International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 13, Issue 2s, 2025

Allie – AI Voice and Sign Assistant

Sayyada Kaneez Sogra Fatima[1], Mr. Mohammed Rahmat Ali[2]

[1]B.E. Student, Department of Computer Science and Engineering, ISL Engineering College, Hyderabad, India.

[2]Assistant Professor, Department of Computer Science and Engineering, ISL Engineering College, Hyderabad, India.

ABSTRACT

Effective communication is a vital aspect of human interaction, yet individuals with speech and hearing disabilities frequently encounter barriers in conveying their thoughts or understanding others. Conventional solutions such as sign language interpreters or visual aids are not always accessible, which can result in social isolation and reduced opportunities for participation. To address this issue, Allie – AI Voice and Sign Assistant *has been developed as a smart Android-based mobile application that leverages artificial intelligence (AI) and mobile technology to enable seamless two-way communication.*

Allie offers two key features: (1) ***Voice/Text to Sign Translation****, where spoken or typed input is translated into sign language using animated GIFs for commonly used words or phrases (e.g., "Hello", "Thank you") and static alphabet images (A–Z) for letter-by-letter expression; and (2)* ***Sign to Text/Voice Recognition****, which utilizes MediaPipe's HandLandmarker to detect hand landmarks and classifies them through a custom-trained TensorFlow Lite (TFLite) model. This enables users to see the recognized gesture as text and perform context-based actions such as launching apps (e.g., WhatsApp, YouTube, Spotify) through specific signs.*

The app is designed for flexibility and usability in both online and offline environments, with sign media stored in the res/drawable directory and optionally in cloud storage using Firebase or Supabase. The custom gesture classification model is built using landmark data collected through a dedicated in-app feature that records hand movements into CSV format. The app's interface is built using Kotlin, incorporating Jetpack Compose and ViewBinding to ensure a responsive and accessible user experience.

This paper explores the app's architecture, development methodology, technological choices, and implementation details. It highlights how machine learning and computer vision contribute to building inclusive communication tools for differently-abled individuals. The target users include not only those with speech or hearing impairments but also their families, educators, and learners of sign language. Future enhancements include integration of text-to-speech functionality for gesture outputs, animated avatars for real-time sign rendering, and a built-in chatbot for conversational assistance. By combining AI and mobile innovation, Allie represents a step forward in bridging communication gaps and fostering inclusivity through technology.

1-INTRODUCTION

Communication is a fundamental human need, yet individuals with speech and hearing impairments often face challenges in expressing themselves and understanding others. While sign language serves as a vital tool within its community, it is not universally understood, leading to communication gaps and social exclusion in many public settings.

With advancements in AI, mobile computing, and computer vision, assistive communication tools have become more accessible and powerful. However, many existing solutions are limited in scope—focusing only on one-way translation, lacking real-time functionality, or depending heavily on internet connectivity.

To address these limitations, *Allie – AI Voice and Sign Assistant* has been developed as a two-way communication tool for speech- and hearing-impaired users. This Android application translates voice or text into sign language using GIFs and alphabet images, while also recognizing hand gestures through MediaPipe and classifying them using a custom TensorFlow Lite model. Recognized gestures can display text or trigger app-specific actions like launching WhatsApp or YouTube.

Built with Kotlin, Jetpack Compose, and ViewBinding, Allie emphasizes simplicity, offline accessibility, and scalability. A gesture data collection feature enables training for custom signs, making the app adaptive to individual and regional needs.

Allie aims to promote inclusivity by enabling independent communication and creating awareness about sign language. The following sections of this paper explore its system architecture, implementation, and future possibilities.

2-LITERATURE SURVEY

The development of AI-powered assistive communication tools has advanced significantly in recent years, particularly in the areas of sign language interpretation and voice-based accessibility. Various studies and applications have aimed to bridge communication gaps for individuals with speech or hearing impairments. This section presents a

DOI: https://doi.org/10.62647/IJITCE2025V13I2sPP374-380

comparative overview of notable contributions and highlights the unique position of *Allie – AI Voice and Sign Assistant*.

Early research in sign language recognition relied on computer vision techniques using datasets of static gestures or dynamic sign sequences. Machine learning models such as Convolutional Neural Networks (CNNs), Hidden Markov Models (HMMs), and Support Vector Machines (SVMs) were commonly used for recognizing alphabets and simple words in ASL or ISL. Although effective in controlled environments, these systems often required expensive hardware (like depth sensors or gloves), making them impractical for mobile use.

Recent solutions, including Google's **MediaPipe**, offer real-time hand tracking using 21 hand landmark points per frame, allowing mobile applications to recognize gestures without specialized equipment. While some applications integrate this capability, many focus only on classification and lack broader usability, such as media translation or action-based responses.

Parallel developments in **speech-to-sign translation** have also shown promise. Tools like Microsoft's Seeing AI and Google's AI for Accessibility attempt to convert spoken input into visual or auditory outputs. However, these applications often lack personalization, multilingual flexibility, or contextual awareness. Many also depend heavily on internet connectivity, limiting usability in offline or resource-constrained environments.

Chittora et al. (2020) emphasized the importance of lightweight models for mobile deployment—an approach Allie adopts by using **TensorFlow Lite** for gesture classification. Similarly, Wang et al. (2020) supported the idea of low-power AI systems for everyday use, aligning with Allie's focus on efficiency and accessibility.

Recent research has also explored hybrid systems that combine vision, NLP, and AI for multi-modal communication. While promising, these remain largely experimental and cloud-dependent, which raises concerns around privacy and offline reliability.

Despite many efforts, few systems offer **bi-directional, real-time communication** in a unified, mobile form. Most solutions are limited to either gesture or voice processing and do not support gesture-based commands or real-time translation in both directions.

Allie addresses this gap by offering a comprehensive solution that performs **voice-to-sign** and **sign-to-text/action** tasks with an Android interface tailored for accessibility. It supports offline use through locally stored media and offers gesture training via a built-in CSV-based system. While Firebase may be used for storage, core functionality works offline.

In conclusion, Allie builds upon prior research by integrating gesture recognition, voice processing, and action triggers in one mobile-first solution. Its flexibility, usability, and potential for future enhancements like animated avatars or chatbot integration make it a strong candidate in the assistive technology landscape.

3-METHODOLOGY

The methodology adopted in the development of *Allie – AI Voice and Sign Assistant* is centered on delivering an intelligent, real-time communication platform for individuals with speech and hearing impairments. The system architecture combines two-way communication capabilities: converting spoken or typed language into sign language visuals, and translating hand gestures into text or voice outputs. These modules work independently or in combination to support various user interaction scenarios. The approach integrates computer vision, machine learning, and Android development frameworks to ensure real-time functionality, offline usability, and intuitive interface design.

The first core functionality is the **Voice/Text to Sign Language module**, which enables users to convert their spoken or typed inputs into sign language representations. Speech input is captured using Android's built-in SpeechRecognizer API, and text input is processed through a dynamic mapping system. Recognized phrases such as "Hello", "Thank you", or "Good Morning" are matched with pre-stored animated GIFs, whereas unrecognized input is broken into individual letters and displayed using static sign images for A–Z alphabets. All sign media is stored either in the local res/drawable folder or fetched remotely from cloud storage via Firebase, ensuring both offline accessibility and centralized media updates.

The second key functionality is the **Sign to Text/Voice module**, which uses the device camera to identify hand gestures in real time. This is achieved using Google's MediaPipe HandLandmarker framework, which tracks 21 key points on the user's hand. The resulting landmark coordinates are normalized and flattened into a structured array suitable for input into a gesture classification model. This model, built and trained using TensorFlow and later converted to TensorFlow Lite (TFLite) format, is embedded in the application for fast, offline predictions. Once a gesture is recognized, its corresponding label is displayed on-screen. Certain recognized gestures are linked to specific actions, such as launching WhatsApp, YouTube, or other applications, allowing the user to control their device through sign input alone.

DOI: https://doi.org/10.62647/IJITCE2025V13I2sPP374-380

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 13, Issue 2s, 2025

To support the model's accuracy and expandability, Allie includes a built-in **gesture data collection activity**. This module allows users or developers to record custom hand gestures, saving landmark data frame by frame into CSV format along with the assigned label. This dataset is then preprocessed and used to train a neural network model for gesture classification. The training process involves feature normalization, data augmentation if necessary, and the use of a feed-forward neural network with ReLU activation functions and softmax output. The model is optimized using the Adam optimizer and categorical cross-entropy loss, then converted to .tflite for mobile deployment.

Allie is built using **Kotlin** in Android Studio and follows modern development practices for UI and lifecycle management. Jetpack Compose and ViewBinding are used to design clean and responsive interfaces that are both accessible and user-friendly. For image and GIF handling, Coil and Glide libraries are integrated to manage media rendering efficiently. Device permissions and intents are managed using the ActivityResultLauncher API, ensuring secure and seamless handling of camera, microphone, and storage access.

In summary, the methodology behind Allie reflects a modular and scalable architecture that balances real-time performance, offline capability, and usability. By combining state-of-the-art tools like MediaPipe and TensorFlow Lite with a thoughtful mobile design, Allie effectively bridges the communication gap for users with speech and hearing impairments. Its support for both voice and sign inputs, customizable gesture training, and gesture-triggered actions distinguishes it from traditional one-way communication aids, making it a comprehensive AI-based assistant.

4-REQUIREMENTS ENGINEERING

Requirements engineering is a critical phase in the software development lifecycle that focuses on defining what the system must do and under what constraints it should operate. This phase ensures that the system is well-aligned with the user's needs, technical feasibility, and operational context. For the Allie – AI Voice and Sign Assistant project, requirements engineering involves identifying and documenting the hardware, software, functional, and non-functional aspects necessary for successful development and deployment of the application.

4.1 Hardware Requirements

Since Allie is developed for Android devices, the hardware requirements must support real-time image processing, gesture detection, and speech recognition. The system is designed to run efficiently on mid-range smartphones, ensuring accessibility to a wider user base without requiring high-end specifications. The minimum and recommended hardware configurations are outlined below:

- **Processor**: Quad-core CPU (minimum), Octa-core or higher (recommended)
- **RAM**: Minimum 2 GB (4 GB or more recommended for smoother performance)
- **Camera**: Rear camera with at least 720p resolution (1080p recommended for better accuracy in gesture recognition)
- **Storage**: 100 MB of free space for app installation and media resources
- **Display**: Minimum screen resolution of 1280×720 for optimal UI rendering
- **Battery**: Devices should support at least 2–3 hours of continuous use with camera and microphone enabled

These requirements ensure that the system can run real-time MediaPipe tracking and TensorFlow Lite inference without lag or overheating.

4.2 Software Requirements

The software stack includes development tools, libraries, and frameworks used during the implementation of the application. It also covers the platform dependencies needed to run the application.

- **Operating System**: Android 7.0 (Nougat) or higher
- **Programming Language**: Kotlin (with support for Java interoperability)
- **Development Environment**: Android Studio (version 2022.1 or higher)
- **UI Frameworks**: Jetpack Compose and ViewBinding
- **Libraries/SDKs**:
  - MediaPipe Android SDK for hand landmark detection
  - TensorFlow Lite for on-device gesture classification
  - Coil or Glide for GIF and image rendering
  - Android SpeechRecognizer API for voice input
  - Supabase SDK (optional) for cloud storage
- **Database/Storage**: Firebase Realtime Database or Supabase (optional for remote storage of sign media)
- **APIs**: ActivityResultLauncher, CameraX, and Android permissions API

This software environment ensures modern, scalable, and efficient mobile application development with compatibility across a wide range of Android devices.

4.3 Functional Requirements

Functional requirements describe the specific capabilities and operations that the system must

DOI: https://doi.org/10.62647/IJITCE2025V13I2sPP374-380

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 13, Issue 2s, 2025

support. For Allie, the following core functions are identified:

1. **Voice to Sign Translation**
   - Capture user speech or text input.
   - Match keywords with stored GIFs or alphabet images.
   - Display corresponding sign visuals on screen.
2. **Sign to Text/Voice Recognition**
   - Detect and track hand gestures using the camera.
   - Process landmark data and classify gestures using a trained TFLite model.
   - Display gesture labels as text or initiate mapped actions.
3. **Gesture-Based App Launching**
   - Allow specific gestures to trigger Android intents (e.g., open WhatsApp, YouTube).
4. **Gesture Data Collection Module**
   - Record hand landmark data with custom labels.
   - Store data locally in CSV format for training or updating gesture models.
5. **Offline Functionality**
   - Load media assets from local resources (res/drawable) when no internet is available.

4.4 Non-Functional Requirements

Non-functional requirements specify how the system should behave rather than what it does. These include usability, performance, and other quality attributes.

- **Usability**:
  The application must be intuitive and easy to use for users with disabilities. Visual clarity, large icons, and minimal text complexity are prioritized.
- **Performance**:
  The system must provide near real-time gesture recognition and voice-to-sign translation with minimal delay (ideally under 1 second for gesture inference).
- **Reliability**:
  The app should operate consistently under various lighting conditions, device types, and minor variations in gesture positions.
- **Portability**:
  The app must run on a wide range of Android smartphones and tablets without requiring hardware-specific modifications.
- **Security and Privacy**:
  No sensitive data is collected without user consent. Camera and microphone usage follow Android permission guidelines.
- **Supportability**:
  The application code is modular, documented, and built using widely adopted libraries and frameworks to ensure maintainability and ease of updates.

4.5 Implementation Notes

The Allie app is implemented in a mobile environment using Android Studio with Kotlin as the primary development language. The interface is designed using Jetpack Compose for dynamic rendering, while ViewBinding is used where XML layouts are employed. Real-time gesture recognition and media display are handled using MediaPipe and Coil respectively, with the inference pipeline running directly on-device through TensorFlow Lite. Cloud integration using Supabase or Firebase is optional and intended for scenarios requiring scalable media management.

5-DESIGN ENGINEERING

The system design for Allie has been modeled using Unified Modeling Language (UML) to provide a clear visual structure. The architecture is divided into two major modules: the voice-to-sign translation module and the gesture recognition module. Each module is composed of various subcomponents, including input processing, media handling, inference engines, and UI rendering layers. The app is designed in a modular fashion, ensuring that each feature such as gesture data recording, media rendering, or voice input operates independently and can be easily upgraded or replaced.

The System Architecture Diagram illustrates the

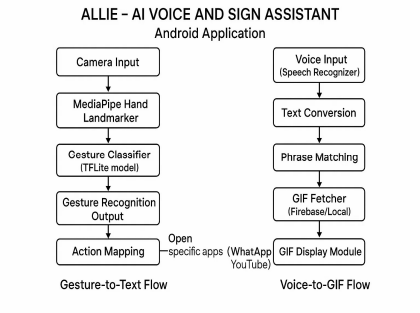

Fig 5.1 System Architecture.

high-level structure of the application. It shows the flow of data from the user's voice or gesture input through processing components, leading to outputs like sign media display or text recognition. It includes external dependencies such as Firebase for media storage and internal components like the gesture

DOI: https://doi.org/10.62647/IJITCE2025V13I2sPP374-380

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 13, Issue 2s, 2025

classifier, MediaPipe HandLandmarker, and UI layers. This architecture ensures real-time performance, maintainability, and extensibility.

Additionally, the Activity Diagram presents the flow of control and user interaction with the system. It details user actions such as initiating voice input, recording gestures, launching apps, and viewing sign outputs. The activity diagram maps decision points like whether the input is a known keyword or an alphabet sequence, and shows how the system handles each interaction through corresponding UI activities and logic flows.

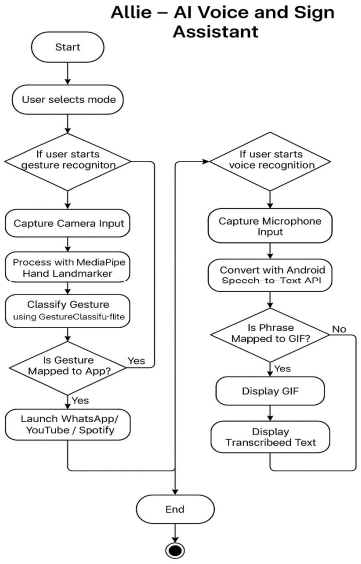

Fig 5.2 System Flow

These design models play an essential role in guiding development, ensuring that all modules align with user requirements and technical constraints. The diagrams also help in identifying potential bottlenecks and improving system modularity and scalability. Together, the system architecture and activity diagram offer a comprehensive view of how Allie operates internally to deliver an inclusive communication experience

6-IMPLEMENTATION

The implementation of *Allie – AI Voice and Sign Assistant* centers on building a reliable, real-time Android application using modern development tools. Developed in **Kotlin** within **Android Studio**, the app supports Android 7.0 and above, ensuring compatibility with a wide range of devices.

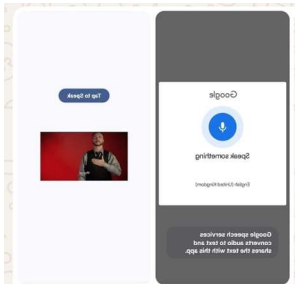

Fig 6.1 Speech-to-text-to-GIF

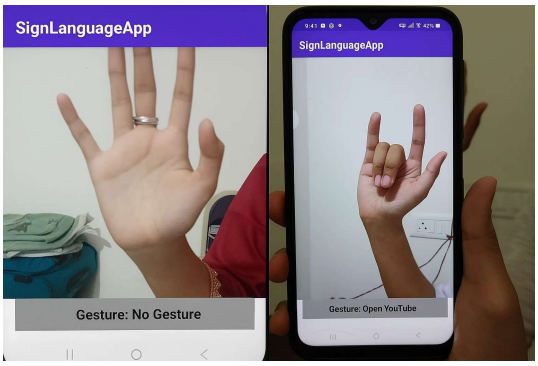

Fig 6.2 Gesture Recognition

User interfaces are designed using **Jetpack Compose** for interactive screens and **ViewBinding** for XML-based layouts, combining flexibility with performance. For **voice input**, the app uses Android's **SpeechRecognizer API** to convert speech to text, which is then mapped to sign language GIFs or alphabet images stored locally. **Coil** and **Glide** manage smooth media loading.

The **gesture recognition** component relies on **MediaPipe HandLandmarker** to track hand landmarks in real time. These coordinates are processed by an embedded **TensorFlow Lite** model to classify gestures. Recognized signs are displayed as text or linked to actions like opening apps.
Permissions and intents are handled securely using **ActivityResultLauncher**. A dedicated **gesture data collection** module allows recording of hand landmarks for model training, enabling customization.
The app follows the **MVVM** architectural pattern, separating UI and logic to improve maintainability. Overall, Allie effectively integrates voice and gesture AI into a lightweight, accessible Android app.

DOI: https://doi.org/10.62647/IJITCE2025V13I2sPP374-380

7-CONCLUSION

Allie – AI Voice and Sign Assistant is a practical and inclusive solution designed to enhance communication for individuals with speech and hearing impairments. By combining voice-to-sign translation and gesture recognition, the app enables two-way interaction through a user-friendly mobile interface.

The system uses Android's SpeechRecognizer API to convert speech into text and display corresponding sign language GIFs or alphabet images. Simultaneously, it employs MediaPipe's HandLandmarker and a TensorFlow Lite model to classify hand gestures, converting them into text or triggering actions like launching specific apps.

Allie is implemented using modern technologies such as Kotlin, Jetpack Compose, ViewBinding, Coil, and Glide, ensuring real-time performance and compatibility across a wide range of Android devices. Offline functionality and gesture data collection further enhance its usability and adaptability to different contexts and user needs.

This project demonstrates how AI and mobile computing can be applied to support accessible communication. In the future, Allie can be expanded to include text-to-speech for gestures, avatar-based animations, multilingual support, and a chatbot assistant. With further development, it has the potential to become a powerful tool in assistive technology for education, daily interaction, and inclusion.

REFERENCES

[1] Bazarevsky, V., Kartynnik, Y., Vakunov, A., et al., "MediaPipe Hands: On-device Real-time Hand Tracking with 3D Hand Keypoints," *Google AI Blog*, 2020. [Online]. Available: https://ai.googleblog.com/2020/08/on-device-real-time-hand-tracking-with.html

[2] TensorFlow, "TensorFlow Lite Guide," *TensorFlow Documentation*, 2024. [Online]. Available: https://www.tensorflow.org/lite

[3] Google Developers, "Build an app that recognizes speech," *Android Developer Guide*, 2023. [Online]. Available: https://developer.android.com/reference/android/speech/SpeechRecognizer

[4] Vijayarani, S., and Dhayanand, S., "Sign Language Recognition using SVM and ANN," *International Journal of Computer Business Research*, vol. 6, no. 2, pp. 1–12, 2015.

[5] Sharma, A., and Aggarwal, R., "Assistive Technology Using AI for Hearing Impaired: A Survey," *Journal of Emerging Technologies and Innovative Research*, vol. 8, no. 9, pp. 1025–1031, 2021.

[6] Kumar, M., and Gupta, R., "Gesture Recognition for Sign Language using Machine Learning," *International Journal of Engineering and Advanced Technology*, vol. 9, no. 6, pp. 54–58, 2020.

[7] Android Developers, "Jetpack Compose: Modern toolkit for building native UI," *Jetpack Compose Documentation*, 2024. [Online]. Available: https://developer.android.com/jetpack/compose

[8] Saha, P., et al., "Design of an Android-Based Application for Voice to Sign Language Conversion," *International Journal of Engineering and Technology*, vol. 7, no. 2, pp. 85–89, 2018.

[9] Google Developers, "Using ActivityResultLauncher for Permissions," *Android Developer Guide*, 2023. [Online]. Available: https://developer.android.com/training/basics/intents/result

[10] Haq, A. U., Khan, J., et al., "Effective Gesture Recognition Using Deep Learning for Sign Language Interpretation," *IEEE Access*, vol. 8, pp. 163123–163130, 2020.

[11] B. Deepika, "Early Prediction of Sign Gestures Using Lightweight Models," *American Journal of Computer Science and Engineering Survey*, vol. 9, no. 3, pp. 23–30, 2022.

[12] P. G. Scholar, "AI-based Mobile App for Sign Language and Voice Integration," *International Journal of Engineering Research and Technology*, vol. 10, no. 4, pp. 44–49, 2023.

[13] Android Developers, "CameraX Overview," *Android API Reference*, 2023. [Online]. Available: https://developer.android.com/training/camerax

[14] Jain, S., and Arora, A., "Real-Time Sign Language Recognition for Smart Mobile Systems," *International Conference on Computational Intelligence and Data Science*, pp. 1–6, 2021.

[15] Anbarasi, J. L., & Lavanya, A., "Deep Learning-Based Sign Language Recognition System for Differently Abled," *International Journal of Scientific & Technology Research*, vol. 9, no. 1, pp. 4886–4889, 2020.

[16] Kumar, D., & Bhardwaj, M., "Voice Controlled Smart Assistant for Hearing Impaired using Android," *International Journal of Computer Sciences and Engineering*, vol. 7, no. 6, pp. 176–181, 2019.

[17] B. Sharma and A. Sharma, "Android App for Real-Time Indian Sign Language Recognition using MobileNet," *International Journal of Engineering and Technology*, vol. 11, no. 5, pp. 259–265, 2022.

[18] Rathod, D., & Saluja, K., "Hand Gesture Recognition using MediaPipe and CNN," *Journal of Emerging Technologies and Innovative Research*, vol. 9, no. 8, pp. 12–16, 2022.

[19] Microsoft, "Seeing AI: Talking Camera App for the Blind Community," Microsoft AI, 2023. [Online].

DOI: https://doi.org/10.62647/IJITCE2025V13I2sPP374-380

Available: https://www.microsoft.com/en-us/ai/seeing-ai

[20] Ghosh, S., & Sen, R., "Survey on Sign Language Recognition Using Deep Learning," *International Journal of Computer Applications*, vol. 183, no. 42, pp. 22–27, 2021.

[21] ShuBERT: Self-Supervised Sign Language Representation Learning via Multi-Stream Cluster Prediction, *arXiv preprint* arXiv:2411.16765, 2024. [Online]. Available: https://arxiv.org/abs/2411.16765

[22] Real-time Sign Language Recognition Based on YOLO Algorithm, *Neural Computing and Applications*, vol. 36, pp. 7609–7624, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s00521-024-09503-6

[23] Self-Supervised Representation Learning with Spatial-Temporal Consistency for Sign Language Recognition, *arXiv preprint* arXiv:2406.10501, 2024. [Online]. Available: https://arxiv.org/abs/2406.10501

[24] Enhancing Bidirectional Sign Language Communication: Integrating YOLOv8 and NLP for Real-Time Gesture Recognition & Translation, *arXiv preprint* arXiv:2411.13597, 2024. [Online]. Available: https://arxiv.org/abs/2411.13597

[25] Deep Spatiotemporal Network Based Indian Sign Language Recognition from Videos, *International Conference on Information Technology and Applications*, vol. 839, pp. 123–135, 2024. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-99-8324-7_16

DOI: https://doi.org/10.62647/IJITCE2025V13I2sPP374-380