INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 3, March 2026 | ISSN: 2456-4184 | IJNRD.ORG

MULTIMODAL SIGN LANGUAGE COMMUNICATION SYSTEM FOR THE HEARING AND SPEECH IMPAIRED

[1]Amruta Naik, [2]Ritesh Choudhary, [3]Mrunal Salekar, [4]Ayush Chaudhari, [5]Yash Dhok

[1]Assistant Professor, [2345] B.Tech Students,
[12345]Department of Information Technology,
[12345] Kavikulguru Institute of Technology & Science, Ramtek, India

Abstract: Communication can be quite a hurdle for those with hearing and speech impairments, especially since they often rely on sign language while most people don't. This paper introduces a Multimodal Sign Language Communication System — a comprehensive web-based platform designed to bridge that gap through three closely integrated modules. The first module focuses on real-time hand gesture recognition: it uses MediaPipe to pull two-dimensional hand landmark coordinates from a live webcam feed. These coordinates are gathered and stored using a custom data collection pipeline, which then trains a Random Forest Classifier from scikit-learn. The resulting model, serialized with Python's pickle library, boasts an impressive classification on the test set and is made available through a Flask-based web server for live inference. The second module is all about translating text to sign and speech to sign: a Django web application takes typed or speech-transcribed text, processes it through an NLTK pipeline that includes tokenization, POS tagging, tense detection, stop word removal, and lemmatization, and then presents the resulting token sequence as sequential ISL animation clips (in MP4 and GIF formats), complete with a character-level alphabetic fallback for any unknown words. The third module captures audio input via the Web Speech API, transcribing speech directly in the browser and sending the result through the same NLP and animation pipeline. Together, these three modules create a single deployable application that enables bidirectional, hardware-free, and real-time sign language communication, all accessible from any standard web browser without the need for specialized software or devices.

IndexTerms - Indian Sign Language, Sign Language Recognition, MediaPipe, Random Forest Classifier, NLTK, Text-to-Sign Translation, Speech-to-Sign, Web Speech API, Flask, Django, scikit-learn, Assistive Technology, Multimodal Interface.

I. INTRODUCTION

For People who are deaf or have speech impairments mainly use sign language to communicate, but unfortunately, most people around them don't know it. This creates a significant communication gap that impacts their access to education, jobs, healthcare, and even basic social interactions. In India, where millions rely on Indian Sign Language (ISL) as their primary means of communication, the lack of affordable and accessible translation technology makes this issue even more pressing. There just aren't enough professional interpreters available, and hiring one can be quite expensive, not to mention they're often unavailable in everyday situations. Over the last twenty years, various tech-based solutions have been suggested, but many of them just aren't practical for real-life use. The early systems needed wearable gadgets like data gloves or colored finger markers, which made them both costly and cumbersome.

More recent deep-learning methods have shown better accuracy, but they require powerful GPU hardware and offline processing, making them inaccessible for the average consumer. Another major drawback of nearly all existing tools is that they only hand le one-way communication: they either recognize signs from a user or convert text into sign animations, but they don't do both. This paper introduces a Multimodal Sign Language Communication System that tackles these challenges head-on.

The system combines three unique but interconnected modules into a single web application. The first module gathers ISL hand gesture data using MediaPipe, trains a Random Forest Classifier with scikit-learn, and deploys the trained model via Flask for real-time sign-to-text recognition. The second module is a Django web application that processes typed input using NLTK and creates ISL animations from a library of pre-recorded MP4 and GIF clips, ensuring full alphabetic fallback coverage. The third module builds on the second by accepting live voice input through the Web Speech API and routing the browser-transcribed speech through the same NLP and animation pipeline. The result is a bidirectional, hardware-free, browser-deployable communication tool accessible to any user with a standard laptop or smartphone. Several mobile applications claiming to offer sign language support exist on app stores, but the majority either cover only a limited set of alphabets, require manual input rather than real gestures, or are designed for American Sign Language (ASL) rather than Indian Sign Language (ISL). ASL and ISL are entirely different languages with different handshapes, grammar, and vocabulary, making ASL-based tools useless for Indian deaf users.

II. LITERATURE SURVEY

The field of sign language technology draws from computer vision, machine learning, and natural language processing. Reviewing prior work in each of these areas reveals both what has been achieved and the specific gaps that motivate the current system.

A. Hardware-Dependent Early Systems

In the early days of automated sign language recognition, the technology relied heavily on direct interaction with the signer. Data glove systems, for instance, utilized flex sensors to measure the angles of the fingers, providing precise hand configuration data. However, this came with the downside of requiring the signer to wear cumbersome hardware. On the other hand, color-marker systems involved placing uniquely colored fingertip covers on the signer and used color histogram segmentation to track hand movements through camera footage. Al-Qurasshi et al. [1] took a closer look at these early methods and pointed out a significant limitation: the accuracy of recognition was heavily dependent on controlled environments that were hard to replicate outside of a lab setting. Unfortunately, neither of these approaches could effectively scale for casual conversations in public spaces.

B. Vision-Based Deep Learning Approaches

The introduction of lightweight pose estimation frameworks like MediaPipe from Google changed the game, allowing for the extraction of detailed hand skeleton information from regular webcam video without the need for any wearable devices. Several studies have leveraged these skeletal representations as input features for traditional machine learning classifiers. For example, Random Forest classifiers trained on flattened MediaPipe landmark vectors have demonstrated impressive accuracy in recognizing Indian Sign Language (ISL) alphabets and common word gestures, all while consuming significantly fewer computational resources than deep convolutional networks. This makes them ideal for real-time applications on browsers and desktops [2]. Sindhu et al. [3] showed that skeleton-based features trained with ensemble classifiers can effectively generalize across various signers and lighting conditions, which is a crucial aspect for any tool intended for real-world use.

C. NLP-Integrated Translation Systems

Creating grammatically correct sign language output from text input involves more than just a simple word-for-word translation. Indian Sign Language, for instance, structures sentences in a Time-Subject-Object-Verb order, unlike the Subject-Verb-Object format of English. Additionally, tense is indicated through specific temporal signs rather than through verb inflections. Yadav et al. [4] addressed this in their Listen framework by integrating an NLP rule engine that identified tense from POS tags and restructured token sequences accordingly before driving a 3D avatar renderer. Their approach achieved high sentence-level accuracy but required substantial 3D graphics infrastructure. Reddy et al. [5] demonstrated that a lighter text-to-animation approach with NLP preprocessing, using NLTK for tokenization, lemmatization, and stop word filtering, is feasible within a standard web server and produces output that hearing-impaired users find comprehensible for common conversational sentences.

D. Audio and Speech Integration

Antad et al. [6] took a closer look at multilingual sign language translation and found that voice input is the most crucial feature missing from current tools. This is because most hearing individuals tend to prefer speaking over typing when they communic ate. The Web Speech API, which is natively supported in all Chromium-based browsers, offers real-time speech-to-text transcription without needing any server-side audio processing. This makes it a perfect fit for sign language translation systems that operate within a browser. Kothwal et al. [7] developed a browser-based ISL translation system that uses pre-recorded animation clips, JavaScript for sequential playback, and a character-level fallback for words that are not recognized, achieving a remarkable 100% input coverage. The current work builds on this framework by incorporating a Random Forest gesture recognition backend and a comprehensive NLP pipeline, all while keeping the animation and fallback features intact.

III. DRAWBACKS OF EXISTING SYSTEMS

Each generation of sign language technology has brought improvements, but a consistent set of limitations continues to restrict practical adoption. Understanding these shortcomings directly shaped the design decisions behind the proposed system.

A. Dependence on Wearable or Specialized Gear

Systems Using glove-based systems or depth cameras can be a real hassle, both physically and financially, making them impractical for every-day, spontaneous conversations. Any setup that requires more than just a standard laptop camera isn't likely to be useful for the many hearing-impaired users who really need it.

B. Sensitivity to Environmental Conditions

Tech that relies on color segmentation and background subtraction tends to falter when the lighting shifts or when the signer is in a busy environment. If a system only works in bright, uncluttered spaces, it can't be counted on for genuine communication situations.

C. Lag and Offline Functionality

A Systems that take a few seconds to process input or that work in batches just don't cut it for conversations, where quick responses are crucial. For real-time communication, you need a model and processing setup that can operate within a single frame cycle.

D. Limited to Static Gesture Recognition

Classifiers that only recognize static hand shapes can't pick up on dynamic signs that rely on movement, changes in hand orientation, or interactions between both hands. This severely restricts the vocabulary to just a tiny portion of any sign language, making these systems inadequate for natural conversations.

E. One-Way Translation

The Most available tools only translate in one direction—either from sign to text or text to sign. This means users have to depend on another tool or a person for the other direction, which defeats the purpose of achieving fully independent communication for deaf and hearing-impaired individuals.

F. Lack of Sentence-Level NLP Processing

Systems that translate words without considering grammar often produce outputs that don't follow the structure of ISL sentences, which diminishes the clarity and effectiveness of the translation. Without features like tense detection, stop word removal, and lemmatization, the resulting sign sequence can end up being confusing or misleading for native ISL users.

IV. PROPOSED SYSTEM AND ARCHITECTURE

The proposed system brings together three independently developed modules into one seamless web application: a sign-to-text gesture recognition module, a text-to-sign NLP and animation module, and a speech-to-sign audio module. Each of these modules focuses on a specific communication direction and technology area, and they can all be accessed through a common browser interface—no software installation or extra devices needed.

A. Overall System Design

The system is divided between two server frameworks. The gesture recognition module operates on a Flask server, which takes in webcam frame submissions, runs inference with the loaded Random Forest model, and sends back predicted sign labels in JSON format. Meanwhile, the text and speech translation modules are hosted on a Django server, which takes care of user authentication, NLP processing, static animation file resolution, and rendering HTML templates. A shared frontend interface, crafted with HTML5, CSS3, and plain JavaScript, connects with both servers to deliver a consistent user experience. This dual-server setup allows each component to utilize its best-suited Python framework while keeping the user interface uniform.

B. Hand Gesture Data Collection

The data collection process is handled in collect_imgs.py and create_dataset.py. MediaPipe Hands is set up with a minimum detection confidence threshold and is applied to each frame captured from the webcam. For every hand detected, MediaPipe provides the normalized x and y coordinates of 21 key landmarks, which include the wrist, four knuckle joints for each finger, and the fingertip of each finger. This results in 42 coordinate values for each hand per frame. The collect_imgs.py script captur es and organizes image sequences for each gesture class into specific directories, while create_dataset.py processes each stored image through MediaPipe to extract landmark arrays and compiles them into a data dictionary saved as data.pickle using Python's pickle module.

C. Training the Random Forest Model

The training process is set up in the train_classifier.py file. First, we load the data from the data.pickle file and use numpy.concatenate to flatten the landmark arrays into a single one-dimensional feature vector for each sample. Next, we split the dataset into training and test sets with the help of train test split from sklearn.model selection, using a test size of 20% and stratified sampling to keep the class balance intact. We then create a Random Forest Classifier from sklearn.ensemble with the default hyperparameters and fit it to the training data. Once the training is complete, we evaluate the model's accuracy on the test set using accuracy score from sklearn.metrics. The model consistently achieves an impressive classification accuracy of 95 to 98 percent on the held-out test set. Finally, we serialize the trained classifier using pickle.dump and save it as model.p for deployment.

D. Real-Time Inference with Flask

The inference module is implemented in app.py within the sign2text component. When the Flask application starts up, it loads the trained Random Forest model from model.p. When a webcam frame is sent via a POST request, the server processes the frame through MediaPipe Hands to extract the landmark coordinates, flattens them into the same feature vector format used during training, and then calls model.predict to get the predicted gesture class. The predicted label is sent back as a JSON response to the browser, where it's displayed in real time as the recognized sign character or word. This inference pipeline runs within a single frame cycle, allowing for smooth real-time recognition without any noticeable delay.

E. NLP Text Preprocessing Pipeline

The When text input arrives at the Django server—whether it's typed by a user or transcribed from speech—it goes through the animation view in views.py. The NLTK word tokenize function breaks the input down into individual tokens. Then, nltk.pos tag assigns Penn Treebank POS tags to each token, and we use the distribution of verb tags to figure out the dominant tense: MD tags point to future tense, VBD and VBN tags indicate past tense, and VBG tags show present continuous. To keep everything in line with ISL grammatical conventions, we prepend corresponding temporal markers ('Will' for future, 'Before' for past, 'Now' for present continuous) to the token list. After that, we apply a custom set of stop words, and then the WordNetLemmatizer from NLTK comes into play, reducing each token to its base form based on the POS tag to choose the right lemmatization strategy.

F. Text-to-Sign Animation Rendering

For every token in the processed list, the Django animation view uses finders.find to search for a matching MP4 animation file in the static assets directory. If it finds a match, the filename gets added to the word list sent to the frontend. If there's no match, the token is broken down character by character, and each character's alphabet animation is included instead. This way, we ensure that every input string results in a visible signed output, no matter the vocabulary coverage. On the frontend, a JavaScript controller plays each animation clip one after the other using the on-ended event, making sure each sign finishes completely before the next one starts, creating a smooth and readable animation sequence.

G. Speech-to-Sign Module

The speech input module kicks in the browser's Speech Recognition interface from the Web Speech API when the user clicks the microphone button. This API streams real-time transcription results straight from the device microphone right in the browser, without sending any audio data to the server. Each finalized transcript segment is appended to the text input field and simultaneously submitted to the Django NLP endpoint, where it is processed identically to typed input. The resulting animation sequence plays through the same frontend controller, allowing hearing individuals to speak naturally while the system renders the equivalent ISL sign sequence for a deaf counterpart.

V. METHODOLOGY

A. Gesture Data Collection Procedure

We gathered ISL gesture data by capturing hand movements in front of a regular consumer webcam under three different lighting scenarios: bright overhead lights, softer ambient lighting, and a mix of natural and artificial light. We used two plain backgrounds for consistency. Five participants helped us out, bringing in a natural variety of hand sizes, skin tones, and signing styles. For each gesture category, we made sure to capture at least 100 images per participant using the collect_imgs.py script. Each image was processed with MediaPipe Hands to extract landmark coordinates, which we saved in a file called data.pickle. In the end, our dataset included ISL representations for the letters A to Z, numbers 0 to 9, and a selection of common conversational words.

B. Model Training and Evaluation

We trained our model using the train_classifier.py script on a standard laptop CPU, without any GPU support, showcasing how accessible the Random Forest method can be. We split our data into an 80/20 train-test ratio with stratified sampling to ensure that both sets had a balanced representation of classes. We used the default hyperparameters from scikit-learn's Random Forest Classifier (100 estimators and Gini impurity criterion), which were more than enough to hit our target accuracy of 95 to 98 percent without needing any manual adjustments. The trained model was saved as model.p and loaded by the Flask server at startup for quick, zero-latency inference.

C. Text and Speech Pipeline Testing

We put the Django NLP pipeline to the test with 100 sentences pulled from educational, healthcare, and everyday conversation topics. To check tense detection accuracy, we compared the added temporal marker against a manually assigned ground truth label for each sentence. We also tested the animation rendering pipeline with an expanded word list that included proper nouns, technical terms, and abbreviations to ensure that the character-level fallback produced correct output for every token without any errors. The speech module was tested across multiple speakers and acoustic environments to confirm consistent transcription quality from the Web Speech API.

D. Technology Stack

The gesture recognition module is built using Python 3.x, with MediaPipe 0.10.21 for extracting landmarks, scikit-learn 1.3.2 for the Random Forest Classifier, NumPy 1.24.4 for handling array operations, and Flask 3.0.3 serving as the web server. For the text and speech module, we utilize Django 4.1.9 as our web framework, NLTK 3.7 for natural language processing, and SQLite for storing user accounts. On the frontend, we stick to plain HTML5, CSS3, and vanilla JavaScript. OpenCV 4.8 comes into play in

our data collection and inference scripts for capturing webcam frames and processing images. All of these libraries are open-source and available for free, and the whole stack operates on CPU hardware without needing a GPU.

It is simply the complete list of all technologies, programming languages, frameworks, libraries, and tools that are used together to build a software application. Our tech stack = Python + Flask + Django + MediaPipe + Random Forest + NLTK + HTML/CSS/JS + Web Speech API.

VI. RESULTS AND DISCUSSION

A. Gesture Recognition Accuracy

The Random Forest Classifier, trained on MediaPipe hand landmark features, consistently hit a classification accuracy between 95 and 98 percent on the 20 percent test set we held out. This impressive accuracy was achieved using the default hyperparameters from scikit-learn, without any manual adjustments, showing that the flattened 2D landmark coordinate vectors from MediaPipe are incredibly useful for classifying ISL gestures. Recognition remained strong across various lighting and background conditions represented in the training data. The most common mix-ups happened between visually similar hand shapes like M/N and R/U, which have nearly identical landmark geometry.

B. Real-Time Inference Performance

The Flask inference endpoint managed to process each submitted frame in under 50 milliseconds on a standard dual-core laptop CPU, which is well within the limits needed for a smooth real-time display. Since the Random Forest model works with a 42-element numerical feature vector instead of raw image data, the inference process is light on resources and doesn't require a GPU. The model.p file that loads at Flask startup is about 3.7 MB in size, leading to minimal memory overhead.

C. NLP Pipeline Output Quality

The tense detection system achieved an impressive match rate of over 90 percent when compared to manually labeled ground truth in a test set of 100 sentences. Most errors occurred in sentences with multiple verbs of different tenses, where the majority-vote method struggled to clear up the confusion. Stop word filtering successfully kept all content words intact in every test case. Lemmatization delivered the expected canonical form for more than 97 percent of tokens, with the remaining errors mainly involving irregular verb forms that the default NLTK WordNet mapping didn't cover.

D. Animation Coverage and Fallback Reliability

The character-level fallback was activated for every single proper noun, technical term, and any words missing from the animation library in the extended test set. Not a single input string resulted in an empty output or an unhandled exception. The on-ended-driven sequential playback ensured smooth animation sequences for sentences containing up to fifteen words, with no clips dropped or overlapping.

E. Comparative Performance

Table I provides a comparison of the proposed system against notable prior approaches based on eight practical deployment criteria.

Table I: Comparison of Proposed System with Prior Approaches

Criteria	Glove-Based	Color Marker	3D Avatar	GIF-Based	Proposed System
Hardware-Free	No	No	Partial	Yes	Yes
Real-Time Speed	Yes	Yes	No	N/A	Yes
Two-Way Translation	No	No	No	No	Yes
Voice Input Support	No	No	No	No	Yes
NLP Text Processing	No	No	Yes	No	Yes
Unknown Word Fallback	No	No	No	Yes	Yes
Runs in Browser	No	No	No	Yes	Yes
ML Classifier Used	Yes	No	CNN/LSTM	No	Random Forest

F. User Acceptance

Informal testing with twelve participants—six who were new to ISL and six from the information technology department—showed that everyone was able to complete three assigned tasks on their first try: translating a sentence into sign animation, using voice input for sign output, and having their hand gestures recognized as text. Feedback after the tasks highlighted bidirectionality and he lack of any installation requirement as the system's standout advantages over tools they had used before.

VII. APPLICATIONS OF PROPOSED SYSTEM

The system is designed to be used right away in educational settings, allowing hearing-impaired students to engage in classroom discussions without needing a dedicated interpreter. In healthcare environments, the speech-to-sign feature lets medical staff communicate naturally while the system translates their words into sign language for deaf patients, enhancing the quality of communication during consultations and emergencies. Public service areas like government offices, bank branches, and transport information kiosks are ideal places for this system to act as a self-service communication tool. In corporate settings, the gesture recognition feature enables deaf employees to join meetings by signing, while their colleagues receive live text translations. Educational video platforms can also incorporate the animation rendering feature to provide ISL dubbing for instructional videos. Thanks to its browser-based design and lack of hardware requirements, all these applications can be implemented without needing to change existing workstation setups.

VIII. CONCLUSION

This paper has introduced the design, implementation, and evaluation of a Multimodal Sign Language Communication System. It combines real-time hand gesture recognition using MediaPipe and a Random Forest Classifier, NLP-driven text-to-sign translation with NLTK and Django, and browser-native speech-to-sign conversion through the Web Speech API into one cohesive web application. The gesture recognition module boasts an impressive classification accuracy of 95 to 98 percent for ISL hand gestures, all thanks to a lightweight Random Forest model that operates in real time on standard CPU hardware, without the need for a GPU or specialized equipment. The text and speech modules ensure comprehensive bidirectional coverage of any input text through an NLP preprocessing chain and a character-level animation fallback. The combined system addresses the core limitations of prior approaches — hardware dependency, unidirectionality, vocabulary gaps, and processing latency — within a freely deployable, browser-accessible platform. The work provides a practical and replicable foundation for continued development toward fully grammar-compliant, mobile-deployable, and community-validated assistive communication technology.

REFERENCES

[1] M. Al-Qurasshi, T. Khalid, and R. Souissi, "Deep Learning for Sign Language Recognition," Institute of Electrical and Electronics Engineers (IEEE), 2021.

[2] K. S. Sindhu, Mehnaaz, N. Biradar, L. Varma, and C. Uddagiri, "Sign Language Recognition and Translation Systems for Enhanced Communication," IEEE International Conference on Innovations in Information and Communication, 2024.

[3] C. C. K. Reddy, C. A. Reddy, and C. S. Rohith, "Real-Time Sign Language and Audio Conversion Using AI," IEEE Journal of Emerging Technologies in Computing, 2024.

[4] S. M. Antad, S. Chakrabarty, S. Bhat, S. Bisen, and S. Jain, "Sign Language Translation Across Multiple Languages," IEEE Transactions on Human-Machine Systems, 2024.

[5] P. Yadav et al., "Harnessing AI to Generate Indian Sign Language from Natural Speech and Text," International Journal of Advanced Computer Science and Applications, vol. 15, no. 4, 2024.

[6] G. N. Kothwal, S. C. Chavan, N. R. Chavan, H. S. Chavan, H. R. Chendwankar, and R. A. Chavan, "Text to Sign Language Translator," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), vol. 5, no. 1, pp. 494–502, Nov. 2025. DOI: 10.48175/IJARSCT-29668.

[7] Indian Patent Application IN201911012932, "Artificial Intelligence-Based Indian Sign Language Translator," Indian Patent Office, 2019.

[8] N. K. Kahlon and W. Singh, "Machine Translation from Text to Sign Language: A Systematic Review," Universal Access in the Information Society, 2023.

[9] T. Ananthanarayana, L. Chaudhary, and I. Nwogu, "SignNet: Single Channel Sign Generation using Metric Embedded Learning," arXiv preprint arXiv:2010.10858, 2020.

Copyright & License:

© Authors retain the copyright of this article. This work is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.