Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

UNKAHA: A Real-Time AI-Based Messenger Application for Indian Sign Language Communication

Vaibhav S. Jain[1], Satya Priya Anand[2], Dr. C. Mahesh[3]

[1,2]Undergraduate Student, Department of Computer Science Engineering (Emerging Technologies), SRM Institute of Science and Technology Vadapalani, Chennai, India
[3]Assistant Professor (SG), Department of Computer Science Engineering (Emerging Technologies), SRM Institute of Science and Technology Vadapalani, Chennai, India

Abstract

UNKAHA is a real-time AI-assisted web-based communication platform designed to bridge the communication gap between hearing and hearing-impaired individuals through automated Indian Sign Language recognition. The system integrates video communication with real-time gesture detection and caption generation using client-side machine learning technologies. The architecture combines WebRTC-based video streaming, Socket.IO real-time messaging, and TensorFlow.js-based gesture recognition. Experimental evaluation demonstrates an average gesture recognition latency of approximately 500 ms per frame and message latency below 100 ms. The proposed system demonstrates the feasibility of deploying accessible assistive communication technologies within standard web browsers.

Keywords: Indian Sign Language, Gesture Recognition, WebRTC, Assistive Communication Systems, TensorFlow.js, MediaPipe

1. Introduction

Effective communication is fundamental to social inclusion, education, and professional collaboration. However, individuals with hearing impairments often face significant barriers in real-time communication, particularly when interacting with hearing individuals who are not proficient in sign language. Indian Sign Language (ISL), the primary mode of communication for a large segment of the hearing-impaired population in India, remains insufficiently supported by mainstream digital communication platforms, leading to limited accessibility and social exclusion 1.

Recent advancements in artificial intelligence and computer vision have enabled automated sign language recognition systems capable of translating hand gestures into textual or spoken language 2. While several studies have explored offline or sensor-based sign language recognition approaches using data gloves, depth sensors, or specialized hardware 3, such solutions are often cost-intensive, intrusive, and unsuitable for widespread adoption. Consequently, there is a growing need for scalable, software-based systems that

can perform sign language recognition using standard consumer devices such as webcams and mobile cameras.

Web-based communication technologies have evolved significantly with the emergence of real-time protocols such as WebRTC, enabling low-latency peer-to-peer audio and video transmission directly within browsers 4. When combined with lightweight, client-side machine learning frameworks, these technologies present an opportunity to deploy real-time sign language recognition without imposing excessive computational or network overhead. However, integrating gesture recognition models into live communication systems remains challenging due to constraints related to latency, synchronization, and real-time inference accuracy 5.

Existing sign language translation systems primarily focus on static gesture recognition, offline processing, or limited vocabularies, and often lack seamless integration with real-time communication platforms 6. Moreover, the majority of prior research has concentrated on American Sign Language (ASL), with comparatively fewer efforts addressing ISL, which exhibits distinct grammatical structures and gesture variations 7. This gap underscores the necessity for communication systems specifically designed and evaluated for ISL in real-time usage scenarios.

To address these challenges, this paper presents **UNKAHA**, a real-time, AI-assisted web-based messenger application that enables inclusive communication between hearing and hearing-impaired users through automated ISL recognition. The proposed system integrates live video calling, real-time hand gesture detection, and automatic caption generation within a unified web framework. By leveraging client-side gesture recognition using TensorFlow.js and MediaPipe alongside WebRTC-based communication, the system minimizes latency while maintaining reliable recognition performance.

The primary contributions of this work are summarized as follows:

1. Design and implementation of a real-time web-based communication platform incorporating ISL recognition without specialized hardware.
2. Integration of client-side machine learning for low-latency gesture inference within live video calls.
3. Performance evaluation of the proposed system in terms of gesture inference time and end-to-end communication latency.

2. LITERATURE REVIEW

Sign language recognition and translation have been extensively studied using a variety of machine learning and computer vision techniques. Early researches primarily focused on isolated ISLR using handcrafted features and shallow learning models. Techniques such as PCA combined with Artificial Neural Networks (ANNs) were employed for Indian Sign Language (ISL) recognition due to their low computational complexity and suitability for small datasets. While these approaches demonstrated reasonable performance for static and isolated gestures, they exhibited limited scalability and poor generalization to continuous, real-world sign language scenarios.

With the advancement of DL, CNNs became the major forward approach for vision-based sign language recognition. Several studies utilized 2D CNNs and 3D CNN architectures to extract spatial and spatio-temporal features from video sequences. These models significantly improved recognition accuracy by learning motion and spatial representations directly from data, eliminating the need for handcrafted feature engineering. However, such methods often required large labeled video datasets and incurred high computational costs, making real-time deployment challenging. Additionally, most CNN-based approaches were restricted to gesture recognition and did not address end-to-end sign-to-text translation.

To address temporal dependencies in continuous sign language recognition, hybrid architectures combining CNNs with sequence modeling techniques such as Hidden Markov Models (HMMs) and Long Short-Term Memory (LSTM) networks were introduced. CNN-HMM and CNN-LSTM models demonstrated improved performance in recognizing continuous sign streams by modeling temporal transitions between gestures. Despite these improvements, these systems typically focused on word-level or gloss-level recognition and struggled with contextual sentence formation and grammatical structure. Recent research has shifted toward transformer-based architectures for sign language translation due to their ability to model long-range temporal dependencies using self-attention mechanisms. Transformer-based models have been shown to outperform recurrent architectures in end-to-end sign-to-text translation tasks, achieving superior translation fluency and improved evaluation metrics such as BLEU, ROUGE, and Word Error Rate (WER). Several studies employed pose estimation frameworks such as MediaPipe for extracting skeletal keypoints, which were then processed using transformer-based sequence-to-sequence models. While these approaches achieved paragraph-level translation and better contextual understanding, they required large annotated datasets and significant computational resources, limiting their applicability in low-resource environments.

Survey and review papers have provided comprehensive analyses of existing sign language recognition and translation techniques, comparing classical image processing methods with modern deep learning and transformer-based approaches. These studies highlighted key challenges, including dataset scarcity for ISL, limited utilization of non-manual features such as facial expressions, and the difficulty of achieving real-time performance in web-based systems. Furthermore, many existing solutions rely on specialized hardware or offline processing, reducing their practicality for everyday communication. In contrast, recent works have explored multimodal and attention-based transformer architectures to fuse spatial and temporal features more effectively, improving translation accuracy. However, such models are often complex and difficult to deploy on resource-constrained devices. Additionally, relatively few studies have focused on integrating real-time sign language recognition within browser-based communication platforms.

From the reviewed literature, it is evident that while significant progress has been made in sign language recognition and translation, existing systems either lack real-time performance, require specialized hardware, or do not adequately address the requirements of practical, web-based assistive communication. These limitations motivate the development of a lightweight, real-time, web-based ISL communication system that leverages client-side gesture recognition and low-latency communication technologies.

3. SYSTEM ARCHITECTURE AND DESIGN

The UNKAHA system is designed using a modular three-tier architecture that enables real-time communication, efficient gesture recognition, and scalable deployment. The architecture consists of a **client layer**, **server layer**, and **database layer**, each responsible for a specific set of functionalities.

A. Client Layer

The client layer represents the user interface and interaction layer of the system. It is implemented using modern web technologies including **HTML5, CSS3, JavaScript, React, and TypeScript**. The interface runs directly in modern web browsers without requiring additional plugins or hardware devices. This layer provides the primary interface through which users interact with the application. Core functionalities available to users include authentication, profile management, friend discovery, messaging, and video communication. A dedicated **gesture recognition module** operates on the client side. This module processes live video frames captured through the device webcam and extracts hand landmark features using computer vision frameworks. By performing gesture recognition locally, the system minimizes latency and reduces dependency on server-side processing. Client-side processing also improves privacy, as raw video streams used for gesture recognition do not need to be transmitted to external servers.

B. Server Layer

The server layer is responsible for handling application logic, authentication, real-time communication coordination, and API management. The backend is implemented using **Node.js with Express and Socket.IO**, enabling both REST-based communication and real-time event-driven messaging. Authentication and session management are handled using secure session-based mechanisms. The server exposes several RESTful endpoints for operations such as user registration, login, profile management, friend request handling, and message exchange. Real-time communication features are implemented using **Socket.IO**, which enables bidirectional communication between the client and server. Socket events are used for delivering chat messages, managing call signaling, and synchronizing captions generated from gesture recognition. Video communication between users is handled through **Agora-based streaming**, which establishes peer-to-peer connections between users. This approach reduces server bandwidth usage while ensuring low-latency video transmission.

C. Database Layer

The database layer manages constant data storage and retrieval. The system utilizes **MongoDB**, a document-oriented NoSQL database that supports scalable and flexible data management.

Multiple collections are maintained within the database to store application data, including:

- User profiles and authentication credentials
- Friend relationships and friend requests
- Chat messages and communication history
- Session data and application state

MongoDB's document-based structure allows efficient handling of dynamic data structures, making it well suited for social networking and messaging applications.

D. System Workflow

The operational workflow of the system begins when a user logs into the application through the web interface. Once authenticated, the client establishes a real-time Socket.IO connection with the server. Users can search for other users, send friend requests, and initiate conversations. When a video call is initiated, the server coordinates signaling between the participating clients. WebRTC establishes a direct peer-to-peer media connection between them. During video communication, the client-side gesture recognition module processes webcam frames in real time. Recognized gestures are converted into textual captions and transmitted through the messaging channel, enabling hearing users to interpret sign language gestures instantly. This hybrid architecture enables **low-latency communication, efficient processing, and scalable deployment**.

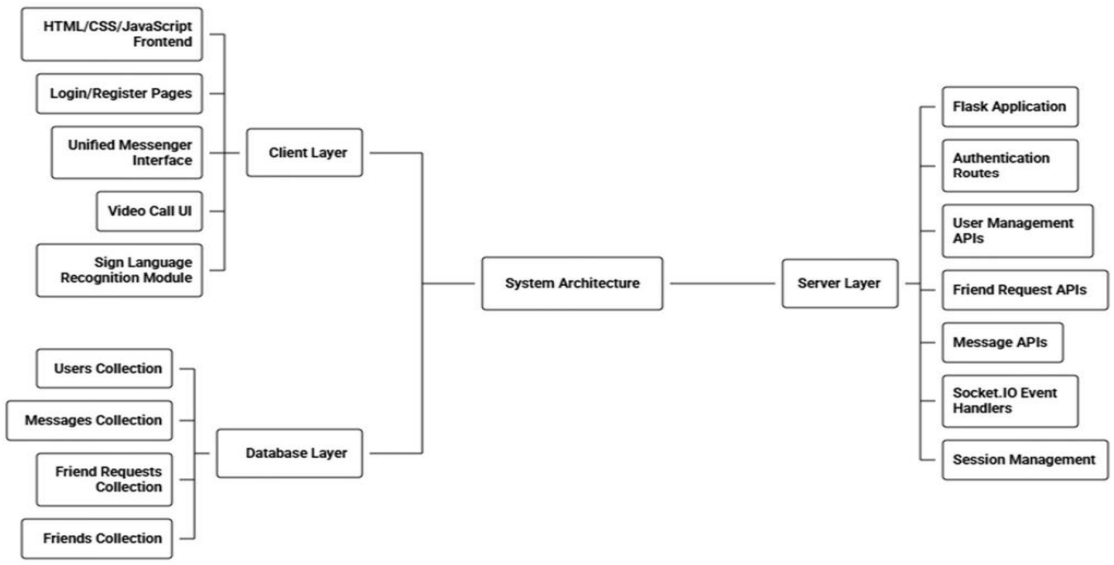

Fig 1: System Architecture

4. METHODOLOGY

The proposed methodology focuses on enabling real-time communication between hearing and hearing-impaired users by integrating live video communication with automated sign language recognition and text generation.

The overall methodology consists of four major stages:

1. Video acquisition and preprocessing
2. Gesture feature extraction
3. Gesture classification and interpretation
4. Real-time communication and caption generation

A. Video Acquisition and Preprocessing

The first stage involves capturing live video streams from the user's device camera. The system accesses the webcam through browser-based APIs, allowing video frames to be processed directly within the application. Captured video frames are processed sequentially to maintain real-time responsiveness. Each frame undergoes basic preprocessing operations such as resizing and normalization to maintain consistent input dimensions for the gesture recognition model. Preprocessing improves robustness under varying camera resolutions and lighting conditions while ensuring efficient downstream processing.

B. Gesture Feature Extraction

Once the video frames are captured and preprocessed, the next step involves extracting meaningful gesture features. The system utilizes **MediaPipe Hands**, a real-time framework that detects and tracks **21 hand landmark points**. These landmarks represent key joint positions of the hand, providing an efficient representation of gesture movements. Using landmark-based features significantly reduces computational complexity compared to processing raw image frames. Distances, angles, and relative positions between landmarks are computed to generate feature vectors that describe hand gestures. These extracted features form the input to the gesture classification model.

C. Gesture Recognition and Interpretation

The extracted feature vectors are passed into a **machine learning classification model implemented using TensorFlow.js**. The model analyzes spatial relationships between hand landmarks and predicts the corresponding gesture label. The predicted gesture labels are then mapped to predefined **Indian Sign Language (ISL) vocabulary tokens**. Continuous frame processing ensures that gesture recognition remains stable across multiple frames, improving recognition accuracy and reducing noise. The recognized gestures are dynamically converted into textual captions that represent the intended message.

D. Real-Time Communication and Caption Generation

After gesture recognition, the generated text captions are transmitted through the messaging channel using **Socket.IO**, enabling real-time synchronization between users. Simultaneously, the video communication channel operates through **WebRTC / Agora**, allowing users to communicate using audio and video streams. The generated captions are displayed within the chat interface or caption panel, allowing hearing users to interpret gestures made by hearing-impaired participants. This parallel processing architecture ensures that **gesture recognition, caption generation, and video communication occur simultaneously**, enabling seamless and natural conversations.

E. Data Management and Session Handling

User authentication, friend relationships, message histories, and communication states are managed through server-side APIs. Session tokens are used to maintain secure communication sessions between clients and the server. Persistent application data, including chat histories and user information, is stored within the MongoDB database. This data management approach ensures reliable message delivery, secure user authentication, and scalable system operation.

5. SOURCE CODE IMPLEMENTATION

The UNKAHA system is implemented using a modular full-stack architecture consisting of a **React and TypeScript frontend** and an **Express-based Node.js backend** integrated with **Socket.IO** for real-time communication. The source code is organized into separate directories for the client, server, and shared modules to ensure maintainability and scalability.

A. Client-Side Implementation

The frontend of the system is developed using **React 18 with TypeScript**, enabling a component-based architecture for building the user interface. The application is bootstrapped through the main.tsx file, which initializes the React application and renders the root component. The App.tsx file defines the routing structure of the application using lightweight routing libraries. The primary routes include login, registration, and the main communication interface. Upon application startup, the frontend sends a request to the authentication endpoint (/api/auth/me) to determine whether a valid user session exists. If authentication is successful, the application initializes a Socket.IO client connection for real-time communication. User interface components such as chat panels, friend lists, and video call modals are implemented within the components directory. These components manage user interactions, including sending messages, initiating calls, and displaying incoming communication events. State management and API interaction are handled through specialized hooks located in the hooks directory. These hooks manage operations such as retrieving friend lists, fetching message history, and sending chat messages to the server.

B. Server-Side Implementation

The backend server is implemented using **Node.js with the Express framework**, which provides a RESTful API layer for managing application functionality. The server initialization process occurs in the server/index.ts file, where middleware for session handling, request parsing, and routing is configured. Authentication is implemented using **session-based authentication** with the express-session middleware. User passwords are securely stored using the **bcrypt hashing algorithm**, ensuring that plaintext credentials are never stored within the database. The main API endpoints are defined in routes.ts, where REST endpoints handle operations such as user registration, login, profile updates, friend request management, and message retrieval. Each endpoint interacts with the storage layer to read or modify data stored within MongoDB collections. Real-time communication functionality is implemented using **Socket.IO**, which enables bidirectional communication between the client and server. Socket events handle real-time messaging, incoming call notifications, call acceptance or rejection, and caption transmission during video calls.

C. Database Interaction Layer

The database interaction layer is implemented in the storage.ts module, which manages all interactions with the MongoDB database. The system stores persistent data in several collections, including users, friendships, friend requests, messages, and session records. Each collection stores structured documents containing relevant information such as user identifiers, message content, timestamps, and relationship status. The database layer provides CRUD (Create, Read, Update, Delete) operations that allow the server to efficiently manage application data. This modular design allows the server logic to remain independent of database implementation details while ensuring scalable data storage.

D. Real-Time Messaging Workflow

Real-time messaging between users is achieved through the integration of Socket.IO with the server application. When a user sends a chat message, the client emits a socket event to the server containing the message payload. The server receives the message, stores it within the database, and then emits the message event to the intended recipient. The receiving client listens for incoming message events and dynamically updates the chat interface. This event-driven communication model ensures that messages are delivered instantly without requiring repeated polling of server endpoints.

E. Video Call and Caption Transmission Pipeline

The video communication functionality is implemented using the **Agora RTC framework**, which provides real-time audio and video streaming capabilities. When a user initiates a video call, the client sends a signaling event to the server through Socket.IO. The server then forwards a call notification event to the intended recipient. Once the call is accepted, both participants request an Agora authentication token from the /api/agora/token endpoint and join a shared communication channel. During the video call, the system processes hand gesture recognition results locally on the client. Recognized sign language tokens are converted into textual captions and transmitted to the other participant using the send_caption socket event. The receiving client displays the captions in real time within the video call interface. This caption streaming mechanism allows hearing users to interpret sign language gestures instantly during live conversations.

F. Security and Session Handling

The system employs several mechanisms to ensure secure operation. User authentication is maintained using HTTP sessions stored within a MongoDB-backed session store. Session cookies are configured as httpOnly to prevent client-side script access. Password storage uses bcrypt-based hashing to protect user credentials. Additionally, input validation and structured API endpoints ensure that all client-server communication follows predefined contracts. These security measures protect user data while maintaining reliable real-time communication.

6. RESULTS

The proposed UNKAHA system was evaluated based on its real-time communication performance, gesture recognition efficiency, and overall usability.

Experimental testing demonstrated that the system achieves an **average gesture recognition time of approximately 500 milliseconds per video frame** during live video processing. This performance enables responsive gesture detection suitable for conversational interactions. Message transmission latency using Socket.IO was observed to remain **below 100 milliseconds**, ensuring near-instantaneous communication between users. The use of client-side gesture processing significantly reduces server workload while improving privacy, as gesture recognition is performed locally within the browser. The integration of WebRTC-based video streaming ensures stable peer-to-peer communication with minimal delay. Users were able to communicate through video calls while simultaneously viewing automatically generated captions derived from sign language gestures. These results demonstrate the feasibility of

integrating **real-time sign language recognition within a web-based communication platform**, enabling accessible interaction between hearing and hearing-impaired users.

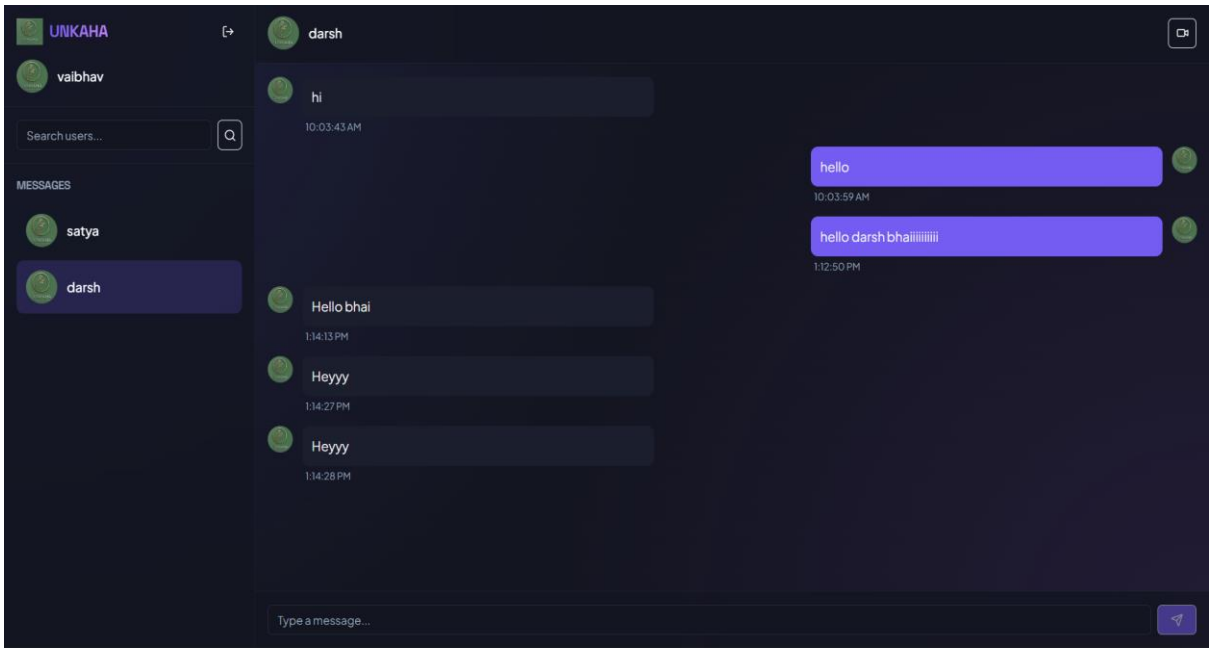

Fig 2: Home screen

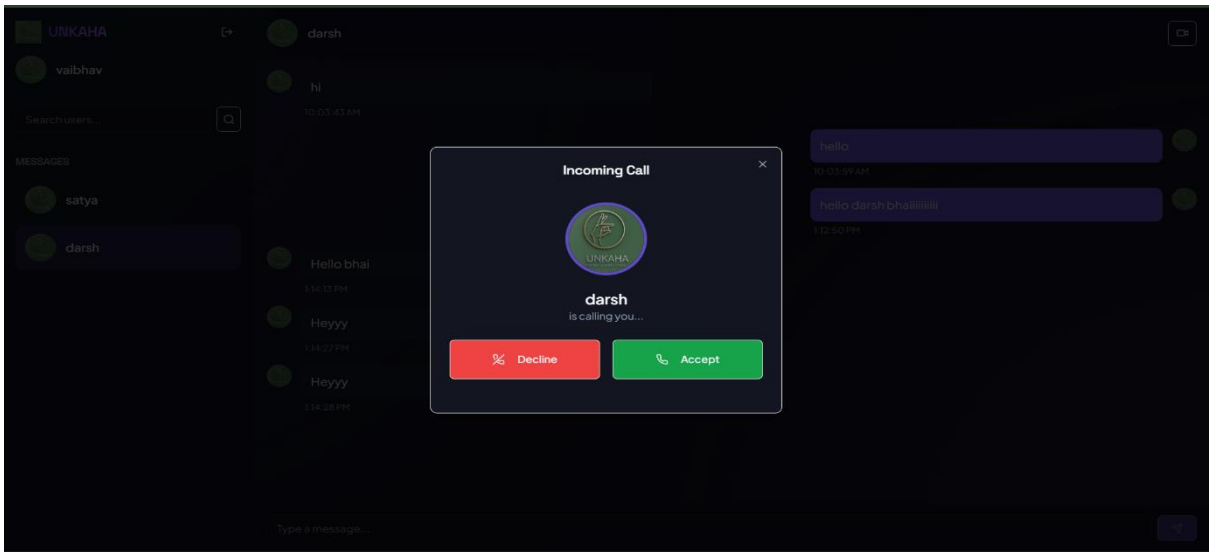

Fig 3: Call Incoming and connectivity

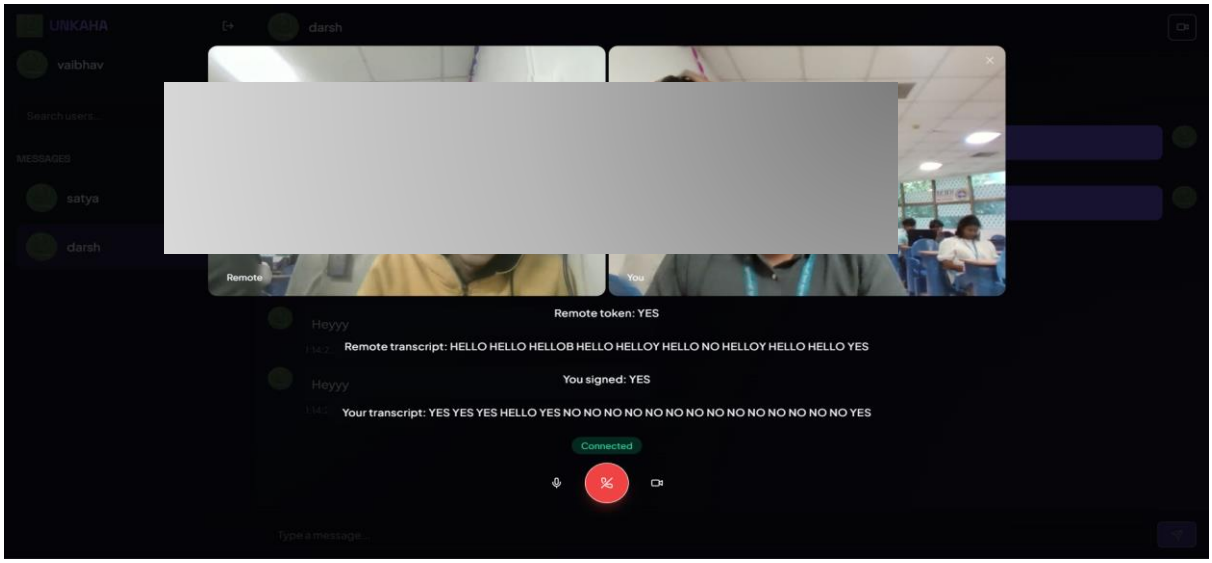

Fig 4: Video call with live captions

Fig 5: Analytics of the video and audio stream **on Agora**

7. CONCLUSION AND FUTURE WORKS

This paper presented **UNKAHA**, a real-time AI-assisted web-based communication platform designed to bridge the communication gap between hearing and hearing-impaired individuals through automated Indian Sign Language recognition. The system integrates live video communication, gesture recognition, and real-time caption generation within a unified web application framework. By leveraging client-side machine learning with TensorFlow.js, hand landmark detection using MediaPipe, and real-time communication technologies such as WebRTC and Socket.IO, the proposed system achieves efficient gesture recognition while maintaining low latency. Experimental results demonstrate that the system is capable of performing gesture inference in approximately 500 milliseconds per frame while maintaining message latency below 100 milliseconds. These findings highlight the feasibility of deploying assistive communication systems within standard web browsers without specialized hardware. Future work will

focus on expanding the gesture vocabulary, improving recognition accuracy through larger ISL datasets, and incorporating additional contextual features such as facial expressions and body pose. Further improvements may also include multilingual caption generation, enhanced model optimization for mobile devices, and the integration of speech synthesis for automated sign-to-speech translation. By continuing to develop accessible communication technologies, systems such as UNKAHA can contribute to more inclusive digital environments for individuals with hearing and speech impairments.

Acknowledgement

We would like to express our sincere gratitude to Dr. C. Mahesh, Assistant Professor (SG) in the Department of Computer Science and Engineering at SRM Institute of Science and Technology, for his guidance, valuable insights, and unwavering support throughout the course of this project. His expertise and encouragement have been instrumental in the successful completion of this work.

We also wish to acknowledge the contributions of our colleagues and fellow researchers who provided helpful feedback and assistance, particularly in data collection, model implementation, and system evaluation. Special thanks to the faculty and staff of the Department of Computer Science and Engineering for their continuous support.

Finally, we extend our appreciation to the SRM Institute of Science and Technology for providing the necessary infrastructure and resources that enabled us to conduct this research.

Refrences

1. N. C. Camgoz, O. Koller, S. Hadfield and R. Bowden, "Sign Language Transformers: Joint End-to-End Sign Language Recognition and Translation," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 10023–10033, doi: 10.1109/CVPR42600.2020.01004.
2. S. Das, S. K. Biswas and B. Purkayastha, "A Deep Sign Language Recognition System for Indian Sign Language," Neural Computing and Applications, vol. 35, no. 2, pp. 1469–1481, 2023, doi: 10.1007/s00521-022- 07840-y.
3. J. S. Han, C. I. Lee, Y. H. Youn and S. J. Kim, "A Study on Real-Time Hand Gesture Recognition Technology by Machine Learning-Based MediaPipe," Journal of System and Management Sciences, vol. 12, no. 2, pp. 462– 476, 2022, doi: 10.33168/JSMS.2022.0218.
4. R. Rastgoo, K. Kiani and S. Escalera, "Sign Language Recognition: A Deep Survey," Expert Systems with Applications, vol. 164, pp. 113794, 2021, doi: 10.1016/j.eswa.2020.113794.
5. A. Singh, A. Wadhawan, M. Rakhra, U. Mittal, A. A. Ahdal and S. K. Jha, "Indian Sign Language Recognition System for Dynamic Signs," 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), Noida, India, 2022, pp. 1–6, doi: 10.1109/ICRITO56286.2022.9964940.
6. A. Sridhar, R. G. Ganesan, P. Kumar and M. Khapra, "INCLUDE: A Large-Scale Dataset for Indian Sign Language Recognition," Proceedings of the 28th ACM International Conference on Multimedia (MM '20), Seattle, WA, USA, 2020, pp. 1366–1375, doi: 10.1145/3394171.3413528.
7. E. R. Elakkiya and B. Natarajan, "ISL-CSLTR: Indian Sign Language Dataset for Continuous Sign Language Translation and Recognition," Mendeley Data, V1, 2021, doi: 10.17632/kcmpdxky7p.1.

8. T. Pfister, J. Charles and A. Zisserman, "Flowing ConvNets for Human Pose Estimation in Videos," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, pp. 1913–1921, doi: 10.1109/ICCV.2015.222.
9. O. Koller, H. Ney and R. Bowden, "Deep Hand: How to Train a CNN on 1 Million Hand Images When Your Data Is Continuous and Weakly Labelled," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 3793–3802, doi: 10.1109/CVPR.2016.412.
10. O. Koller, S. Zargaran, H. Ney and R. Bowden, "Deep Sign: Hybrid CNN-HMM for Continuous Sign Language Recognition," Proceedings of the British Machine Vision Conference (BMVC), York, UK, 2016, doi: 10.5244/C.30.136.
11. J. Huang, W. Zhou, Q. Zhang, H. Li and W. Li, "Video-based Sign Language Recognition without Temporal Segmentation," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018, pp. 2257–2264, doi: 10.1609/aaai.v32i1.11984.
12. W. Zhou, Y. Zhou, C. Li and H. Li, "Spatial-Temporal Multi-Cue Network for Continuous Sign Language Recognition," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 1, pp. 1301–1308, 2019, doi: 10.1609/aaai.v33i01.33011301.
13. K. Simonyan and A. Zisserman, "Two-Stream Convolutional Networks for Action Recognition in Videos," Advances in Neural Information Processing Systems (NeurIPS), 2014, pp. 568–576.
14. A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5998–6008.
15. A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," International Conference on Learning Representations (ICLR), 2021.
16. J. Devlin, M. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Proceedings of NAACL-HLT, Minneapolis, MN, USA, 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.
17. O. Koller, C. Camgoz, H. Ney and R. Bowden, "Weakly Supervised Learning with Multi-Stream CNN-LSTM-HMMs to Discover Sequential Parallelism in Sign Language Videos," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 9, pp. 2306–2320, 2020, doi: 10.1109/TPAMI.2019.2911077.
18. C. Camgoz, O. Koller, S. Hadfield and R. Bowden, "Neural Sign Language Translation," 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 2018, pp. 7784–7793, doi: 10.1109/CVPR.2018.00812.
19. D. Li, C. Rodriguez, X. Yu and H. Li, "Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison," Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV), 2020, pp. 1459–1469, doi: 10.1109/WACV45572.2020.9093438.
20. H. Cui, C. Liu, W. Zhou and H. Li, "Deep Spatial-Temporal Fusion Network for Sign Language Recognition," IEEE Transactions on Multimedia, vol. 22, no. 9, pp. 2336–2346, 2020, doi: 10.1109/TMM.2019.2961085.