iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** III **Month of publication:** March 2026

DOI: https://doi.org/10.22214/ijraset.2026.78789

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

Real-Time Indian Sign Language Recognition and Multilingual AI Chatbot

Dr. D. Anusha[1], P. Sravya[2], N.L. Prasanna Kumar[3], K. Yaswanth[4], MD. Yasin[5]
[1]*Head of Department of Computer Science Engineering(AI&ML), SRK Institute of Technology, Andhra Pradesh, India*
[2, 3, 4, 5]*Department of computer science and Engineering (AI & ML), SRK Institute of Technology, Andhra Pradesh, India*

Abstract: ***Over six million deaf and hard-of-hearing individuals in India rely on Indian Sign Language (ISL) as their primary mode of communication. Despite this, they continue to face significant barriers in accessing healthcare, education, and public services, largely because real-time multilingual translation tools remain unavailable at scale. Existing systems typically produce static text output in English alone. This paper presents a fully web-based ISL multilingual chatbot that addresses this gap end-to-end. The system employs a MobileNetV2 convolutional neural network trained on a dataset of hand gesture images spanning A–Z and 0–9 signs, achieving high accuracy in real-time webcam-based recognition. The recognized gesture text is passed directly to a cohere's command language model, which generates natural, context-aware conversational responses. These responses can then be translated into multiple Indian regional languages — including Hindi, Tamil, Telugu, Kannada, Malayalam, Marathi, Bengali, and others — and delivered as both text and synthesized speech. The complete system is deployed using a FastAPI backend and a React frontend, with all components designed for low latency and real-world usability.***

Keywords: Indian Sign Language, Sign Language Recognition, Multilingual Chatbot, MobileNetV2, Transfer Learning, Cohere AI, FastAPI, React, Text-to-Speech.

I. INTRODUCTION

Communication is a fundamental human right, yet millions of deaf and hard-of-hearing individuals worldwide are routinely excluded from digital systems that assume text or voice as the only valid input. For ISL users in India, this exclusion is particularly acute. Modern AI-powered chatbots and conversational agents — however capable — are designed around typing or speaking, making them largely inaccessible to those who communicate through hand gestures.

Recent progress in deep learning and computer vision has created an opening to close this gap. Convolutional Neural Networks (CNNs) trained with transfer learning can now recognize complex visual patterns efficiently on consumer hardware. Lightweight pretrained models such as MobileNetV2 make real-time inference feasible in web-deployed backends. Meanwhile, large language models like command can generate fluent, context-sensitive responses from short or fragmented input — which is exactly what gesture-derived text often looks like. This project builds on these capabilities to create a system with one clear goal: a person who knows ISL should be able to open a browser, point a webcam at their hand, sign a message, and receive an intelligent response — without needing to type a single character. The system extends this further by offering translation of responses into major Indian regional languages with voice output, ensuring that family members and caregivers who may not speak English can also participate in the interaction. The remainder of this paper describes the system architecture, the model training pipeline, the frontend and backend implementation, the language support features, and an honest assessment of the current limitations alongside a roadmap for future improvement.

II. RELATED WORK

A. CNN-Based Static Gesture Recognition

A substantial body of prior work has applied convolutional neural networks to the problem of static hand gesture classification. These systems extract spatial features from individual frames and classify them into discrete gesture categories. Early approaches used hand-crafted features such as histogram of oriented gradients or skin-color segmentation. More recent work has moved to end-to-end CNN training, often leveraging ImageNet-pretrained backbones through transfer learning. Models such as VGG-16, ResNet-50, and MobileNetV2 have all been applied to sign language recognition, with MobileNetV2 gaining traction in deployment scenarios due to its favourable accuracy-to-latency tradeoff.

Most published ISL recognition systems focus on a limited vocabulary — typically A–Z alphabets and digits 0–9 — and report accuracy on held-out test splits of their own datasets. Comparisons across systems are difficult because dataset size, signer diversity, background conditions, and evaluation methodology vary widely.

B. Limitations of Existing ISL Systems

Despite steady progress in recognition accuracy, existing ISL systems share a common set of limitations that prevent real-world deployment. First, output is almost universally static English text. There is no mechanism for the recognized text to flow into a conversational system that can respond meaningfully. Second, systems typically require clean, controlled input conditions — plain backgrounds, fixed camera angles, and consistent lighting — that do not reflect real-world use. Third, and most critically, no existing publicly available system integrates gesture recognition with a large language model backend capable of natural dialogue.

C. Conversational AI Integration

The integration of speech and text recognition with large language models for conversational interfaces has advanced rapidly, driven by systems like OpenAI's GPT series and Google's Gemini family. These models can handle ambiguous or incomplete input gracefully, inferring intent from context. For ISL users, whose signed input may produce short, fragmented, or phonetically unusual text sequences, this tolerance for imperfect input is especially valuable. However, no prior work has directly connected ISL gesture recognition to an LLM conversational backend in a deployed web system.

D. Multilingual and Accessibility Considerations

India's linguistic diversity means that a system serving ISL users must think beyond English output. Prior accessibility-focused NLP work has demonstrated the feasibility of real-time translation into Indian regional languages using services such as Google Translate. Text-to-speech systems capable of producing intelligible output in Hindi, Tamil, Telugu, Kannada, Malayalam, and other languages are available as both offline libraries (pyttsx3) and cloud APIs. Combining these capabilities within a single user interface represents a meaningful step toward genuine inclusivity.

III. PROBLEM STATEMENT

The core problem this work addresses is straightforward: there is no accessible, freely usable, web-based system through which a deaf or hard-of-hearing individual can communicate in Indian Sign Language and receive an intelligent AI response in their preferred Indian language.

Breaking this down into specific technical requirements, the system must:

1) Recognize ISL hand gestures from a standard webcam in real time without specialized hardware
2) Convert recognized gestures into text that accurately reflects the user's intended message
3) Pass that text to a conversational AI that can respond helpfully, even when the input is abbreviated or letter-by-letter
4) Translate the AI response into the user's preferred Indian regional language
5) Deliver output as both readable text and audible speech
6) Run in a standard web browser without requiring software installation

IV. SYSTEM ARCHITECTURE

The system follows a standard client-server architecture with a clear separation of concerns. Figure 1 illustrates the high-level data flow.

A. Frontend — React Application

The user interface is a React single-page application served locally via Vite. The frontend is responsible for webcam access through the browser's MediaDevices API, frame capture and encoding, real-time display of recognition results, the text builder interface, and the chat display.

Frames captured from the webcam are cropped to a 70% center square — the region where the user's hand naturally appears — and resized to 224×224 pixels before being base64-encoded and sent to the backend prediction endpoint. This preprocessing step is critical: it ensures that the image received by the model matches the spatial distribution of the training data, which consisted of hand-only images rather than full-frame captures.

The text builder accumulates recognized letters and allows the user to type additional characters freely, effectively blending sign recognition with keyboard input. Once the user is satisfied with their message, a single button sends it to the Cohere AI backend and displays the response in a chat interface. Every message — both user input and AI response — carries inline translate and speak buttons that operate on that specific message without affecting the rest of the conversation.

B. Backend — FastAPI Service

The backend is a FastAPI application running under Uvicorn. It exposes the following endpoints:

1) /predict/base64 — accepts a base64-encoded JPEG frame and returns the predicted gesture label, confidence score, top-3 candidates, and inference latency
2) /chat — accepts the user's message and conversation history, forwards them to Cohere, and returns the AI response
3) /translate — accepts a text string and target language code, and returns the translated text via deep-translator
4) /tts — accepts a text string and returns a synthesized WAV audio file via pyttsx3
5) /health — returns server status, model load confirmation, and session statistics

At server startup, the TensorFlow model is loaded once from disk using keras.models.load_model() with compile=False to avoid the verbose architecture dump and reduce startup time. A single warm-up inference pass is made immediately after loading to ensure that the first real prediction is not penalized by JIT compilation overhead. All TensorFlow logging is suppressed at the environment level to keep server output clean.

C. Model — MobileNetV2 with Transfer Learning

The gesture recognition model uses MobileNetV2 pretrained on ImageNet as a frozen feature extractor, with a custom classification head consisting of a GlobalAveragePooling2D layer, a Dense layer with 256 units and ReLU activation, Batch Normalization, Dropout at 0.4, and a final softmax Dense layer over the number of gesture classes. An explicit linear Activation layer with dtype='float32' is placed after the softmax output to prevent the TopKCategoricalAccuracy metric from receiving float16 tensors on GPU — a common source of runtime errors when training on Colab's T4 GPU.

Training proceeds in two phases. In Phase 1, the MobileNetV2 backbone is frozen and only the classification head is trained for up to 20 epochs using Adam at a learning rate of 1e-3. In Phase 2, the top layers of the backbone (from layer 100 onward) are unfrozen and the entire network is fine-tuned for up to 15 additional epochs at a much lower learning rate of 1e-5 to avoid destabilizing the pretrained weights. ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau callbacks are active throughout both phases.

The model is saved in Keras 3 native .keras format, which is required for compatibility with the version of TensorFlow shipped in current Colab environments (Python 3.12). The older SavedModel directory format is no longer supported for loading in Keras 3.

D. Language AI — Cohere AI

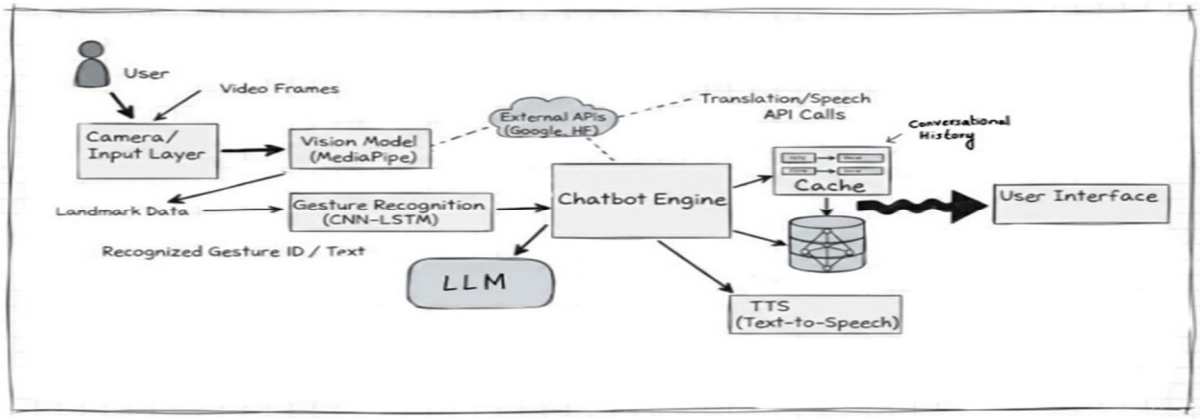
For conversational responses, the system uses Cohere’s command model. The older google-generativeai library is deprecated and no longer reliable for production use. The chat endpoint constructs a full contents list from the conversation history on every request, allowing Cohere to maintain context across multiple turns. A system instruction informs the model that input comes from ISL gesture recognition and may be abbreviated or letter-by-letter, prompting it to infer intent gracefully and respond concisely.

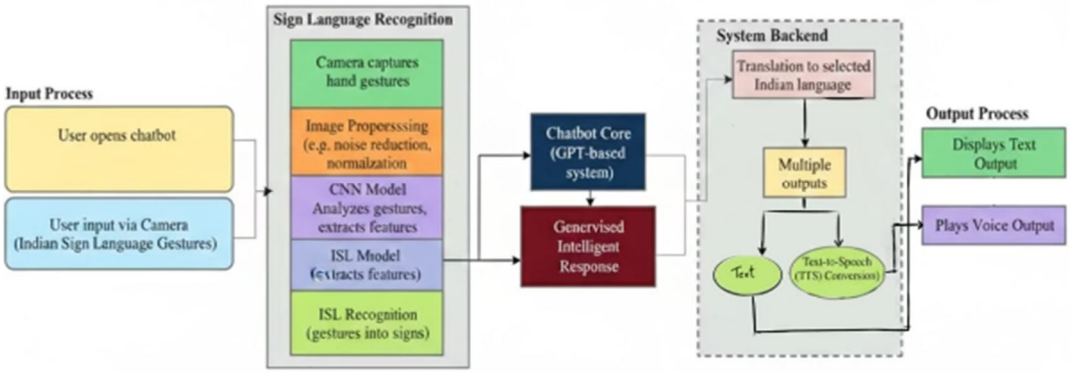
V. WORKFLOW

The first step in this project's workflow is to make a sign recognition system that collects and processes gesture videos to figure out how hands move. A hand-tracking method is used to find important points on the hand. These points are then put together in sequences that show both the shape of the hand and how it moves over time. This helps the system learn and recognize gestures well. When the model is stable, it is added to a backend service that does all the processing and makes predictions in real time.

The user interface is web-based and uses the camera to record live video and keep track of hand movements all the time. The system analyses the extracted information and sends back the recognized gesture. A conversational module then uses this text to create meaningful responses, which makes the interaction more natural and helpful. To make sure everyone can use it, the response can be translated into several Indian languages, making it easier for people from different backgrounds to talk to each other.

All interactions, including recognized gestures and responses that were made, are saved in a database for later use and to make the system better. The final output, along with its translation, is shown to the user right away on the interface. This makes for a smooth and seamless communication experience from gesture input to meaningful response.





VI. DATASET

The model was trained on a static gesture dataset organized into class folders, one per gesture label. The dataset covers the 26 English alphabets (A–Z) and digits 0–9, giving 36 classes in total. Each class folder contains hand gesture images captured from a webcam perspective against varied backgrounds.

The dataset is loaded using a custom tf.data pipeline that reads image paths and integer labels, applies MobileNetV2-compatible preprocessing (pixel scaling to [-1, 1]), and applies lightweight augmentations during training including random horizontal flip, random brightness shift (±15%), and random contrast adjustment (±15%). Importantly, caching is disabled in the pipeline to prevent out-of-memory errors on Colab's free tier — the trade-off being slightly slower epoch times in exchange for stable training without process termination.

The data is split 80/20 into training and validation sets using stratified sampling to preserve class balance. No held-out test set is used separately; validation accuracy after fine-tuning is used as the primary performance metric.

Parameter	Value	Notes
Input size	224 × 224 × 3	RGB, MobileNetV2 standard
Classes	36	A–Z + 0–9
Train/Val split	80% / 20%	Stratified
Batch size	16	Reduced for Colab RAM
Augmentation	Flip, Brightness, Contrast	Training only
Normalization	MobileNetV2 preprocess	Scales to [-1, 1]

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

VII. IMPLEMENTATION DETAILS

A. Training Environment

All model training was performed on Google Colab using a T4 GPU. TensorFlow mixed precision was explicitly set to float32 throughout training — not mixed_float16 — because the TopKCategoricalAccuracy metric passes predictions to tf.math.in_top_k, which requires float32 input and raises a TypeError when it receives float16 tensors. The model output layer includes an explicit Activation('linear', dtype='float32') layer to cast the softmax output back to float32 before it reaches any metric computation.

The training pipeline is implemented as a series of modular Python files: dataset_loader.py handles data discovery and tf.data pipeline construction; model_builder.py handles MobileNetV2 instantiation and head construction; utils.py provides callbacks, save utilities, and inference helpers; and train.py orchestrates the two-phase training loop. The entire pipeline is parameterized via command-line arguments so that backbone selection, batch size, epoch counts, dropout rate, and confidence threshold can be adjusted without modifying source code.

B. Backend Environment

The FastAPI backend runs under Python 3.11.7 in a virtual environment. The critical dependency constraint is numpy==1.26.4, which must be installed before tensorflow==2.15.0. NumPy 2.x is incompatible with TensorFlow 2.15.0's compiled C extensions on Windows, causing an AttributeError: _ARRAY_API not found at import time. Pinning NumPy to 1.26.x resolves this entirely.

The .env file used to supply environment variables must not wrap values in quotes. Python's os.getenv() reads values literally, so COHERE_API_KEY='abc' passes the string 'abc' including the quotes, which renders the API key invalid. The correct format is COHERE_API_KEY=abc with no surrounding punctuation. The python-dotenv library is used to load the .env file at startup via load_dotenv().

C. Frontend Environment

The React frontend is built with Vite and targets modern browsers. Webcam access requires HTTPS or localhost — the application works without a certificate in local development but would need TLS termination for any public deployment. The inference loop runs at 600ms intervals using setInterval; the captured frame is cropped to the center 70% of the video feed before encoding, which significantly improves recognition accuracy by excluding the background and the user's face from the model input.

VIII. RESULTS AND OBSERVATIONS

The model training consistently reaches validation accuracy near 1.0 on the static gesture dataset after Phase 1 training alone for many classes. This reflects both the relatively constrained nature of the classification task (36 distinct classes, static single-hand gestures) and the power of MobileNetV2's pretrained feature representations.

In practice, recognition quality under live webcam conditions is more nuanced. Several factors affect real-world performance:

1) Lighting conditions — the model performs significantly better under uniform, adequate lighting. Shadows on the hand or overexposure both reduce accuracy.

2) Background complexity — a plain, uniform background behind the hand reduces false positives. Cluttered backgrounds cause the model to occasionally misclassify.

3) Hand positioning — the user must place their hand within the dashed ROI box displayed on screen. The frontend crops to the center 70% of the frame before sending to the model; gestures made at the frame edges will not be captured correctly.

4) Gesture hold time — the recognition loop requires the same gesture to appear in two consecutive inference cycles (1.2 seconds) before appending the letter. This reduces noise-driven false appends but means users must hold each sign briefly.

The Cohere AI integration produces natural, contextually appropriate responses to signed input. Because gesture-derived text is often a sequence of individual letters rather than complete words, the system prompt explicitly instructs the model to infer intended meaning from letter sequences and respond naturally.

In testing, Gemini handles inputs like 'HELLO' (individual letters H-E-L-L-O) correctly, treating them as the word 'hello' rather than a random character sequence.

Translation via deep-translator works reliably for all supported Indian languages. TTS via pyttsx3 works for English output; voice quality for Indian language text depends on the system TTS voices installed on the host machine and varies by platform.

IX. FUTURE WORK

The current system handles static, single-hand gestures for individual letters and digits. Several directions would meaningfully extend its capability:

1) Dynamic gesture recognition — incorporating temporal modeling through LSTM or transformer-based sequence models would allow the system to recognize multi-frame gestures representing complete words or phrases rather than individual letters.

2) MediaPipe hand landmark integration — replacing raw image input with MediaPipe's 21-point hand landmark coordinates would make the recognition model invariant to skin tone, lighting variation, and background complexity, all of which currently limit real-world accuracy.

3) Expanded ISL vocabulary — the dataset covers only A–Z and 0–9. Adding common ISL words and phrases (greetings, health terms, emergency phrases) would make the system genuinely useful for everyday communication.

4) Mobile deployment — wrapping the React frontend as a progressive web app with on-device TensorFlow.js inference would remove the server dependency and allow the system to run entirely on a smartphone.

5) User feedback loop — logging confidence scores and user corrections would create a dataset for continual model improvement through active learning.

6) Bidirectional communication — adding an avatar or video output that produces ISL signs from text input would allow hearing users to communicate back to ISL users within the same interface.

X. CONCLUSION

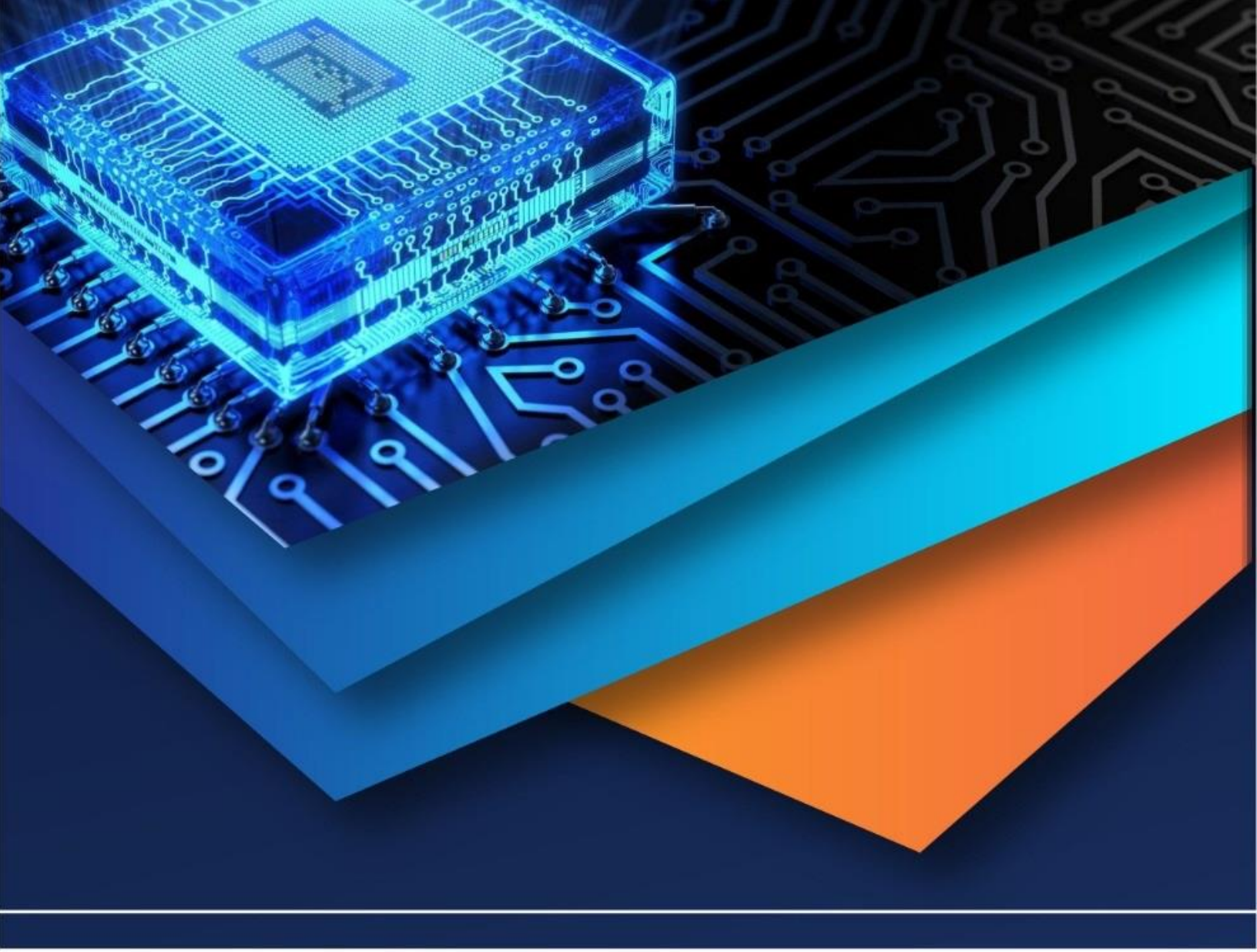
This paper has described the design, implementation, and deployment of a real-time Indian Sign Language chatbot that connects gesture recognition directly to a large language model conversational backend. The system is built entirely from open and freely available components — MobileNetV2 for visual recognition, Cohere AI for conversational AI, deep-translator for regional language translation, and pyttsx3 for voice output — and runs in a standard web browser without specialized hardware.

A significant portion of this work involved resolving practical engineering challenges that arise when deploying machine learning systems in real production environments: NumPy version conflicts, Keras 3 format changes, TensorFlow GPU precision issues, API SDK deprecations, and the gap between validation accuracy and real-world recognition performance. These challenges are rarely discussed in research papers but represent the majority of the work required to move from a trained model to a usable system.

The result is a functional prototype that demonstrates the feasibility of the core concept: a deaf or hard-of-hearing individual can open a browser, sign a message using ISL gestures, and receive an intelligent AI response in their preferred Indian language with voice output. Extending this prototype into a robust, widely deployable application — with dynamic gesture support, a broader ISL vocabulary, and mobile access — represents a clear and achievable path toward genuinely inclusive AI communication.

REFERENCES

[1] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861, 2017.
[2] M. Sandler et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Proc. IEEE CVPR, 2018.
[3] Google LLC, "MediaPipe Hands: On-device Real-time Hand Tracking," arXiv:2006.10214, 2020.
[4] S. Gupta, R. Sharma, and A. Kumar, "Indian Sign Language Detection for Real-Time Translation Using CNN with MediaPipe," arXiv preprint arXiv:2507.20414,Jul. 2025. [Online]. Available: https://arxiv.org/abs/2507.20414 [web:65]
[5] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[6] TensorFlow Development Team, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: https://www.tensorflow.org
[7] S. Ramachandran, "Indian Sign Language Dataset," Kaggle, 2021. [Online]. Available: https://www.kaggle.com
[8] FastAPI Documentation, Sebastián Ramírez, 2024. [Online]. Available: https://fastapi.tiangolo.com
[9] React Documentation, Meta Open Source, 2024. [Online]. Available: https://react.dev
[10] deep-translator Library, Nidhal Baccouri, 2023. [Online]. Available: https://github.com/nidhaloff/deep-translator
[11] R. Kadwade, A. Tangade, and N. Pakhare, "Indian Sign Language Recognition System," International Journal of Engineering Research&Technology,vol.12,no.5,pp.789-798,May2023. [Online]. Available: https://www.ijert.org/indian-sign-language-recognition-system [web:63]
[12] S. Kanade et al., "Indian Sign Language recognition system using SURF with polynomial classifier," Software Impacts, vol.12, pp.100.112,Apr.2022.[Online].Available:https://www.sciencedirect.com/science/article/pii/S2590005622000121 [web:60]
[13] V. Narayanan et al., "A Comprehensive Approach to Indian Sign Language Recognition Using Sequential LSTM with MediaPipe Holistic," EAI Endorsed Transactions on AI and Robotics, vol.2,no.1,pp.1-12, May 2025. [Online]. Available: https://publications.eai.eu/index.php/airo/article/view/8693 [web:69]
[14] IIT Kanpur CS365 Team, "Indian Sign Language Character Recognition," CS365 Project Report, Dept. of Computer Science,Indian Institute of Technology Kanpur,2015.[Online]. Available:https://cse.iitk.ac.in/users/cs365/2015/_submissions/vinsam/report.pdf [web:66]



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY