hands seen by the camera that indicates the end of the sentence so that the user may try signing again to output a new, more correct translation.

On the physical product side, bluetooth functionality actually became a significant concern. During testing, the pairing between Arduino and web app would work on some days while on others be completely ineffective, and the latter would occasionally last for over 48 hours. Because this occurred during the final integration phases of our schedule, we considered reverting to wire connection and eliminating the bluetooth feature entirely. However, fortunately we resolved this by trying pairing web app to Arduino on different environments such as on another team member's computer or using Android instead of iPhone.

9 ETHICAL ISSUES

In a worst-case scenario, our product could fail its accuracy and latency requirement in a medical emergency. Since we can never know where accidents could take place, our web app could lose connection to the Internet in remote regions. Without proper Wifi connection, our product is unable to generate any translations, or it outputs

Table 1: Bill of Materials

Description	Manufacturer	Quantity	Cost @	Total
Arduino Nano 33 BLE Sense Rev2	Arduino	1	\$40.09	\$40.09
Lithium Polymer Battery	Adafruit	1	\$5.95	\$5.95
Adafruit Li-Po Backpack	Adafruit	1	\$4.95	\$4.95
OLED 2.42" Display Module	Diymore	1	\$19.99	\$19.99
Breadboard Jumper Wires Kit	Aypzuke	1	\$9.99	\$9.99
Arduino Nano 33 BLE	Arduino	1	\$29.69	\$29.69
Lithium Polymer Battery	EEMB	1	\$13.89	\$13.89
Adafruit Li-Po Backpack	Adafruit	1	\$4.95	\$4.95
E-Ink 2.7" Display Module	Waveshare	1	\$23.99	\$23.99
6in Micro USB Cable	StarTech	2	\$5.99	\$11.98
				\$165.47

error-prone, largely delayed results. Moreover, people easily panic under medical emergencies, and their mental or physical activities are severely disrupted due to pain or extreme stress. The hardware components could also get mechanically broken due to external forces in chaos, which could also contribute to the serious glitch and even complete dysfunction of our product.

Individuals who are deaf or hard of hearing and rely on sign language as their primary mode of communication would be particularly vulnerable to being on the losing end of this harm. Since the sign language translator is intended to facilitate communication between individuals ASL users and non ASL users, any failure or malfunction of the translator could disproportionately affect the deaf or hard of hearing community. These individuals may already face communication barriers in healthcare settings, and a breakdown in communication due to technical issues with the translator could further compromise their access to critical medical care, especially in regions where access to immediate medical assistance may be limited.

Users expect to be able to confidently rely on our product as a method of communication, but if it fails as described in this severe edge-case scenario, then it's inevitable that users will feel they cannot believe in us or our tech anymore. Moreover, considering the severity of medical emergencies and the hypothetical in which users depend on our product, a malfunction can deprive our users of their autonomy since they may end up unable to communicate for themselves if an ASL translator is unavailable.

To mitigate these potential adverse effects, we could implement a progressive web app as they are able to function offline and store data locally. Hence, even without Wifi or some internet connection, our web app is still usable in dire circumstances. We could additionally secure and pad the hardware components to lessen risk of damage.

10 RELATED WORK

A simple Google search for "ASL translators" results in multiple one-way English to ASL translators in which sentences are converted into hand symbols that spell out the words in the ASL alphabet, but not into the actual gestures ASL users make for the word.

There are existing efforts to further refine real-time ASL translations for seamless communication. We find there are public projects working on two-way communication within open-source platforms such as Github or shared on Youtube. However, they are either ASL alphabet recognition translators or very primitive models that can only recognize extremely few gestures, both which are not operational for full translations nor practical as they require a user to prepare their environment, download code, and run the program on a computer with a camera. Plus, they use very controlled video input.

Some competing technologies are BrightSign, Hand Talk, Slait.ai, and Jeenie. BrightSign is a glove that recognizes sign language gestures through sensors and translates them into text or spoken language. BrightSign requires users to wear a glove, but our app + attachment solution is portable and can be easily attached to any smartphone, making it accessible to a wider range of users [20]. Hand Talk is an app that translates written text or spoken language into sign language using avatars, to encourage non-ASL users to learn sign language. However, our solution translates from sign language to written text to be used reliably by ASL users [21]. Slait.ai is an application for web and mobile that uses artificial intelligence to transcribe ASL into text for the hearing person to read, and is currently undergoing beta testing [22]. However, given its solely software implementation, the user must be situated by a flat, stable surface to operate it. If they are on their phone, they must figure out how to set it down or only translate with one hand. Jeenie is an app that connects users to human interpreters for real-time translation of spoken and sign language [23]. While Jeenie provides access to human interpreters, our app offers instant and automated translation of sign language into written English, eliminating the need to wait for a human interpreter's availability.

Our approach addressing the limitations of these competing technologies by prioritizing accessibility, convenience, and reliability in facilitating communication between ASL users and non-ASL users.

11 SUMMARY

Our system was able to meet most of the design specifications. First, results for distance requirements of MediaPipe landmarks showed 100% accuracy within the specified range of 1-3.9 feet. Accuracy requirements for gesture recognition were met with landmarks properly displayed 95% of the time, though challenged by human distractors. Translation accuracy fell slightly below the 95% requirement, with phrases achieving 90% accuracy and sentences 76%, attributed to domain shifts and overfitting. Translation latency met the requirement for the web application but exceeded it for the OLED screen due to serial communication constraints. User satisfaction was high, with a perfect rating for web application usability and an overall rating of 4.7 out of 5, indicating a successful meeting of the 90% satisfaction requirement. To improve the system, collecting more diverse training data and and mitigating overfitting and noise could enhance translation accuracy. Implementing alternative communication protocols or optimizing existing ones could reduce translation latency on the OLED screen.

11.1 Future Works

11.1.1 ASL Vocabulary Expansion and Model Refinement

Expanding our dataset beyond the initial 10 words for dynamic signing is essential to improve the translator's vocabulary coverage and accuracy. This expansion could involve collecting more diverse sign language data, including words spelled from static letter signing and gestures for complex phrases or expressions. Additionally, we expect to devote more training time to scale up the ML model without losing accuracy, possibly through techniques such as transfer learning or data augmentation, which will contribute to improved translation quality and robustness.

11.1.2 Sign Language Linguistic Diversity

Moreover, we plan to expand our sign language recognition system to include multiple sign language variants, such as international, Chinese, Korean, and more. This expansion will involve collecting extensive datasets for each language variant and training our models to accurately recognize gestures across diverse linguistic and cultural contexts. Additionally, we aim to implement adaptive learning algorithms to enhance the system's ability to adapt to individual user preferences and regional variations in sign language usage. By incorporating these enhancements, we aspire to create a more inclusive and accessible sign language recognition solution for a global audience.

11.1.3 Speech to Text Feature

The speech-to-text feature was set as our reach goal but we did not accomplish it eventually due to the limited time. So, we aim to integrate this feature in future works, to provide a seamless bidirectional communication experience for users. This addition would enable non-ASL speakers to speak to our product in spoken language, which is interpreted and displayed in text on the ASL user faced side. Implementing this feature involves integrating speech recognition algorithms and natural language processing techniques to accurately transcribe spoken language into text and subsequently structure the sentences.

11.2 Lessons Learned

11.2.1 Overestimation/Underestimation

During the project, we encountered instances where our initial estimations of performance metrics and/or time required either underestimated or overestimated the actual outcomes. For instance, while we researched on multiple datasets and tried to expand training sources as much as possible, we underestimated the impact of complex and mixed datasets on prediction accuracy and latency. This lesson highlights the importance of a thorough and realistic assessment of potential challenges and variables that may influence system performance.

11.2.2 Time Allocation

Effective time management proved to be essential throughout the project progress. We learned that allocating adequate time for each phase was essential for meeting project milestones and ensuring the quality of deliverables. Notably, major difficulties could arise during integration, so in order to spot issues early and spare time for a thorough decision making, we could have continuously conducted intermediate integration and started early on final system integraion.

11.2.3 What We Want vs. What Is Possible

Throughout the development process, sometimes we found it hard to balance ambitious project goals with the practical limitations of technology and resources. While we aimed for over 90% translation accuracy and minimal latency across all communication channels, we encountered constraints such as domain shifts, overfitting, and serial communication limitations. This experience highlights the importance of aligning project objectives with realistic expectations, acknowledging constraints, and conditionally refining goals based on achievable outcomes.

11.2.4 Being Resourceful

Lastly, it is essential to take an effective use of resources from all domains. Leveraging available tools, such as alternative communication protocols and optimization techniques, enabled us to address translation latency issues on the OLED screen and enhance overall user satisfaction. This lesson underscores the significance of adaptability and proactive problem-solving in cracking obstacles and maximizing project outcomes within existing constraints.

Give Me A Sign packages the latest computer vision and machine learning technology into a simple phone attachment and mobile application. Gesture recognition and neural networks identify and interpret a user's movements into proper, grammatical English text, essentially delivering live "subtitles" of the user's ASL to the conversant the user is engaging with.

We expect our primary challenges are the implementations and polishing of our respective tasks. Numerous tests will have to be repeated with incremental changes and improvements made, but we are prepared for these hurdles and have established our Minimum Viable Product to be a translator at a satisfactory level, executing basic ASL alphabet and word recognition and text transmission.

If we complete our MVP earlier than expected, we defined three reach goals to improve our product. One is to implement a speech-to-text function so that the speaking recipient can also communicate through text to the non-hearing user, equalizing communication so the user does not have to rely on reading lips or other methods for understanding. Second, integrating a signal for the end of the user's sentence. In settings where many people are together, this feature can be turned on or off so that other people know the user has communicated. For the hearing community, other people can listen and automatically face someone to engage because of voice and noise, so this element in our product is to provide such an alert whether by an audio notification or a blink of light. Third, adding facial recognition because ASL incorporates much facial expression to instill meaning in its words.

Our product breaks down communication barriers with efficiency, practicality, and portability. It require no screws, glue, or coding efforts from the user. By propping the phone on a surface, the front-facing camera captures the user's ASL and the dual-screen technology channels translations to both user and recipient. Simply put on a phone, the product is naturally carried anywhere and everywhere, and just as easily removable. Fulfilling values of diversity, equity, and inclusion, Give Me A Sign aims to bridge the speaking, deaf, and hard-of-hearing communities together.

Glossary of Acronyms

- 3D-CNN - 3D Convolutional Neural Network
- ASL – American Sign Language
- AWS - Amazon Web Service
- BLE - Bluetooth Low Energy
- CLI - Command-line Interface
- CNN - Convolutional Neural Network
- CV – Computer Vision
- FPS - Frames Per Second
- GATT - General Attribute Profile
- GPT - Generative Pre-trained Transformer
- GRU - Gated Recurrent Unit
- IDE - Integrated Development Environment
- LCD - Liquid Crystal Display
- LiPo - Lithium Polymer
- LLM - Large Language Model
- LSTM - Long Short Term Memory networks
- ML - Machine Learning
- NLP - Natural Language Processing
- OLED - Organic Light Emittion Diode
- PLA - Polyactic Acid
- RNN - Recurrent Neural Network
- RPi – Raspberry Pi
- SPI - Serial Peripheral Interface
- TTFB - Time-to-First-Byte

References

[1] *NC DHHS: north carolina department of health and human services.* URL: `https://www.ncdhhs.gov/`.

[2] *What is Screen Distance?* URL: `https://support.apple.com/en-us/105007`.

[3] *Read Our Ultimate Guide To Lux vs Lumens vs Watts For Lighting Installations | Warehouse & Factory Lighting.* URL: `https://greenbusinesslight.com/resources/lighting-lux-lumens-watts/`.

[4] Huaizhong Zhu, Chao Deng, and Yuguang Zhu. "MediaPipe based Gesture Recognition System for English Letters". In: *Proceedings of the 2022 11th International Conference on Networks, Communication and Computing.* ICNCC '22. New York, NY, USA: Association for Computing Machinery, Apr. 2023, pp. 24–30. ISBN: 9781450398039. DOI: `10.1145/3579895.3579900`. URL: `https://doi.org/10.1145/3579895.3579900`.

[5] Anil Kag, Ziming Zhang, and Venkatesh Saligrama. *RNNs Evolving on an Equilibrium Manifold: A Panacea for Vanishing and Exploding Gradients?* Aug. 2019. URL: `https://arxiv.org/abs/1908.08574v2`.

[6] *Azure AI Translator | Microsoft Azure.* URL: `https://azure.microsoft.com/en-us/products/ai-services/ai-translator`.

[7] *Hand landmarks detection guide | MediaPipe.* URL: `https://developers.google.com/mediapipe/solutions/vision/hand_landmarker`.

[8] *Pose landmark detection guide | MediaPipe.* URL: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker.

[9] *OpenCV: Smoothing Images.* URL: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html#:~:text=Gaussian%20Blurring&text=It%20is%20done%20with%20the,directions%2C%20sigmaX%20and%20sigmaY%20respectively..

[10] *Opencv: gesture recognition.* URL: https://docs.opencv.org/4.x/d9/db7/group__datasets__gr.html.

[11] Ruchi Manish Gurav and Premanand K. Kadbe. "Real time finger tracking and contour detection for gesture recognition using OpenCV". In: *2015 International Conference on Industrial Instrumentation and Control (ICIC).* May 2015, pp. 974–977. DOI: 10.1109/IIC.2015.7150886. URL: https://ieeexplore.ieee.org/document/7150886.

[12] *LLM vs. NLP: 6 Key Differences and Using Them Together — kolena.com.* URL: https://www.kolena.com/blog/llm-vs-nlp-6-key-differences-and-using-them-together#llm-vs-nlp-6-key-differencesnbsp.

[13] *Core ML.* en-US. URL: https://docs.developer.apple.com/documentation/coreml (visited on 05/03/2024).

[14] *Leo Rover Blog - Raspberry Pi or Arduino – when to choose which?* URL: https://www.leorover.tech/post/raspberry-pi-or-arduino-when-to-choose-which.

[15] *Hand landmarks detection guide for Web | MediaPipe.* en. URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker/web_js (visited on 05/04/2024).

[16] *128 by 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller.* SSD1309. Rev 1.1. Solomon Systech Limited. 2011. URL: https://www.hpinfotech.ro/SSD1309.pdf.

[17] Syed Fouzan Iftekar et al. "Advancements and Limitations in 3D Printing Materials and Technologies: A Critical Review". In: *Polymers* 15.11 (May 2023), p. 2519. ISSN: 2073-4360. DOI: 10.3390/polym15112519. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10255598/.

[18] Thingiverse.com. *Modular Mounting System by HeyVye.* URL: https://www.thingiverse.com/thing:2194278.

[19] *Nano 33 BLE.* ABX00030. Rev 2, 2022. Arduino S.r.l. 2022. URL: https://docs.arduino.cc/resources/datasheets/ABX00030-datasheet.pdf.

[20] BrightSign Technology Limited. *Brightsign - about us.* URL: https://www.brightsignglove.com/about.

[21] *Hand Talk: your website accessible in ASL.* URL: https://www.handtalk.me/en/.

[22] *SLAIT – Real-time Sign Language Translator with AI — slait.ai.* URL: https://slait.ai/.

[23] *Jeenie | terms of use.* Aug. 2021. URL: https://jeenie.com/terms-of-use/.

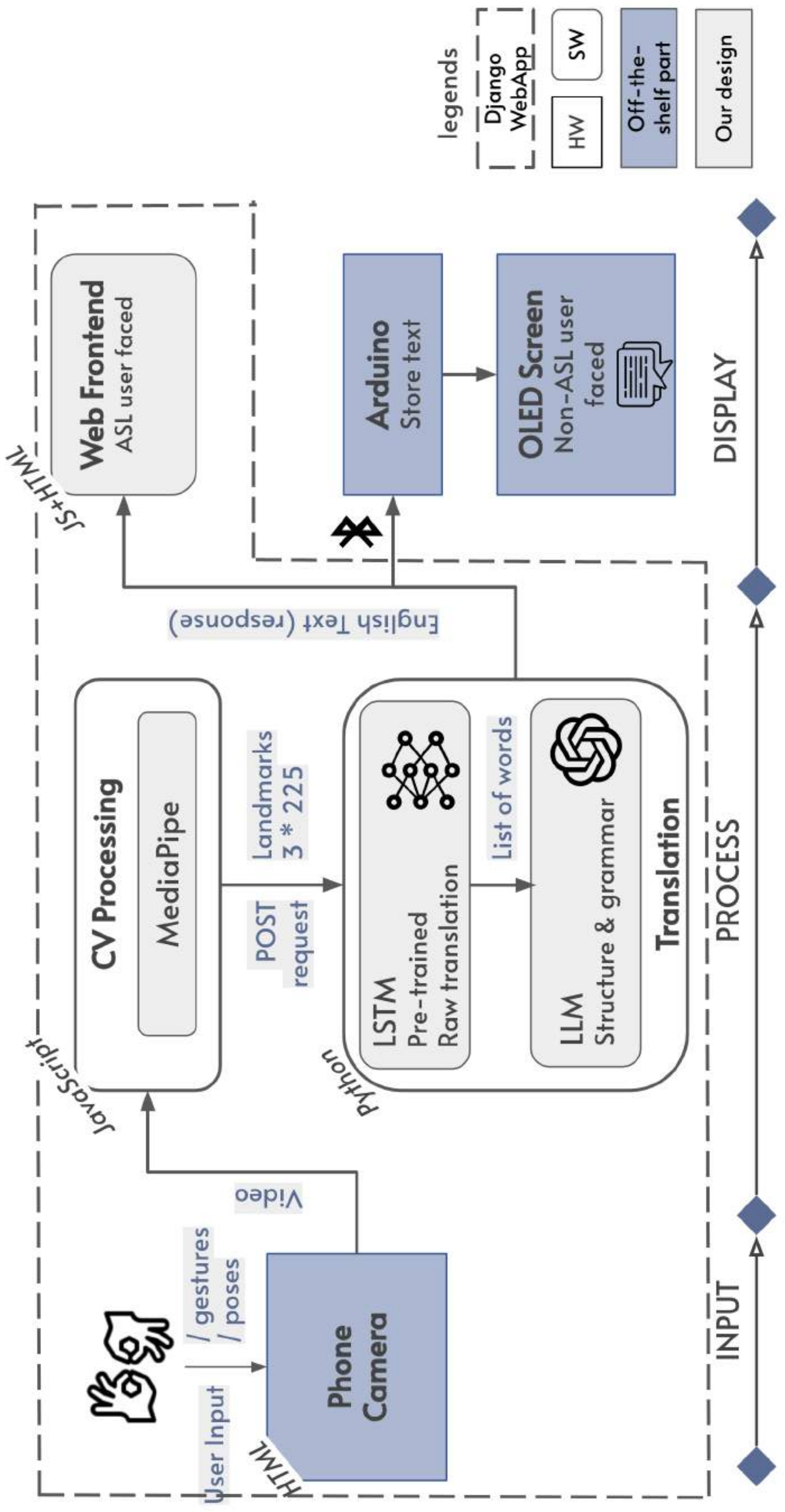


Figure 19: A full-page version of the same system block diagram as depicted earlier.

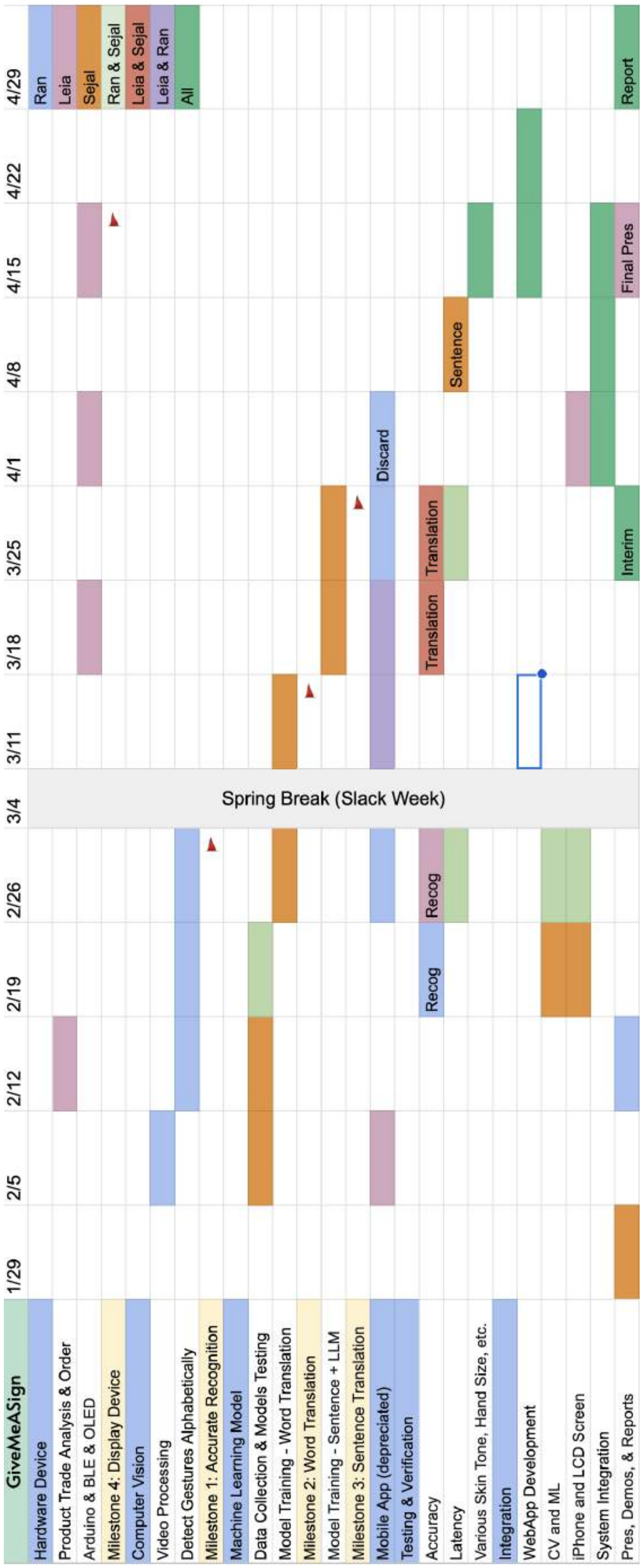


Figure 20: Gantt Chart