ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.1375

# SignVision: a web-based platform for Puerto Rican Sign language (PRSL) in real-time using AI

Merchan-Rueda, Diego; Ortiz, Anthony; Alvear-Suarez, Alcides; Vergara-Laurens, Idalides; Carvajal-Jiménez, Carmen; Masalmah, Yahya
Universidad Ana G. Méndez, Recinto de Gurabo, Puerto Rico
*dmerchan1@email.uagm.edu, aortiz615@email.uagm.edu, aalvear@uagm.edu, ivergara@uagm.edu*
*carvajalc1@uagm.edu, yamasalmah@uagm.edu*

***Abstract*– Communication barriers between deaf, nonverbal individuals and hearing people remain significant across various aspects of daily life. This article introduces an application for translating individual hand signs from the alphabet, based on the Puerto Rican Sign Language, into written text, as well as converting spoken language into text through speech recognition technologies. By integrating both gesture-based and voice-based input, the system promotes inclusive communication, especially in educational settings. Its main objective is to reduce communication barriers by offering an intuitive, real-time platform accessible through a standard webcam and microphone. The sign recognition process is based on the use of Artificial Intelligence techniques. The selected architecture combined the strengths of Multi-Layer Perceptron (MLP) and Multi-Layer Perceptron (MLP), allowing the system to learn both spatial patterns from individual frames and temporal dependencies across gesture sequences. To provide users with an accessible and real-time translation experience, a web application was developed using Django, a Python web framework. The system achieved accuracy exceeding 84%, demonstrating its capacity to recognize both static and dynamic hand gestures with high reliability**

***Keywords: sign language, artificial intelligence, machine learning, computer vision, software development.***

## I. Introduction

Sign language is a fully developed linguistic system that operates through visual-manual modality rather than auditory-vocal channels [1]. Unlike simple gestures, sign languages follow structured grammatical rules and possess rich phonological and syntactic components. Research has shown that sign languages are not universal but instead are culturally and linguistically distinct. Studies in gesture and sign language have demonstrated that these languages evolve naturally within Deaf communities and serve the same cognitive and communicative roles as spoken language. Their status as natural languages have been validated. However, in Puerto Rico, the Deaf and hard-of-hearing community faces substantial communication challenges. While early sign language acquisition is vital for cognitive and social development, systemic barriers limit access to interpreters, educational tools, and support services. Reports indicate a critical shortage of certified sign language interpreters across the island, depriving Deaf individuals of access to essential services in education, healthcare, and legal systems [2]. In addition, advocacy groups like LatinoJustice have filed complaints against agencies such as FEMA for failing to provide adequate communication resources during emergency responses in Puerto Rico, citing violations of civil rights protections [3]. These recurring issues underscore the urgent need for policy reform, professional training programs, and technological tools that facilitate communication equity for the Deaf population.

Multiple studies have taken various approaches to address the challenges of sign language recognition using artificial intelligence and deep learning. Chenghong Lu et al. [4] integrates bending sensors with vision-based systems, utilizing Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (BiLSTM) to process spatial and temporal features, achieving a recognition accuracy of 84.13% for Japanese Sign Language. On the other hand, Teena Varma et al. [5], focused on recognizing American Sign Language (ASL) using Convolutional Neural Networks. The researchers applied preprocessing techniques such as grayscale conversion, skin masking, and edge detection to enhance image quality before training CNN models, achieving up to 98% accuracy for static hand gestures of the ASL alphabet.

This work presents an application for translating individual hand signs from the alphabet into written text, as well as converting spoken language into text through speech recognition technologies. The project structure comprises two core topics: (1) sign language recognition via webcam and (2) voice recognition via microphone. Image frames captured through the webcam are analyzed using trained models to identify PRSL gestures, while speech input is transcribed using a voice-to-text engine. By integrating both gesture-based and voice-based input, the system promotes inclusive communication, especially in educational settings. Its primary objective is to reduce communication barriers by offering an intuitive, real-time platform accessible through a standard webcam and microphone. Its layout is intuitive, allowing users to easily access translation tools, account settings, and instructional content from a centralized menu.

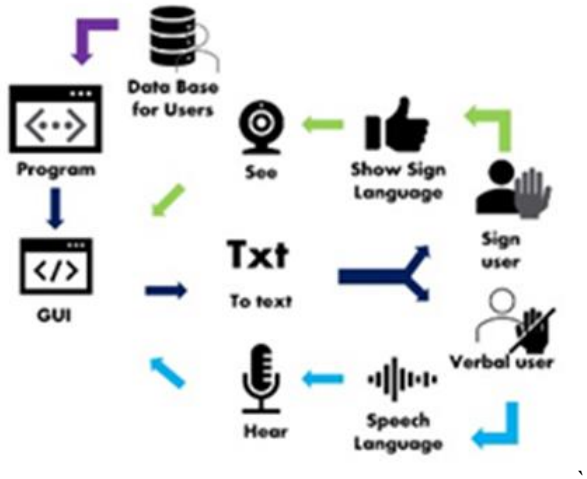


Fig 1. System schematic

Figure 1 represents the main features of the project. Sign language users will be able to access the system through the web application, and use their camera to display signs, and the trained model will recognize these signs and translate them into text, displaying the output to the user. Meanwhile, verbal users can use a microphone to record their voice, which will also be translated into text.

The system was designed focusing on accessibility, ensuring ease of use for individuals with varying technical expertise, especially those in educational and accessibility-focused settings. This user-centered design supports the project's mission to reduce communication barriers through a functional and inclusive digital tool. When the user activates sign language input via webcam, each video frame is captured in real time, encoded, and sent to the back end through AJAX. The server then decodes the frame, uses MediaPipe to extract hand signs, and converts them into feature vectors. This vector is passed to the appropriate trained model. The prediction is returned to the frontend as a JSON response and displayed as text on the screen. This real-time feedback loop ensures users receive immediate responses to their input, enabling smooth interaction between the user interface and the AI models. The recognition system is built upon three AI architectures. All three models were first tested independently to evaluate accuracy and then combined into a unified system to enhance robustness and versatility.

1) *Convolutional Neural Network (CNN).* Used for static image recognition, leveraging convolution and max pooling layers with ReLU activations to extract spatial features.

2) *Long Short-Term Memory (LSTM).* Handles temporal image sequences, using 10-frame video segments and a pre-trained CNN to extract frame-level features for input into the LSTM network.

3) *Multi-Layer Perceptron (MLP).* Processes 63-dimensional keypoint vectors (21 hand landmarks × 3D) extracted via MediaPipe and organized into CSV datasets. The model includes dense layers with ReLU activations and a softmax classifier.

The rest of the paper presents the technical background in section II, followed by a description of the Web application in section III. Then, section IV presents the machine learning model developed for this project. In supervised machine learning, data acquisition and data preprocessing are crucial, therefore, section V and VI present these important issues. Section VII introduces a description of the data visualization process followed by the model evaluation in section VIII. Since this is a software development project, the development methodology is presented in section IX. Finally, sections X and XI provide the conclusions and future work respectively.

## II. Background

This section introduces the three fundamental neural network architectures used in this project: Convolutional Neural Networks (CNN), Multi-Layer Perceptrons (MLP), and Long Short-Term Memory networks (LSTM). These three architectures form the computational backbone of SignVision, each selected for their strengths in handling image data, spatial keypoints, or sequential gesture patterns.

### *A. Convolutional Neural Network (CNN)*

A Convolutional Neural Network is a type of deep learning model specifically designed for image classification and pat-tern recognition in visual data [6]. CNNs are highly effective for spatial data, such as full-color images, and are often used in computer vision tasks. According to literature, CNNs consist of multiple layers, including: 1) Convolutional Layers to apply filters (kernels) to detect features such as edges, shapes, or textures within an image. 2) Activation Functions, typically ReLU (Rectified Linear Unit), are used to introduce non-linearity. 3) Pooling Layers to reduce dimensionality and computation. 4) Fully Connected Layers, at the end of the network, these layers map the extracted features to class probabil-ities.

### *B. Long Short-Term Memory (LSTM)*

LSTM is a specialized type of Recurrent Neural Network (RNN) designed to process sequential data and learn temporal patterns over time [7]. Unlike traditional RNNs, LSTMs have internal memory cells that help retain important information over long sequences and avoid the vanishing gradient problem. In this project, LSTMs are used to identify letters represented by movements across multiple frames such as "J" or "Ñ" in Puerto Rican Sign Language. Key components include 1) The input gate decides which new information to store. 2) The forgetting gate removes irrelevant information from memory. 3) The output gate determines what to output based on current memory state.

### C. *Multi-Layer Perceptron (MLP)*

A Multi-Layer Perceptron is a fully connected feedforward neural network composed of multiple layers of neurons. MLPs are best suited for tasks involving structured data like numerical vectors [8]. In this project, MLPs process keypoint features extracted from hand gestures, offering a lightweight and effective classification mechanism. A MLP is composed of 1) The input layer receives raw features (in this case, flattened key-point vectors). 2) The hidden layers contain multiple neurons, each computing weighted sum followed by an activation function like ReLU. 3) The output layer uses SoftMax activation to assign probabilities to each class.

## III. The Web Application

To provide users with an accessible and real-time translation experience, a web application was developed using Django, a Python web framework. The system was designed to process both visual input (sign language) and audio input (spoken Spanish) through an intuitive interface. This section presents the design and implementation of the front-end, backend, input processing, and routing components that support the translation prototype.

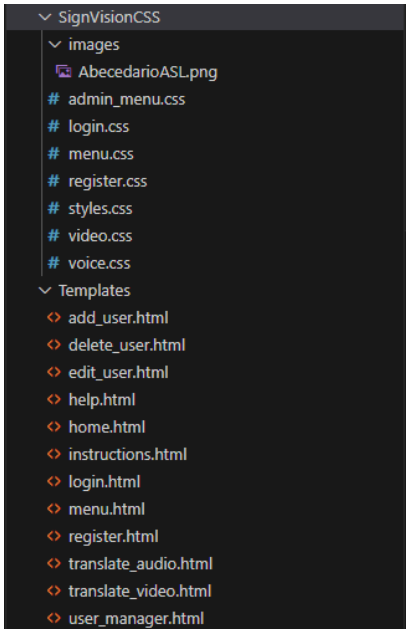


Fig 2. HTML and CSS Archives.

### A. *Frontend*

The frontend was created using HTML, CSS, and JavaScript to ensure a responsive and user-friendly interface. It includes a live video feed from the user's webcam, microphone input for speech recognition, and display elements to show translated text. Navigation options were also implemented to allow users to return to the main menu or refresh the interface. The interface prioritizes usability and clarity, making it accessible to users of varying technical backgrounds. Figure 7 shows the folder architecture for the front-end. As shown, the system allows to user management-create, edit and delete- as well as translate audio and video into text as shown in figures 3 and 4.


Fig 3. Audio translation into text interface.

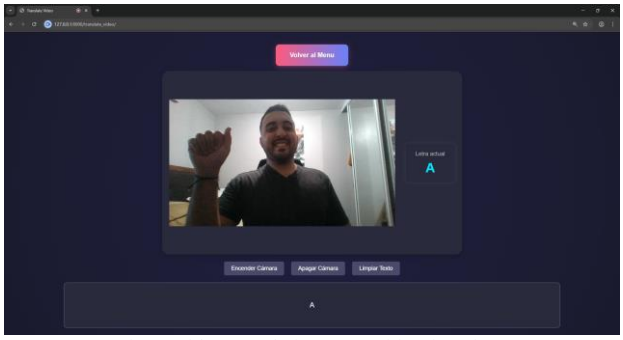


Fig 4. Video translation recognition interface.

### B. *Backend Architecture*

The backend of the SignVision system is developed using Django, a web framework based on Python. It integrates computer vision and audio processing components to interpret sign language and spoken Spanish. Some Django view functions were developed to handle various tasks such as routing, user authentication, real-time image and audio processing, and database interaction. The main translation function, process_frame, receives base64-encoded web-cam images from the frontend, decodes and preprocesses them using OpenCV, extracts hand keypoints with MediaPipe, and sends the resulting feature vector to a trained MLP model for classification. The predicted output is returned as a JSON response.

The main objective is to receive input data (images or audio), process it through AI models, and return the corresponding translated text. Image frames are captured via the user's webcam and sent to the server as base64-encoded strings using JavaScript. Once received by the Django view, each frame is decoded and processed using OpenCV and MediaPipe. MediaPipe detects the hand and extracts 21 hand landmarks, each with x, y, and z coordinates. These keypoints are then flattened into a feature vector and passed to a pre-trained machine learning model for classification. The models used include a Multi-Layer Perceptron (MLP) and a Long

Short-Term Memory (LSTM) network, both stored in the keras format.

The *navigator.mediaDevices.getUserMedia* is used to access the webcam and microphone, and asynchronous JavaScript (AJAX) requests to ensure minimal latency. This real-time interaction allows immediate visual feedback and enables users to alternate between sign and voice inputs seamlessly. Audio input is handled in parallel using another route that captures speech through the browser's built-in speech recognition engine. Specifically, the system uses the Web Speech API's SpeechRecognition interface, which enables real-time transcription of directly within supported browsers such as Google Chrome. This allows for lightweight voice processing without the need for external cloud services or backend audio analysis. For voice processing, the system includes a route (translate_audio) that records audio through the browser and uses the Google Cloud Speech-to-Text API for transcription. User management is handled through protected views such as add_user, edit_user, and delete_user, all requiring admin-level access.

The model loading is abstracted into a wrapper class called SignLanguageModel, which ensures that the trained model (stored as modelo_MLP_ver10.keras) is loaded once and reused efficiently during runtime. Exception handling, error messages, and AJAX responses are also integrated to support asynchronous communication.

### C. *Routing and Endpoint Handling*

All routing logic and endpoint definitions in the system are configured through Django's urls.py file. This file serves as the mapping layer between frontend requests and backend view functions. Each URL path is associated with a specific functionality, allowing modular and maintainable code structure. The /translate/ endpoint renders the main interface, displaying the webcam feed, translation results, and input controls. The /process_frame/ endpoint handles image data sent via AJAX, extracts key points, classifies the sign, and returns a JSON response. The /speech_input/ endpoint handles voice input, transcribes it using Google's API, and displays the result. These endpoints operate asynchronously, ensuring a fluid user experience without full page reloads. The separation of logic across views promotes clarity, scalability, and easier debugging.

## IV. Machine Learning Model

Throughout the development of this project, multiple supervised learning architectures were evaluated to address the task of letter recognition in Puerto Rican Sign Language (PRSL). This section describes the models considered, reasons for their selection or exclusion, and the architectural decisions regarding their internal layers.

Convolutional Neural Networks (CNNs) were initially used for their proven effectiveness in image classification tasks. The proposed architecture included several Conv2D layers with (3,3) filters and ReLU activation functions, followed by MaxPooling2D layers to reduce the spatial dimensions. A Flatten layer was used to convert the feature maps into a vector, which was passed through two dense layers: one with 128 units and another with 64 units. Finally, a SoftMax layer handled the classification. Despite its strengths, this model was ultimately discarded for real-time translation due to two primary limitations. First, it required large datasets containing full images to generalize effectively. Second, its computational cost was significantly higher, particularly when modeling subtle hand movements and variations in gesture shape. These constraints made CNN impractical for the limited and varied dataset available for PRSL.

The Multi-Layer Perceptron was incorporated as part of dual branch architecture with LSTM. Each hand gesture frame was processed to extract 21 hand landmarks, each with x, y, and z coordinates, totaling 63 features per image. These were fed into an MLP branch consisting of a Dense (128) layer with ReLU activation to learn complex spatial patterns, followed by a Dropout (0.5) layer to reduce overfitting, and another Dense (64) layer to refine the learned features. The output of the MLP was later concatenated with the output from the LSTM branch (see next subsection). This architecture allowed the model to jointly learn spatial representations from the MLP and temporal dynamics from the LSTM. A final Dense(num_classes) layer with SoftMax activation produced the predicted class probabilities. This configuration effectively balanced model depth and regularization, making it well-suited for sign recognition tasks using moderate-sized datasets.

LSTM networks were particularly important for recognizing dynamic signs involving sequential movements, such as J and Ñ. Each sample was reshaped into a sequence of shape (21, 3), where each of the 21-time steps represented a landmark with its 3D coordinates. The LSTM architecture included a single LSTM (64) layer to capture the temporal dependencies between frames, followed by a Dropout (0.5) layer to prevent overfitting. A final Dense layer was added for classification purposes. Although CNN+LSTM combinations were considered, they were ruled out due to their high computational cost when applied to full images.

The selected architecture combined the strengths of MLP and LSTM, allowing it to learn both spatial patterns from individual frames and temporal dependencies across gesture sequences. Each input consisted of a sequence of key points in the shape of (21, 3), capturing the full 3D structure of the hand as extracted by MediaPipe. The LSTM branch contained a LSTM(64) layer followed by a Dropout (0.4) to learn temporal motion patterns and mitigate overfitting. Simultaneously, the same input was flattened using a Reshape layer and processed through the MLP path with a Dense (128) ReLU layer, a Dropout(0.5), and another Dense(64) layer. Both branches were merged using a Concatenate() layer. The merged features passed through a Dense(64) ReLU layer and an additional Dropout, followed by a final Dense layer with SoftMax activation for multiclass classification. This architecture

achieved a validation accuracy above 84%, proving highly effective in recognizing both static and dynamic signs. It provided a well-balanced trade-off between performance, generalization, and computational efficiency.

## V. Data Collection

While sign language systems vary across countries, the alphabet component remains relatively consistent due to its universal nature. In Puerto Rico, PRSL (Puerto Rican Sign Language) is taught primarily using ASL (American Sign Language). Initially, the pre-existing ASL datasets from Kaggle, a public repository offering a wide range of image-based datasets, were used to train the system. These datasets provided good image quality and helped us develop our first training sets. However, three additional letters, J, Ñ, and Z, were manually included in the dataset, reflecting how they are signed locally. During the data collection sessions, several individuals, some of whom had prior knowledge of ASL from previous training, were photographed. These individuals clarified that, although ASL forms the foundation of PRSL, certain letters such as Ñ, LL, and RR are uniquely used. For this project, the double letters (LL and RR) were excluded to focus on core alphabetic signs guidance from the Puerto Rico Department of Education to align hand signs with local standards.

The initial dataset was designed for use with a CNN model, which required images taken from multiple angles under different lighting conditions, and from hands of various shapes, sizes, and skin tones. After determining that the original dataset was unsuitable for MediaPipe's hand detection, a key requirement for extracting keypoints we conducted new photo sessions [9]. From Kaggle's ASL datasets, we selectively extracted usable images and integrated them with our custom photo sets to train both MLP and MLP+LSTM models.

## VI. DATA PREPROCESSING

To train the machine learning models effectively, all collected images needed to be systematically organized in a local directory. This organization ensured easy access and simplified manipulation of data through code. Specifically, a folder was created for each letter of the alphabet, resulting in a total of 28 subfolders within a parent directory named dataset_train. Each subfolder represented a specific letter and contained either raw images or structured frame sequences, depending on the model requirements.

For the MLP and CNN models, each letter folder contained approximately 520 individual image samples. These images were stored as single frames since these models process static data. In contrast, when MLP or CNN were used in conjunction with LSTM—models that operate on sequential inputs—each letter folder contained 52 subdirectories. Each of these subdirectories housed a set of 10 consecutive frames, representing a short sequence of motion. This structure was designed to accommodate the temporal nature of the LSTM model, which relies on detecting changes across a sequence of frames.

Beyond physical data organization, the preprocessing procedure varied according to the type of model being used. For CNN-based models, the raw images were read using the OpenCV library, resized to a uniform resolution of 64x64 pixels, and normalized by dividing pixel values by 255. This ensured that input data remained consistent in size and scale across all training samples. For MLP-based models, the process focuses on extracting keypoints rather than using raw pixel data. Each image was passed through the MediaPipe library to detect 21 hand landmarks. The (x, y, z) coordinates of these landmarks were flattened into a 63-element feature vector per image. When LSTM was integrated, these vectors were grouped into sequences of 10 frames to match the temporal input structure expected by the model.

Ultimately, all preprocessed data—regardless of whether they came from CNN, MLP, or combined models—were stored in NumPy arrays (X for features and y for labels). This enabled efficient and scalable training of the machine learning models across all architecture variations.

### *A. CNN Preprocessing*

For the CNN model, the preprocessing stage operated directly on full color images. Each image was loaded using the Python library OpenCV and resized to a standardized dimension of 64x64 pixels to ensure consistency across the dataset. The pixel values were then normalized by dividing by 255.0, effectively scaling the RGB values to a range between 0 and 1. This normalization not only ensured uniformity across samples but also helped reduce memory and computational usage during training. However, it is important to note that reducing the resolution and scaling the pixel values may slightly diminish image quality, potentially affecting model performance. Once preprocessing was completed, the processed image data was stored in NumPy arrays (X), along with their corresponding labels (y), enabling efficient and structured training input. This format allowed the model to handle large datasets without excessive computational burden. When combining CNN with an LSTM model, the pre-processing approach shifted. Instead of processing images individually, the data was structured into sequences of 10 consecutive frames per set. These sequences were encoded

### *B. MLP Preprocessing*

Unlike CNNs, which rely on raw pixel data from images, the preprocessing stage for the MLP model was centered on extracting numerical key point data from hand landmarks. Instead of feeding the model full color images, MediaPipe Hands was used to detect 21 specific hand landmarks in each image, capturing spatial coordinates in the (x, y, z) format. As a result, each image was transformed into a flat vector of 63 values (21 landmarks × 3 coordinates), which formed the input

features for the MLP model. This process was conducted by traversing the dataset folder by folder and image by image. For every valid image, the hand was identified using MediaPipe, and if detected, its 21 key points were extracted and flattened. These key point vectors were then stored in NumPy arrays (X), while their corresponding class labels were encoded and stored in array (y). This structured format allowed efficient storage and training for the neural network.

When the MLP was integrated with an LSTM model, preprocessing followed a slightly different path. Since LSTMs require sequential data, key points were grouped in 10-frames sets to reflect a time dependent movement or gesture. Each group of 10 sequential hand poses was compiled into a single sample. The final shape for each input sample became (10, 63), capturing the temporal evolution of the hand's key point positions. To ensure high quality in the data, only images where hands could be reliably detected by MediaPipe were included. This filtering was critical, as undetected or partially detected hands could result in incomplete or invalid input vectors, degrading model performance. Ultimately, this approach allowed the MLP to effectively learn spatial patterns from individual hand gestures, while the LSTM component learned temporal patterns across frames. The resulting preprocessing pipeline ensured consistent input structure, model readiness, and optimal use of computational resources.

## VII. Data Visualization

To obtain a deeper understanding of the data used across different models, the shape of each NumPy array was verified to ensure that the expected dimensions and class counts were correctly maintained. This verification step was crucial to confirm that no data was missing or misaligned during preprocessing. Beyond this, specific visualizations were implemented for each model type to further explore the structure and integrity of the input data.

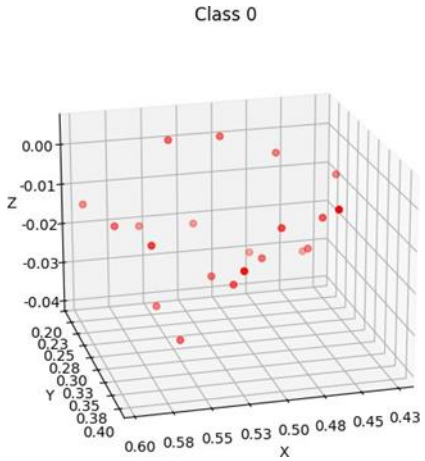


Fig 5. Preview of the visualization of key points for class A

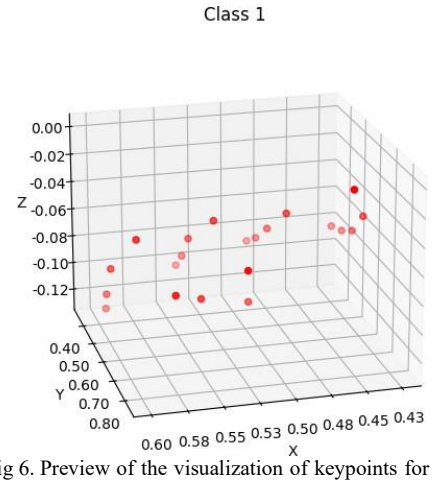


Fig 6. Preview of the visualization of keypoints for class B

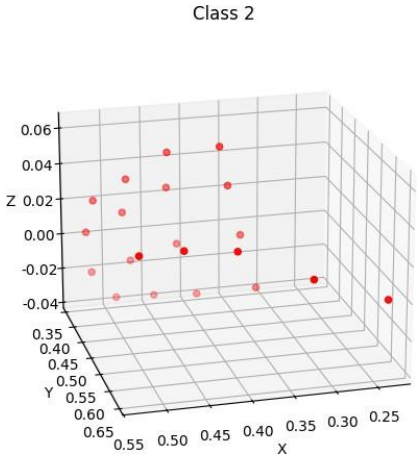


Fig 7. Preview of the visualization of keypoints for class C

### A. *MLP and LSTM Visualization*

For the MLP and LSTM models, which rely on keypoint data extracted via MediaPipe, 3D scatter plots were created using Python's extttmatplotlib library. These plots visualized the spatial distribution of the 21 hand signs in (x, y, z) coordinates for each example. Such visualizations were particularly helpful in comparing classes, for instance, class A versus class B and C, as shown on figures 5, 6 and 7—and ensured that the MediaPipe extraction process correctly captured meaningful hand structures. These plots also served to identify potential issues in the dataset, such as corrupt or inconsistent entries.

### B. *CNN Visualization*

In the case of the CNN model, which processes raw image data, visualization focused on confirming that images were properly resized to 64imes64 pixels, normalized, and correctly loaded into memory. By randomly sampling and displaying a subset of preprocessed images, we could verify that the conversion and formatting steps (e.g., resizing and scaling RGB values between 0 and 1) had been applied uniformly across the dataset.

### C. *Training Visualization*

Throughout the training phase of each model, performance was continuously monitored and visualized. Several metrics were plotted such as *extttttrain_accuracy*, *extttval_accuracy*, *extttttrain_loss*, and *extttval_loss* over the course of multiple epochs to better understand the model's learning behavior. These plots provided insights into potential issues like overfitting, underfitting, or convergence stagnation, allowing for timely adjustments in model architecture or training strategy.

## VIII. Model Evaluation

The selected architecture combined the strengths of MLP and LSTM, allowing it to learn both spatial patterns from individual frames and temporal dependencies across gesture sequences. This architecture achieved a validation accuracy above 84%, proving highly effective in recognizing both static and dynamic signs. It provided a well-balanced trade-off between performance, generalization, and computational efficiency. Figure 8 shows the accuracy progression during training and validation over 50 epochs. The training accuracy (blue) increases steadily, while the validation accuracy (orange) converges above 80%, indicating strong generalization and learning capacity of the model without signs of overfitting.

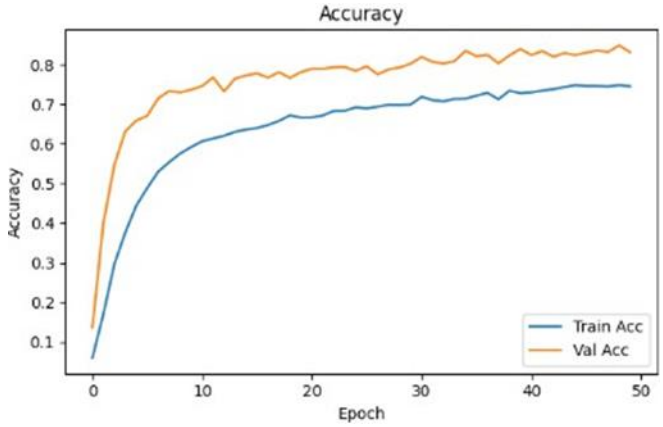


Fig 8. Model accuracy

Figure 6 shows the training (blue) and validation (orange) loss over the same 50 epochs. Both curves show a consistent decrease, with validation loss stabilizing around 0.5. This confirms that the model is effectively minimizing errors and converging during training.

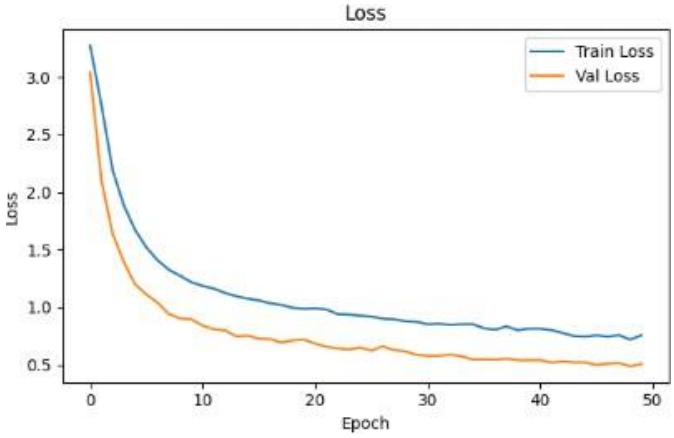


Fig 9. Validation loss

## IX. Development Technique

To ensure efficient collaboration and task organization throughout the project development, the *ScrumBan* methodology, supported by Jira, was adopted. This approach allowed for continuous progress tracking, flexibility in adapting priorities, and seamless coordination even when unforeseen issues delayed specific tasks. Jira served as a centralized platform where team members could plan development sprints, assign responsibilities, and visualize progress. Tasks that encountered delays or unresolved technical problems were marked as "Blocked" and temporarily paused, allowing the team to focus on other tasks in parallel. Once the issues were resolved, those tasks were resumed without disrupting the overall workflow. This agile strategy made it easier to coordinate meetings, review code collaboratively, and manage deliverables effectively.

## X. Conclusions

This project successfully designed and implemented a multimodal system capable of translating Puerto Rican Sign Language (PRSL) and spoken Spanish text using machine learning and web technologies. The combination of Convolutional Neural Networks (CNN), Multi-Layer Perceptrons (MLP), and Long Short-Term Memory (LSTM) architectures allowed for extensive experimentation with various data modalities, ultimately leading to the selection of the MLP + LSTM hybrid model for its superior balance of performance, com-putational efficiency, and versatility. The system achieved an accuracy exceeding 84%, demonstrating its capacity to recognize both static and dynamic hand gestures with high reliability. The preprocessing strategy that transformed raw image data into structured keypoint vectors using MediaPipe is a key factor for this project. This, along with careful model tuning and sequential training techniques, enabled the models to generalize effectively across different user inputs and environmental conditions.

A user-friendly web application was developed using Django, HTML, CSS, and JavaScript, providing a uniform interface for sign and voice input through webcam and microphone. The real-time feedback loop, integrated through AJAX and Web APIs, ensures that users receive immediate results while maintaining smooth interaction. Additionally, the project's database design allows for robust user management, logging, and future scalability via MySQL. The adoption of an agile Scrumban methodology, supported by Jira, was instrumental in managing the complexity of the project. This workflow ensured timely delivery, flexibility in adjusting priorities, and clarity in team coordination. Overall, the SignVision system represents a significant contribution toward inclusive communication technologies. It bridges the gap between hearing individuals and the Deaf or non-verbal community, particularly in educational settings. The system is both scalable and adaptable, opening the door for future improvements, such as real-time sentence construction, expanded gesture datasets, mobile compatibility, and advanced natural language processing. Through a combination of machine learning, web development, and human-centered design, this project lays the groundwork for future efforts in accessible, AI-driven communication platforms.

## X. Future work

To enhance the SignVision system performance, several improvements and expansions are proposed for both the machine learning models and the web application. These future enhancements aim to make the system more robust, scalable, and user-friendly while further improving accessibility and performance across diverse user environments

### *A. For the Machine Learning Model*

- Obtain more photos to enrich the training dataset. Use more diverse data for improved model generalization.
- Collect data from different people to capture a wider variety of hand shapes and gestures.
- Improve the performance of the current model by optimizing its architecture and training strategies.

### B. For the Web Application

- Ensure compatibility across multiple devices (desktop, tablet, and mobile).
- Improve the voice recognition functionality to support clearer and faster transcription.
- Make the application accessible on multiple browsers beyond Google Chrome.
- Implement local voice recognition to reduce dependency on third-party APIs and enhance privacy.

## References


[1] J. O. Barreto Abrams y L. L. Dowtin, "Deaf Early Intervention in Puerto Rico: A Qualitative Study," The Journal of Early Hearing Detection and Intervention, vol. 7, no. 1, pp. 108–120, 2022. Disponible: https://digitalcollections.sit.edu

[2] B. Belcher, "An analysis of communication-related information and services offered to parents of deaf children in Puerto Rico," Cap-stone Collection, no. 3283, SIT Graduate Institute, 2023. Disponible: https://digitalcollections.sit.edu/capstones/3283

[3] A. S. Gaafar, J. M. Dahr, y A. K. Hamoud, "Comparative Analysis of Performance of Deep Learning Classification Approach Based on LSTM-RNN for Textual and Image Datasets," Informatica, vol. 46, pp. 21–28, 2022. Disponible en: https://doi.org/10.31449/inf.v46i5.3872.

[4] C. Lu, M. Kozakai, and L. Jing, "Sign Language Recognition with Multimodal Sensors and Deep Learning Methods," Electron-ics, vol. 12, no. 4827, pp. 1–15, Nov. 2023. [Online]. Available: https://doi.org/10.3390/electronics12234827

[5] T. Varma, R. Baptista, D. Chithirai Pandi, and R. Coutinho, "Sign Language Detection using Image Processing and Deep Learning," In-ternational Research Journal of Engineering and Technology (IRJET), vol. 7, no. 11, pp. 338–344, Nov. 2020.

[6] Shahzadi, F. Meriadeau, T. B. Tang, and A. Quyyum, "CNN-LSTM: Cascaded Framework for Brain Tumour Classification," 2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES), pp. 633–637, 2018. DOI: 10.1109/IECBES.2018.8641994.

[7] T. Tatsunami and T. Taki, "Sequencer Deep LSTM for Image Clas-sification," 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pp. 490–494, 2019. DOI: 10.1109/ICAIIC.2019.8669002.

[8] T. Lv, C. Bai, and C. Wang, "MDMLP: Image Classification from Scratch on Small Datasets with MLP," arXiv preprint, 2022. [Online]. Available: https://arxiv.org/abs/2205.14477.

[9] Google AI, "MediaPipe Hands," Medi-aPipe Documentation. [Online]. Available: https://mediapipe.readthedocs.io/en/latest/solutions/hands.html