AI-Based Sign Language Translation Enhancing Human Interaction

Niranjan Vajramushti, Nivedita Shainaj Nair, Steven Godinez, Sutton Spindler
CS5100 - Foundation of Artificial Intelligence
Northeastern University, Boston, MA

Abstract

Millions of sign language users worldwide often face barriers in communication due to the lack of accessible, multilingual sign language translation tools. This project presents an AI-driven solution that recognizes and translates signs from American Sign Language (ASL), German Sign Language (DGS), and Spanish Sign Language (LSE) using a unified deep learning pipeline. Leveraging a convolutional neural network (CNN) trained on a custom-curated and standardized dataset, the system integrates MediaPipe for real-time hand tracking, TensorFlow for gesture classification, and natural language processing (NLP) for output refinement and translation. By accommodating overlapping visual gestures across languages and correcting for class imbalances, the model offers robust, multilingual recognition. The system bridges the gap between signed and spoken communication, making strides toward inclusive and adaptive accessibility technologies for the deaf and hard-of-hearing communities.

I Introduction

Sign language recognition and translation continues to be a multifaceted challenge that requires not only accurate hand gesture classification, but also an understanding of linguistic context, user intent, and language-specific nuances. Traditional models often focus on a single sign language, limiting their usefulness in real-world, multilingual contexts. This project aims to address that gap by developing a unified system capable of recognizing and translating signs from three distinct sign languages, American Sign Language (ASL), German Sign Language (DGS), and Spanish Sign Language (LSE), within a single machine learning model.

At the core of this system is a deep learning pipeline built around a convolutional neural network (CNN) architecture. The model is trained on preprocessed datasets of ASL, DGS, and LSE signs, all standardized to consistent input formats and merged where signs are visually identical. The unified dataset allows the model not only to classify signs but also to infer the signed language being used by the user based on hand shape in live video input.

To capture the user's signing in real-time, the project integrates MediaPipe for precise hand landmark detection, TensorFlow for model training and inference, and natural language processing (NLP) techniques for post-processing the output. Once the signed input is transcribed, contextual NLP models are used to correct likely misclassifications, such as misinterpreted letters or missing spaces, by referencing patterns in language usage. Finally, the cleaned transcription is translated into the user's target spoken language, making the system a complete pipeline for multilingual sign language recognition and translation.

This project not only advances accessibility tools for deaf and hard-of-hearing communities but also lays the groundwork for future AI systems that operate across language boundaries and adapt to diverse user inputs in real time.

II Related Works

The domain of sign language recognition has evolved considerably with advancements in computer vision and deep learning. Traditional models typically focus on recognizing signs from a single language, limiting cross-linguistic application. Early efforts primarily used handcrafted feature extraction and sensor-based methods (e.g., data gloves or Kinect sensors), which, while accurate, were impractical for real-world use due to hardware dependencies.

Recent trends have shifted toward vision-based models powered by convolutional neural networks (CNNs), which allow for real-time classification of hand gestures from images or videos. Systems like Sign Language MNIST and various Kaggle datasets have provided a foundation for ASL recognition using image-based classification. However, they often fail to generalize across multiple sign languages or account for signs that appear visually similar across languages but differ semantically.

Several studies have built on this CNN foundation with different augmentations. One approach combined CNNs with Long Short-Term Memory (LSTM) networks to capture temporal dynamics in continuous signing [1]. That system also integrated speech recognition and OpenCV-based hand tracking to support both recognition and generation of sign language, aiming for a more complete communication pipeline. While promising, the model was still language-specific and required well-framed inputs to maintain accuracy.

Other projects have emphasized accessibility and real-time responsiveness. For instance, Kumar and Singh designed a lightweight pipeline using Google's MediaPipe framework to extract hand landmarks, then fed those into a Random Forest classifier to predict sign labels [2]. They further translated the recognized text into speech using gTTS, targeting real-time applications on constrained devices. However, despite its re-

sponsiveness, the approach was limited in handling ambiguous or overlapping signs across different regional sign languages.

The issue of regional variation is especially pronounced. Bewoor et al. highlighted how most models struggle with differences in hand positioning, occlusions, and a lack of representative datasets across sign languages [3]. To address this, they proposed a ResNet100-based CNN trained on augmented data to recognize signs despite minor distortions or visual noise. Yet even this more robust architecture treated the sign language system as monolithic, without accounting for cross-linguistic relationships.

Some systems have explored alternative platforms entirely. Torres and Alday, for example, developed a smartphone-based application that uses Kotlin and MediaPipe to translate live camera input into sign text. This mobile approach brought gesture recognition to more casual settings but, like others, still focused on one language and lacked higher-level linguistic correction.

Taken together, these projects illustrate the field's shift from hardware-intensive solutions toward lightweight, vision-based models powered by CNNs or hybrid architectures. However, very few attempt to bridge multiple sign languages in a single system, or to disambiguate visually similar signs that differ in meaning across linguistic contexts. Moreover, while NLP has occasionally been applied to improve translation fluency, it is rarely integrated directly into the sign recognition pipeline.

This project builds upon such prior work by introducing a unified model capable of recognizing and distinguishing signs across ASL, DGS, and LSE. It uniquely addresses class overlap by merging visually identical signs across languages, introduces robust data preprocessing using MediaPipe, and applies NLP for contextual translation correction. This represents a novel step forward in multilingual sign language translation using AI.

III Implementation Details

1. CNN Model Training

The proposed method makes use of a CNN to classify static images of signs drawn from American Sign Language (ASL), German Sign Language (DGS), and Spanish Sign Language (LSE). The dataset consisted of preprocessed hand gesture images of these three sign languages which was sourced from publicly available Kaggle datasets. A few weeks into training and testing, it was determined that the CNN model was biased against predicting ASL images due to the fact that the images sourced were darkly lit. As a result we recreated the ASL image dataset ourselves. There was a particular focus on combining visually identical signs into unified class folders when appropriate. For example, "E" is different in all the three sign languages, which meant 3 "E" classes, one corresponding to each language. "A" on the other hand has an identical sign across every language, which meant all the "A" dataset images were combined into a single class. Images were included only if a hand was detected using MediaPipe Hands, otherwise they were rejected and not used during training.

Since each individual class originally had 100 images per language (eg, 100 ASL "A" images, 100 DGS "A" images, etc.), combining them into a shared class introduced class imbalance. For instance, the unified "A" class had 300 images, while a distinct class like "B" from LSE remained at only 100. To address this imbalance, we applied class weighting during training to ensure that less frequent classes had proportionally higher influence in the loss function, mitigating the risk of underrepresentation.

For each class, images were uniformly resized to 128x128 pixels and normalized for model input. All training images were of right hands, and so that the model would work for left-handed users in addition to right-handed users, all images were flipped horizontally and re-added to the preprocessed set for training.

The classification model was implemented using TensorFlow and Keras. It consists of sequential convolutional layers with ReLU activation, batch normalization, and max pooling, followed by dropout for regularization and fully connected layers for classification. The final layer uses a softmax activation function to output class probabilities across all sign categories.

A custom data generator was developed to load and augment images in real-time. Augmentations such as random flips, brightness changes, and slight rotations were applied to the training set to enhance generalization. The generator also split the data into training and validation subsets with a 90/10 split ratio and applied shuffling per epoch.

The model was trained using the Adam optimizer with a learning rate of 0.001 and categorical cross-entropy loss. Class weights were calculated and applied during training to compensate for remaining class imbalances. The model was trained for 30 epochs using a batch size of 64. Performance was monitored on a held-out validation set, and the final model was saved upon completion.

2. Object Detection and Feature Extraction

The sign language detection system implements a sophisticated approach to hand tracking and feature extraction using MediaPipe Hands, an open source framework that provides precise hand landmark detection. This component serves as the critical first stage of the pipeline, responsible for capturing and pre-processing visual input before classification.

Hand Detection and Landmark Tracking: The system initializes MediaPipe Hands with specific parameters optimized for sign language recognition:

- `static_image_mode=False`: Enables continuous tracking between frames.
- `max_num_hands=1`: Focuses on a single hand to reduce ambiguity.
- `min_detection_confidence=0.7`: Sets a high confidence threshold to minimize false positives.

These parameters ensure robust tracking even under varying lighting conditions and complex backgrounds. When a hand is detected in the camera feed, MediaPipe extracts 21 precise

landmarks representing key points on the hand, including the finger joints, fingertips, and the center of the palm.

Image Processing Pipeline: Once hand landmarks are detected, the system implements a sophisticated preprocessing pipeline:

- Bounding Box Extraction: The system calculates the minimum and maximum coordinates across all hand landmarks to create a dynamic bounding box around the hand.
- Adaptive Padding: A 30-pixel padding is added to the bounding box to ensure that the entire hand is captured, even during gesture transitions.
- Aspect Ratio Preservation: The system implements an algorithm that maintains a square crop while preserving the hand's aspect ratio:

```
if crop_width > crop_height:
  # Width is the larger dimension,
  # adjust based on width
  center_y = (y_min + y_max) // 2
  half_crop = crop_width // 2
  y_min = max(0, center_y - half_crop)
  y_max = min(h, center_y + half_crop)
else:
  # Height is the larger dimension,
  # adjust based on height
  center_x = (x_min + x_max) // 2
  half_crop = crop_height // 2
  x_min = max(0, center_x - half_crop)
  x_max = min(w, center_x + half_crop)
```

- Normalization and Resizing: The cropped hand image is resized to a standardized 128×128 pixel format and normalized to pixel values between 0 and 1, matching the input requirements of the CNN classifier.

Stability Enhancement with Temporal Delay: To mitigate misclassifications resulting from transient or unstable hand poses, the system incorporates a temporal stabilization mechanism. Upon initial detection of a hand, a timer is triggered. The system continuously monitors the consistency of the detected hand landmarks over a predefined duration (2 seconds). If the hand remains within acceptable positional thresholds for the entire interval, the system sets the flag `ready_for_detection = True`, signaling readiness for classification. This delay introduces a crucial buffer period, allowing users to stabilize their hand gestures. Additionally, the interface provides real-time visual feedback by changing landmark color from cyan to green, confirming successful stabilization and readiness for sign recognition.

3. Text Processing and NLP Integration

The main challenges to consider when implement the pipeline was, how do we determine what language of the three is being used? How do we obtain the dataset? how do we get the dataset? How do we know where to segment the words? And how do we spell check?

The first question tackled was obtaining the dataset. The foundation of the pipeline had to be based off of words, and transitions. A Reddit scrapper that scrapes from specifics sub-reddits was the most optimal solution. Sub-reddits contain informal text that are used in day to day dialogue. This will also mean our dataset will include slang, informal text, and depending on the sub-reddit, formal text. After many trial and errors on how the dataset should be format, two types of storage were decided. A dictionary and a dictionary of dictionaries, which we will get back to. Our dataset is trained on the following sub-reddits: AskReddit, neu, todayilearned, AmIthe[Jerk], CasualConversation, and confession.

The nature of a Natural Language Processing pipeline is to process human language and in our case mimic its structure. Initially, I believed a Hidden Markov Model would fit most appropriately as [4] they are modeled to find the "probabilistic relationship between a sequence of observations and a sequence of hidden states". However, due to the inclusion and complexity of day to day conversation. A model that assumes each emission, observed word / token, is independent of the rest lacks to capture transitions states between words or better known as, context. My solution was to transition the classic HMM to a Bigram Hidden Markov Model.

$$P(o_t|s_t, o_{t-1})$$

A Bigram Hidden Markov models introduces dependencies to the previous observation, o_{t-1}, that help capture word state transitions in a sequence, while eliminating emissions; each word its is own state and our algorithm tracks transition probabilities between words. This alongside with its limitations, gave a solution to state transitions that is able to keep context as the most probable next word sequence using a Viterbi algorithm that has a faster runtime. However, the BHMM is still limited as it cannot accurately capture long term sequences with out it being computationally expensive where it slows down outputs, ruining the quality of the software.

This way we can have our data set into 4 files for each language. Start Count, Start Probability, transition count and transition probability where having a dictionary of word-probabilities and a dictionary of word to most likely next probable word fuels our bigram model.

Our word segmentation implement a Viterbi-based algorithm that determines the most probable sequence of words for the given input. The algorithm calculates different possible segmentation based on probabilities, transition and start. For each segmentation we then get a combined score that includes a bonus for recognized words that promotes splitting up of words which ensures valid words are favored over unknown sequences. This is induced with a fail safe as if the input of characters fails to find a segmentation, a greedy approach segments based on max word probability. This is then passed to the spell correction and finally to the Bigram HMM to build coherent sentences. This pipeline effectively process multilingual signs with high accuracy as tested with many sequences including inputs like "igofast" to "i go fast" and "iloveyo" to "i love you"

From here, the next challenge was figuring out what language is being used to use the corresponding data set. We

developed a confidence counter that counts every sign detected and increments the given language, if a sign shares multiple language, those are all incremented as well. Once the user passes the input by clicking enter, the model is able to extract the most probable language that is then passed to the pipeline for the bigram model.

Because our data set is compact enough to run locally, It was almost impossible to create an adequate spell checker that was based off words our model knew. To fix this issue I implemented an existing library, SpellChecker, to handle that portion.

When compiling the final pipeline fail safes were placed where the model, segmentation, or spell check might fail. For example if the model could not determine what language was used, maybe the confidence counter had two equal entries, our NLP would default to using english. Moreover, if the user was inputting a word our model has never encountered before the NLP simply returns the same exact sequence. This was done to consider the fact that new slang, or a small data set might fail if project is used extensively.

Ultimately, we developed a NLP pipeline that supports multilingual sign languages, ASL, DGS, and LSE; Using a Bigram Hidden Markov Model implementation, we were able to successfully capture word relationships with context, segment for accurate sequences, correct spelling, and most probable sequence predition.

4. Application Interface and Workflow

The sign language detection system utilizes an intuitive graphical user interface built with Pygame that facilitates real-time sign recognition and language processing. The application follows a structured workflow to ensure accurate detection and translation of sign language input.

System Initialization: When launched, the application performs the following setup procedures:

- Initializes Pygame for interface rendering
- Establishes connection to the webcam for video input
- Loads the pre-trained CNN model for sign classification
- Sets up MediaPipe Hands for landmark detection
- Initializes the NLP pipeline components for text processing

Main Application Loop: The core functionality operates within a continuous loop that:

- Captures frames from the webcam
- Processes frames to detect and track hand landmarks
- Implements a temporal delay mechanism for stable detection
- Classifies detected hand signs when stability conditions are met
- Updates the accumulated text based on detection results
- Renders the GUI components with real-time feedback
- Handles user interaction through keyboard and mouse events

User Interface Components: The GUI is organized into distinct panels that provide visual feedback:

- A header section with application title and instructions
- A main camera feed panel showing the detected hand with landmark overlays
- A hand crop panel displaying the processed hand image used for classification
- A detection results panel showing the current sign and confidence score
- A text accumulation panel displaying the constructed phrase with editing options

Multilingual Processing: The system incorporates language detection and processing capabilities:

- As signs are detected, a confidence counter tracks language probabilities
- The system infers whether ASL, DGS (German), or LSE (Spanish) is being used
- When the user presses Enter, the appropriate language-specific NLP pipeline processes the text
- Word segmentation, spelling correction, and linguistic refinement are applied
- The processed text is displayed in the text accumulation panel

User Interaction Flow: The application provides intuitive controls for user interaction:

- Visual feedback through color-coded landmarks (cyan for positioning, green for ready)
- Countdown timer display during hand positioning
- Cooldown period after detection to prevent unintended duplicate entries
- Keyboard shortcuts: Enter to process text, C to clear, B for backspace, Q to quit
- On-screen buttons for text editing operations

IV Results

The model finished the 30-epoch training with a self-assigned 98% validation accuracy. To evaluate the performance of the trained CNN model, we used the held-out validation set constructed during preprocessing. The dataset contained a mix of unified and language-specific sign classes, enabling us to assess the model's ability to generalize across signs that are either visually identical or subtly different across ASL, DGS, and LSE.

The confusion matrix in Figure 1 shows the difference between prediced labels and true labels when the model performed the static images that was part of the validation sets. The model performed the best on unified sign classes such as "A", that had consistent hand shapes across languages and benefited from a more balanced, combined datasets. Classification accuracy was also strong for distinct signs such as B in LSE. Although LSE B is not a combined class and thus had less training images, no other sign looks similar to it which causes the model to identify it easily.

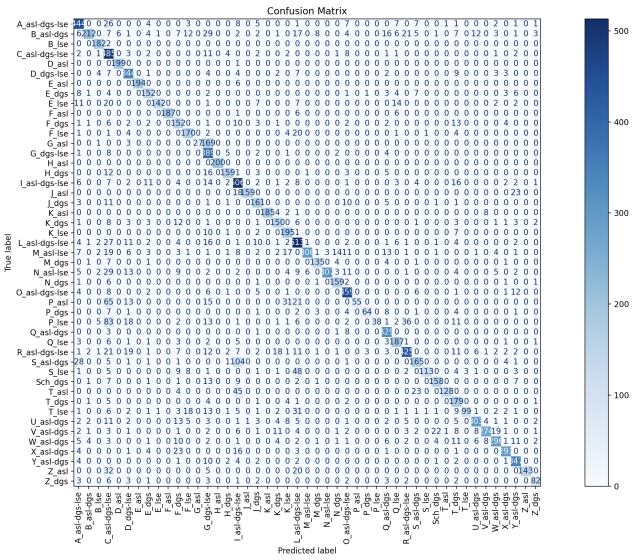

Figure 1. Confusion matrix of final model, with combined classes

However, the model struggled more often with signs that were unique to their language and had smaller distinctive features. Although the confusion matrix does not indicate this, when testing with a live video feed the ASL sign for “Z,” which involves a closed fist with an outstretched index finger, was frequently misclassified as “Y,” which uses an extended thumb and pinky.

In addition to improvements in visual classification, the overall user experience was enhanced by a redesigned NLP module for context-aware translation and auto-correction. Earlier attempts at using a basic Hidden Markov Model (HMM) for sequence prediction proved unreliable on longer or more complex sign sequences, particularly due to the limited scope of our scraped dataset. To address this, we implemented a Bigram Hidden Markov Model (BHMM), which removed the need for emissions and leveraged probabilistic transitions to better retain linguistic context. This change significantly improved the accuracy and fluidity of final transcriptions, particularly when signs were strung together in real-time.

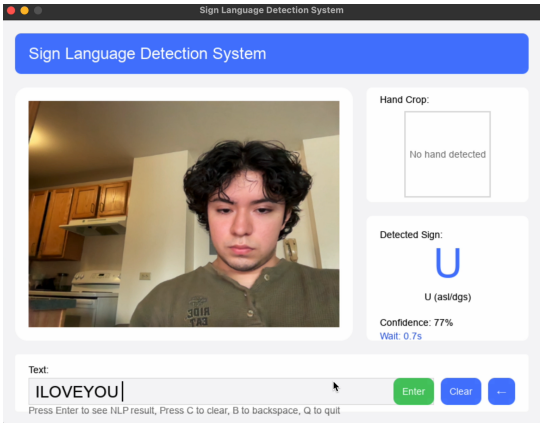

Figure 2. GUI display before NLP segmentation

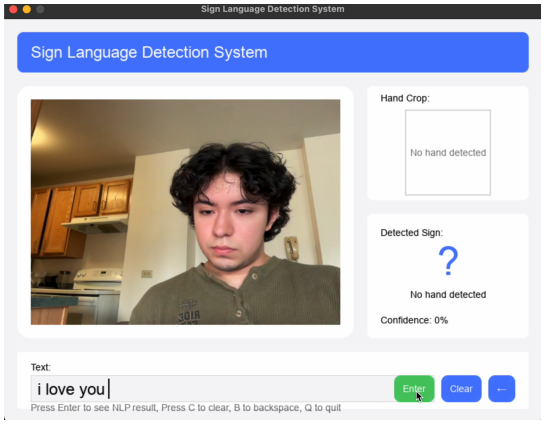

Figure 3. GUI display after NLP segmentation

From the following image segmentation occurs by cutting into the text input character by character until an initial word is found that exist in word probabilities folder. From here it is assigned a score, this is done for all words found in the input, if no word is found or no potential segmentation it is given a low score. This is ran multiple times for different possible word segmentation to find if a better score exist for the given segmentation. if no score exist you default to returning the best guess.

V Discussion/Conclusion

As described in the results, the model is most accurate when predicting unified signs and struggles most with visually similar but distinct classes across languages.

Although the confusion matrix did not show the ASL "Z" training images being mistaken for "Y", this was in fact happening when using the live video feed. This meant that some other component was discouraging the model from predicting the ASL class. The ASL dataset images were all taken from the right hand of the same individual, whereas the DGS dataset images and especially combined classes (like "A") used the hands of many individuals, which would prevent bias toward factors such as hand shape, texture, or skin tone.

This is likely the reason the model performed best with the combined classes. Going forward, the model will likely perform better if it is trained on a more diverse collection of dataset images. Training on a wider variety of lighting, hand positions, and real-world settings would help improve generalization.

Despite project and outside limitations, this project provides a useful foundation for static sign recognition across multiple sign languages with natural language processing. With future extensions to handle motioned signs and longer sequences of character for context, this work could support more inclusive, multilingual sign language translation systems.

Future work should focus on implementing a predictive NLP model that runs parallel with sign detection, rather than sequentially. This enhancement would allow the system to suggest word completions and predict subsequent words while the user is still signing, significantly reducing the time required

to construct sentences and providing real-time feedback during the communication process.

VI Team Contributions

Niranjan developed the GUI integrating all components into a user-friendly interface with real-time feedback, stabilization, and error handling.

Sutton and Nivedita built the preprocessing pipeline using MediaPipe, applied augmentations, addressed ASL dataset bias by curating new data, and led CNN training with hyperparameter tuning.

Steven enhanced the NLP module for context-aware translation and auto-correction, significantly improving transcription accuracy and overall user experience.

VII Link to code repository

To view the source code and read details about running this project, visit our GitHub repository: github.com/godinezsteven1/AI-SignLanguage

References

[1] S. Zhang, C. Zhang, and Z. Liu, "Sign language recognition based on computer vision," in *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, 2021, pp. 1221–1225.

[2] A. Kumar and A. Singh, "Real-time hand sign language translation: Text and speech conversion," in *2024 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, 2024, pp. 112–117.

[3] M. Bewoor, A. Mehta, A. Ukey, H. Gade, and D. Dixit, "Ai & machine learning to convert sign language to english script," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, vol. 11, no. 12, pp. 1487–1494, 2023. [Online]. Available: https://www.researchgate.net/publication/382066607_AI_MACHINE_LEARNING_TO_CONVERT_SIGN_LANGUAGE_TO_ENGLISH_SCRIPT

[4] GeeksforGeeks, "Hidden markov model in machine learning," https://www.geeksforgeeks.org/hidden-markov-model-in-machine-learning/, 2023, accessed: 2025-04-16.